Apports des méthodes d'optimisation et du calcul haute performance à la théorie de la modélisation et de la simulation : application à la gestion des ressources halieutiques

Nicolas Poiron-Guidoni

UNIVERSITE DE CORSE-PASCAL PAOLI
ECOLE DOCTORALE ENVIRONNEMENT ET SOCIETE
UMR CNRS 6134 (SPE)

Thèse présentée pour l'obtention du grade de

DOCTEUR EN INFORMATIQUE

Soutenue publiquement par

NICOLAS POIRON-GUIDONI

Le 03 DECEMBRE 2021

Apports des méthodes d'optimisation et du calcul haute performance à la théorie de la modélisation et de la simulation : application à la gestion des ressources halieutiques

---

Directeurs :

M Paul-Antoine BISGAMBIGLIA, Pr, Université de Corse
M Paul-Antoine BISGAMBIGLIA, Dr-HDR, Université de Corse

Rapporteurs :

M Patrick SIARRY, Pr, Université de Paris-Est Créteil
M Sébastien VEREL, Pr, Université du Littoral côte d'Opale

Jury

M Bruno BACHELET, Dr-HDR, Université Clermont Auvergne
M Dominique BARBOLOSI, Pr, Université Aix-Marseille
M Patrick SIARRY, Pr, Université Paris-Est Créteil
M Sébastien VEREL, Pr, Université du Littoral côte d'Opale
M Bastien POGGI, Dr, Université de Corse
M Paul-Antoine BISGAMBIGLIA, Dr-HDR, Université de Corse
M Paul-Antoine BISGAMBIGLIA, Pr, Université de Corse

REMERCIEMENTS

Je remercie en tout premier lieu le Pr Paul-Antoine Bisgambiglia pour m'avoir tout d'abord accueilli en stage de L3 au sein de l'UMR SPE, puis pour sa proposition d'alternance de master et enfin pour cette thèse. Choses qui ont été possibles grâce au soutien infaillible de mon co-directeur de thèse, le Dr-HDR Paul-Antoine Bisgambiglia (non je n'ai pas copié/collé le mauvais nom...). Les discussions n'ont pas manqué et ont toujours permis à la fois d'avancer et de passer de bons moments. Un grand merci !

J'exprime également ma gratitude envers les Pr. Patrick Siarry et Sébastien Verel pour avoir accepté d'être rapporteurs sur ce travail. Merci également aux membres du jury : Pr Bruno Bachelet, Pr Dominique Barbolosi, Dr Bastien Poggi, jury également constitué de mes directeurs et rapporteurs.

Continuons avec David Araujo que j'ai connu pendant mon BTS. Il m'a accompagné pendant ce fameux stage, alternance, puis en tant qu'ingénieur d'étude et enfin doctorant au laboratoire. Nous n'oublierons jamais les réflexions matinales en regardant les pokémons légendaires de l'IUT.

Jean-Pierre Jouault a un parcours presque similaire puisqu'il m'a accompagné depuis la L3 jusqu'à cette fin de thèse, d'abord en cours et alternance, puis en tant qu'ingénieur d'étude, mais surtout en tant qu'ami.

François-Marie Manicacci, frère de pâtes au kinder country, étudiant avec moi en Master puis également doctorant. C'est certainement avec lui que l'on a eu les idées les plus prometteuses de nos vies, mais nous n'en dirons pas plus par confidentialité pour un potentiel futur dépôt de brevet dans le domaine de la robotique et l'aide à la personne. Un grand merci à eux trois pour tous ces bons moments !

Parlons maintenant de ma famille d'adoption. À commencer par Paul-Henri Martelloni doctorant dès mon arrivée en alternance. Il m'a accompagné à la fois en informatique, mathématiques, dans nos mauvaises idées, au café, à la montagne, mais surtout pour les phrases à double sens toujours très poétiques. Encore plus associable que moi quand on s'est connu, le café a su nous rapprocher des meilleures personnes qu'on a pu rencontrer. Parlons donc de l'homme à appeler quand il faut monter un meuble en kit... Guillaume Gerandi ! Heureusement il sait se rattraper, d'abord aux échecs quand on s'est connu

enfant, puis à la belote quand on s'est retrouvé en thèse. Comment ne pas parler également de notre mami, Anais Pannequin et son ballon moustachu. Toujours à prendre soin de nous avec ses petits plats et ses conseils, tant que vous ne la battez pas à la belote, elle devrait être un amour. Enfin... gare à vous si vous n'alignez pas les figures sur vos slides ou qu'elles sont trop vides !

Enfin, après des dizaines de pauses cafés prises en même temps pendant notre année de M2, j'ai fait la connaissance de celle qui deviendra ma petite sœur de thèse, Karina Meerpoel-Pietri. Toujours fidèle aux pauses cafés, mais également dans les mauvais plans (#laPince) qui resteront inoubliables, elle a pu me rendre fou avec ses traitements d'images, je me vengeais en ayant besoin de ses conseils de vie. Un énorme merci à toi pour ta patience, ta gentillesse, ton écoute et ton amitié indéfectible.

Merci toute la famille !

Merci également à mes parents, grand-parents et famille en général, pour m'avoir supporté de manière indéfectible depuis tout ce temps.
Un merci particulier à Claudie Fonteneau pour sa relecture !
Merci également à Marie-Jeanne Andreani pour sa traduction en Corse de mon résumé vulgarisé.

Un grand merci à Camille Rubio pour son amitié ! J'espère que ton histoire de patate durera éternellement !
Merci également à Alois Beck, en espérant que la pelle te montrera le chemin.
Merci à Alexandre Zamboni pour toutes les bonnes soirées, réflexions philosophiques et corrections ! Le monde n'oubliera jamais le K divin et la constante de Zamboni !
Merci également à David Perez, on se souviendra longtemps de mon retour de Toulouse ! Enfin...

Un merci particulier à Marie pour m'avoir supporté si longtemps et aidé à sa façon ! L'Allemagne s'en souvient. J'attends toujours la défaite !

Merci à Cédric pour tout le temps à rigoler ensemble et pour la semaine inoubliable lors de ma première conférence. Promis j'arrive bientôt et plus motivé que jamais, objectif 25k et SR.
Merci à Marc, bro a.k.a. petit p. vigoureux ! Merci pour ton amitié qui m'a beaucoup aidé ces 10 dernières années. Merci également pour ta relecture, tes blagues nulles, et pour tous nos délires de challenges insensés, mais passionnants à réaliser !
Merci également à Marc P, pour les réflexions incompréhensibles sous décalage horaire.

Merci également au PRANZU ! Le point de ralliement du bien manger à Corte à condition de ne pas être au régime.
Merci à la déesse de l'olympe et à la reine de chez Casanova pour leur fidèle contribution à mon taux de cholestérol sous forme de bruschette.
Merci également au Cyrnea et spécialement à Henry pour sa façon de doser bien particulière, mais appréciée.

Merci aux enseignants-chercheurs qui m'ont inspiré tels que la MCF Marie-Laure Nivet, la MCF Évelyne Vittori, le MCF Bastien Poggi et le chercheur CNRS Jean-Baptiste Filippi.
Merci également à Frédéric Bosseur pour sa gentillesse et réactivité à chaque fois qu'il y a eu besoin de calculs.

Merci enfin à tous ceux qui ont fait que mes années à Corte et au sein de l'UMR SPE sont certainement les meilleures que j'ai passées jusqu'à maintenant.

À Caro

Résumé

Le projet informatique (SiSU) de l'Unité Mixe de Recherche CNRS Science pour l'Environnement conçoit des méthodes d'aide à la décision pour aider à une meilleure gestion des systèmes complexes environnementaux. Ces travaux de thèse s'inscrivent dans ce contexte. Ils ont pour objectif d'étudier les apports de plusieurs types de méthodes informatiques afin d'améliorer nos connaissances sur les systèmes complexes et ainsi de fournir une aide à leur gestion en situation de fortes incertitudes. En effet, les systèmes complexes environnementaux ne peuvent pas toujours être connus et modélisés avec précision. C'est par exemple le cas en biologie halieutique où des méthodes de gestion doivent être proposées malgré un manque de connaissances sur le système observé, dans notre cas d'étude : la pêche côtière Corse. Nos premiers travaux ont porté sur la calibration de modèles, c'est-à-dire le recherche de valeurs de paramètres permettant à nos modèles de représenter au mieux la dynamique du système. Ils ont montré les limites des approches habituelles et la nécessité d'utiliser des approches probabilistes basées sur de grandes quantités de simulations. Elles apportent une aide précieuse quant à l'acquisition de connaissances, notamment en délimitant des ensembles de solutions. Ceux-ci peuvent alors être utilisés dans des méthodes d'optimisation robuste, voire d'optimisation robuste ajustable. Ces approches permettent non seulement de prendre en compte les incertitudes, mais également de quantifier la réduction d'incertitude que de nouvelles années de données pourront apporter, afin de proposer des stratégies de plus en plus précises à long terme. L'optimisation est donc utilisable efficacement à l'échelle des décideurs. Cependant, la petite pêche côtière Corse, est un système sur lequel agissent un grand nombre d'acteurs avec des comportements différents et difficilement prévisibles et contrôlables. L'optimisation ne semble pas adaptée à l'étude de cette échelle de par la quantité de paramètres et le nombre infini de transitions stochastiques engendrées. Pour cela, des méthodes basées sur l'apprentissage profond par renforcement ont été proposées. Ces approches nous ont permis dans un premier temps de proposer un modèle gérant à la fois décideurs et pêcheurs, les uns cherchant à réduire l'impact écologique, les autres à maximiser leurs gains. À partir de cela, nous avons pu montrer que de faibles connaissances suffisent pour la maximisation des gains des pêcheurs. De plus, cette approche, couplée à de l'optimisation, a permis d'obtenir des décisions d'instauration de quotas efficaces. Enfin, ce système nous a permis d'étudier l'impact de certains comportements individuels de maximisation des gains au détriment du respect des recommandations des décideurs. Il est alors apparu que des politiques de gestion efficaces et adaptées peuvent permettre de pallier l'impact écologique d'une quantité non négligeable de ces comportements. Ainsi, nous avons pu contribuer de manière théorique à élargir les domaines d'application de la théorie de la modélisation et de la simulation, proposer un ensemble d'outils d'optimisation et d'apprentissage automatique à la gestion de systèmes dynamiques partiellement observables, mais également applicative pour la problématique de la gestion de la pêche en Corse.

Summary

The computer science project (SiSU) of the CNRS Science for the Environment Joint Research Unit designs decision support methods to help better management of complex environmental systems. This thesis work is part of this context. They aim to study the contributions of several types of computer methods to improve our knowledge of complex systems and thus provide assistance in their management in situations of high uncertainty. Indeed, complex environmental systems cannot always be known and modeled with precision. This is for example the case in fisheries biology where management methods must be proposed despite a lack of knowledge on the observed system, in our case study : the Corsican coastal fishery. Our first work focused on the calibration of models, i.e. the search for parameter values allowing our models to best represent the dynamics of the system. They have shown the limits of the usual approaches and the need to use probabilistic approaches based on large quantities of simulations. They bring a precious help for the acquisition of knowledge, in particular by delimiting sets of solutions. These sets can then be used in robust optimization methods, or even in adjustable robust optimization. These approaches allow not only to take into account the uncertainties, but also to quantify the reduction of uncertainty that new years of data can bring, in order to propose more and more precise strategies in the long term. Optimization can therefore be used effectively at the level of decision makers. However, the small-scale coastal fishery in Corsica is a system in which a large number of actors act with different behaviors that are difficult to predict and control. Optimization does not seem adapted to the study of this scale because of the quantity of parameters and the infinite number of stochastic transitions generated. For this, methods based on deep reinforcement learning have been proposed. These approaches allowed us to propose a model that manages both decision-makers and fishermen, the former seeking to reduce the ecological impact, the latter to maximize their gains. From this, we were able to show that little knowledge is sufficient for the maximization of the fishermen's gains. Moreover, this approach, coupled with optimization, allowed us to obtain efficient quota decisions. Finally, this system allowed us to study the impact of certain individual behaviors of maximizing gains to the detriment of respecting the recommendations of the decision makers. It then appeared that effective and adapted management policies can help to mitigate the ecological impact of a significant amount of these behaviors. Thus, we were able to contribute in a theoretical way to broaden the application domains of the theory of modeling and simulation, to propose a set of optimization and machine learning tools for the management of dynamic systems partially observable, but also applicative for the problem of fisheries management in Corsica.

Riassuntu

Issi travagli di tesa anu per oghjettivu di studià u purtà di parechji genari d'arnesi infurmatichi per aiutà à gistisce situazione quandu e cuniscenze sò debbule è incerte. Hè u casu di e pesche, custì e ricumandazione devenu esse pruposte ben quellu sia difficiule di stimà a quantità di pesci di manera precisa. Sta prublematica hè dunque propria adatta à u nostru studiu.

Indè un primu tempu, i nostri travagli indè u duminiu di l'uttimisazione anu permessu di prupone mettuti novi da acquistà e cuniscenze nantu à e spezie studiate. Puru sottumesse sempre à l'incertitutine forte, c'anu permessu di prupone mettuti fundive per assicurà un impattu ecologicu debbule mantinendu una certa rentabilità ecunomica à i sfruttanti.

Ma, ste strategie sottuponenu un cuntrollu sanu nantu à a spluttazione. I cumpurtamenti individuali, per un dettu indè u duminiu di a pesca, sò disficiule à cuntrullà di manera precisa. È puru, e strategie pruposte da quelli chì dicidenu ponu ùn esse micca realizevule nantu à u terrenu. Da mudelizà di manera efficace stu fattu, hè statu necessariu d'aduprà d'altri genari di mettuti : l'amparera da u rinfurzimentu. C'hà permessu di fà vede ch'ellu ùn era micca sempre pussibule di realizà e precunisazione di quelli chì decidenu. Ci semu appughjati nantu à e poche infurmazione per pudè prupone e strategie efficace à tutte e scale, soprattuttu in accupiendu l'uttimisazione è l'amparera. Per compie, avemu pussutu studià l'impattu ch'elli anu l'individui chì ùn rispettenu micca i rigulamenti nantu à a sfruttera è prupone e strategie chì permettenu di gistisce stu prublema di manera efficace.

TABLE DES MATIÈRES

INTRODUCTION

La prise de décision est un processus complexe que nous, humains, expérimentons tous les jours à différentes échelles. Il repose sur des critères d'analyse d'une problématique, de ses enjeux, des choix possibles pour conduire à un choix final décidé et conscient visant à satisfaire un objectif. Le décideur peut alors aussi bien être un individu seul, ayant à prendre une décision de faible importance, ou un conseil organisé ayant à prendre des décisions d'ampleur nationale. La prise de décision revêt donc un aspect primordial dans la plupart des systèmes gérés par l'humain. Dans Legaré et al. (2003), les auteurs la décrivent comme pouvant être organisée suivant des protocoles plus ou moins stricts et précis comme les procédures à suivre pour aboutir à un diagnostic médical, individuel ou partagé. Les décisions humaines peuvent cependant souvent être influencées par les émotions Van Hoorebeke (2008) Coget et al. (2009) ou l'intuition Sinclair and Ashkanasy (2005) Salas et al. (2010). C'est un processus difficilement modélisable par une approche informatique.

L'aide à la décision, notamment via l'informatique décisionnelle et la recherche opérationnelle, vise à fournir aux décideurs le maximum d'informations et d'aides possibles afin de réaliser leur prise de décision de la manière la plus efficace et éclairée possible. Son but n'est donc pas de prendre la décision directement, mais d'éclairer l'humain, de clarifier sa vision d'un contexte souvent complexe, incertain, et à partir duquel les impacts des décisions sont peu prévisibles. La simulation est alors un formidable outil permettant l'évaluation des conséquences de la prise de décision, mais également l'utilisation directe d'autres méthodes, telles que l'optimisation via simulation ou l'apprentissage par renforcement. Elle nécessite toutefois un travail exigeant, souvent à la croisée de plusieurs domaines, comme c'est le cas en sciences environnementales, où informaticiens, écologues, biologistes et économistes doivent collaborer pour concevoir des outils et des modèles adaptés aux problématiques de chacun des domaines, souvent méconnues et peu intuitives pour les collaborateurs.

Dans un contexte de sur-exploitation, la gestion des ressources naturelles, particulièrement des ressources marines, est d'une importance capitale. Ces problématiques sont au cœur des recherches de notre laboratoire Sciences Pour l'Environnement (UMR SPE 6134). Protéger les populations et les habitats d'espèces considérées en danger par l'union internationale pour la conservation de la nature Hoffmann et al. (2008) ou surexploitées, nécessite la mise en place de politiques de régulation. Ces dernières sont

souvent difficiles à évaluer et à faire respecter, car elles impliquent des conflits d'intérêts entre scientifiques, gestionnaires et pêcheurs, entraînant une perte financière potentielle pour les professionnels. De plus, l'institut français de recherche pour l'exploitation de la mer a publié un rapport IFREMER (2019) montrant que la surexploitation touche environ un quart des stocks de poissons pêchés en France. Il est alors primordial de proposer rapidement des stratégies de gestion des stocks efficaces. De ces constats alarmants, découle une volonté de mettre en place des modèles bio-écnomiques focalisés sur les pêcheries. L'objectif de ces derniers est de pouvoir à la fois évaluer les conséquences environnementales d'un ou plusieurs métiers de pêche afin de proposer des mesures de gestion adaptées, et également de proposer des stratégies de pêche plus durables. Associer ces modèles bio-économiques avec l'optimisation et l'apprentissage permet non seulement une évaluation des conséquences de l'exploitation, mais également une modélisation des objectifs, ainsi qu'une proposition automatique de stratégies visant à les atteindre.

Contexte

Notre travail a été effectué au sein de l'*UMR CNRS 6134 SPE* qui est une unité habituée aux travaux pluridisciplinaires depuis maintenant de nombreuses années, comme en témoignent notamment ses travaux dans de nombreux projets :

— Énergies renouvelables (Mattei et al. (2006); Voyant et al. (2017)).

— Feux de forêts (Balbi et al. (2009); Bisgambiglia et al. (2017)).

— Ressources naturelles (Nothias et al. (2020)).

— Gestion et valorisation des Eaux en Méditerranée (Koeck et al. (2015)).

Le projet Simulation Informatique et Systèmes Ubiquitaires (SiSU), au sein duquel nous avons travaillé, est fondé sur la définition d'approches génériques de Modélisation et Simulation (M&S) pour l'étude de systèmes complexes. L'objectif est de les mettre en œuvre sur des problèmes liés à l'environnement afin d'apporter un éclairage sur des problèmes concrets émanant de la société ou de l'industrie, mais également de la recherche, au travers de modèles purement théoriques. Ainsi, les travaux s'orientent de plus en plus vers les méthodes d'aide à la décision et de recherche opérationnelle. C'est dans ce contexte que nos travaux de recherche s'inscrivent.

Une première thèse portant sur l'optimisation via simulation Poggi (2014) a déjà été réalisée. Elle visait principalement à l'incorporation de ces méthodes au sein d'un formalisme de modélisation et de simulation. Par la suite, Franceschini (2017) mène des recherches pour l'utilisation de systèmes multi-agents, travaux qui seront complétés par Martelloni (2021) [1] en y incorporant la notion de système cognitif dans l'esprit des agents, pour permettre d'intégrer de premiers mécanismes de prise de décisions. Ceux-ci pouvant être basés sur des méthodes d'apprentissage ou d'optimisation. Nos travaux viennent compléter ces approches en cherchant à identifier les complémentarités entre les méthodes d'optimisation et d'apprentissage en situation de faibles connaissances et de fortes incertitudes. Cette problématique s'appliquera notamment aux questions de gestion des pêches du programme PO-FEDER [2] *MoonFish*.

1. Thèse à paraître au moment de la rédaction de ce manuscrit
2. PO-FEDER : Programme opérationnel Fonds européen de développement régional

Problématiques et objectifs

La nature incertaine, difficilement contrôlable à grande échelle et dirigée par l'appât du gain des décisions individuelles humaines rend la problématique de gestion des ressources naturelles partagées particulièrement difficile. Quand une ressource naturelle est partagée entre différents exploitants, il est fréquent de voir apparaître des comportements gloutons, c'est-à-dire motivés par des gains directs importants, pensant principalement à maximiser les bénéfices individuels à court terme. Ces actions ont un impact très rapide sur l'ensemble de l'exploitation, conduisant à une baisse de productivité économique, mais également à un fort impact écologique. C'est la tragédie des biens communs mise en avant dans Hardin (1968). Le but des gestionnaires, est alors de proposer des restrictions, contrôles, stratégies d'exploitation, etc, permettant de concilier conservation écologique et gains économiques et d'atteindre un niveau d'exploitation optimal durable.

Ainsi, un grand nombre de travaux ont déjà été menés Ostrom (2008); Aguilera (2018) sur la gestion des systèmes de ressources communes de multiples façons comme par des études en laboratoire Kimbrough and Vostroknutov (2015) comme via des simulations Janssen (2002). Un grand nombre de facteurs d'influence ont pu être identifiés, comme la communication entre les individus Hackett et al. (1994) ou la considération du futur Brandt et al. (2012) et bien d'autres von der Osten et al. (2017).

En raison de la difficulté d'acquisition des données en sciences marines et des incertitudes qui y sont associées, le domaine des sciences halieutiques est actuellement très dépendant des approches basées sur l'expertise Chrysafi and Kuparinen (2016). Les données peuvent être exploitées, en tenant compte de leur imprécision, pour aboutir à des stratégies sûres. Des approches par optimisation ont pu être proposées Azadivar et al. (2002) mais, à notre connaissance, les méthodes d'apprentissage n'ont encore jamais été utilisées dans ce contexte. Ce travail est donc un premier pas démontrant l'intérêt de lier ces deux domaines.

Les problématiques et objectifs de notre travail se sont dessinés de façon incrémentale et ont été fortement dirigés par l'application. Dans un premier temps, le choix d'un modèle de dynamique de population de la littérature, puis notamment son calage, nous ont permis d'identifier la nature théorique du problème auquel nous aurions à faire face. Cette phase a mis en lumière un très important problème d'équifinalité lors du calage du modèle, dirigeant nos applications vers les méthodes permettant la prise de décision dans un contexte de fortes incertitudes. Celles-ci peuvent également se répercuter sur la calibration, de par le manque de données, ou leur incohérence, sur une espèce, rendant la calibration impossible. Des méthodes d'identification des incohérences et d'amélioration des données ont donc dû être développées.

Finalement, notre étude devra donc permettre d'identifier l'apport des méthodes d'optimisation et d'apprentissage par renforcement, qu'elles soient individuelles ou complémentaires, à la théorie de la modélisation et de la simulation, en situation de faibles connaissances et de forte incertitudes. Ces dernières pouvant être présentes dans un premier temps dans les données permettant la calibration, mais également à l'issue de cette phase. Les méthodes d'aide à la décision que nous proposerons devront donc les prendre en compte directement.

Organisation de ce mémoire

Dans le chapitre 1, nous introduirons les différentes méthodes informatiques utilisées pour étudier nos problématiques et objectifs. Nous nous concentrerons dans un premier temps sur l'optimisation, en rappelant les caractéristiques des problèmes d'optimisation puis en introduisant les différentes variantes d'optimisation utilisables en fonction des problématiques.

Le chapitre 2 commencera par présenter les différents modèles de dynamique de population et le choix de l'utilisation de l'un d'eux. Celui-ci sera ensuite étudié en détail, afin de définir le cadre théorique du problème, d'en dégager ses difficultés et les méthodes qui pourraient permettre de les gérer. Cette étude se fera, pour commencer, sur des données simulées que nous avons générées pour contrôler la validité de nos expériences et résultats. Néanmoins, nous montrerons également les limites d'un cas réel et proposerons des méthodes pour aider à les surpasser.

Une fois la phase de calage terminée, le chapitre 3 détaillera ce qu'il est possible de faire à partir de ces résultats. Nous nous intéresserons donc à l'optimisation robuste et à l'optimisation robuste ajustable, semblant parfaitement correspondre aux problématiques de gestion des incertitudes étudiées. Nous proposerons également une approche par caractérisation de l'espace des solutions au calage pour pallier certains problèmes et nous identifierons les limites de ces approches.

Cela nous mènera à l'utilisation de méthodes d'apprentissage. Dans le chapitre 4, nous nous intéresserons particulièrement aux problèmes de bandits manchots en introduisant d'abord les processus de décision markoviens et leurs différentes variantes permettant la modélisation et prise de décision dans des systèmes partiellement observables compétitifs.

Après avoir montré les apports de ces méthodes et leurs limites, nous nous intéresserons finalement aux méthodes d'apprentissage profond par renforcement dans le chapitre 5. Nous réutiliserons les environnements du chapitre précédent pour proposer une approche n'utilisant pas directement la phase de calage. Celle-ci se base sur l'observation de résultats de pêche individuelle, afin d'estimer efficacement les conséquences des actions, à la fois à l'échelle du pêcheur, comme à l'échelle du décideur. Ses résultats seront ensuite analysés en différentes situations, afin d'identifier l'impact de certains comportements individuels imprévisibles et d'aider à les gérer. Le tout dans le but de permettre de proposer des outils d'aide à la décision efficaces pour tous les acteurs de l'exploitation.

Enfin, les apports de nos travaux ainsi que leurs perspectives seront énoncés dans une conclusion générale.

Chapitre 1

DESCRIPTION DES PROCESSUS D'OPTIMISATION

Pour aider à la prise de décision dans un contexte où il est difficile d'acquérir des connaissances fiables sur le système réel, il est nécessaire de fournir des méthodes robustes capables d'intégrer la prise en compte d'incertitudes. Dans ce chapitre, nous allons donc présenter les concepts d'optimisation afin de mieux définir dans quelles conditions ces méthodes s'appliquent ainsi que leur utilité dans notre contexte.

De façon générale, l'optimisation est l'ensemble des méthodes mathématiques et/ou numériques permettant de résoudre les problèmes de minimisation ou maximisation de fonctions. Elle s'applique dans de très nombreux domaines tels que la biologie synthétique Naseri and Koffas (2020), la génétique Thomas et al. (2019), la cosmologie Haggag et al. (2017), l'énergétique Tian et al. (2018) Zakaria et al. (2020), la finance Soler-Dominguez et al. (2017) et bien d'autres. Un grand nombre de méthodes existent afin de gérer le plus de problèmes et situations possibles. L'optimisation globale considère la recherche de l'optimum global, plutôt qu'un optimum local, c'est-à-dire que nous ne chercherons pas simplement à converger vers une solution qui semble parfaite au regard de son voisinage proche, mais explorerons largement l'espace des solutions à la recherche de la meilleure possible. L'optimisation sous contrainte, elle, recentre la question autour de la gestion de contraintes, pouvant être exprimées par des fonctions mathématiques simples réduisant directement l'espace des solutions, ou moins directes et plus floues comme peuvent l'être l'intégration de connaissances expertes. L'optimisation multiobjectif, s'intéresse à la gestion des compromis entre différents objectifs. Elle est beaucoup utilisée dans les domaines de la décision multi-critères Ridha et al. (2021) afin d'identifier l'ensemble des compromis possibles, appelé front de Pareto, à un ensemble d'objectifs souvent contradictoires. Enfin, nous nous intéresserons à l'optimisation robuste et l'optimisation robuste ajustable, permettant de prendre en compte des incertitudes au sein du processus d'optimisation et donc de fournir des solutions plus adaptées à l'imperfection des connaissances sur le système étudié. Ces méthodes s'intéressent particulièrement à l'étude des pires cas afin d'assurer que le pire résultat sera toujours acceptable selon les critères fixés. Bien que principalement utilisées dans l'industrie Beyer and Sendhoff (2007), ces méthodes semblent parfaitement adaptées

à des questions environnementales où les incertitudes sont souvent importantes et où il est nécessaire de garantir la qualité de la solution retenue.

Nous commencerons donc par caractériser les problèmes d'optimisation afin de montrer l'utilité que ce type de méthodes peut avoir dans notre contexte. Nous montrerons l'intérêt et les limites des méthodes basées sur les gradients avant de montrer en quoi les métaheuristiques peuvent nous permettre de pallier la majorité des problèmes. Enfin, nous parlerons plus en détails de chacun des différents types d'optimisation qui nous intéresserons par la suite : globale, sous contraintes, multimodale, multiobjectif, robuste et robuste ajustable.

1.1 Caractérisation des problèmes d'optimisation

De façon générale, l'optimisation est un processus cherchant à déterminer le meilleur ensemble de paramètres possible sur un problème et des critères donnés. Ces critères peuvent être :

- des objectifs : fonctions de coûts f, dont les valeurs de retour doivent être minimisées ou maximisées ;
- des contraintes :
  - fonctions g représentant des contraintes d'inégalité à respecter ;
  - fonctions h représentant des contraintes d'égalité.

Partant de ce constat, l'optimisation peut s'appliquer à une infinité de problèmes tant que ceux-ci peuvent être formalisés de cette manière. Tous les problèmes ne sont cependant pas équivalents. Ils vont dépendre du nombre de transitions possibles associées à chaque état du problème et donc de leur complexité. Par conséquent, dans le cas où une seule transition est envisageable, les problèmes peuvent être modélisés/résolus par une machine de Turing déterministe en un temps polynomial. On parle alors de problème de classe P. Dans le cas contraire où plusieurs choix sont possibles, la résolution fait appel à une machine de Turing non déterministe et l'on parle alors de classe NP (*Nondeterministic Polynomial time*). Les calculs d'un problème P forment une suite, alors que ceux d'un problème NP forment un arbre. Il devient alors difficile d'en estimer le temps de résolution.

Cependant, et bien que la complexité des calculs soit différente, une machine de Turing non déterministe peut être simulée par une machine déterministe. De plus, dans le cas particulier où l'on déciderait de restreindre l'évolution des états au meilleur choix possible, on aurait alors une équivalence avec un problème de type P. La résolution se ferait alors en un temps polynomial. Cela pourrait être le cas avec une machine de Turing non déterministe qui aurait toujours effectué les bons choix.

La classe des problèmes NP-complet est un sous-ensemble de NP pour lesquels tout problème peut être réduit à tout autre de cette classe par réduction polynomiale. C'est-à-dire que pour un problème donné, il existe une fonction calculable en temps polynomial permettant de le transformer en un autre de cette classe Ruiz-Vanoye et al. (2011). Ils sont un sous-ensemble des problèmes NP-difficiles, aussi réductibles en temps polynomial par *many-one reduction*. Un problème NP-complet est toujours NP-difficile, c'est-à-dire qu'il n'existe pas de constante n telle que le temps de résolution soit borné par un polynôme de degré n. C'est généralement sur ce type de problèmes que s'appliquent les méthodes d'optimisation.

En plus de sa complexité, un problème peut être défini par un grand nombre de caractéristiques telles que son nombre d'objectifs et de contraintes ainsi que leur linéarité. La convexité représente le fait que la fonction objectif ainsi que l'ensemble des solutions soient convexes ou non. Cette caractéristique peut grandement impacter la méthode de résolution et la difficulté du problème.

On parle de problème dynamique quand une notion de temporalité intervient. Cela est typiquement le cas en optimisation par simulation de systèmes naturels. À l'inverse, un exemple de problème statique peut être de trouver le minimum d'une fonction mathématique classique comme la fonction de De Jong (section 6.5.1) : $f(x) = \sum_{i=1}^{n} x_i^2$.

On parle d'optimisation discrète lorsque les variables à optimiser ne peuvent prendre que des valeurs discrètes. L'optimisation combinatoire, est un sous-ensemble de l'optimisation discrète dans lequel les ensembles de définitions des variables sont bornés, constituant ainsi un nombre fini de solutions potentielles. On peut par exemple penser au problème du sac à dos (section 1.4). À l'inverse, un problème d'optimisation est dit continu lorsque ses variables peuvent prendre des valeurs continues sur des intervalles bornés ou non.

Ces caractéristiques sont résumées dans le tableau 1.1.

Caractéristique	Caractéristique opposée
Mono-objectif	multiobjectif
Linéaire	Non-linéaire
Convexe	Non-convexe
Statique	Dynamique
Continu	Discret
Unidimensionnelle	Multidimensionnelle
Contraints	Non-contraints
Déterministe	Stochastique

TABLE 1.1 – Caractéristiques des problèmes d'optimisation

Il existe un très grand nombre de méthodes d'optimisation Rao and Rao (2009). Ce nombre ne cesse d'augmenter d'année en année, il est donc nécessaire de les classifier. La figure 1.1 résume la classification des méthodes et donne quelques exemples.

Comme pour les machines de Turing, les méthodes d'optimisation peuvent être déterministes ou stochastiques.

Dans les premières, on peut par exemple citer la traditionnelle programmation linéaire Dantzig (2016) Wagner (1959), ou encore la descente de gradient Rumelhart et al. (1986) très utilisée en machine learning Bousquet and Bottou (2008) et pour des problèmes non linéaires.

Les méthodes basées sur les gradients consistent, à partir d'un point aléatoire dans l'espace des solutions, à calculer le gradient correspondant à la fonction objectif en ce point, et orienter la recherche dans la direction de la plus forte pente. Ces étapes sont alors réitérées jusqu'à satisfaction d'un critère d'arrêt. Ces méthodes ne s'appliquent que sur des problèmes d'optimisation continue. Elles ont l'avantage d'être très rapides et peuvent permettre de résoudre des problèmes à très grandes dimensions Ruder (2016) mais peuvent ne converger que vers un minimum local Dauphin et al. (2014). Elles peuvent également

être instables Burdakov et al. (2019) et sont peu adaptées à des problèmes comportant des contraintes bien que certaines contraintes d'inégalités puissent être gérées Chen et al. (2019).

Si pour les méthodes déterministes, les solutions obtenues à partir des mêmes conditions initiales seront toujours identiques, il n'en est pas de même pour les méthodes non-déterministes. Celles-ci vont dépendre des conditions initiales, du choix des générateurs aléatoires qui seront utilisés et de la graine [1] (*seed* en anglais) associée.

Dans le domaine de l'optimisation, des heuristiques ont été développées pour trouver des solutions acceptables, voire optimales, mais celles-ci sont spécifiques au problème à traiter. À l'inverse, les métaheuristiques peuvent s'appliquer facilement à toutes sortes de problèmes. Elles sont les algorithmes d'optimisation stochastique les plus connus et peuvent s'appliquer à toutes sortes de problèmes combinatoires, mais également aux problèmes continus.

Ces algorithmes font partie des méthodes d'optimisation approchée (voir figure 1.1) : elles ne peuvent pas garantir de toujours trouver l'optimum global, c'est-à-dire le point de l'espace des solutions qui satisfait le mieux les objectifs modélisés, d'un problème contrairement aux méthodes de résolution exacte. Ces méthodes sont cependant utilisées pour résoudre des problèmes spécifiques ou permettant une exploration exhaustive de l'ensemble des solutions, ce qui est impossible dans la plupart des cas du fait de la complexité des problèmes à traiter. Dans le cas de méthodes exactes, on peut parler de résolution plutôt que d'optimisation.

Les métaheuristiques se différencient des méthodes exactes par le fait qu'elles ne visent pas forcément à obtenir la solution optimale, mais une solution acceptable en un temps de calcul raisonnable. Elles peuvent être séparées en deux groupes :

— Les métaheuristiques de voisinage qui n'utilisent qu'une solution sur laquelle des transformations seront appliquées afin de l'améliorer.

— Les métaheuristiques distribuées, qui se basent sur un ensemble de solutions, appelé population, qui seront modifiées à chaque itération afin de les améliorer au regard de la fonction d'évaluation.

Le principe consiste à faire évoluer de manière aléatoire un ensemble de solutions en n'en conservant que certaines, souvent les meilleures et en s'en servant pour en générer de nouvelles plus performantes, ceci dans l'optique de trouver des optimums (voir figure 1.5).

Toutes ces méthodes, qui sont partiellement ou totalement guidées par des processus stochastiques, permettent, lorsque l'on considère un problème NP-difficile, de ne pas avoir à explorer toutes les possibilités. Nous nous intéresserons donc particulièrement aux méthodes basées sur les gradients, montrerons leur intérêt et limites avant de développer les métaheuristiques.

1. Nombre ou ensemble de nombres utilisé pour initialiser un générateur de nombre pseudo-aléatoire. Fixable par l'utilisateur pour aider à la reproductibilité numérique, ou déterminée automatiquement en fonction de l'état du système au moment de l'initialisation.

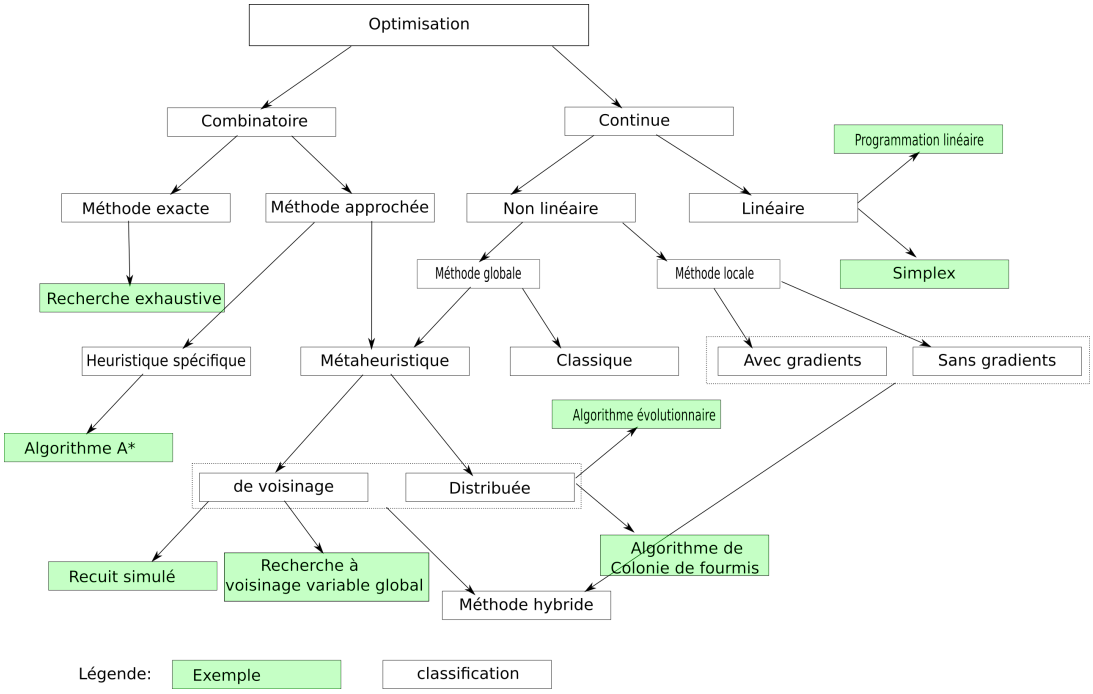


FIGURE 1.1 – Classification des méthodes d'optimisation, inspirée de Siarry (2014)

1.2 Méthodes d'optimisation basées sur les gradients

Les algorithmes basés sur les gradients permettent de minimiser une fonction objectif $f(x)$ avec x les paramètres d'un modèle, généralement $x \in R^n$, en mettant à jour les paramètres dans la direction opposée du gradient de la fonction objectif $\nabla_x f(x)$. La fonction objectif doit donc être différentiable. Quand le gradient est impossible à calculer de manière analytique, comme c'est souvent le cas en optimisation via simulation, la méthode des différences finies permet de l'approximer Zavriev (1993). La méthode historique Cauchy et al. (1847), appelée algorithme du gradient, ou descente de gradient, ne permet que converger vers un optimum local.

De nombreuses variantes de cet algorithme existent et il a été amélioré au cours du temps. Elles sont aujourd'hui extrêmement utilisées pour l'optimisation des paramètres de réseaux de neurones lors d'apprentissage Ruder (2016) par exemple. Les méthodes basées sur les gradients sont donc extrêmement utilisées en matière d'ajustement de courbe. On peut alors distinguer trois types de méthodes de gradients selon qu'elles utilisent tout ou partie des données à disposition pour calculer le gradient.

1.2.1 Descente de gradient par lot

En utilisant toutes les données à disposition, on parle, de descente de gradient par lot, ou en anglais, de *batch gradient descent*, ou *vanilla gradient descent*. Comme nous devons calculer les gradients pour l'ensemble du jeu de données afin d'effectuer une seule mise à jour, cette méthode peut être lente et est

irréalisable pour les jeux de données qui ne tiennent pas en mémoire. Elle ne permet pas non plus de mettre à jour notre modèle en ligne, c'est-à-dire avec de nouveaux exemples à la volée. C'est pourquoi elle n'est pas privilégiée dans le domaine de l'apprentissage.

En revanche, elle est très utilisée en optimisation. De nombreux algorithmes ont pu être proposés au cours du temps afin de pallier les problèmes de la descente de gradient historique. Le plus utilisé semble être l'algorithme de Levenberg-Marquardt Gavin (2019) et les algorithmes basés sur les régions de confiance Yuan (2000). Dans le cas de fonction vectorielle, ces algorithmes se basent généralement sur les matrices jacobiennes. La matrice jacobienne d'une fonction vectorielle à plusieurs variables généralise le gradient d'une fonction scalaire en plusieurs variables. Ces méthodes, semblent optimales quand un problème peut être mis sous la forme d'un simple problème des moindres carrés Lawson and Hanson (1995). Les problèmes de moindres carrés apparaissent dans le contexte de l'ajustement d'un modèle mathématique paramétré à un ensemble de points de données en minimisant un objectif exprimé comme la somme des carrés des erreurs entre la fonction du modèle et un ensemble de points de données.

Elle est également très utilisée en optimisation continue non contrainte Barzilai and Borwein (1988) ou peu contrainte Chen et al. (2019). Son instabilité nécessite cependant des études approfondies pour mener à des méthodes de plus en plus stables Burdakov et al. (2019) mais sa stabilité peut difficilement être garantie pour tous les problèmes.

1.2.2 Descente de gradient stochastique

À l'inverse des méthodes précédentes, il est possible de calculer le gradient pour seulement un exemple à la fois. On parle alors de descente de gradient stochastique. Elle peut être considérée comme une approximation stochastique de l'optimisation par descente de gradient, puisqu'elle remplace le gradient réel (calculé à partir de l'ensemble des données) par une estimation de celui-ci (calculée à partir d'un sous-ensemble des données sélectionné de manière aléatoire). En particulier dans les problèmes d'optimisation à haute dimension, cela réduit la charge de calcul, permettant des itérations plus rapides en échange d'un taux de convergence plus faible Bottou and Bousquet (2011). Comme nous ne considérons qu'un seul exemple à la fois, le coût fluctuera au fil des exemples, ne diminuera pas nécessairement à chaque fois, mais le processus sera efficace sur le long terme.

1.2.3 Descente de gradient par mini-lots

Cette méthode propose d'utiliser un sous-ensemble des données à chaque itération. De cette façon il réduit la variance des mises à jour des paramètres, ce qui peut conduire à une convergence plus stable. Il peut également utiliser les optimisations matricielles, comme les matrices jacobiennes utilisées dans l'algorithme de levenberg-marquardt, hautement optimisées. La descente de gradient par mini-lots est généralement l'algorithme de choix lors de l'entraînement d'un réseau de neurones et le terme de descente de gradient stochastique est généralement employé également lorsque des mini-lots sont utilisés.

Récemment, un très grand nombre de méthodes ont été développées dans ce contexte. L'algorithme le plus utilisé est ADAM Kingma and Ba (2014) mais certains auteurs proposent de l'améliorer notamment

en ajoutant une gestion différente de l'élan (*momentum*) comme l'élan de Nesterov Dozat (2016) ou encore en permettant une moindre influence des hyperparamètres comme le taux d'apprentissage Liu et al. (2019).

1.2.4 Forces et faiblesses des méthodes basées sur les gradients

Ces méthodes sont extrêmement efficaces pour trouver un optimum local d'une fonction convexe différentiable même en très haute dimension. Par exemple, Burdakov et al. (2019) réalisent des benchmarks d'une méthode qu'ils proposent sur des problèmes atteignant le million de dimensions. Leur stabilité n'est cependant pas toujours garantie et le point de départ peut avoir une importance significative dans la convergence de l'algorithme. Les méthodes de Levenberg-marquardt et de régions de confiances récentes permettent de résoudre ces problèmes dans une certaine mesure. Leur application optimale est cependant limitée à des problèmes non contraints. La prise en compte d'un intervalle de définition des paramètres peut déjà s'avérer problématique. L'ajout de contraintes complexes est donc un véritable problème pour ces algorithmes. De plus, la gestion de plusieurs objectifs semble possible Fliege et al. (2019) mais ne semble pas faire consensus.

Dans la suite de ce document, nous utiliserons quand même certaines de ces méthodes, notamment pour des problèmes d'ajustement de courbe lors de la phase de calibration de modèle ou d'estimation de lois de probabilité par maximum de vraisemblance. Les limites que nous venons de présenter nous pousseront cependant souvent à l'utilisation de métaheuristiques.

1.3 L'optimisation par métaheuristiques

Les métaheuristiques constituent une famille d'algorithmes adaptables à un grand nombre de problèmes d'optimisation. Elles se différencient des méthodes exactes par le fait qu'elles ne visent pas forcément à obtenir la solution optimale, mais une solution acceptable en un temps de calcul raisonnable.

Un premier critère de différenciation est le fait d'utiliser une ou plusieurs solutions potentielles. On distingue alors :

— les métaheuristiques à solution unique, aussi appelées de voisinage, qui n'utilisent qu'une solution sur laquelle des transformations seront appliquées afin de l'améliorer de plus en plus.

— les métaheuristiques à population de solutions, aussi appelées distribuées, qui se basent sur un ensemble de solutions qui seront utilisées afin de les modifier au cours des itérations et de proposer un ensemble de solutions de meilleure qualité.

Parmi ces dernières, les algorithmes évolutionnaires constituent une classe de métaheuristiques qui s'inspirent de la théorie de l'évolution. Le principe consiste à faire évoluer de manière aléatoire une famille de solutions par croisement et mutation en ne conservant qu'une partie des solutions en fonction de leur qualité, ceci dans l'optique de trouver des optimums. Les algorithmes d'intelligence en essaim correspondent à une classe qui s'inspire du comportement des colonies d'insectes Dorigo and Stützle (2019) Parsopoulos and Vrahatis (2002) Parsopoulos et al. (2002) Wang et al. (2020), dans lesquelles

des agents autonomes au comportement simple et sans stratégie prédéfinie cherchent à évoluer vers des *attracteurs* correspondant à de *bonnes* situations. Comme pour les algorithmes évolutionnaires, ils sont particulièrement bien adaptés lorsque l'environnement est changeant et pour traiter des fonctions continues.

Dans les deux cas, elles se basent toujours sur au moins une fonction objectif qui se doit de représenter au mieux le problème et les solutions souhaitées. Par exemple, si le problème consiste à retrouver des données réelles à partir de simulations, la fonction d'évaluation pourra être $f(x) = \sum_i^{nDonnees} |Dreelle_i - Dsimulee_i|$, quantifiant ainsi les différences entre données réelles $Dreelle_i$ et simulées $Dsimulee_i$. On préférera cependant généralement l'élévation au carré plutôt que la valeur absolue pour accentuer les différences et profiter de sa dérivabilité en tout point, permettant l'utilisation de gradients. L'algorithme cherchera à minimiser (ou maximiser) cette fonction en modifiant la/les solution(s) au cours des itérations.

Ces méthodes incluent l'utilisation de processus stochastiques ce qui permet de pallier l'explosion combinatoire dans une certaine mesure.

Enfin, les métaheuristiques sont soumises à un théorème appelé "no free lunch theorem" Wolpert and Macready (1997) énonçant qu'il est impossible de déterminer à l'avance quelle métaheuristique sera la plus efficace sur un problème donné. Il est donc important de pouvoir en tester plusieurs afin de déterminer laquelle utiliser pour traiter le problème.

Elles ont de plus le grand avantage d'être très génériques et modulaires, permettant de facilement les adapter à un grand nombre de problèmes de toute sorte.

Même si chaque métaheuristique a ses particularités, elles se basent presque toutes sur un socle commun présenté en figure 1.2. Dans ce schéma, les phases d'encodage et décodage sont optionnelles en fonction du type de représentation utilisé. Celle-ci doit respecter le principe de complétude, c'est-à-dire, que la représentation des solutions doit permettre de couvrir l'ensemble de l'espace de recherche afin de n'ignorer aucune solution potentielle. Elle peut également jouer sur l'efficacité des calculs de voisinage. La représentation des solutions et les méthodes d'exploration de l'espace de solutions, telle que l'exploration de voisinage, doivent être cohérentes entre elles (figure 1.3). Dans le cas d'optimisation continue, on peut par exemple définir une distance maximale. Dans ce cas, l'ensemble des voisins serait donc une hypersphère comme représentée dans la figure 1.3a. Pour un problème où on pourrait représenter les solutions sous forme de vecteur binaire, tel que le problème du sac à dos (section 1.4), l'inversion d'un bit pourrait être une notion de voisinage cohérente (figure 1.3b).

Dans le cas de types complexes, la solution peut être encodée en un type intermédiaire, plus facilement utilisable par les opérateurs de la métaheuristique utilisée. On respecte alors la généricité des métaheuristiques. La phase de traduction peut cependant avoir un coût de calcul. À l'inverse, l'approche directe propose d'utiliser directement une solution de type complexe, mais nécessite une adaptation des différents opérateurs de l'algorithme.

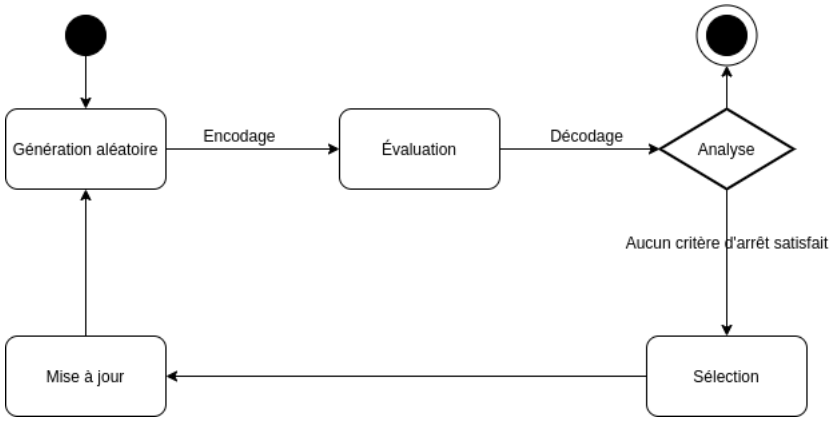
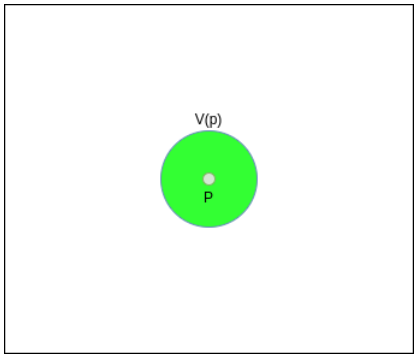
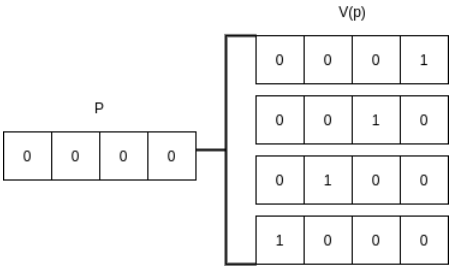


FIGURE 1.2 – Déroulement classique d'une métaheuristique



(a) Voisinage par distance dans un espace métrique continu



(b) Voisinage par inversion binaire

FIGURE 1.3 – Notion de voisinage

Génération aléatoire La première étape de toute métaheuristique consiste à générer la population initiale de façon aléatoire. Bien que la distribution utilisée puisse varier en fonction des cas et des connaissances sur le problème, la plupart des méthodes cherchent à répartir la population initiale de façon uniforme dans l'espace de recherche afin d'explorer toutes les caractéristiques. Cela s'applique particulièrement aux algorithmes à population de solutions.

Ainsi, le choix d'un générateur de nombres pseudo-aléatoires de qualité est nécessaire. L'algorithme de Mersenne Twister Matsumoto and Nishimura (1998) est aujourd'hui un choix éprouvé, mais d'autres sont bien sûr possibles Van Toan Dao et al. (2014).

Évaluation Cette phase est certainement la plus importante de toutes. Elle est commune à tout algorithme d'optimisation. C'est elle qui permet de définir si une solution est de qualité ou non. Elle se

base sur la/les fonction(s) d'évaluation définie(s) par l'utilisateur de la méthode afin d'attribuer une/des note(s) représentant la qualité de la solution par rapport au(x) critère(s) à optimiser.

Une fonction d'évaluation mal définie ne sera pas néfaste au déroulement de la métaheuristique en soi. Le problème est que l'algorithme ne réfléchit pas par lui-même et ne fait que ce qu'on lui demande. Il optimisera donc les solutions dans le sens que lui donne cette fonction. Or, si elle n'est pas le reflet de ce que l'utilisateur considère comme une bonne solution, les solutions de sortie seront mauvaises même si l'algorithme les considère optimales. C'est pourquoi la phase de détermination de l'objectif est toujours une phase cruciale dans les processus d'optimisation.

Prenons l'exemple d'un problème où le but est de maximiser les profits d'une entreprise. Une simulation est réalisée sur 10 ans et la fonction d'évaluation est la somme des gains de chaque année. Dans ce cas, l'algorithme pourrait proposer de vendre tous les biens de l'entreprise et de mettre tous les employés au chômage la dernière année car après tout, c'est ce qui permettra de gagner le plus possible sur cet intervalle de temps. L'algorithme ne considérera pas le futur en dehors du temps de simulation.

C'est également durant la phase d'évaluation que sont généralement prises en compte les contraintes. Cela peut se faire par plusieurs méthodes telles que la peine de mort ou la pénalisation. Ces méthodes seront détaillées en section 1.5.

Les différentes sorties de la phase d'évaluation permettent de savoir quelles solutions se rapprochent de l'optimum, permettant ainsi de déterminer lesquelles utiliser ou non pour la suite : la phase d'analyse.

Analyse La phase d'analyse permet de déterminer s'il faut continuer le déroulement de l'algorithme. Si les solutions trouvées sont acceptables, selon un critère défini au préalable, l'algorithme s'arrête. Il faut bien sûr définir la notion de solution acceptable, ce qui peut être difficile si nous ne connaissons pas l'espace de définition de la fonction d'évaluation. En effet, la plupart du temps, on cherche à atteindre un certain niveau de qualité des solutions. Or, la fonction objectif peut ne pas être explicitement bornée.

Dans le cas de la maximisation des gains d'une entreprise, celui ci ne peut pas être infini, mais il est très difficile de savoir quel est le gain maximum possible, car cela sous-entendrait bien souvent de savoir comment l'atteindre et donc l'optimisation deviendrait inutile.

D'autres critères d'arrêt peuvent alors être définis, comme simplement un nombre d'itérations maximum. En pratique, il est souvent difficile d'estimer le nombre optimal d'itérations à réaliser. Ce critère n'est donc utilisable de façon optimale qu'en situation de benchmarking d'algorithmes de même nature. On utilise donc souvent la vitesse de convergence, c'est-à-dire, si les solutions continuent d'évoluer au cours des itérations et à quel point.

Si aucun critère d'arrêt n'est satisfait, l'algorithme se dirige vers la phase de sélection.

Sélection La phase de sélection permet de déterminer quelles solutions devront être utilisées pour la suite de l'algorithme. Dans le cas d'algorithmes à solution unique, cette phase vise à déterminer si la nouvelle solution calculée doit remplacer la solution initiale. Bien souvent, notamment dans les algorithmes évolutionnaires ou génétiques, cette phase peut être stochastique afin d'éviter une convergence trop rapide et ainsi de garder une certaine diversité dans la population. Des sélections déterministes peuvent cependant être utilisées, même dans ces algorithmes.

On peut par exemple citer la sélection élitiste utilisée par certaines variantes d'algorithmes génétiques. Elle consiste à conserver au moins les meilleures solutions de la population.

Plus de détails sont donnés sur ces algorithmes et leur façon d'effectuer les sélections en annexe 6.4.2.

Enfin, l'algorithme passe à la phase de mise à jour.

Mise à jour C'est lors de cette dernière phase que les solutions sont modifiées afin d'en considérer de nouvelles, potentiellement plus intéressantes. On se sert des solutions présentes dans la population pour croiser les caractéristiques de chacune, essayant ainsi de combiner les avantages que chaque solution peut avoir. Dans le cas d'un algorithme à solution unique, une perturbation est appliquée à la solution afin d'effectuer une recherche de voisinage.

Puisque cette phase est spécifique à chaque algorithme, nous renvoyons le lecteur intéressé vers l'annexe consacrée au détail de chaque algorithme implémenté dans notre bibliothèque : l'annexe 6.4.

C'est également pendant cette phase que les contraintes peuvent être prises en compte via des méthodes plus complexes que la simple pénalisation. On peut notamment penser à l'utilisation de méthodes de satisfaction de contraintes spécifiques au problème, dirigeant ainsi l'utilisateur vers des méthodes hybrides métaheuristiques/heuristiques spécifiques. Plus de détails sont donnés en annexe 6.3.1.

Le paramétrage des métaheuristiques est une phase souvent complexe. En effet, chacune possède son jeu de paramètres dont dépend fortement son efficacité et qui ne peuvent être optimaux pour chaque problème. La littérature propose bien souvent des exemples éprouvés sur des études comparatives, mais sans réelle justification mathématique ni garantie que cela sera efficace sur un autre problème inédit.

Ainsi, des méthodes de paramétrage ont été proposées Smit and Eiben (2009) Eiben and Smit (2011) et peuvent être classées en deux branches, voir figure 1.4.

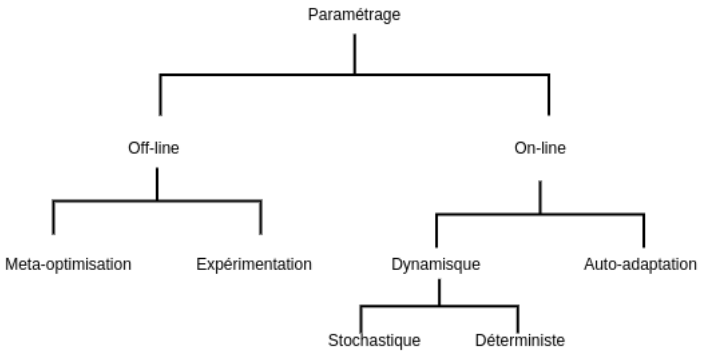


FIGURE 1.4 – Type de paramétrage des métaheuristiques

Les méthodes off-line supposent un paramétrage en amont de l'optimisation, souvent via expérimentation et tâtonnement. Cela peut également se faire via une analyse théorique approfondie du problème et par rapprochement avec d'autres problèmes connus.

Les méthodes on-line, sont plus souples et permettent une mise à jour dynamique des paramètres au cours de l'optimisation. Cela n'est évidemment pas toujours possible, mais certaines méthodes telles que

la règle des 1/5 (annexe 6.4.2 Rechenberg (1973)) utilisée dans les algorithmes génétiques avec mutation gaussienne, peuvent être trouvées dans la littérature.

Dans notre étude, les métaheuristiques seront un outil particulièrement utile, non seulement au vu de leur efficacité, mais également leur généricité.

Nous allons maintenant nous intéresser à différents types d'optimisation et les problèmes qu'ils permettent de résoudre, à commencer par l'optimisation globale.

1.4 Optimisation globale

Par abus de langage, on parle souvent d'optimisation globale pour désigner l'optimisation mono-objectif. Ce terme désigne en réalité toute optimisation recherchant l'optimum global.

De façon générale, elle désigne tout type de problème pouvant être défini sous la forme :

$$Minimise\ f_i(s),\ (\forall i \in [0, l[),\ s \in S$$

$$Sous\ contraintes :$$

$$g_i(s),\ (\forall i \in [0, m[)$$

$$h_i(s),\ (\forall i \in [0, n[)$$

avec :

— s : solution proposée ;

— S : espace des solutions ;

— f_i : $i$ème fonction d'évaluation ;

— g_i : $i$ème fonction de contrainte d'inégalité ;

— h_i : $i$ème fonction de contrainte d'égalité ;

— l : nombre de fonctions d'évaluation ;

— m : nombre de contraintes d'inégalité ;

— n : nombre de contraintes d'égalité.

Ainsi, l'optimum global $s*$, s'il existe [2], peut être défini comme :

$$s* \in S,\ \forall s \in S,\ f_i(s*) \leq f_i(s)\ \forall i \in [0, l[\tag{1.1}$$

Nous reviendrons plus en détails sur l'optimisation sous contraintes en section 1.5

Bien souvent, les problèmes d'optimisation difficile présentent un grand nombre d'optimums locaux, comme on peut le voir sur la figure 1.5. Les méthodes doivent donc inclure des mécanismes d'extraction des optimums locaux.

2. Ce n'est pas toujours le cas notamment en optimisation multiobjectif où il existe un optimum pour chaque fonction d'évaluation, mais l'optimum global n'est que théorique et est même parfois caractérisée de *solution utopique.*

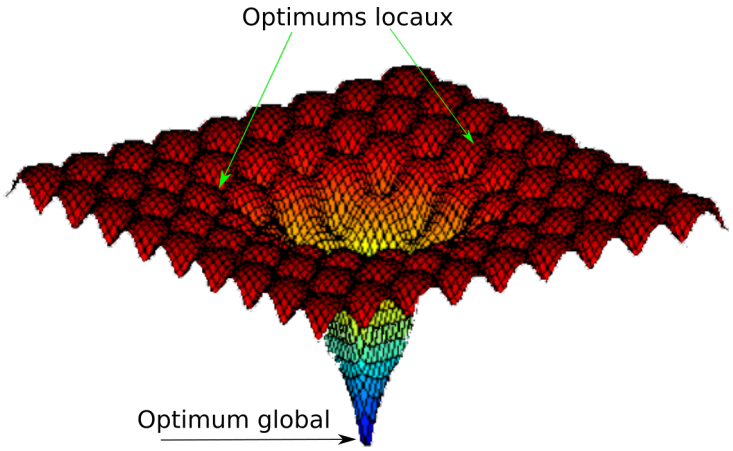


FIGURE 1.5 – Fonction de Ackley en dimension 2 : illustration du concept d'optimum global, local

Un problème se pose alors : nous ne connaissons pas à l'avance la taille et la profondeur de ces optimums. Il est donc difficile de s'assurer de pouvoir sortir de tous les optimums locaux sans compromettre l'efficacité de la méthode. En effet, une méthode d'optimisation doit gérer le compromis entre exploration et approfondissement. La première composante représente la diversité des recherches dans l'espace des solutions, alors que la seconde indique jusqu'à quel point les recherches seront approfondies au voisinage des solutions jugées intéressantes. L'augmentation de la composante exploratrice est une bonne solution pour s'assurer de l'extraction des optimums locaux, mais cela engendre souvent un temps de convergence plus élevé et une diminution de l'efficacité de la méthode lorsque cela est mal géré.

Les méthodes d'optimisation permettant d'obtenir une solution optimale consistent généralement à explorer la totalité de l'espace, comme pour les méthodes exhaustives pour des problèmes combinatoires. Cependant, ces méthodes ne permettent de résoudre que des problèmes de petite taille, comme nous le montrons sur les deux exemples suivants, où il est nécessaire de s'orienter vers des méthodes approchées.

Problème du sac à dos Le problème du sac à dos est un problème NP-complet très connu qui a été formalisé pour la première fois en 1972 dans Karp (1972).

Dans sa version la plus simple, le problème consiste à sélectionner certains objets possédant certaines caractéristiques, comme le poids et la valeur, parmi un ensemble donné. Ces objets doivent être mis dans un sac ayant une capacité limitée. L'objectif étant, dans ce cas, de maximiser la valeur obtenue sans toutefois dépasser un poids maximum, la métaphore étant faite avec un sac à dos ne pouvant supporter qu'un certain poids.

Pour cette version, mais également d'autres plus complexes, des méthodes de résolutions exactes peuvent être envisagées Jorge (2010). On peut cependant imaginer augmenter le nombre de contraintes à satisfaire et de valeurs à maximiser. On parle alors de sac à dos multidimensionnel, lors de l'ajout de contraintes, et de sac à dos multiobjectif, lors de l'ajout de valeurs aux objets.

Lorsque la complexité du problème devient trop importante, on privilégiera des méthodes stochastiques, approchées telles que les métaheuristiques Kong et al. (2008) Chu and Beasley (1998) Raidl (1998).

Bien sûr, l'utilité de ce problème s'étend bien au-delà du simple remplissage d'un sac à dos. Pouvant aller de l'optimisation des porte-containers ou des avions à la gestion d'actifs financiers Kleywegt and Papastavrou (1998).

Problème du voyageur de commerce Comme pour le sac à dos, le problème du voyageur de commerce semble très simple de prime abord, mais on se rend vite compte qu'il est un exemple parfait d'explosion combinatoire.

Ce problème consiste, étant donné une liste de villes toutes reliées entre elles par des chemins, à trouver le plus court chemin qui visite une seule fois toutes les villes et qui retourne, in fine, sur la ville de départ.

Le nombre de chemins possibles étant $\frac{1}{2}(n-1)!$, on arrive très vite à un nombre de possibilités impossible à explorer de façon exhaustive. Par exemple, pour seulement 14 villes, on atteint déjà 3 milliards de chemins possibles.

Ainsi, une approche par métaheuristique semble particulièrement adaptée Reinelt (1994) Dorigo and Gambardella (1997). Cependant, des heuristiques comme les 2, k-opt Croes (1958) Lin and Kernighan (1973) Helsgaun (2000) Nilsson (2003) qui servent à optimiser des sous chemins, permettent de grandement améliorer les solutions. Des approches hybrides, utilisant métaheuristiques et heuristiques spécialisées Puchinger and Raidl (2005), ont donc été proposées Mahi et al. (2015).

Comme pour le problème du sac à dos, de nombreuses applications réelles sont possibles. On peut par exemple penser au parcours de bus scolaire Angel et al. (1972) qui est une de ces inspirations. Le lecteur intéressé pourra notamment se reporter à Matai et al. (2010) pour trouver un grand nombre d'applications.

Comme nous venons de le présenter, même si les problèmes d'optimisation combinatoire peuvent être faciles à définir, ils sont très souvent difficiles à résoudre. Étant donné qu'un grand nombre d'applications pratiques peuvent se définir comme des problèmes d'optimisation, de nombreuses méthodes approchées ont été développées. Les métaheuristiques font partie de cette catégorie. Elles nous permettront donc de résoudre de nombreux problèmes d'optimisation globale. De plus, les problématiques auxquelles nous aurons à faire face dans la suite de ce document, nécessiteront l'utilisation de contraintes, ce qui fera l'objet de la prochaine section.

1.5 Optimisation sous contraintes

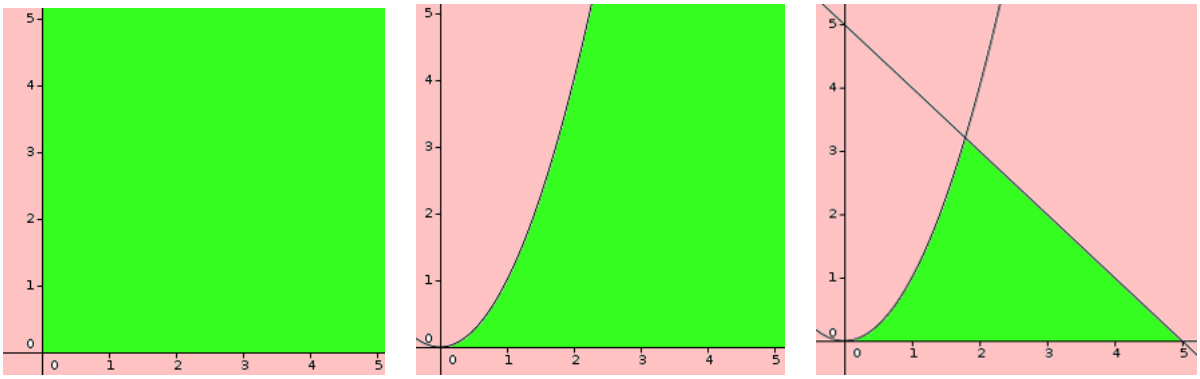
L'optimisation sous contraintes traite des problèmes où l'espace de validité des solutions est limité par des contraintes. Elles sont exprimées via des fonctions mathématiques comme peuvent l'être les fonctions d'évaluation. On distingue :

— les fonctions d'inégalité de la forme : $g_i(s) < 0,\ (\forall i \in [0, m])$.

— les fonctions d'égalité de la forme : $h_i(s) = 0,\ (\forall i \in [0, n])$.

La figure 1.6 montre un exemple de restriction de l'espace des solutions que peuvent imposer des contraintes. La zone en vert est appelée espace réalisable et correspond aux valeurs qui peuvent être

prises par f(x). Les zones en rose correspondent à l'espace irréalisable, où les solutions ne peuvent pas être prises.



$x \in [0, 5.2], f(x) \in R^+$ $g_1 : \ f(x) < x^2$ $g_1 : \ f(x) < x^2$ $g_2 : f(x) < 5 - x$

FIGURE 1.6 – Exemple de restriction de l'espace de recherche imposé par des contraintes

Une méthode naïve de prise en compte des contraintes pourrait consister à vérifier leur satisfaction à chaque modification d'une solution potentielle et à la modifier en cas de non respect de celles-ci.

Cette approche peut suffire dans les cas où les contraintes sont facilement satisfaisables ou que nous connaissons des méthodes de satisfaction de contraintes spécifiques (annexe 6.3.1). Or, la plupart du temps, ce n'est pas le cas. D'autres méthodes sont alors utilisées telles que :

— les méthodes de pénalisation. Elles transforment le problème initial en ajoutant des fonctions de pénalité influençant l'évaluation en fonction du degré de violation des contraintes. Elles posent cependant le problème de la gestion des pénalités pour orienter convenablement les recherches.

— la méthode de la peine de mort. Elle Résout le problème de sous-pénalisation en fixant une pénalité infinie aux solutions non réalisable. Son efficacité est cependant trop dépendante de la difficulté de satisfaction des contraintes, elle n'est donc que rarement privilégiée.

— les pénalisations dynamiques, adaptatives et auto-adaptatives permettent de gérer un grand nombre de cas en définissant des coefficients de pénalité optimaux en fonction d'intervalle de niveau de violation de contraintes Homaifar et al. (1994), de faiblement pénaliser au début pour favoriser l'exploration, puis grandement ensuite Joines and Houck (1994), ou encore en fonction de la proportion solutions réalisables/non réalisables Hamida and Schoenauer (2000) par exemple.

— les méthodes de supériorité des individus réalisables. Elles se basent généralement sur des méthodes de classement des solutions afin de toujours considérer qu'une solution réalisable sera meilleure qu'une non réalisable indépendamment de l'évaluation. C'est le cas notamment du Stochastic Ranking proposé par Runarsson and Yao (2000), méthode que nous détaillerons en annexe 6.4.14.

— des méthodes hybrides telles que la préservation de la faisabilité des solutions par ajout d'heuristiques de satisfaction de contraintes problème-spécifiques ou encore la modification préalable de la topologie de l'espace des solutions afin de rendre l'espace de faisabilité convexe et d'y définir des

opérateurs fermés comme le système GENOCOP Michalewicz and Janikow (1991), Michalewicz and Nazhiyath (1995)

Le détail de l'ensemble de ces méthodes est donné en annexe 6.3.1 Dans notre étude, nous nous intéresserons particulièrement au stochastic ranking qui permet une grande généricité et adaptabilité aux contraintes qui pourront représenter des connaissances expertes lors de la phase de calibration par exemple. Cela permet une meilleure adaptabilité de la méthode aux problèmes. D'autres méthodes de préservation de la satisfaisabilité des solutions via heuristiques spécifiques seront cependant également utilisées.

L'ensemble des méthodes présentées jusque-là supposait l'existence d'un optimum global unique, ou au moins que la découverte de multiples optimums n'était pas nécessaire. Certains problèmes présentent cependant plusieurs solutions d'évaluation équivalente Qu et al. (2016). Il est alors nécessaire d'utiliser des méthodes d'optimisation multimodale.

1.6 Optimisation multimodale

Elle consiste à identifier plusieurs optimums, potentiellement globaux, en même temps. Via optimisation unimodale, la population est rapidement menée à converger vers un seul optimum. Cette convergence vers une unique solution est restrictive, car la présence d'un seul individu au voisinage de l'optimum global serait suffisante. Les autres membres de la population pourraient être répartis dans d'autres régions de l'espace et donner ainsi de multiples renseignements utiles par exemple sur la forme de la fonction objectif. La figure 1.7 montre une représentation du problème de rastringin pour deux paramètres $x, y \in [0, 1]^2$ et $f(x, y) = -(20 + 9cos(6\pi x) + 9cos(8\pi y))$ à minimiser. L'échelle de couleur représente la qualité de $z = f(x, y)$. En jaune, on peut voir que toutes les solutions proposées par une optimisation globale ont convergé vers la même solution alors que la population finale d'un algorithme d'optimisation multimodale, en bleu clair, converge vers tous les optimums.

Cela peut également mener à la découverte de solutions moins bonnes théoriquement, mais avec certaines caractéristiques différentes intéressantes. En effet, la traduction d'un problème d'optimisation avec tous ses aspects (performance oui, mais aussi sensibilité, facilité de fabrication, prix de revient, robustesse aux incertitudes, ...) sous forme d'une fonction à optimiser est une grande difficulté dans l'utilisation efficace de l'optimisation en situation réelle. Lorsque l'utilisateur ne cherche pas uniquement la performance, il appréciera la multiplicité des possibilités. On en vient alors à l'idée de localiser toutes les régions intéressantes de l'espace des solutions.

L'optimisation multimodale consiste donc à incorporer des méthodes de spéciations, permettant de classer les individus en sous-populations, ou de nichages permettant ainsi de séparer des sous-populations dans des niches *écologiques*. Ce terme, comme souvent en optimisation, s'inspire des écosystèmes naturels où les espèces évoluent de façon à remplir une niche écologique. Dans chaque niche, les ressources naturelles sont limitées et doivent être partagées entre les individus. Par analogie, les méthodes de nichages des algorithmes multimodaux tendent à reproduire l'émergence de niches et d'espèces dans l'environnement. Une niche se réfère donc naturellement à un optimum de la fonction objectif, sa valeur représentant la

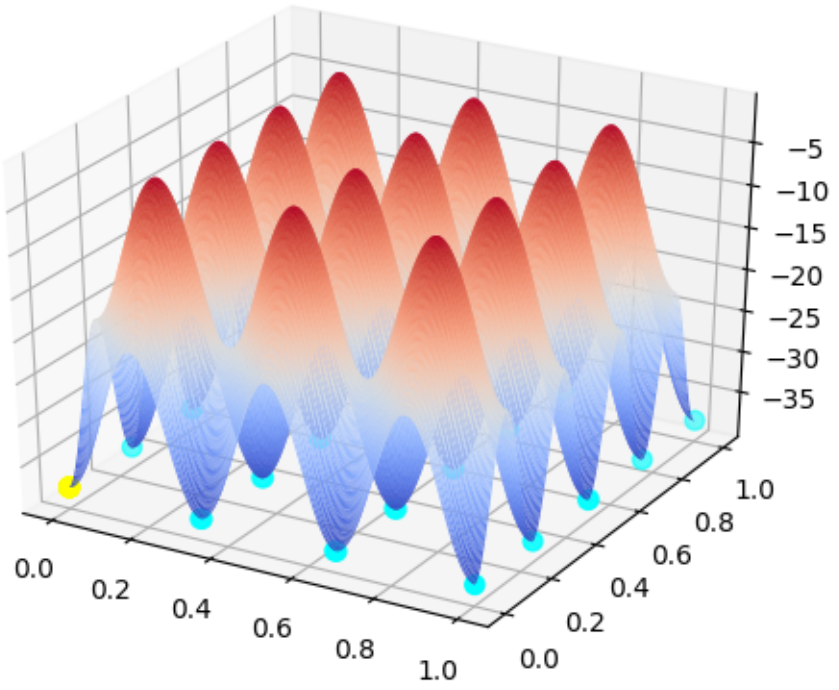


FIGURE 1.7 – Problème multimodal de Rastringin modifié

ressource et les espèces sont les groupes d'individus similaires partageant cette ressource. On est donc placé dans un espace métrique où une mesure de distance permet de calculer la similarité entre les individus Sareni (1999) Deb and Goldberg (1989).

On distingue alors plusieurs types de méthodes :

— de nichage par surpeuplement : ces méthodes ont semblé peu efficaces sur des problèmes complexes jusqu'aux travaux récents de Thomsen (2004) proposant, CrowdingDE plus tard améliorée par Wong et al. (2012), une variante d'évolution différentielle pour l'optimisation multimodale et démontrant de très bonnes performances.

— de partage, consistant à ajuster l'évaluation des solutions en fonction du nombre d'individus dans leur voisinage Goldberg et al. (1987).

— d'éclaircissement, consistant cette fois à ne pas partager, mais plutôt attribuer l'évaluation à la meilleure solution de chaque région dense, éliminant ainsi les autres Petrowski et al. (1997)Pétrowski et al. (1997).

— de clustering, couplant ces méthodes de machine learning à l'optimisation afin de séparer efficacement les solutions. Ces méthodes semblent les plus populaires dans la littérature récente comme le montrent les nombreuses variantes d'essaim particulaire Mehmood et al. (2018) ou d'évolution différentielle Wang et al. (2017) Wang et al. (2019b) qui ont été couplées à différents types de clustering Rana et al. (2013) Niu et al. (2015) Bouyer and Hatamlou (2018)

Ces méthodes nous serviront à proposer différentes solutions à des problèmes de calibration soumis à un problème d'équifinalité. Nous nous appuierons notamment sur l'algorithme Dual Strategy Differential Evolution (DSDE) présenté dans Wang et al. (2017).

Enfin ces méthodes ont également pu être déclinées pour résoudre des problèmes d'optimisation multimodale multiobjectif Tanabe and Ishibuchi (2019).

1.7 Optimisation multiobjectif

L'optimisation multiobjectif Deb (2001) est un cas particulier de l'optimisation globale où plusieurs fonctions objectif doivent être minimisées ou maximisées. Elle est beaucoup utilisée dans des processus d'aide à la décision. En effet, les problèmes réels présentent souvent des objectifs contradictoires et/ou des facteurs qui ne peuvent pas être formalisés mathématiquement tels que les questions d'éthique. Ainsi, la principale différence avec les méthodes mono objectif est qu'il n'y a pas de relation d'ordre total [3] entre les solutions. Il n'existe donc pas, dans la plupart des cas pratiques, de solution optimale à proprement parler. Dans ce cas, il peut être nécessaire de proposer un ensemble de solutions, appelé front de Pareto, permettant de montrer le meilleur résultat possible pour chaque critère, mais également les compromis possibles entre eux.

La notion de front de Pareto se base sur la dominance de Pareto. Soient deux solutions $s1$ et $s2$ d'un problème multiobjectif, $s1_i$ désignant la $i$ème fonction objectif et $i \leq l$, l désignant le nombre d'objectifs du problème. Dans le cas de la minimisation de tous les objectifs on dit que la solution s_1 domine ($\prec$) la solution s_2 si on a :

$$s1 \prec s2 \iff \forall i \in [1,l]\ s1_i \leq s2_i\ et\ \exists j \in [1,l]\ :\ s1_j < s2_j \tag{1.2}$$

Ainsi, une solution est dite optimale au sens de Pareto si elle ne peut pas être dominée. L'ensemble des solutions non dominables constituent le front de Pareto. La figure 1.8 montre un exemple illustrant cette notion.

Une méthode simple consiste à ramener l'ensemble des fonctions objectif à une seule évaluation globale en effectuant une sommation pondérée par l'importance de chaque objectif, appelée poids et notée w. À chaque ensemble de poids w, de sorte que la somme de ces composants soit égale à 1, correspond un point du front de Pareto. Le problème peut alors être transformé en cherchant à optimiser la somme pondérée

3. relation d'ordre total : toute relation définie sur un ensemble telle que tous éléments peuvent être comparés deux à deux suivant cette relation

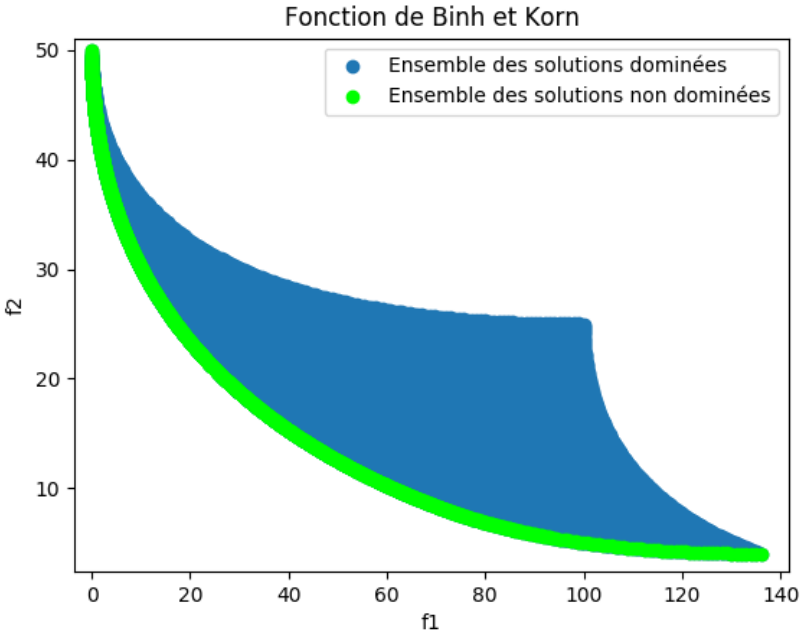


FIGURE 1.8 – Front de Pareto de la fonction de Binh et Korn

des objectifs :

$$G1(x|w)) = \sum_{i=1}^{l} \omega_i f_i(x) \tag{1.3}$$

Le problème se ramène à une optimisation globale classique. Ainsi, une approche naïve serait d'effectuer un grand nombre d'optimisations, chacune avec des jeux de poids différents. Cela permettrait d'obtenir une solution Pareto optimale pour chaque optimisation, constituant ainsi facilement une grande partie, voire l'entièreté, du front de Pareto.

Cette approche présente souvent de bons résultats, mais demande un temps de calcul très important. De plus, elle ne permet pas d'obtenir des solutions Pareto optimales situées sur les parties concaves du front Siarry (2014).

La scalarisation via la distance de Chebyshev Cantrell (2000) pondérée introduit un point de référence p ayant pour coordonnées la meilleure évaluation possible sur chaque objectif (voir figure 1.9). La fonction d'agrégation devient alors :

$$G_\infty(x|w,p) = \max_{i=1}^{l} \omega_i |f_i(x) - p_i| \tag{1.4}$$

G_∞ est appelé la distance de Chebyshev qui devra être minimisée.

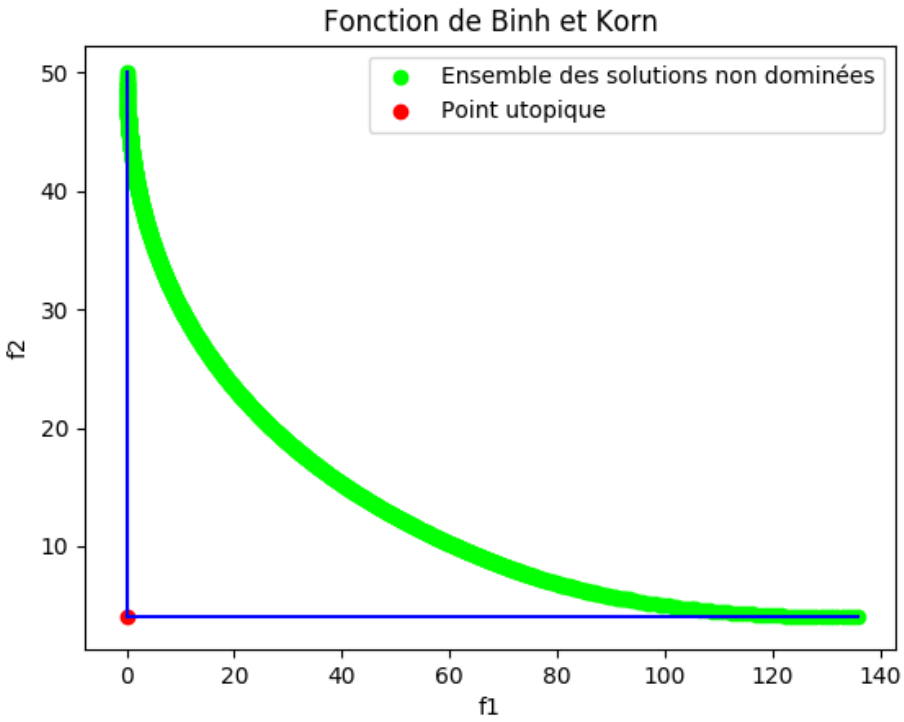


FIGURE 1.9 – Détermination du point utopique de la fonction Binh and Korn

Le calcul du point de référence, aussi appelé point utopique Gunantara (2018), se fait via une première optimisation cherchant à améliorer chaque objectif indépendamment des autres tout en respectant les contraintes du système. Cette procédure se fait en attribuant des poids nuls à toutes les fonctions objectif sauf une et en répétant l'opération pour chacune. La solution devient alors généralement très simple à déterminer. Ce point est bien souvent inatteignable en pratique.

Cette approche permet d'atteindre l'intégralité du front de Pareto, elle permet cependant également à des solutions non pareto-optimales d'émerger.

De nombreuses méthodes utilisant la scalarisation des objectifs ont été proposées. Ainsi, le lecteur intéressé pourra se référer à :

— $\epsilon - MOEA$ stationnaire : Deb et al. (2003), cherchant à rapidement trouver des solutions non dominées représentatives du front de Pareto. Cette méthode se base sur la notion d'ϵ-dominance.

— MOEA/D : Zhang and Li (2007), basé sur la notion de décomposition des objectifs. Cette solution semble être particulièrement adaptée aux problèmes présentant des fronts de Pareto complexes Li and Zhang (2009).

Les approches utilisant un classement de Pareto s'appliquent principalement aux algorithmes évolutionnaires (détaillés en annexe 6.4.2) en incluant la notion de dominance de Pareto dans le classement des solutions proposées, permettant ainsi de ne garder, d'une itération à l'autre, que les individus intéressants au sens

de la dominance de Pareto. Historiquement, la première méthode utilisant ces notions a été introduite par Goldberg (1989).

Un très grand nombre de méthodes basées sur les algorithmes génétiques ont été proposées au cours du temps. On peut notamment penser à MOGA Fonseca et al. (1993), NPGA Horn et al. (1994), NSGA Srinivas and Deb (1994) , SPEA Zitzler and Thiele (1999) ou encore NSGA-II Deb et al. (2002).

Cette dernière utilise une méthode de classement de Pareto de moindre complexité ($O(\mu^2 c)$ au lieu de $O(\mu^3 c)$ pour NSGA classique par exemple) et une technique de nichage sans paramètres, simplifiant son utilisation. Enfin, la méthode de sélection utilisée permet d'accélérer la convergence des solutions.

Ainsi, l'optimisation multiobjectif, notamment via l'algorithme NSGA-II, nous permettra de montrer l'ensemble des compromis possibles entre économie et écologie dans un contexte de gestion des ressources halieutiques. Cela se fera notamment par la représentation des fronts de pareto niveau de biomasse de l'espèce/quantité de captures pour les pêcheurs ou encore gains/sensibilité des solutions aux incertitudes, permettant ainsi d'évaluer leur robustesse.

1.8 Optimisation robuste

L'optimisation robuste est dédiée à la résolution de problèmes d'optimisation soumis à incertitudes. Elle est très utilisée notamment dans l'industrie Beyer and Sendhoff (2007) où, par exemple, des contraintes de conception doivent être satisfaites pour toutes les valeurs des paramètres incertains dans un ensemble d'incertitude donné. Les ensembles d'incertitude peuvent être modélisés comme des ensembles déterministes, auquel cas le problème d'optimisation robuste peut être reformulé par une analyse du pire cas, ou comme des familles de distributions. Un défi de l'optimisation robuste est de reformuler ou d'approximer les contraintes robustes de sorte que le problème d'optimisation incertain soit transformé en un problème d'optimisation déterministe tractable [4].

On peut définir un problème d'optimisation robuste de la manière suivante :

$$Minimise\ f(x),\ x \in \S$$

$$Sujet\ à :$$

$$c(x;u) \leq 0, \forall u \in \mathcal{U}(x)$$

avec :

— $x \in \mathcal{X}$, l'ensemble des variables de décision formant une solution, de l'espace des solutions réalisables $\mathcal{X}$. Dans le cas d'optimisation continue, on a $s \in \mathbb{R}^n$ et $\mathcal{X} \subseteq \mathbb{R}^n$

— $u \in \mathbb{R}^p$ l'ensemble des paramètres incertains $\in \mathcal{U}(\S)$ l'ensemble d'incertitude

— $c(u;x)$ modélise l'impact des incertitudes.

Nous supposons également que $\mathcal{U}(x)$ est un ensemble non vide et compact pour tout $x \in \mathcal{X}$. S'il était vide pour tout $x \in \mathcal{X}$, alors la contrainte d'incertitude pourrait être supprimée et on reviendrait à

4. Qui peut être résolu en pratique, pas seulement donnant une solution optimale théorique.

une optimisation classique. L'hypothèse de compacité garantit que les incertitudes sont bornées. Quand les incertitudes, $u \in \mathcal{U}(x)$ ne sont pas dépendantes de variables de décision x, l'ensemble d'incertitudes $\mathcal{U}(x)$ est alors écrit $\mathcal{U}$. De plus, si les contraintes sont séparables en x et u, $c(x;u) = g(x) + d(u) \leq 0$ la contrainte d'incertitude peut être remplacée par une contrainte déterministe $d^* = maxd(u), u \in \mathcal{U}$, la contrainte d'incertitude devient alors $c(x;u) = g(x) + d^* \leq 0$.

On définit la valeur nominale $\hat{u} \in \mathcal{U}$ comme la valeur que prendrait les paramètres incertains en l'absence d'incertitudes. Il est généralement le centre de l'intervalle, c'est-à-dire, pour $\mathcal{U} := [l, u] \subset \mathbb{R}^p$ un hyper-intervalle, $\hat{u} = (l + u)/2$. Quand les incertitudes dépendent de x, la valeur nominale devient alors une fonction $\hat{u}(x)$.

De plus, les incertitudes peuvent également influencer la fonction de sortie Pauwels (2016), d'évaluation. On a alors :

$$f = f(x, u) \tag{1.5}$$

Le but est alors de gérer le système de sorte à obtenir les sorties désirées, f, dépendantes des variables de décisions, x, et des quantités d'entrées incertaines u. Il est alors important de distinguer les différents types d'incertitudes. La figure 1.10 en montre une représentation. Ceux-ci sont historiquement utilisés dans le design de processus industriels Beyer and Sendhoff (2007), mais se généralisent facilement à tout système d'optimisation robuste via simulation.

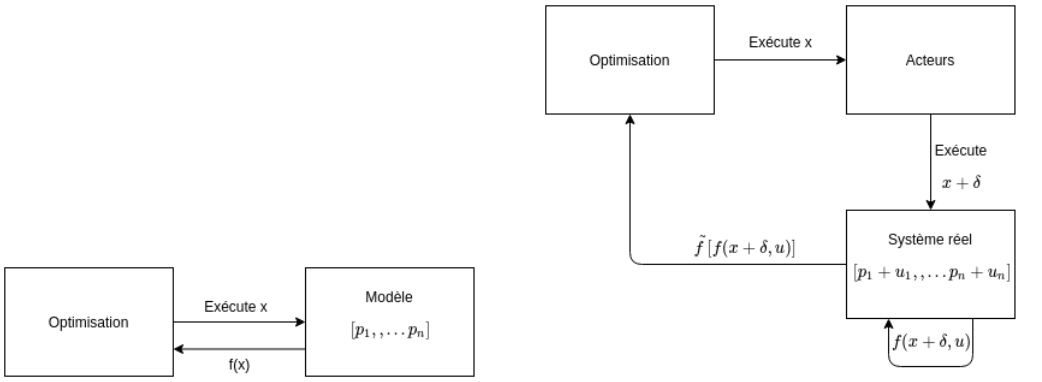
— Changement dans l'environnement où des conditions d'exploitation : ce type d'incertitude entre directement en compte via u.

— Tolérance de production et imprécision des actionneurs : dans les processus industriels, ce type d'incertitude intervient quand le design d'un produit ne peut être réalisé qu'avec un certain degré de précision. La haute précision industrielle a parfois un coût non négligeable, un design plus tolérant aux imprécisions réduit les coûts. Pour des applications de gestion de ressources, ce type d'incertitude pourrait s'appliquer aux actions humaines, réduisant la précision de la solution proposée. Dans ce cas, les contrôles pour le respect des stratégies de gestion proposées peuvent avoir un coût très important Akinbulire et al. (2017) Akinbulire et al. (2018), une solution moins sensible à la précision peut alors être préférable. Dans ce cas, les incertitudes entrent directement dans la fonction f en tant que perturbations δ, $f = f(x + \delta, u)$ qui peut également dépendre de x, on exprime alors $\delta = \epsilon x$.

— Incertitudes dans les sorties du système : celles-ci sont dues aux imprécisions dans l'évaluation du système. Cela peut être des erreurs de mesures ou d'approximations dues à l'utilisation de modèles plutôt que d'objets physiques réels par exemple. On a alors une fonction d'évaluation aléatoire $\tilde{f}$ dépendante de f : $\tilde{f} = \tilde{f}[f(x + \delta, u)]$.

— Incertitudes de faisabilité : les incertitudes concernant le respect des contraintes auxquelles les variables de décisions doivent obéir. Celles-ci ne sont pas directement prises en compte dans f mais sur l'espace des solutions réalisables. Elles pourraient s'inscrire directement dans $c(x;u)$, contraintes à respecter formellement réduisant donc l'espace des solutions réalisables.

On distingue également plusieurs façons de représenter les incertitudes mathématiquement :



(a) Système sans incertitudes (b) Système avec tous les types d'incertitudes

FIGURE 1.10 – Optimisation via simulation en présence d'incertitudes

— déterministe : définit un ensemble dans lequel u et δ peut varier ;

— probabiliste : décrit les mesures de probabilité exprimant la probabilité qu'un certain événement se produise ;

— possible : définit des mesures floues décrivant les possibilités ou taux d'appartenance qu'un événement puisse apparaître.

En fonction des systèmes et besoins, on distinguera alors plusieurs méthodes de résolution. Très souvent, on cherchera à étudier le pire cas et à garantir une certaine qualité pour celui-ci. On peut également étudier l'espérance de réalisation et la réduction de la variabilité. Une mesure probabiliste de la robustesse par seuil peut aussi être utilisée en considérant la distribution de f directement. Le but est alors, pour un seuil q, de maximiser le nombre de cas où f le dépasse. Il est généralement impossible de calculer $P(f|x)$ de manière analytique et le temps de calcul serait trop important numériquement. Il est alors nécessaire d'utiliser des simulations Monte-Carlo qui sont la boucle interne de l'algorithme d'optimisation et représente la majorité de la complexité algorithmique de celui-ci. De plus, dans ce type d'approche, fixer une valeur pour q est très souvent un problème de décision difficile en lui-même.

Certains problèmes d'optimisation robuste peuvent être résolus par des approches exactes en temps polynomial. On parle alors de programmation mathématique comme pour des problèmes d'optimisation linéaire, quadratique, ou conique par exemple Ben-Tal and Nemirovski (2002) Ben-Tal et al. (2002). Ceux-ci sont cependant limités et de nombreux problèmes nécessitent un autre type d'approche. On différencie alors deux approches : déterministe et probabiliste. La première calcule directement les mesures de robustesse et les contraintes associées explicitement. En résulte un problème d'optimisation classique résolvable par méthode d'optimisation locale ou globale. La seconde va chercher à prendre en compte les incertitudes en optimisant les fonctions d'évaluation et de contraintes bruitées par les incertitudes directement. On parle alors d'approche randomisée ou méthode Monte-Carlo. De par la nature des problèmes que nous étudierons dans la suite de ce document, c'est à cette dernière approche que nous nous intéresserons particulièrement.

Celle-ci est souvent appelée approche directe d'optimisation robuste dans le sens où les incertitudes

sont directement incorporées à un problème d'optimisation générique. Dans le cas d'incertitudes de type probabilistes, la fonction objectif $f(x, u)$ devient alors une fonction aléatoire $\tilde{f}$ et les effets des incertitudes peuvent alors être évalués directement dans la valeur de la fonction objectif. Les mesures de robustesse doivent directement être déterminées à partir de $\tilde{f}$. On peut alors différencier trois catégories :

— les stratégies Monte-Carlo Liu et al. (2001) Papadrakakis et al. (2004) Sandgren and Cameron (2002) Martin and Simpson (2006), qui réalisent un grand nombre de simulations pour un même point x afin d'obtenir une représentativité statistique suffisante et utilisent les statistiques obtenues comme entrée d'un modèle d'optimisation déterministe.

— les approches par méta-modèle Jin et al. (2003) Chang et al. (1999), construisant un méta-modèle qui utilise un ensemble de points x choisis de façon à représenter réellement la robustesse du système.

— utiliser les valeurs de $\tilde{f}$ directement en entrée d'un algorithme d'optimisation spécialement conçu pour gérer ce type de fonction.

Cette dernière est très souvent couplée à des approches basées sur les métaheuristiques. Par exemple, dans Sorensen (2001) l'auteur propose l'utilisation d'une recherche par tabou pour trouver des solutions robustes soumises à des incertitudes de précision. Les algorithmes évolutionnaires sont également très souvent utilisés dans un contexte de robustesse Rakshit et al. (2017) Sevaux and Le Quéré (2003) Ong et al. (2006) Ray and Smith (2006).

Ce type d'optimisation nous permettra de gérer les incertitudes dues à une calibration non optimale dans la phase d'optimisation des stratégies de pêche. Cela n'est cependant pas toujours suffisant pour obtenir des résultats acceptables. Notre problématique incluant une notion de temporalité dans l'exécution des variables de décision, l'utilisation de méthode d'optimisation robuste ajustable semble particulièrement appropriée.

1.9 Optimisation robuste ajustable

L'optimisation robuste ajustable (ARO), est une branche de l'optimisation robuste où certaines des variables de décision peuvent être ajustées après qu'une partie des données incertaines se soient révélées. L'optimisation robuste ajustable donne généralement une meilleure valeur de fonction objectif que l'optimisation robuste statique, car elle donne lieu à des décisions ajustables (ou attentistes) plus flexibles. L'ARO a de nombreuses applications dans la vie réelle Dashti et al. (2016) Tang and Wang (2015) Yanıkoğlu et al. (2019) et constitue une méthodologie calculable pour un grand nombre de variables de décision ajustables et d'ensembles d'incertitudes paramétrés. L'optimisation robuste est basée sur trois principes Yanıkoğlu et al. (2019) :

— les variables de décision $x \in \mathbb{R}^n$ sont dites *here-and-now decisions*

— le modèle d'optimisation prend l'ensemble des paramètres dans l'ensemble d'incertitudes préspécifié $\mathcal{U}$ uniquement

— les contraintes sont dures, c'est-à-dire qu'elles ne peuvent jamais être violées.

En optimisation robuste ajustable, le premier principe est relâché. Par exemple, la mise en place de quotas de régulation d'exploitation pour l'année $t+1$ n'est pas une *here-and-now* décision. Il serait plus cohérent de la modéliser comme une *wait-and-see* décision dépendante des résultats de l'année t. Certaines variables de décision peuvent donc être ajustées à un moment ultérieur en fonction d'une règle de décision, qui est fonction de tout ou partie de l'incertitude des données. On peut alors représenter le problème comme :

$$Minimise\ f(x,y,u),\ x \in \mathbb{R}^n, y \in \mathbb{R}^k$$

$$Sujet\ à :$$

$$A(\zeta)x + By(\zeta) \leq d, \forall \zeta \in \mathcal{Z}$$

avec :

— $x \in \mathbb{R}^n$ les variables de décision *here-and-now*, déterminées directement avant que ζ soient réalisées ;

— $y \in \mathbb{R}^k$ les variables de décision *wait-and-see*, ajustées après révélation de certaines données incertaines ;

— $u \in \mathbb{R}^n$, $A \in R^{m \times n}$, $B \in \mathbb{R}^{m \times k}$, $d \in R^m$, représente tous les coefficients incertains tels que $(u, A, B, d) \in \mathcal{U}$ l'ensemble d'incertitudes ;

— ζ les réalisations incertaines telles que $\zeta \in \mathcal{Z}$ l'ensemble des réalisations possibles de ζ et $\mathcal{Z} \subset \mathbb{R}^l$

1.10 Conclusion du chapitre

Dans ce chapitre, nous avons vu que l'optimisation permet de formaliser et de résoudre de nombreux types de problèmes. Elle propose alors des méthodes adaptées à chacun. Ainsi dans un premier temps, nous avons vu les différentes caractéristiques d'un problème d'optimisation. Ensuite, nous avons développé les deux types de méthodes les plus couramment utilisées : les méthodes basées sur les gradients et les métaheuristiques. Nous avons finalement détaillé les différents types de problèmes d'optimisation qui seront rencontrés dans la suite de ce document.

Dans un premier temps, les méthodes à base de gradient nous permettront d'étudier des problèmes simples de calibration Gavin (2019). Ces problèmes pourront être approfondis via des métaheuristiques adaptées à l'optimisation globale puis à l'optimisation sous contraintes, permettant l'inclusion de connaissances expertes aux problèmes de calibration. Celles-ci pourront également servir à l'optimisation de stratégies de pêche via simulation. Dans ce cadre, l'optimisation multiobjectif permettra la proposition de front de pareto des compromis entre objectifs économiques et écologiques. Enfin, l'optimisation robuste et l'optimisation robuste ajustable permettront de gérer les incertitudes liées à la phase de calibration qui peut se révéler imprécise du fait de problèmes d'équifinalité.

Elle est donc un formidable outil, notamment à destination des décideurs. En revanche, elle semble peu adaptée à la représentation des comportements humains à petite échelle. L'optimisation des stratégies de pêche est également très dépendante de la qualité de la calibration, introduisant une grande quantité

d'incertitudes qui n'est pas toujours gérable de façon optimale. L'utilisation de méthodes d'apprentissage, et notamment d'apprentissage par renforcement pourrait permettre de proposer une formalisation différente des problèmes pour représenter plus efficacement certains comportements et s'affranchir dans une certaine mesure des contraintes de qualité de la calibration.

Nous allons à présent développer deux chapitres qui utiliseront les méthodes d'optimisation, le premier pour calibrer le modèle de dynamique de population qui sera utilisé dans le deuxième pour optimiser les stratégies de pêche.

Chapitre 2

APPLICATION AUX CALAGES DE MODÈLES

Les aspects généraux du processus de modélisation et de simulation sont : l'acquisition de données sur le système à étudier, la définition du cadre expérimental et la réalisation du modèle, la définition des expériences de simulation, la simulation et enfin l'exploitation des résultats (Traoré and Muzy (2006)).

Chaque étape est importante, mais dans ce travail nous allons nous intéresser en particulier aux modèles. Les modèles sont généralement caractérisés par de nombreux paramètres qui déterminent la dynamique globale du système simulé. L'un des aspects importants dans la modélisation est lié à la mise au point des paramètres du modèle (Calvez (2007)). Ces paramètres peuvent être fixés par étude de sensibilité ou expertise puis affinés lors de la phase de calibration. Nous proposons de voir la calibration comme un problème d'optimisation et notre objectif est d'identifier les jeux de paramètres qui optimisent nos résultats de simulation. Dans ce cas, la simulation numérique, couplée au processus d'optimisation, doit permettre d'identifier les meilleures solutions applicables.

La modélisation permet de passer d'un système réel à une représentation informatique plus ou moins précise. Le modèle conceptuel permet, via utilisation d'un ensemble d'hypothèses et de règles, d'approcher le fonctionnement ou le comportement du phénomène étudié. Une phase de programmation nous permet ensuite de passer d'un modèle conceptuel à un code de calcul générique, pour lequel il est nécessaire, au cours de la phase de construction, de caler les paramètres des modèles afin qu'ils soient adaptés au système réel étudié. Le calage est donc une procédure itérative qui consiste à modifier les paramètres du code de calcul jusqu'à ce que les résultats de la simulation collent le plus possible à un jeu de données de référence. Il est cependant important de préciser qu'un modèle est par nature imparfait, un niveau de correspondance réaliste doit donc être défini (Vidal (2005)).

On peut alors séparer deux approches non exclusives, manuelle et automatique.

La première propose, en général, de réaliser une première simulation à partir d'une estimation des paramètres, dans le but de retrouver des résultats expérimentaux passés. Un expert compare ensuite visuellement les résultats réels et simulés afin de modifier les paramètres grâce à son expertise. Il réitérera

l'opération jusqu'à satisfaction. Sous condition du niveau d'expertise de l'utilisateur, ce processus présente souvent des résultats fiables (Vidal (2005)). Il pose cependant le problème de la non reproductibilité du processus, implique une parfaite confiance en l'expert et nécessite de bonnes connaissances préliminaires. Dans le cadre de la dynamique des populations halieutiques, les connaissances peuvent souvent s'avérer limitées, rendant ainsi cette approche difficile.

Les méthodes automatiques, elles, sont principalement réalisées via l'utilisation de méthodes d'optimisation. On se base alors sur trois points :

— une fonction objectif : représentant la qualité de la solution proposée, en général proportionnelle à l'écart entre données de référence et de simulation.

— un algorithme d'optimisation, dont le but est de minimiser [1] la fonction objectif.

— un test d'arrêt, visant à arrêter le processus, généralement un critère de vraisemblance, de convergence, ou un temps maximum d'exécution.

Le calage par optimisation est très populaire dans la littérature. Par exemple, (Moles et al. (2003)) propose une comparaison de différentes méthodes d'optimisation globale dans le contexte de calage de modèles biochimiques sous contraintes. Par les différences de résultats entre les méthodes comparées, il montre l'importance du choix de la méthode utilisée.

Un des problèmes principaux que présentent les méthodes d'optimisation est l'équifinalité, défini pour la première fois par (Von Bertalanffy (1956)). On parle d'équifinalité lorsque plusieurs, très souvent une infinité, de solutions peuvent permettre de résoudre un problème donné. Dans notre cas, nous ne pouvons pas nous intéresser uniquement aux solutions parfaites du fait de la non exactitude des données d'entrée. Il est donc impossible de considérer que la seule solution acceptable est l'optimum global même en supposant son unicité. Nous aurons donc forcément un ensemble de solutions et donc un problème d'équifinalité à prendre en compte.

Ainsi, il peut être nécessaire de finalement proposer l'ensemble des solutions à une expertise manuelle. De plus, cette approche est très sensible à la qualité des données d'entrée (Anderson (2002)).

Il est également possible de proposer une approche plus statistique telle que la méthode Generalized Likelihood for Uncertainty Estimation (GLUE) (Beven and Binley (1992)). Cette méthode se base sur des approches statistiques afin d'estimer la vraisemblance des solutions proposées, voire de l'ensemble des paramètres. Elle peut donc être utilisée afin de pallier, dans une certaine mesure, le problème d'équifinalité (Schulz et al. (1999)). En revanche, même si ces méthodes peuvent s'avérer efficaces, certains résultats (Rankinen et al. (2006)) suggèrent que l'utilisation d'une plus grande quantité de connaissances préalables est un moyen de réduire efficacement ce problème. Ainsi, d'après Seibert and McDonnell (2002), même si les modélisateurs préfèrent disposer de données précises pour le calage de modèles (*hard data*), l'utilisation de données peu précises supplémentaires (*soft data*) peut permettre de bien meilleurs résultats.

Il semble donc approprié de se pencher sur les méthodes d'optimisation et l'utilisation de connaissances expertes. Nos recherches, bien que théoriques, visent directement à être appliquées dans le contexte de la pêche en Corse. C'est pourquoi, dans une première section nous détaillerons les différents modèles

1. Cette fonction peut aussi être maximisée mais, sauf précision inverse, on supposera par la suite un processus de minimisation

de dynamique de population et le choix d'un modèle pour la Corse. Ensuite nous étudierons ce modèle afin de dégager le problème théorique sous-jacent et toutes les implications à prendre en compte lors de notre processus d'optimisation. Nous montrons dans un premier temps la nécessité, dans certains cas, d'utilisation de méthodes d'optimisation difficile dès la phase de calage, notamment après introduction de connaissances expertes contraignantes. Cette première étude a été entièrement réalisée sur des données autogénérées à partir du modèle afin de s'assurer de leur cohérence et fiabilité dans un premier temps. Naturellement, par la suite, nous montrons les limites supplémentaires engendrées par l'utilisation de données réelles et proposons des méthodes pour les gérer.

2.1 Modèles de dynamique de population

2.1.1 Modèles biologiques de croissance

Il existe plusieurs variantes de ce type de modèle. Notre choix sera dicté par les contraintes de notre application et surtout les données à notre disposition.

Nous pouvons citer le modèle de Fox Jr (1970) qui est un modèle logarithmique basé sur le modèle de croissance de *Gompertz* présenté dans Tjørve and Tjørve (2017)).

Dans ses travaux Roopnarine (2013) teste les hypothèses du modèle de Hardin (1968) sur le partage de ressources communes en l'intégrant à deux modèles mathématiques : un modèle écologique de croissance démographique et un modèle proies-prédateurs.

Le modèle de Pella and Tomlinson (1969) est lui une généralisation du modèle de *Graham-Schaefer*. Il permet de tenir compte de l'asymétrie de la courbe de production des stocks en fonction de la taille de la population et propose un ajustement qui détermine la courbe de production du stock pour une population exploitée en utilisant uniquement l'historique des captures et l'effort de pêche.

Le modèle de *Graham-Schaefer* Graham (1935); Schaefer (1954, 1959) est une évolution du modèle de Verhulst (1845), modèle logistique d'accroissement de population, pour lequel on introduit une pêcherie. C'est un modèle bio-économique qui permet d'estimer l'évolution d'une population ou d'une biomasse soumise à une activité de pêche.

Il est basé sur le modèle initial de Graham (1935) auquel Schaefer (1954, 1959) a ajouté une composante modélisant l'activité de pêche.

Il s'exprime de la manière suivante :

$$\frac{dB(t)}{dt} = r\left(1 - \frac{B(t)}{k}\right)B(t) - qE(t)B(t) \tag{2.1}$$

Avec :

— k : la biomasse à l'équilibre, ou capacité de charge (en masse) ;

— r : le taux de croissance ;

— $E(t)$: l'effort de pêche déployé sur une unité de temps, et

— q : le coefficient de capturabilité ;

— on peut définir $C(t) = qE(t)B(t)$: les captures.

La résolution de cette équation nécessite la connaissance d'un paramètre n'apparaissant pas explicitement à savoir la valeur de la biomasse à l'instant initial, notée B_0.

Ce modèle a l'avantage d'être simple à utiliser et répertorie l'ensemble des paramètres écosystémiques qui décrivent l'évolution de la population et qui permettent l'élaboration d'une stratégie de pêche à appliquer.

D'autres versions du modèle de *Graham-Schaefer* ont été étudiées plus récemment, en prenant certaines conditions en compte, notamment l'aléatoire comme dans Shah and Sharma (2003).

D'autres *familles* de modèles existent comme les modèles proie-prédateur dont le plus connu est le modèle de Lotka (1956). Celui-ci a été étudié et décliné de nombreuses fois Padua and Ontoy (2010) Tahara et al. (2018). On peut également penser aux modèles structuraux dont le principal représentant est le modèle de Von Bertalanffy (1938) ou encore des modèles plus axés informatique comme les variantes de DYNFISH Versmisse (2008); Versmisse et al. (2007).

Parmi toutes ces propositions de nombreuses pistes nous intéressent, mais aucun modèle n'est adapté aux spécificités de la pêche corse.

2.1.2 Un modèle pour la Corse

Fournir un modèle bio-économique spatialisé de confiance pour la gestion des pêcheries en Corse est un enjeu majeur. La mise à disposition d'un tel modèle pourra fournir aux gestionnaires du milieu halieutique un outil efficace pour une évaluation dynamique des conséquences des mesures envisagées, telles que la modification des périodes de pêches, la réévaluation des quotas, la modification des maillages, ou encore la délimitation de zones protégées.

Un modèle bio-économique doit permettre de mesurer les conséquences environnementales, mais aussi économiques des mesures proposées. Si l'aspect artisanal des pêcheries et la complexité des courants présents autour de l'île imposent l'élaboration d'un modèle dédié, des besoins similaires sont observés sur l'ensemble des zones où intérêts économiques et protection de l'environnement cohabitent.

L'exploitation en Corse est basée sur une pêcherie aux petits métiers et multi-espèces, c'est-à-dire qu'à la différence de certaines pêcheries industrielles mono-spécifiques, il y a un grand nombre d'espèces d'intérêt exploitées par la plupart des pêcheurs. Cette spécificité rend l'acquisition des données associées aux prises réelles très difficile. En effet, là où une pêcherie mono-spécifique peut se permettre des études précises sur la dynamique de population d'une espèce en particulier, il nous est impossible de réaliser cela sur les 6 espèces d'intérêt principales de notre projet et encore moins sur l'ensemble des 50 réellement exploitées.

C'est pourquoi dans ces travaux, nous nous sommes concentrés sur des modèles globaux qui ne prennent en considération que l'ensemble de la biomasse (stock) à un temps donné. Ils nécessitent peu de connaissances par rapport à des modèles plus précis comme les modèles structuraux ; seuls les indices d'abondance des stocks et les captures correspondantes sont nécessaires.

En conséquence, ils peuvent être utilisés même si les données sont limitées, alors que les modèles structuraux nécessiteront plus de connaissances et des données plus spécifiques qui ne sont pas forcément toutes mesurables ou déductibles empiriquement comme l'a montré Chaloupka and Balazs (2007). Même si les modèles globaux sont moins réalistes que les modèles structuraux, ils sont utiles et fournissent de bonnes indications comme les grandes tendances encore utilisées pour fixer les quotas de pêche.

Les modèles structuraux peuvent présenter des résultats plus réalistes en prenant en compte des paramètres spécifiques aux espèces étudiées, mais dans le cas de pêcherie multi-espèces, l'intérêt d'utiliser ce type de modèle représenterait un coût d'acquisition de données très important comparé aux résultats que l'on pourrait obtenir.

Dans le cas de notre projet, il est difficile, voire impossible de représenter une ressource multi-espèces selon une unique caractéristique, car les données nous manquent. Ainsi, nous ne pouvons pas, à l'heure actuelle, nous permettre l'utilisation d'un modèle structural. Notre choix s'est donc porté vers l'utilisation du modèle de *Graham-Schaefer* Graham (1935); Schaefer (1954, 1959), qui semble le plus adapté aux données que nous possédons. En effet, la campagne d'acquisition de données de programme *MoonFish* n'est pas terminée, et ne sera vraisemblablement pas suffisante. Nous basons donc nos premiers essais sur des données trouvées dans la littérature issue de Le Manacha et al. (2011).

Dans cet article, les auteurs présentent :

— une estimation des prises de certaines espèces d'intérêt de 1950 à 2008 ;

— une estimation de la pêche récréative de 1950 à 2008 ;

— l'évolution du nombre d'exploitants de 1950 à 2008 ;

Leurs estimations sont discutables, en effet il semble y avoir des années manquantes ou non cohérentes dans les données des sardines et leur protocole d'estimation n'est pas détaillé.

Elles sont cependant un exemple représentatif de ce que nous aurons à l'issue de la campagne de collecte de données de *MoonFish*.

Ainsi, leurs estimations nous serviront de base pour tester nos approches. Nous avons donc choisi un modèle et possédons des données préliminaires de référence.

2.2 Vers les méthodes d'optimisation difficile : définition du problème

Dans cette section, nous allons étudier les caractéristiques du modèle de dynamique de population précédemment choisi. Cela nous permettra de dégager le problème sous-jacent et de mieux définir la portée théorique de notre travail.

2.2.1 Première approche de calage : algorithme de Levenberg-Marquardt sur données parfaites autogénérées

Dans sa formalisation la plus simple du problème de calibration du modèle, celui-ci peut se résumer à un problème convexe de type moindres carrés. Nous allons donc commencer par étudier sa résolution par un algorithme de Levenberg-Marquardt Moré (1978).

Le modèle étant exprimé sous forme d'équation différentielle, le choix d'un schéma d'analyse numérique est nécessaire à sa simulation. La figure 2.1 montre un exemple de comparaison de l'évolution de la biomasse suivant différents schémas d'analyse. On peut voir que de très légères différences sont présentes par moment, mais sont vite compensées et ne mènent pas à une instabilité numérique. Dans le cas où les efforts de pêche sont très importants et que la population tend vers 0, il peut arriver que les différences ne soient plus négligeables entre la méthode d'Euler et les méthodes Runge-Kutta d'ordre 2 ou 4. Ces dernières sont cependant toujours superposées. Dans la suite de notre étude, nous utiliserons donc une méthode de type Runge-Kutta d'ordre 2.

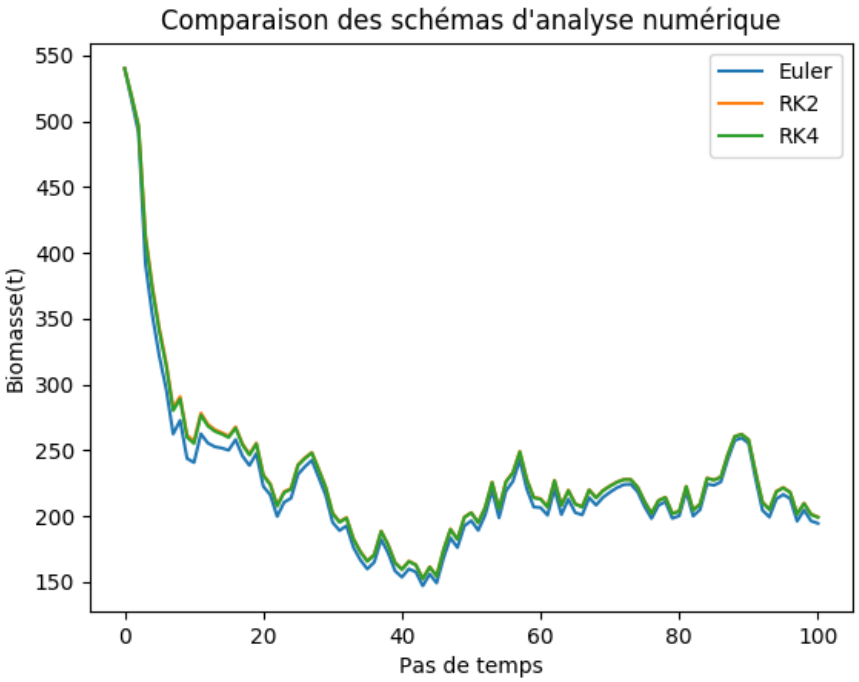


FIGURE 2.1 – Exemple d'évolution de la biomasse par simulation du modèle de Graham-Schaefer suivant différents schémas d'analyse numérique

Si l'on dispose d'une série de captures, C, et des efforts de pêche associés, E, il est possible d'en déduire, via des méthodes de régressions linéaires, les valeurs des paramètres biologiques p=(r, q, k, B0) qui définissent le modèle d'évolution de la population. L'algorithme de Levenberg-Marquardt est une

méthode itérative de minimisation qui consiste, à partir d'un point initial p_0, à calculer un nouveau point qui diminue la valeur de la fonction objectif. La fonction que nous chercherons à minimiser est l'erreur résiduelle pondérée entre les différentes captures effectuées au cours du temps et des valeurs de la fonction C(t) du modèle de Graham-Schaefer aux temps t :

$$\delta^2(p) = \sum_{i=0}^{n} \left(\frac{C_i - C(t_i, p)}{\omega_i} \right)^2 \tag{2.2}$$

Avec :

- n : nombre de relevés de pêche ;
- C_i : captures réalisées lors du ième relevé ;
- $C(t_i, p)$: captures simulées pour le point p pour le ième relevé ;
- ω_i : pondération associée à chaque mesure, indiquant la fiabilité que l'on accorde à la mesure. En général égale à 1 $\forall$ i.

La méthode du gradient consiste à fixer la perturbation dans la direction de la plus forte pente fournie par le gradient de la fonction C en p_i. Dans ce cas le point initial ne doit pas être trop éloigné du minimum global, pour ne pas être arrêté par un minimum local, et le pas de la perturbation pas trop important pour éviter un comportement chaotique ou divergeant.

Le gradient de $\delta = \frac{\partial \delta^2(p)}{\partial p} = -2[C_i^{'} - C_i(p)]^t W[\frac{\partial C_i(p)}{\partial p}] = -2J^T W[C_i^{'} - C_i(p)]$ permet d'obtenir l'estimation de la perturbation $eta_g = \alpha J^T W[C_i^{'} - C_i(P_j)]$.

La méthode de Gauss Newton est bien plus stable que celle de descente du gradient, elle permet la convergence dans la plupart des cas, quel que soit le pas et le point d'initialisation de la fonction. Elle consiste à approximer la fonction à minimiser par une fonction quadratique au voisinage de la solution grâce à son développement limité :

$$C_i(P_{j+1}) \approx C_i(p_j) + \frac{\partial C_i(p_j)}{\partial p} \eta_{gn} = C_i(p_j + J\eta_{gn}) \tag{2.3}$$

Donc :

$$\delta^2(p_{j+1}) \approx C_i^{'t} W C_i^{'} - 2C_i^{'t} W C_i(p_j) - 2C_i^{'t} W J \eta_{gn} + [C_i(p_j) + J\eta gn]^t W [C_i(p_j) + J\eta_{gn}] \tag{2.4}$$

$$\frac{\partial \delta^2(p_{j+1})}{\partial \eta_{gn}} \approx -2[C_i^{'} - C_i(p_j)]^2 W J + 2\eta_{gn}^t J^T W J \tag{2.5}$$

La valeur de la perturbation qui minimise l'erreur quadratique est donnée par l'annulation de cette dérivée :

$$\eta_{gn} = [J^T W J]^{-1} J^T W [C_i^{'} - C_i(p_j)] \tag{2.6}$$

L'inconvénient de cette méthode est qu'elle est plus lente que celle du gradient. La méthode de

Levenberg-Marquardt, présente la particularité d'évoluer entre les méthodes de Gauss-Newton et de descente du gradient. Cette méthode fonctionne très bien dans la plupart des domaines auxquels elle s'applique et constitue donc un standard pour la résolution des problèmes des moindres carrés pour des modèles non linéaires. L'équation de la perturbation est donnée par :

$$\eta_{lm} = [J^T W J + \lambda diag(J^T W J)]^{-1} J^T W [C_i^{'} - C_i(p_j)] \tag{2.7}$$

— W est la matrice des poids affectés à chaque valeur ;

— J est la matrice jacobienne associée au système ;

— λ est une variable servant à accélérer ou diminuer les variations.

Elle se comporte comme la méthode de Gauss-Newton lorsque l'on est éloigné de la solution et plus on s'en rapproche, plus la méthode évolue vers une méthode du gradient adaptatif. En effet, en jouant sur la valeur de λ on peut passer progressivement d'une méthode à l'autre. Si λ est grand, les coefficients de la diagonale sont importants et $[J^T W J]$ est négligeable devant la matrice diagonale $\lambda diag(J^T W J)$ ce qui revient à une perturbation de type gradient. Si λ est petit le terme se comporte comme $[J^T W J]$ et on se trouve dans le cas de la méthode de Gauss-Newton.

Le calcul de la perturbation ne dépend donc que des dérivées premières de la fonction C(t) au point p et de l'inversion d'une matrice $m \times m$ ou m est le nombre de paramètres à estimer. De plus si nous choisissons d'affecter le même poids à toutes les mesures la matrice W disparaît des équations. La valeur de la perturbation qui minimise l'erreur quadratique est donc donnée par :

$$\eta_{lm} = [J^T J + \lambda diag(J^T J)]^{-1} J^T [C_i^{'} - C_i(p_j)] \tag{2.8}$$

Avec les valeurs de la matrice jacobienne suivantes :

$$\frac{\partial C(t)}{\partial B_0} = \left(\frac{kX}{YB_{0]}}\right)^2 qE(t)e^{-rXt} \tag{2.9}$$

$$\frac{\partial C(t)}{\partial k} = \frac{qEX}{Y}\left(1 - \frac{Xk}{YB_0}e^{-rXt}\right) \tag{2.10}$$

$$\frac{\partial C(t)}{\partial q} = \frac{Ek}{Y}\left(X - qE\left(\frac{1}{r} + \frac{X}{Y}e^{-rXt}\left(\frac{kXt}{B_0} - t - \frac{k}{rB_0}\right)\right)\right) \tag{2.11}$$

$$\frac{\partial C(t)}{\partial r} = \frac{qEk}{Y}\left(\frac{qE}{r^2} - \frac{X}{Y}e^{-rXt}\left(\frac{kqE}{B_0 r^2} + \left(1 - \frac{kX}{B_0}\right)t\right)\right) \tag{2.12}$$

Avec :

$$X = 1 - \frac{qE(t)}{r} \tag{2.13}$$

$$Y = 1 + \left(\frac{kX}{B_0} - 1\right)e^{-rXt} \tag{2.14}$$

Finalement nous utilisons l'algorithme 1. En fixant le point p et l'ensemble E d'efforts à des valeurs arbitraires, il est possible de générer les captures C correspondantes en faisant simplement tourner le modèle. On obtient ainsi des jeux de données d'efforts et de captures parfaits.

La figure 2.2 montre la convergence de l'algorithme de Levenberg-Marquardt sur un exemple. On peut constater que la convergence est très rapide, seulement une dizaine d'itérations et les résultats obtenus sont très proches de l'optimal théorique.

Algorithme 1 Algorithme de Levenberg-Marquardt

Fixe une valeur pour $\lambda = max(diag(J^T J))$; $\lambda_{up} = 11$; $\lambda_{dw} = 9$
Fixe la précision ϵ et un point de démarrage $p = p_0$; continue=true
tant que continue **faire**
  Calcul de la perturbation $\eta_{lm} = [J^T(p)J(p) + \lambda diag(J^T(p)J(p))]^{-1} J^T[C' - C_i(p_j)]$
  si $||\delta^2(p) - \delta^2(p + \eta_{lm}|| < \epsilon||p||$ **alors**
    continue=false
  sinon
    Calcul du facteur d'acceptabilité $p(\eta_{lm}) = ||\delta^2(p)|| - \frac{||\delta^2(p+\eta_{lm})||}{2\eta_{lm}(\lambda\eta_{lm} + J^T(\overline{X} - R_i(p))}$
    si $p(\eta_{lm}) > \epsilon$ **alors**
      $\lambda = max(\frac{\lambda}{\lambda dw}, \frac{1}{\lambda_{max}})$
      $p = p + \eta_{lm}$
    sinon
      $\lambda = min(\lambda * \lambda_{up}, \lambda_{max})$
    fin si
  fin si
fin tant que

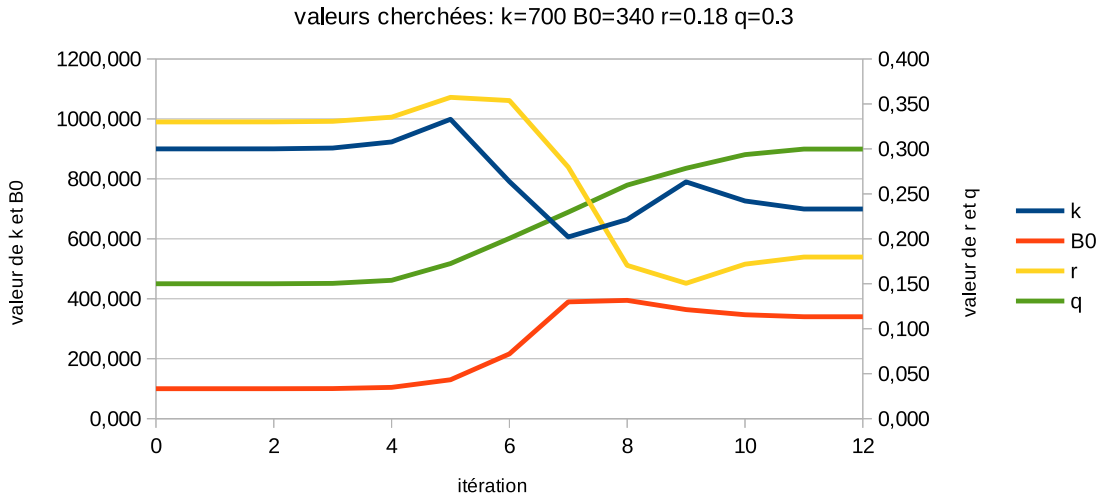


FIGURE 2.2 – Convergence de l'algorithme de Levenberg-Marquardt

Un premier problème se pose alors : même avec une connaissance parfaite des ensembles E et C,

plusieurs solutions peuvent être possibles.

2.2.2 Étude de l'espace des solutions

Étude de l'espace des solutions avec connaissances parfaites

L'espace des paramètres étant continu, il est impossible de réaliser une recherche exhaustive complète pour parfaitement caractériser l'espace des solutions. Cependant, une discrétisation de l'espace des paramètres permet de se faire une bonne idée. L'algorithme 2 permet de rechercher l'ensemble des solutions possibles à un calage dans l'espace des paramètres discrétisé.

Algorithme 2 Recherche dans l'espace des paramètres discrétisé

Variables $nbAnnees, E[nbAnnees], C[nbAnnes], nbIntervalles$
$mink, maxk, minB0, maxB0, minq, maxq, minr, maxr$
pour $i = 0; i < nbIntervalles; i++$ **faire**
  $k = minK + (maxK - minK) * i/(nbIntervalles - 1)$
  pour $j = 0; j < nbIntervalles; j++$ **faire**
    $B = minB + (maxB - minB) * j/(nbIntervalles - 1)$
    pour $\_k = 0; \_k < nbIntervalles; \_k++$ **faire**
      $q = minQ + (maxQ - minQ) * \_k/(nbIntervalles - 1)$
      pour $l = 0; l < nbIntervalles; l++$ **faire**
        $r = minR + (maxR - minR) * l/(nbIntervalles - 1)$
        $ok = simulationSchaefer(k, B0, q, r, E, C)$
        si ok **alors**
          $enregistre(k, B0, q, r)$
        fin si
      fin pour
    fin pour
  fin pour
fin pour

En exécutant cet algorithme avec les bornes de paramètres, arbitrairement choisies pour ce test, suivant le tableau 2.1, 10 ans de simulation et 100 intervalles par paramètre, on obtient les solutions présentées dans le tableau 2.2.

	min	max
k	200	2000
B0	100	1000
q	0	1
r	0	1
E $\forall t$	1	1
C $\forall t$	200	200

TABLE 2.1 – Intervalles de l'espace des paramètres

k	B0	q	r
1200	300	2/3	8/9
1200	600	1/3	2/3
1200	900	2/9	8/9
1400	300	2/3	0,85
1800	900	2/9	4/9
2000	900	2/9	0,4

TABLE 2.2 – Solutions trouvées par discrétisation en 100 intervalles

On peut déjà observer que plusieurs solutions sont possibles, montrant déjà un problème d'équifinalité. De plus, plusieurs solutions présentent une même valeur pour le paramètre k. De même, plusieurs solutions possèdent une même valeur de paramètre $B0$, q ou r. Il semble alors naturel de se demander si l'ensemble des solutions est réellement discret ou si au contraire cet espace est continu.

Les valeurs de E et C, fixes au cours du temps, nous permettent de nous placer dans le cas d'une exploitation stable durable. Ce cas particulier permet une étude mathématique plus simple de la situation. Il permet de fixer $\frac{dB(t)}{dt} = 0$, permettant de simplifier les calculs car :

$$C(t) = qE(t)B(t) \tag{2.15}$$

$$\frac{\partial C(t)}{\partial B(t)} = qE(t) \tag{2.16}$$

$$\partial C(t) = qE(t)\partial B(t) \tag{2.17}$$

Donc si q et $E(t)$ sont fixes non nuls, le seul moyen d'avoir une variation de captures nulle au cours du temps est d'avoir une variation de biomasse nulle au cours du temps. De plus, ce cas pourrait arriver en réalité, les conclusions auxquelles mènera ce cas particulier devront donc être prises en compte.

Dans ce cas, l'étude de la variation d'un seul pas de temps suffit à être valable sur toute la durée de simulation. Il n'est donc pas nécessaire de prendre en compte les variations de $B(t)$ au cours du temps pour exprimer les différents paramètres. Ici on peut donc facilement exprimer les paramètres en fonction des autres ainsi que la variation de chacun en fonction de la variation d'un autre. Par exemple :

$$q = \frac{C}{EB_0} \tag{2.18}$$

$$\frac{\partial q}{\partial B_0} = -\frac{C}{EB_0^2} \Leftrightarrow \partial q = -\frac{C}{EB_0^2}\partial B_0 \tag{2.19}$$

Ceci est faisable pour chacune des variables du problème. Ainsi, une solution est bonne tant qu'il existe des valeurs pour lesquelles ces calculs rentrent dans l'intervalle de définition de chacune des variables. De plus, si une solution existe et que l'espace des paramètres est continu et non réduit à un point, il est toujours possible de compenser une variation infinitésimale d'un paramètre par une va-

riation infinitésimale d'un autre paramètre ou couple de paramètres. Il est donc possible de dire que dans ce cas là, l'ensemble des solutions est un espace continu contenant une infinité de solutions. En revanche, cet espace est non convexe. Reprenons les solutions trouvées dans le tableau 2.2. Par définition, un espace, Co, est convexe si $\forall A, B \in Co^2, \forall t \in [0,1], (1-t)A + tB \in Co$. Un espace n'est donc pas convexe si $\exists t \in [0,1], (1-t)A + tB \notin Co$. Nous avons vu que dans notre exemple de cas particulier, une solution est valide si $C = 200$ et $\frac{dB(t=0)}{dt} = 0$ et que les valeurs des paramètres sont inclues dans leur espace de définition. En prenant les points $A = (k = 1200, B_0 = 300, q = 2/3, r = 8/9)$ et $B = (k = 1200, B_0 = 600, q = 1/3, r = 2/3)$ et en posant $t = 0.2$, l'application de la définition d'un espace convexe nous donne le point $p = (k = 1200, B_0 = 360, r = 0.8444..., q = 0.6)$. On a alors $C_p = 216$ donc $p \notin Co$. L'espace des solutions n'est donc pas convexe. L'utilisation de ce point nécessite au moins une adaptation de la valeur de E.

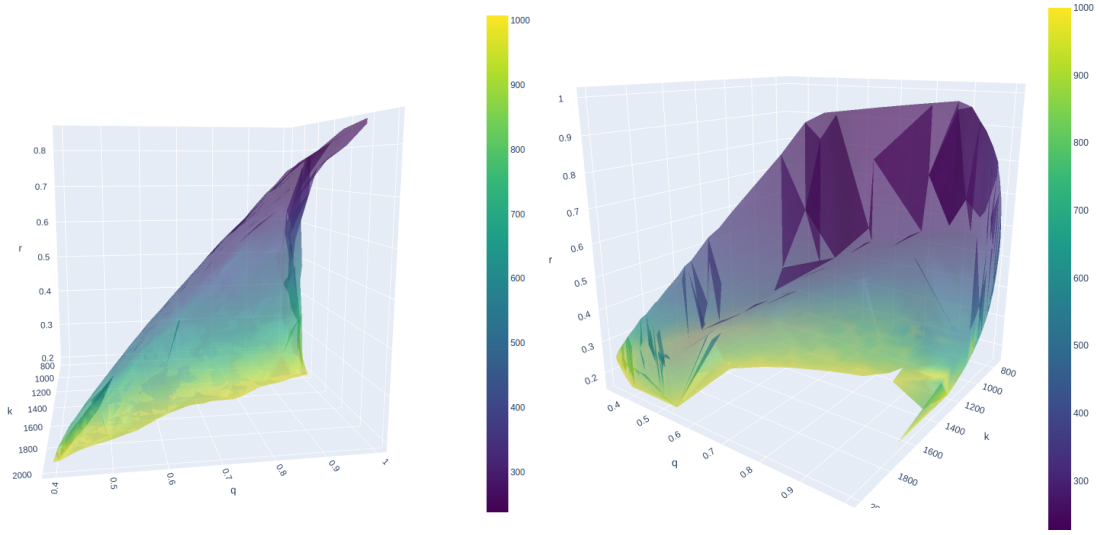
Nous allons donc maintenant nous intéresser à l'étude d'un calage avec incertitudes sur les données d'entrée.

Calage avec incertitudes sur les données

Nous venons donc de voir que même avec une connaissance parfaite des séries temporelles d'efforts et de captures, une infinité de solutions peut exister. Dans un cadre réel, il est impossible de connaître ces valeurs de manières parfaites. Ainsi, en réalité, il faut prendre en compte un intervalle de valeurs possible pour chacune des valeurs d'entrée.

Or, par définition, en utilisant un intervalle de valeur, $[Emin_t, Emax_t], [Cmin_t, Cmax_t] \forall t$ tel que $E(t) \in [Emin_t, Emax_t], C(t) \in [Cmin_t, Cmax_t] \forall t$ plutôt qu'uniquement les valeurs $E(t)$ et $C(t)$, alors si $\exists f(E,C)$ tel que $f(E,C) \rightarrow S_p$, S_p un ensemble, correspondant ici à l'ensemble des solutions à un calage parfait, $\int_{Emin_0}^{Emax_0} \int_{Cmin_0}^{Cmax_0} ... \int_{Emin_n}^{Emax_n} \int_{Cmin_n}^{Cmax_n} f(E,C) = S_c$ et $S_p \subseteq S_c$.

En généralisant et de façon plus simple, agrandir l'espace des paramètres ne peut mener qu'à un espace des solutions au moins aussi grand qu'avant. La figure 2.3 montre les espaces de solutions d'un calage avec une faible variation, 2%, possible d'effort (figure 2.3a) et une avec une variation d'effort relativement importante, 20% (figure 2.3b). On peut bien voir que l'ensemble des solutions de a sont incluses dans b. De plus, dans le cas de l'effort restreint, l'espace des solutions couvrent 0.005253% de l'espace des paramètres. Dans le cas d'une large plage de valeur d'effort possible, celui-ci représente 0.548264% de l'espace des paramètres. Le problème d'équifinalité s'accentue donc de manière très forte avec l'augmentation des incertitudes dans les données d'entrée.

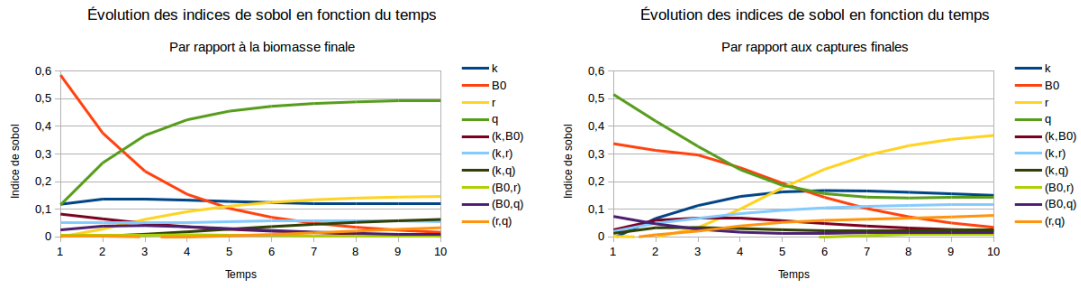


(a) $E_t \in [0.98, 1.02]\ \forall t$ (b) $E(t) \in [0.8, 1.2] \forall t$

FIGURE 2.3 – Espace des solutions d'un calage avec $k \in [200, 2000]$, $B_0 \in [100, 1000]$, $q, r \in [0, 1]^2, C(t) = 200 \forall t$

Pour ces graphiques et pour tous ceux de cette forme qui seront présents dans la suite de ce document, sauf précision contraire, nous avons fait le choix de ne pas représenter le paramètre de biomasse initiale car après quelques années de simulation, il n'a plus réellement d'importance. Il semble donc moins important de connaître sa valeur précisément. Nous le représentons donc par une échelle de couleur.

La figure 2.4a montre l'évolution des indices de sobol associés à chaque variable et couple de variables en fonction du nombre d'années de simulation par rapport à la valeur de biomasse finale. De même la figure 2.4b montre l'évolution des indices de sobol associés à chaque variable, et couple de variables, en fonction du nombre d'années de simulation par rapport à la valeur de capture finale. Dans les deux cas, on peut voir que la biomasse initiale ne joue qu'un faible rôle.



(a) Importance des variables par rapport à la valeur de biomasse finale (b) Importance des variables par rapport à la valeur de captures finales

FIGURE 2.4 – Évolution des indices de sobol en fonction du temps de simulation

Paramètre	Min	Max	C
k	200	2000	
B0	100	1000	
q	0	1	
r	0	1	
E0	0,85	0,95	145
E1	1,13	1,23	166
E2	1,13	1,23	142
E3	1,09	1,19	121
E4	1,05	1,15	106
E5	1	1,1	94
E6	0,85	0,95	77
E7	1,03	1,13	91
E8	0,88	0,98	75
E9	1	1,1	85

TABLE 2.3 – Valeurs des paramètres de test pour le calage d'une exploitation variable

La connaissance de cette sensibilité peut cependant permettre d'adapter l'algorithme 2 pour diminuer ou augmenter le nombre d'intervalles de discrétisation de chaque paramètre en fonction de son importance. Cela peut permettre une diminution de la quantité de calcul pour les variables de moindre importance et une augmentation pour les plus importantes, améliorant ainsi la précision des résultats.

Généralisation à tous les cas

Nous venons donc de voir que dans le cas où $\frac{dB(t)}{dt} = 0 \ \forall t$, il existe généralement un espace continu contenant une infinité de solutions. De même, l'utilisation d'intervalles de valeurs de validité pour E et C ne provoque qu'un agrandissement de l'espace des solutions. Mesurons maintenant quel impact une exploitation variable a sur ces résultats.

Dans ce cas, l'expression de chacune des variables en fonction des autres dépend de l'évolution de B, E et C au cours du temps. Il devient alors trop difficile de les exprimer mathématiquement. Nous n'utiliserons donc dans cette partie que des résultats de simulations et calibrations.

Nous sommes donc repartis du protocole précédent en utilisant des données de captures et d'efforts générées à partir d'un modèle déjà calibré. L'espace des paramètres est donné en tableau 2.3.

La figures 2.5 montre l'espace des solutions du calage utilisant les paramètres précédents. L'espace des solutions ne couvre alors que 0.002749% de l'espace des paramètres alors que les intervalles d'efforts étaient relativement étendus.

Ces figures nous permettent de constater que nous avons cette fois encore un espace des solutions continu. Cela ne constitue pas une preuve formelle en soi, mais au vu des analyses précédentes, nous pensons qu'il est raisonnable de supposer que cela est le cas pour toutes les situations où des solutions existent.

Dans tous les cas, la couverture de 0.002749% de l'espace des paramètres par l'espace des solutions, continu ou pas, et l'étendue des valeurs engendrée, représentent un espace d'incertitude beaucoup trop

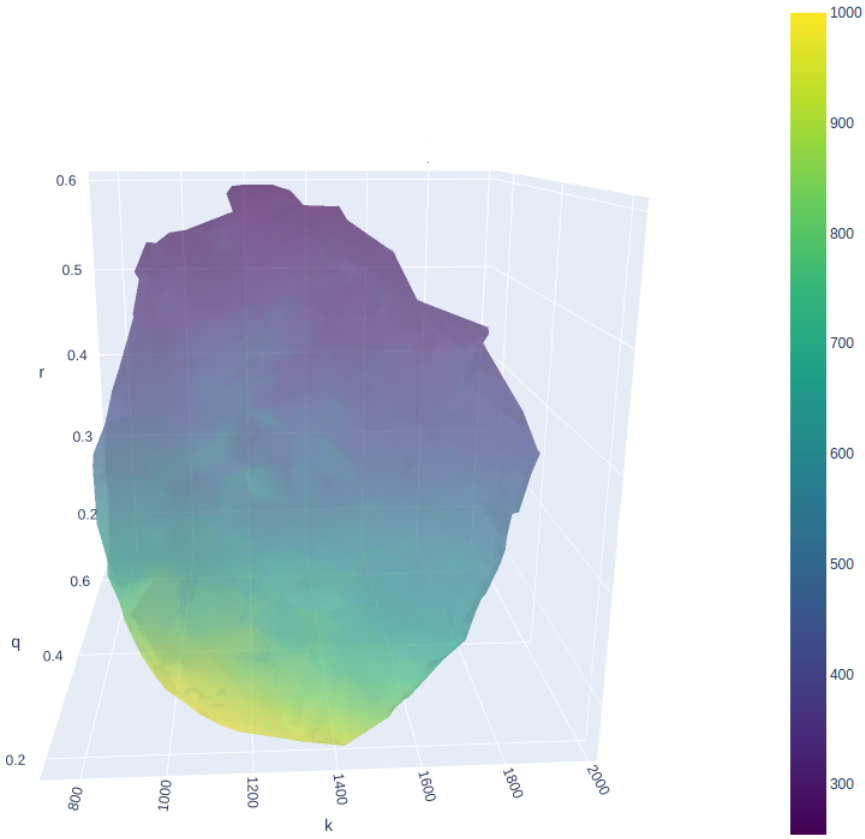


FIGURE 2.5 – Espace des solutions d'un problème de calage d'une exploitation variable

important pour considérer un calage automatique à partir de séries temporelles de captures et d'efforts de pêche uniquement. Nous allons donc nous pencher sur les méthodes utilisant des connaissances expertes.

2.2.3 Utilisation de connaissances expertes

Les ensembles de solutions pouvant s'appliquer à la calibration à partir de séries temporelles de captures et d'efforts au cours du temps sont généralement trop grands pour être utilisables. Nous reviendrons plus en détails sur cette affirmation dans le chapitre 3.1

L'intégration de connaissances expertes peut se faire directement sous la forme de contraintes à respecter. La communication entre experts d'un domaine spécifique et informatique peut s'avérer problématique et la traduction de connaissances approximatives en langage informatique peut parfois être un défi en soi. Le cas le plus simple et classique est certainement des contraintes limitant l'étendue de l'espace des paramètres par réduction de l'intervalle de définition d'un paramètre. Ces contraintes restreignent donc indubitablement l'espace des solutions à un sous-ensemble de l'espace des paramètres initial. Ce cas ne pose de difficulté de prise en compte que pour les méthodes à base de gradient les moins robustes, nous

ne nous attarderons donc pas sur ce cas.

Cette réduction peut cependant s'avérer indirecte. Par exemple, pour une étude pouvant être menée sur plusieurs espèces à la fois, même en restant sur notre modèle simple, un modélisateur pourrait exprimer des connaissances comparant les espèces, avec en langage humain "l'espèce a se reproduit moins vite que l'espèce b". On aurait alors informatiquement $r_a < r_b$. Se pose alors d'autres questions telle que "leur exploitation se fait-elle toujours conjointement ?", phénomène impliquant que $E_a(t) = E_b(t) \forall t$ restreignant grandement l'espace des paramètres, et que donc les différences de captures ne sont impactées que par $q_a, q_b, B_a(t) et B_b(t)$. À l'inverse on peut obtenir la connaissance qu'une certaine proportion, α, des exploitants pêchent les deux espèces conjointement avec le même engin et d'autres, de proportion inconnue, ne ciblent qu'une des deux. Cela pourrait être notamment le cas lorsque la pêche récréative sur une espèce joue un rôle non négligeable par exemple. Dans ce cas, l'espace des paramètres s'agrandit artificiellement par une augmentation du nombre de paramètres mais ceux-ci peuvent finalement être agrégés en l'ensemble des paramètres initiaux. La complexification du problème engendré provoque donc quand même une réduction de l'espace des solutions une fois les composantes agrégées.

Ainsi, si précédemment l'ensemble des solutions était toujours continu, ceci peut ne plus être valide dans ce type d'étude.

De plus, la prise en compte des contraintes, notamment quand certains paramètres influencent la validité d'autres, peut grandement mettre à mal l'utilisation de certains algorithmes. Il devient par exemple très difficile, voire impossible, d'exprimer des gradients de manière analytique. Problème rendant difficile l'utilisation de méthodes de type descente de gradient ou algorithme de Levenberg-Marquardt. Quand les contraintes sont relativement simples et qu'il est raisonnable de supposer qu'une des séries temporelles C ou E est parfaite, l'expression de la deuxième en fonction de la première peut permettre une recherche exhaustive dans l'espace des paramètres discrétisé comme présenté précédemment. Cette approche arrive cependant très vite à ses limites à cause de l'explosion combinatoire.

Il devient donc rapidement nécessaire de s'orienter vers des méthodes d'optimisation difficile de type métaheuristiques. Nécessité d'autant plus accentuée lors de l'utilisation de données réelles comme nous le verrons dans la section suivante.

2.3 Limites d'un cas réel

2.3.1 Données réelles et problématique

Les données présentées dans (Le Manacha et al. (2011)) sont discutables, car les captures présentées ne sont que des estimations de la réalité et les mesures d'efforts ne sont quantifiables qu'en terme de nombre de bateaux. Elles sont cependant un bon exemple de ce que l'on peut espérer avoir. Dans nos tests nous partirons du postulat que leurs estimations sont cohérentes afin de tester nos algorithmes et pouvoir mettre en lumière leur efficacité, mais également leurs problèmes.

On peut se servir des données de captures totales pour estimer les valeurs de C sur la période étudiée (1950-2008) et estimer les efforts de pêche associés en nous basant sur le nombre de bateaux professionnels,

auxquels on ajoutera la proportion de pêche récréative.

La formule suivante est alors utilisée :

$$Effort(t) = \frac{nombreBateaux(t)}{1 - priseRecreative(t)/priseGlobales(t)} \tag{2.20}$$

De plus, comme il est plus facile et cohérent de manipuler un effort relatif (Rafalimanana (2003)), nous utiliserons l'effort calculé pour l'année de départ, 1950, comme référence. On obtient alors la formule :

$$Effort(t) = \frac{nombreBateaux(t)}{1 - priseRecreative(t)/priseGlobales(t)} \times \frac{1}{Effort(1950)} \tag{2.21}$$

Les estimations des prises pour chaque espèce sont également intéressantes, mais il est irréaliste de supposer que l'effort global est représentatif de chaque espèce. Nous n'avons donc pour l'instant aucune information quant à l'effort de pêche appliqué sur chaque espèce par intervalle de temps. Nous nous intéresserons donc à la calibration de la dynamique du stock global.

Une recherche exhaustive dans l'espace des paramètres discrétisé ne nous donne aucune solution même jusqu'à une valeur de $nbIntervalles = 1000$ soit 1000^4 solutions testées, uniformément réparties au sein de l'espace des paramètres. Il semble donc qu'aucune solution ne soit valide pour ces données d'entrée. On arrive alors à une des limites de ce genre d'approche. En effet, l'approche par discrétisation de l'espace des paramètres était utilisable auparavant pour une raison : il était possible de calculer $E(t)$ en fonction de $C(t)$ et inversement, en fonction de nos connaissances. Or dans ce cas, le fait de ne pas avoir de solution parfaite nous force à chercher une solution optimale non parfaite. Le calcul de $E(t)$ en fonction de $C(t)$ est alors impossible, car il impliquerait de supposer que les estimations sont parfaites jusqu'à impossibilité de calcul, donc divergence, et donc rendrait impossible l'exploration d'une très grande majorité de l'espace des paramètres. Ceux-ci doivent donc être ajoutés à l'espace des paramètres, augmentant donc de manière exponentielle le nombre de solutions à tester, rendant donc cette approche inutilisable.

De plus, comme nous l'avons déjà vu, pour une exploitation variable et impactant les stocks, donc sans $E(t_1) = E(t_2), C(t_1) = C(t_2) \forall t_1, t_2$ $et \frac{dB(t)}{dt} = 0 \forall t$, la formulation des gradients devient trop complexe analytiquement, l'utilisation de la méthode des différences finies est alors nécessaire, augmentant le nombre de simulations de manière linéaire avec le nombre de paramètres et la prise en compte de contraintes, même simples, peut s'avérer un véritable défi d'adaptation au cas par cas. Toutes ces raisons rendent l'utilisation de ce type de méthodes peu intéressante ici. Nous devons donc nous orienter vers d'autres types de méthodes d'optimisation telles que les métaheuristiques.

2.3.2 Approche par métaheuristiques

Comme mentionné précédemment, il existe un grand nombre de métaheuristiques et rien ne permet de prédire laquelle est la plus adaptée à une classe de problème. Pour cela, nous avons fait le choix de tester un grand nombre de métaheuristiques. Leurs implémentations et paramétrages sont détaillés en annexe 6.4. Après initialisation de la population initiale, à chaque itération et pour chaque solution potentielle, une simulation de la pêcherie est réalisée en se servant des efforts de pêche connus. Cela va déterminer

des captures associées à la dynamique de population proposée. La solution proposée est alors évaluée en comparant captures réelles, C, et captures simulées, Cs, suivant la fonction suivante :

$$f(x) = \sum_{i=0}^{nbAnnees} [C_i - Cs_i]^2 \tag{2.22}$$

Avant de l'appliquer sur les données réelles, nous avons voulu tester l'efficacité de chacun des algorithmes. Les benchmarks utilisés pour valider les implémentations sont détaillés en annexes 6.5.1. Le détail des benchmarks relatifs au calage de modèle et la paramétrisation des méthodes sont développés en annexes 6.6. Nous avons défini plusieurs configurations de ce problème :

— 10 ou 58 ans de données et donc de simulations, pour coller avec les durées d'estimation de Le Manacha et al. (2011) et ce qu'on pourrait espérer avoir en plus sur une période plus récente ;

— un niveau de connaissance de l'effort de pêche : parfait ; précis (bornes de ± 10% de la valeur à retrouver) ; peu précis (bornes de ± 20% de la vrai valeur).

Sur chacun, nous avons testé 9 algorithmes de la littérature : recherche à voisinage variable généralisée Siarry (2014) Mladenović et al. (2008) (RVVG) ; colonie d'abeilles artificielle Karaboga and Akay (2009) (ABC) ; évolution différentielle Qin et al. (2008) (ED) ; une variante d'essaim particulaire Clerc (2012a) (SPSO2007) ; algorithme des loups gris (GW) Mirjalili et al. (2014) ; algorithme des loups gris amélioré Wang and Li (2019) (IGW) ; une variante d'algorithme génétique (GA) Hamdan (2012) ; algorithme d'optimisation des baleines Mirjalili and Lewis (2016) (WOA).

Pour chacun, nous avons utilisé les mêmes paramètres que dans la publication associée à leur benchmarking mais également d'autres jeux de paramètres qui sont détaillés en annexe 6.6. De plus, nous limitons le temps de calcul à une unité standard de calcul (*Standard Time Unit* (STU) Shcherbina et al. (2002)) qui est équivalent à environ 10s sur un thread d'un processeur Intel i7 de 4GHz.

La figure 2.6 représente le processus d'optimisation du problème avec des connaissances précises. Sans connaissance, nous fixons $E_t > 0 \ \forall t$. Avec une connaissance parfaite, l'ensemble E est fixé et géré par la simulation.

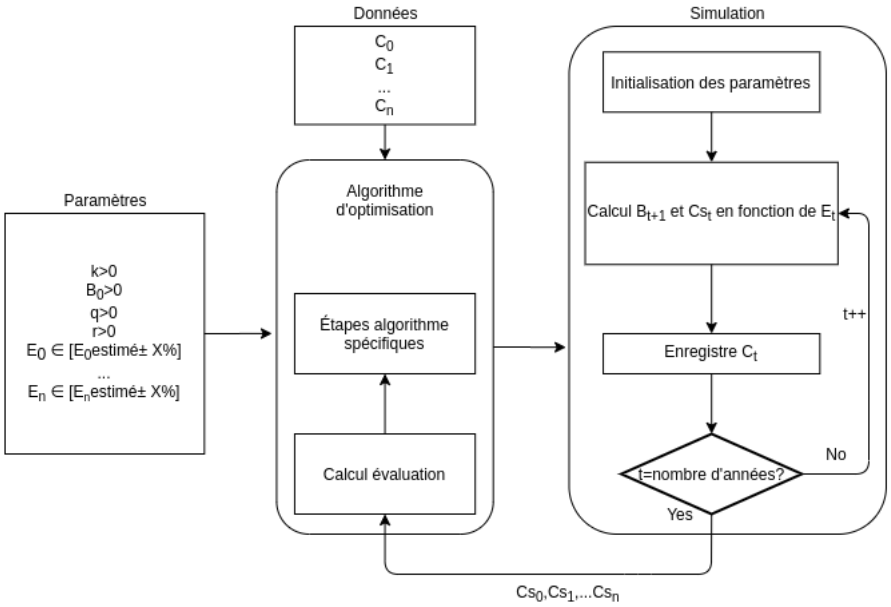


FIGURE 2.6 – Problème de calibration

Le tableau 2.4 montre les résultats de différents algorithmes sur des problèmes de calage avec différents taux de variation dans les efforts au cours du temps ainsi que différentes durées de simulation et différents niveaux de connaissances. Ils sont comparés en utilisant une méthode de Friedman par rang Hodges and Lehmann (2012). De plus, les moyennes et écart-types de ces tests sont présentées en annexe 6.6.3 par le tableau 6.27. Le test de Friedman nous permet de constater que quand le nombre de paramètres est faible, c'est-à-dire ici les cas où les efforts sont parfaitement connus, l'essaim particulaire de 2007 et l'évolution différentielle sont très proches dans la plupart des cas. Nous nous sommes donc intéressé à leur temps de convergence. Le tableau 2.5 montre que dans la plupart des cas il est très difficile de trancher entre les deux algorithmes.

En revanche, dès l'introduction d'incertitudes dans les valeurs d'efforts de pêche, ils trouvent leurs limites au profit de la colonie d'abeilles artificielle ou de la recherche à voisinage variable générale.

Les problèmes avec 58 ans de simulations et de trop fortes variabilités ne présente cependant pas d'algorithmes dont les performances sont significativement meilleures que les autres.

Plus de détails sur ces tests et sur les paramètres utilisés pour chaque algorithmes sont donnés en annexe 6.6.

t	var	u	ABC	PSO	GW	IGW	Baleines	RVVG	DE	P-value
10	0	0	5,65	**2,2**	3,4	3,85	6,45	3,7	2,75	3,11E-15
10	5	0	4,8	1,8	4,2	4,4	5,5	5,55	**1,75**	1,11E-16
10	10	0	4,85	**1,6**	3,8	4,2	6,2	5,5	1,85	1,11E-16
10	20	0	4,9	1,85	3,7	3,55	6,45	6,05	**1,5**	1,11E-16
10	30	0	5,3	1,65	3,65	3,95	6,7	5,4	**1,35**	1,11E-16
58	0	0	6,05	**1,85**	4,15	3,95	5,85	3,9	2,25	1,11E-16
58	5	0	4,4	**1,475**	3,85	5,1	5,9	5,75	1,525	1,11E-16
58	10	0	5	1,55	3,75	4,25	6,25	5,75	**1,45**	1,11E-16
58	20	0	5,65	1,85	3,5	3,5	6,2	5,85	**1,45**	1,11E-16
58	30	0	5,8	**1,5**	3,3	4,25	6	5,65	**1,5**	1,11E-16
10	0	10	4,45	2,15	4,35	4,4	6,9	**1**	4,75	1,11E-16
10	5	10	4,7	2,1	4,25	4,15	6,8	**1**	5	1,11E-16
10	10	10	4,3	2,05	3,9	4,1	6,85	**1,3**	5,5	1,11E-16
10	20	10	3,3	**1,5**	4,75	3,8	6,3	3,95	4,4	2,86E-13
10	30	10	3,15	**1,25**	4,95	3,8	6,75	4	4,1	1,11E-16
58	0	10	2	5,8	4,1	5,05	5,5	**1,25**	4,3	1,11E-16
58	5	10	2,25	5,65	4,2	4,65	5,45	**1**	4,8	1,11E-16
58	10	10	2,1	5,6	3,85	4,35	6,1	**1,1**	4,9	1,11E-16
58	20	10	3,85	4,75	4,5	**2,85**	4,35	3,85	3,85	1,22E-01
58	30	10	3,4	5,9	**2,9**	3,45	4,75	3,4	4,2	3,40E-05
10	0	20	5	2	3,55	4,35	6,8	**1,3**	5	1,11E-16
10	5	20	4,65	2,05	3,75	4,1	6,85	**1,05**	5,55	1,11E-16
10	10	20	4,9	1,95	3,35	4,4	6,95	**1,2**	5,25	1,11E-16
10	20	20	3,45	**1**	4,5	3,55	5,85	5,3	4,35	1,11E-16
10	30	20	3,7	**1,15**	4,5	4,05	5,7	4,65	4,25	7,93E-12
58	0	20	2,4	6	4,35	4,35	5,25	**1,1**	4,55	1,11E-16
58	5	20	2	5,7	4,25	4,1	5,55	**1,05**	5,35	1,11E-16
58	10	20	2,15	6,05	3,45	4	5,95	**1,1**	5,3	1,11E-16
58	20	20	4,35	4,7	4,05	3,5	**3,15**	3,75	4,5	2,27E-01
58	30	20	3,5	4,15	4,2	3,7	**3,3**	4,65	4,5	3,52E-01

TABLE 2.4 – Résultats de différents algorithmes sur différents problèmes de calage, comparaison par test de friedman

En appliquant ces algorithmes à nos données, nous n'obtenons pas mieux que les résultats présentés en figure 2.7

Deux explications s'offrent à nous. La première étant que le modèle n'est pas assez précis pour représenter la complexité d'un écosystème marin réel. La deuxième étant que les données d'entrée sont imparfaites. Il nous est impossible de conclure quant à la première. En revanche, nous avons montré (voir section 2.2.1) que si les relevés de captures/efforts sont cohérents entre eux, il existera toujours des solutions au calage. L'acquisition et les estimations de ce type de données étant particulièrement complexes et sujettes à imprécisions, il est normal que des erreurs soient présentes. Ainsi il est nécessaire de proposer des méthodes pour pallier ce problème, au moins de façons théoriques.

			PSO			DE		
t	var	u	moyenne	ecart-type	Temps	moyenne	ecart-type	Temps
10	0	0	**6,68E-08**	**2,71E-08**	1,36E-02	4,14E-03	1,80E-02	**7,96E-03**
10	5	0	**9,29E-04**	**3,81E-03**	2,45E-02	1,19E-03	4,71E-03	**1,55E-02**
10	10	0	**2,20E-04**	**7,35E-04**	6,72E-02	5,65E-04	2,26E-03	**1,03E-02**
10	20	0	3,62E-02	1,09E-01	4,92E-02	**7,40E-08**	**1,94E-08**	**1,07E-02**
10	30	0	1,17E-07	2,05E-07	3,03E-02	**6,94E-08**	**2,30E-08**	**9,43E-03**
58	0	0	**6,10E-08**	**2,85E-08**	3,23E-02	8,50E-07	1,63E-06	**1,77E-02**
58	5	0	4,37E-06	1,88E-05	4,42E-02	**5,81E-06**	**2,50E-05**	**3,61E-02**
58	10	0	5,97E-08	2,93E-08	3,74E-02	**6,05E-08**	**2,46E-08**	**1,93E-02**
58	20	0	6,45E+00	2,81E+01	2,72E-02	**6,86E-08**	**2,54E-08**	**1,32E-02**
58	30	0	**6,24E-08**	2,48E-08	2,86E-02	7,09E-08	**1,98E-08**	**1,52E-02**
10	20	10	1,18E+01	5,15E+01	**4,49E-02**	**6,14E-01**	**1,23E+00**	5,17E-01
10	30	10	**8,41E-08**	**1,57E-08**	**4,09E-02**	2,68E-01	3,16E-01	4,57E-01
10	20	20	**8,05E-08**	**1,49E-08**	**3,61E-02**	4,66E+00	7,96E+00	2,33E-01
10	30	20	**8,32E-08**	**1,00E-08**	**3,48E-02**	3,57E+00	8,64E+00	4,76E-01

TABLE 2.5 – Comparaison des temps de convergence de DE et PSO sur les problèmes sur lesquels ils sont en compétitions

2.3.3 Ajustement automatique des données d'entrée

(Jaqaman and Danuser (2006)) proposent de modifier des données d'entrée en utilisant des méthodes de régression. Dans le même esprit, nous partons du postulat que le calage est correct et nous en servons pour progressivement modifier les données d'entrée. Une progression lente est nécessaire afin d'éviter des biais comme la divergence ou les comportements chaotiques comme cela peut être le cas dans les descentes de gradient. Il n'est pas pertinent de considérer les résultats du premier calage comme parfaits si on considère que les données d'entrée ne le sont pas. Ainsi, une trop grande modification dans le sens de l'optimisation pourrait biaiser les résultats dans ce sens et ne pas permettre de trouver une solution plus adéquate située entre les données réelles et le premier meilleur résultat trouvé.

Algorithme 3 Modification des données en fonction des résultats de l'optimisation

Entrée: $variationMax \geq 0$
tant que Critère d'arrêt, 200 itérations pour nous **faire**
Générer 1000 solutions au calage
Filtrer ces solutions selon leur cohérence (critère d'évaluation ou filtre selon la densité de l'espace des solutions)
Parmi les solutions cohérentes, identifier quelle année, A, simulée est la plus loin des données
$captures(A) = captures(A) + random(0, variationMax) \times (MoyenneCapturesSimulées(A) - captures(A))$
fin tant que

Pour tester cette approche, nous sommes repartis de données simulées, et avons volontairement ajouté des erreurs. Dans un cas sans contrainte autre que les intervalles de définition des paramètres, les impossibilités de calage viennent soit d'une quantité de captures provoquant une chute à 0 de la biomasse, soit

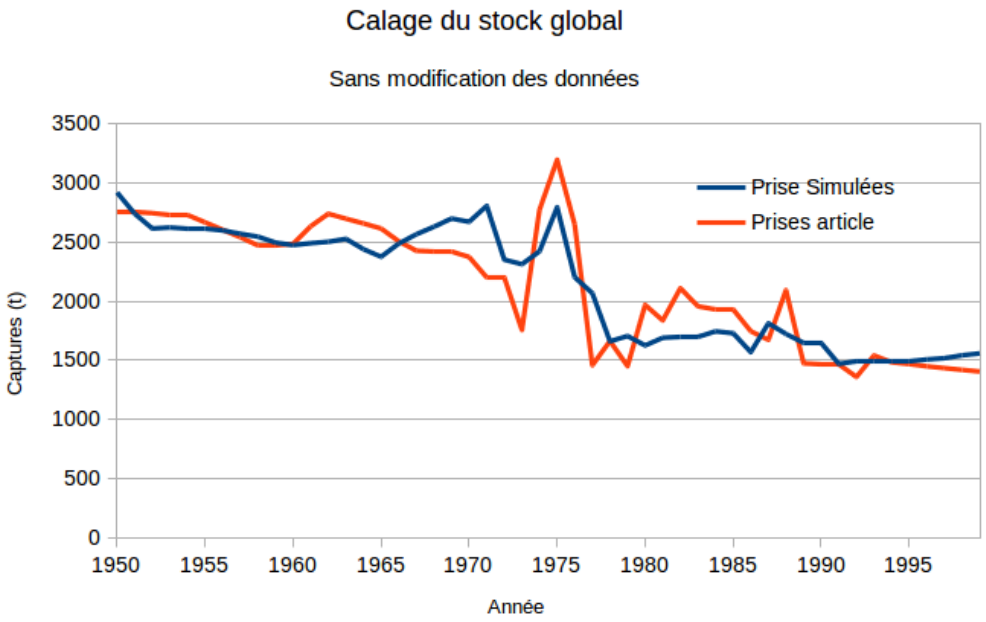


FIGURE 2.7 – Résultat calage par rapport aux prises globales

d'une impossibilité d'obtenir les captures à partir de l'ensemble de définition de l'effort à au moins un instant donné. Des pics imprévus de valeur de captures trop faible ou trop élevée devraient donc mener à une impossibilité de calage.

La figure 2.8 (a) montre le résultat du calage sur ces données. On y constate également des erreurs de calage. En revanche, sur la figure 2.8 (b), représentant les différences entre les données avant modification et les données modifiées auxquelles nous avons appliqué 200 itérations de l'algorithme 3, on peut voir que nous sommes en mesure de retrouver presque parfaitement les données d'origine et donc un calage optimal.

Ainsi, après 200 itérations sur les données de prises globales, nous obtenons les résultats visibles en figure 2.9. Bien que nous ne pouvons pas prouver qu'elles correspondent mieux à la réalité, elles semblent toutefois mieux correspondre au modèle. Par exemple, le pic de 1975 a été atténué et, bien que toujours présent, celui-ci est plus facilement justifiable d'un point de vue biologique.

2.3.4 Apport des connaissances expertes

À partir de ces données, on ne peut déterminer de mesure d'efforts fiable que pour le stock global. Or, il est insuffisant de ne se baser que sur cela pour proposer des méthodes d'aide à la pêche en Corse. Nous allons donc chercher à calibrer les dynamiques de population de chacune des espèces indépendemment.

À l'échelle d'une seule espèce, il est difficilement justifiable d'utiliser une mesure de l'effort de pêche uniquement basée sur le nombre de bateaux. En effet, certaines espèces sont plus intéressantes commer-

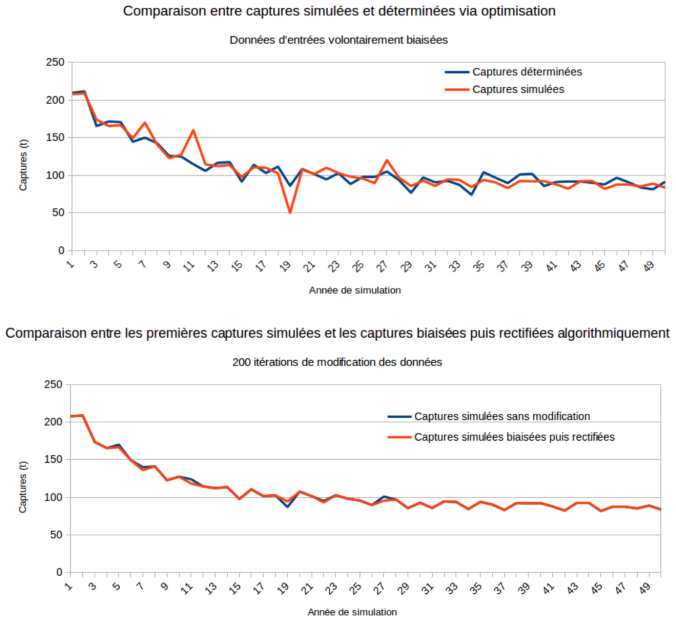


FIGURE 2.8 – Validation de l'algorithme de modification de données

cialement que d'autres, plus facilement ciblables par un engin donné, etc. nous devons donc nous même déterminer l'effort de pêche dans notre optimisation.

Les données présentées en annexes de (Le Manacha et al. (2011)) montrent l'évolution des captures au cours du temps pour 5 espèces d'intérêt. Ces espèces font également partie de la liste des espèces d'intérêt du projet MoonFish. Ainsi, nous pouvons utiliser des estimations peu précises de chaque paramètre afin de fortement limiter l'espace des solutions et ainsi, pallier l'accentuation du problème d'équifinalité engendré par la détermination via optimisation des efforts de pêche. Nous avons donc réutilisé les méthodes présentées en section 2.3.2 en ajoutant la liste d'efforts de pêche à la liste de paramètres à caler et en bornant les paramètres selon les connaissances expertes à notre disposition. On sait par exemple que sur certaines espèces, l'effort ne varie pas de plus de 20% entre les années. Nous obtenons donc l'ensemble des dynamiques de population possible de chacune de ces espèces.

Par la suite, cette section va se révéler très théorique, elle n'est là que pour montrer ce que l'on pourrait faire avec un minimum de connaissances expertes (connaissances que nous n'avons pas encore à l'heure actuelle). Nous ferons donc des suppositions afin de pouvoir montrer l'intérêt théorique de cette méthode. Nous ne sommes cependant pas en mesure de l'appliquer en situation réelle. Nous allons donc supposer la connaissance de bornes quant à la biomasse de chacune des espèces d'intérêt. Connaissant la difficulté d'une telle estimation, ces bornes sont relativement larges par soucis de crédibilité avec le système réel. On a donc $borneSup = 4 * borneInf$.

Pour avoir une représentation relativement fine de l'activité de pêche en Corse, nous ne pouvons pas nous baser sur seulement 5 espèces. Ainsi, aux 5 espèces calibrées précédemment, nous ajoutons un total

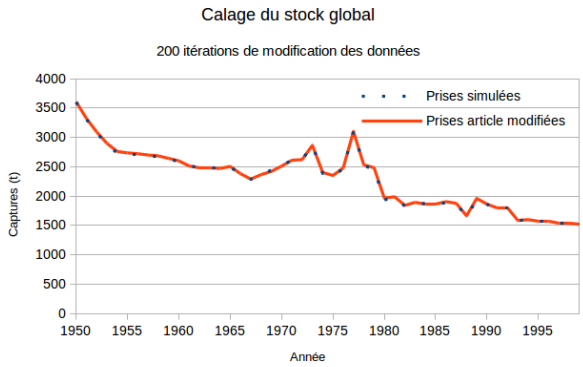


FIGURE 2.9 – Résultat calage par rapport aux prises globales après modification

Paramètre	Minimum	Maximum
k	1	2000
B1950	x	4*x
r	0.001	0.5
q	0.001	0.5
Effort[58]	0	5

TABLE 2.6 – Bornes des paramètres de chacune des espèces d'intérêt

de 11 espèces théoriques, 10 serviront à représenter chacune une espèces d'intérêt commercial spécifique et une dernière représentera l'ensemble des autres espèces modélisées.

Le tableau 2.6 présente les bornes utilisées pour les paramètres de chacune des 11 espèces théoriques qui ne sont pas encore calées. Pour la dernière, nous autorisons des bornes bien plus élevées pour k et B1950, c'est-à-dire la biomasse à l'instant où commencent les simulations permettant les évaluations des calibrations, car ces paramètres représentent un groupement d'espèces et non une espèce unique. Enfin, nous réutilisons les dynamiques de populations des 5 espèces précédemment calées telles quelles.

Ne connaissant que l'évolution de la biomasse globale et des captures globales au cours du temps, la fonction d'évaluation va naturellement chercher à représenter ces deux aspects. Ayant deux objectifs, il semble naturel d'utiliser un algorithme d'optimisation multiobjectif. Or, comme ces données sont censées représenter la réalité, nous savons qu'un optimum de Pareto existe. Il n'est donc pas nécessaire de chercher le front de Pareto, représentant l'ensemble des compromis possibles entre les objectifs. Un algorithme d'optimisation globale avec une fonction d'évaluation scalarisant les deux objectifs est donc acceptable. Nous proposons la fonction d'évaluation suivante :

$$f(x) = \sum_{i=0}^{nbAnnee} \left[B_{global}(i) - \sum_{j=0}^{nbEspece} B_j(i) \right]^2 + \sum_{i=0}^{nbAnnee} \left[C_{global}(i) - \sum_{j=0}^{nbEspece} C_j(i) \right]^4 \tag{2.23}$$

Parmi les algorithmes que nous avons testés sur ce problème, la Recherche à Voisinage Variable

Générale (Siarry (2014)) est la plus performante. Bien qu'il y ait beaucoup de paramètres, l'évaluation n'est pas gourmande en temps de calcul. Cette méthode peut donc être utilisée même si elle nécessite un très grand nombre d'évaluations. Elle n'est cependant pas recommandée dans le cas d'un modèle plus complexe pouvant demander un grand temps de calcul par évaluation.

Nous obtenons ainsi les résultats présentés en figure 2.10 et 2.11. On peut y voir que les captures cumulées suivent parfaitement celles présentées en figure 2.9.

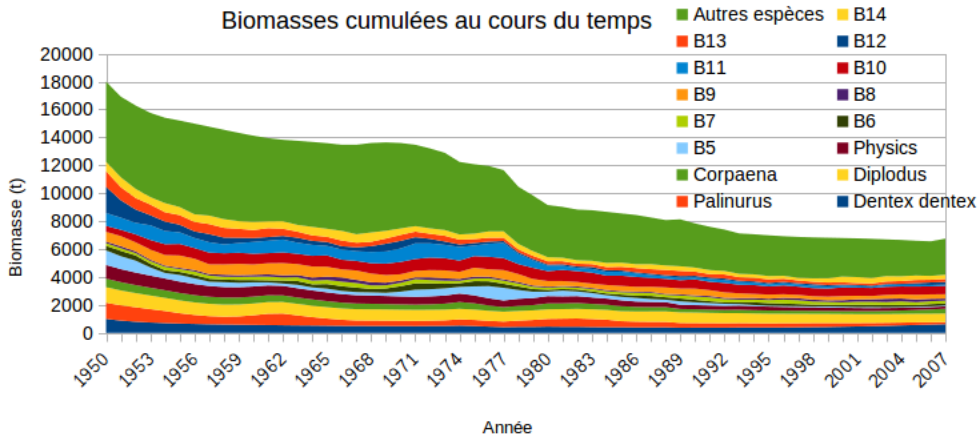


FIGURE 2.10 – Biomasse cumulée de 15 espèces d'intérêt au cours du temps

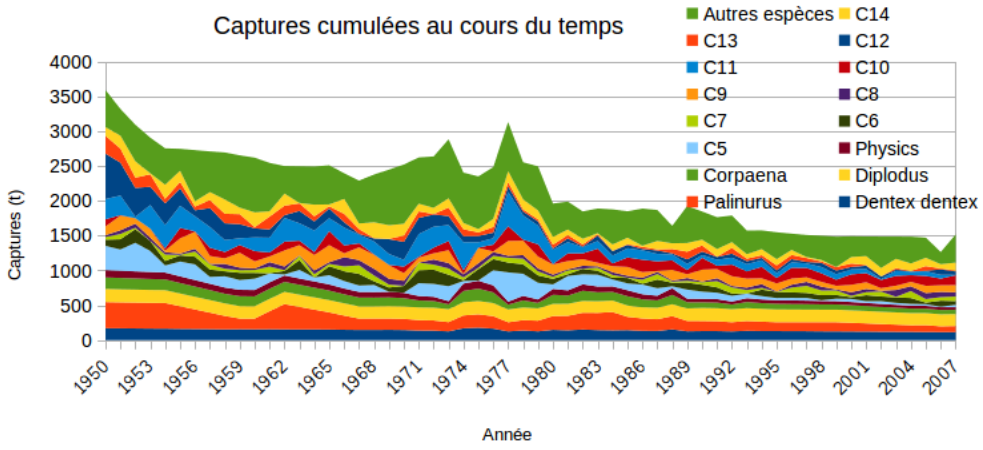


FIGURE 2.11 – Captures cumulées de 15 espèces d'intérêt au cours du temps

Ainsi, l'utilisation de la puissance des méthodes d'optimisations alliée à des connaissances expertes peut permettre l'émergence de connaissances difficiles à déterminer telle que la biomasse de chacune des espèces. Cela permettant, à terme de représenter fidèlement l'évolution des stocks et de l'exploitation marine.

L'utilisation de bornes pour les relevés de captures et d'efforts et la considération de validité de la solution proposée si chaque paramètre les respecte, supposent que chaque valeur des intervalles a la même cohérence et que celle-ci s'arrête complètement à la frontière. Cette hypothèse semble peu réaliste. Pour pallier ce problème, nous allons à présent proposer des méthodes de calibration, détection et correction des données basées sur une approche probabiliste.

2.4 Approche probabiliste

2.4.1 Description formelle de l'approche

Nous partons de deux séries temporelles notées $C(t)$ et $E(t)$ tel que $\forall t, C(t) \sim P_{C_t}(x), E(t) \sim P_{E_t}(x), \forall x \in R^+$ avec P une loi de probabilité quelconque connue, potentiellement différente pour chaque donnée. Tout P est discrétisé en intervalles de valeurs pour lesquels nous calculons la probabilité de tirer une valeur dans cet intervalle. La largeur nécessaire pour chacun de ces intervalles peut être déterminée via une analyse de sensibilité. De plus, via simulation, pour une série E connue, nous pouvons calculer $C(t)$ correspondant $\forall t$ (voir algorithme 4 ligne 8). Ainsi, nous pouvons calculer la probabilité $f(C, E)$ (ligne 10 et 15) qu'un ensemble de valeurs (C, E) corresponde à la réalité :

$$f(C, E) = \prod_t min(P_{C_t}(C(t)), P_{E_t}(E(t))) \tag{2.24}$$

En simulant tous les scénarios possibles pour la série E (ligne 5 à 13), nous pouvons calculer la probabilité que la solution proposée soit valable peu importe la série temporelle réelle (ligne 15) :

$$F(q, r, k, B0) = \sum_{i=0}^{nSim} f(C_i, E_i) \tag{2.25}$$

Ce type d'évaluation est une variante des estimations par maximum de vraisemblance Aldrich (1997), sur lequel nous reviendrons en section 4.4.4. Il peut mener à des valeurs de probabilités trop faibles pour être calculées de manière fiable à cause des problèmes d'arithmétique flottante Higham (2002). Pour éviter ces problèmes, il est possible d'utiliser une fonction basée sur la somme des logarithmes des probabilités plutôt que sur leur produit. On obtient alors :

$$f(C, E) = -\frac{1}{t} \sum_t log(min(P_{C_t}(C(t)), P_{E_t}(E(t)))) \tag{2.26}$$

De plus, si les deux séries temporelles sont cohérentes entre elles, simuler l'ensemble des scénarios possibles selon $E(t) \sim P_{E_t}(x)$ doit nous permettre de retrouver P_{C_t} dans le cas d'une solution parfaite. Sinon, $\forall x \in P_{C_t}$, $\sum P_{C_t} calculee(x) = F \Rightarrow \frac{P_{C_t} calculee(x)}{F} = P_{C_t} estimee(x)$. Des différences significatives entre $P_{C_t} estimee$ et $\frac{P_{C_t} calculee}{F}$ seront alors le signe d'incohérences entre les séries temporelles et donc d'erreurs d'estimation sur l'année t.

Cette approche est hautement parallélisable sur GPU ce qui nous permet de facilement pouvoir

explorer l'ensemble des scénarios possibles. Dans le cas où il est impossible de simuler tous les scénarios, un ensemble significatif pourra être utilisé efficacement. Il est de plus toujours possible de se baser sur des simulations Monte Carlo. Notre but va donc être de trouver $(q, r, k, B0)$ maximisant F, obtenant ainsi la solution la plus cohérente par rapport à nos données. Cette approche a été publiée dans Poiron-guidoni et al. (2020). En sortie de chaque simulation, nous obtenons donc un ensemble de valeurs de biomasses finales probabilisées pouvant servir d'entrée pour un futur processus d'optimisation robuste. Cela permet de connaître l'évolution du stock dans le meilleur et pire cas mais également dans l'ensemble des cas les plus probables.

Pour valider notre approche, les deux prochaines sections porteront d'abord sur une étude de cas théorique arbitrairement défini, cohérent avec la réalité.

Algorithme 4 Calcul de l'évaluation probabiliste

Entrée: $P_{C_t} P_{E_t} \forall t, (q, r, k, B0)$, n le nombre d'année de simulation, $nsimu$ le nombre total de simulations, $E[nsimu][n]$ l'effort à appliquer pour chaque année, $Bfinal[nsimu]$ ensemble pour enregistrer la biomasse finale de chaque simulation.

Chaque thread GPU :
pour $j = threadIndex; j < nsimulation; j+ = threadStride$ **faire**
  B=B0
  proba[j]=1
  pour $i = 0; i < n; i++$ **faire**
    $c = q * B * E[j][i]$
    $B = B + r * (1 - B/k) * B - c$
    $proba[j] = proba[j] * min(P_{C_i}(c), P_{E_i}(E[j][i]))$
  fin pour
  Bfinal[j]=B
fin pour
Synchronisation des threads
$fitness = \sum_{i=0}^{nsimu} proba[i]$

2.4.2 Application sur données cohérentes

Pour tester notre approche, nous avons décidé de partir de données théoriques que nous avons générées en simulant l'évolution d'une espèce aléatoire. Ainsi, un premier test sur des estimations parfaites permettra de vérifier la cohérence de l'approche.

Pour cela, nous partons d'un scénario arbitraire qui pourrait arriver en pratique :

— Nous avons 10 années, d'estimation des efforts et des captures

— 6 d'entre elles, yr_i sont très fiables

— les 4 autres, ye_i sont des estimations qui suivent une loi normale, centrée sur la valeur réelle et dont nous avons fait varier σ au cours de différents essais.

— les estimations d'efforts et de captures sont cohérentes entre elles.

Pour les années fiables, on considère que $E(t)$ est parfaitement juste et que l'estimation de $C(t)$ ne diffère pas de plus de 20% de la réalité.

σ_C	σ_E	Fmoyen	σ_C	σ_E	Foyen
C(t)/5	E(t)/5	0.874579	C(t)/15	E(t)/5	0.015499
C(t)/5	E(t)/10	0.965582	C(t)/15	E(t)/10	0.208958
C(t)/10	E(t)/5	0.075843	C(t)/15	E(t)/15	0.893316
C(t)/10	E(t)/10	0.857864	C(t)/15	E(t)/20	0.979369
C(t)/10	E(t)/15	0.991102	C(t)/15	E(t)/25	0.996413

TABLE 2.7 – Résultats en fonction de la variabilité de P_{C_t} et P_{E_t}

Comme précédemment, il est possible que les paramètres se compensent et donc qu'il y ait plusieurs solutions. Nous avons donc fait le choix d'un algorithme d'optimisation multimodal nous permettant ainsi de proposer plusieurs solutions pour expertise finale et validation.

Nous réutilisons l'algorithme Dual Strategy Differential Evolution (DSDE) présenté dans Wang et al. (2017). Une solution est uniquement composée des paramètres $(q, r, k, B0)$ du modèle de Graham-Schaefer et est évaluée en suivant l'algorithme 4.

Le tableau 2.7 montre les résultats obtenus en fonction de la variabilité des lois de probabilité. On remarque que plus les estimations d'efforts sont précises (σ_E faible), plus les résultats sont bons. Par contre, une précision plus importante sur C que sur E donne des évaluations relativement mauvaises. Ce résultat peut cependant facilement s'expliquer. En effet, comme nous explorons en fonctions de E, si celui-ci couvre une large plage de valeurs alors que C en couvre une petite, un grand nombre de scénarios aura des valeurs de C non cohérentes et donc $f(C, E) = 0$. La solution proposée n'est cependant pas forcément à rejeter et cela pourra permettre d'identifier les années de données nécessitant une réestimation, voire de les corriger (section 2.4.3).

La figure 2.12 montre la répartition des solutions finales dans l'espace composé des paramètres (q, r, B), le paramètre k a été retiré pour permettre une représentation graphique. Nous obtenons un ensemble de solutions de fitness quasi-équivalentes. Leur nombre relativement restreint et leur dispersion peut facilement permettre à un expert d'analyser les résultats et de nous aiguiller vers la solution la plus cohérente biologiquement.

De plus, la figure 2.13 représente l'évolution de la biomasse au cours du temps pour l'ensemble des scénarios simulés cohérents d'une des solutions proposées. La biomasse finale varie d'environ 10% entre les différents scénarios. Cette valeur peut différer en fonction des données et de la variabilité des estimations, mais elle est assez représentative de ceux-ci. De plus, pour chacune de ces valeurs, nous connaissons la probabilité qu'elle soit juste. Nous pouvons donc facilement nous en servir dans une future optimisation robuste visant à améliorer les stratégies d'exploitation.

Enfin, la figure 2.14 montre les lois de probabilité P_{C_t} estimées (en rouge) et obtenues via simulation en bleu pour les 4 années non fiables (respectivement t=2, 3, 5, 8). Comme nous nous y attendions pour le cas d'estimations parfaitement cohérentes entre elles, nous retrouvons quasi parfaitement les lois estimées.

Nous allons maintenant nous baser sur ces premiers résultats pour repérer des erreurs de cohérence dans les données et les corriger.

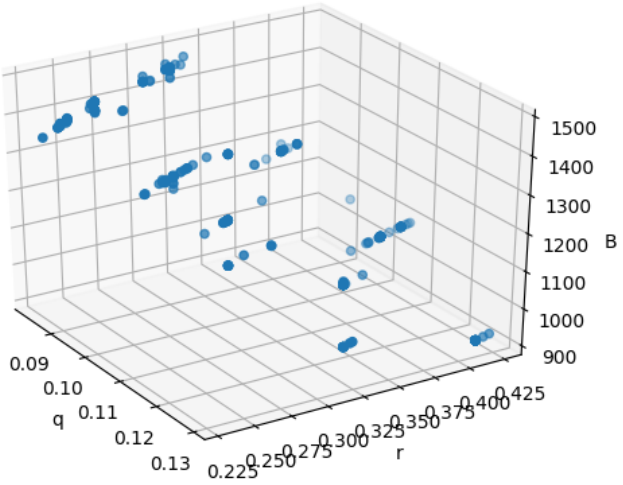


FIGURE 2.12 – Solution de la calibration avec une parfaite connaissance des lois de probabilité

2.4.3 Identification et correction de données incohérentes

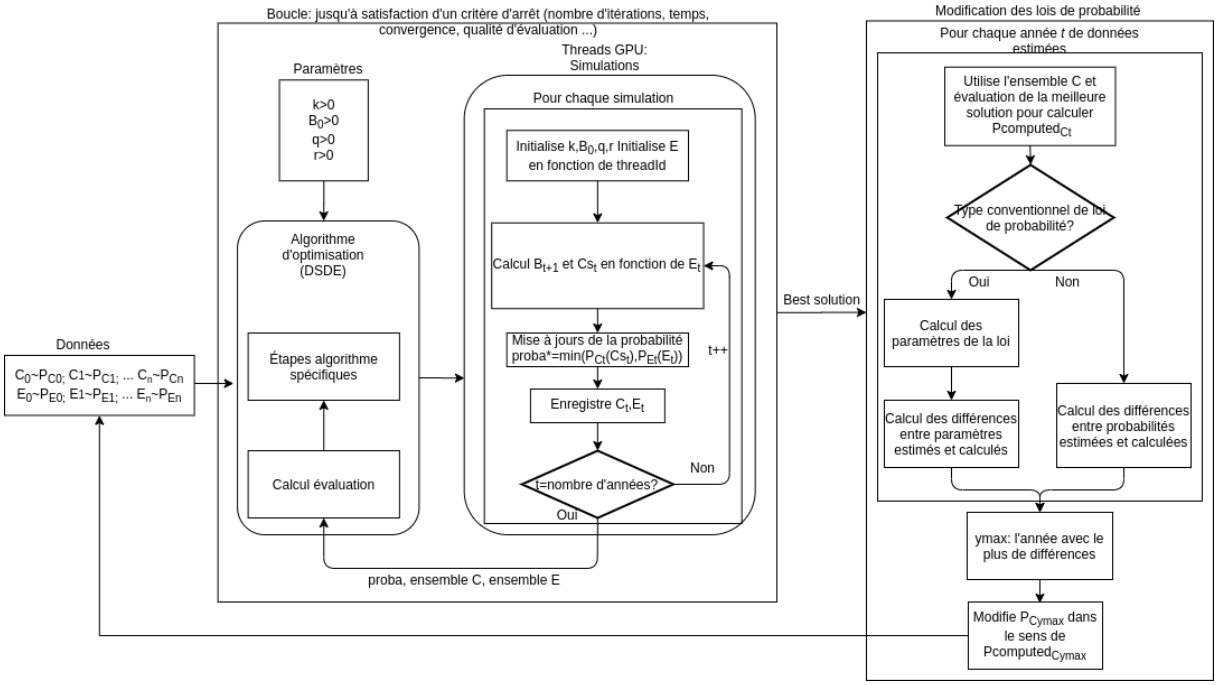


FIGURE 2.15 – Diagramme complet du calage par approche probabiliste.

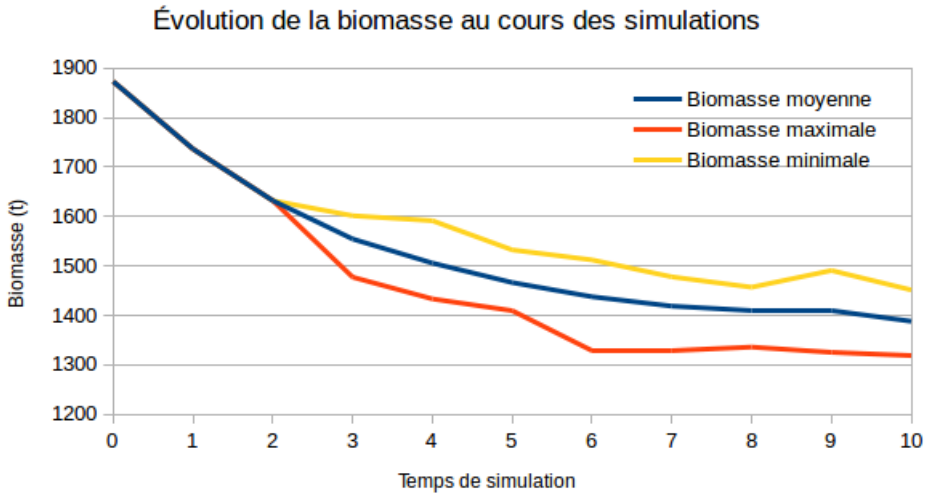


FIGURE 2.13 – Évolution des biomasses moyenne, maximale et minimale au cours du temps

Nous repartons du scénario précédent sur lequel nous avons apporté les modifications suivantes :

— parmi ye, la première estimation de capture est centrée sur la bonne valeur, les 3 autres sont surestimées. L'utilisateur ne le sait pas.

— les estimations d'efforts étant plus facilement faisables, elles seront plus fiables que les estimations de captures.

Les surestimations varient entre $\sigma(t)/4$ et $2 * \sigma(t)$ permettant ainsi de couvrir une plage de variabilité de très faible à très élevée. Étonnement, la variabilité de la surestimation n'impacte que très peu les résultats sur cette plage. Passée cette limite supérieure, les probabilités estimées deviennent tellement faibles que la réussite de l'approche proposée devient très aléatoire.

Le but ici va être de modifier la loi de probabilité estimée P_{C_t} la plus éloignée de la loi de probabilité calculée $Pcomputed_{C_t}$ afin de se rapprocher le plus possible d'une solution cohérente. Ici, nous allons laisser l'algorithme 5 se dérouler de manière automatique. En situation réelle, il pourra évidemment être utilisé pour repérer les erreurs potentielles et laisser un expert mettre à jour les lois de probabilités lui-même.

On commence (ligne 1) par calculer l'ensemble des solutions en suivant la méthode définie précédemment. Pour chaque année de simulation, on met à jour $Pcomputed_{C_t}$ suivant l'équation 2.25 (ligne 2 à 6). Par la suite, si les lois de probabilité le permettent, nous calculons les paramètres correspondant à $Pcomputed_{C_t}$. Un calcul de distance relative entre les paramètres estimés et calculés permet ensuite d'identifier l'année $ymax$ pour laquelle l'estimation est la pire (ligne 7 à 11), signe d'une erreur d'estimation. Ces paramètres sont finalement modifiés dans le sens de ceux de $Pcomputed_{C_t}$ (ligne 12). À noter qu'une mauvaise estimation peut facilement impacter les lois de probabilités des autres années, il est donc important de n'en modifier qu'une par itération.

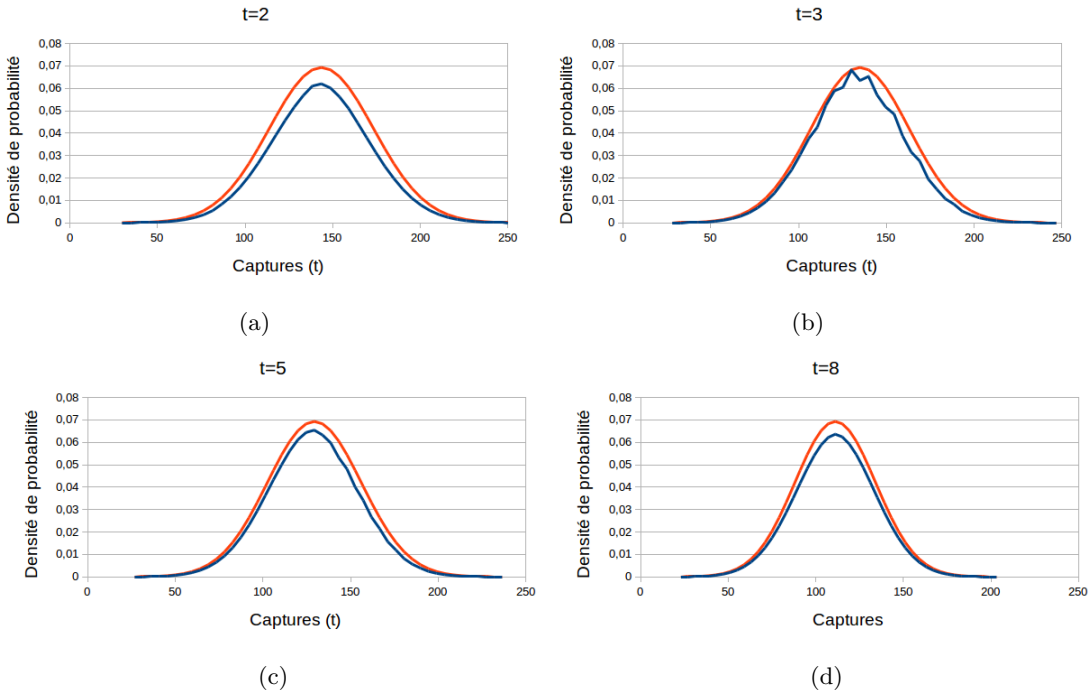


FIGURE 2.14 – Comparaison entre lois de probabilité estimées (rouge) et calculées (bleue)

Dans le cas où les lois de probabilité ne permettent pas un calcul rapide des paramètres la régissant, ces étapes peuvent être remplacées par un calcul de distance entre les différentes valeurs des intervalles de la discrétisation puis une mise à jour directe de ces valeurs.

La figure 2.15 montre un diagramme du système complet incluant les algorithmes 4 et 5.

La figure 2.16 montre les résultats de cet algorithme pour les différentes années d'estimation non fiables avec :

— En jaune, la loi de probabilité que nous avons utilisée dans l'application précédente et qu'il faudrait retrouver.

— En rouge, la loi de probabilité $Pcomputed_{C_t}$ à la première itération de l'algorithme

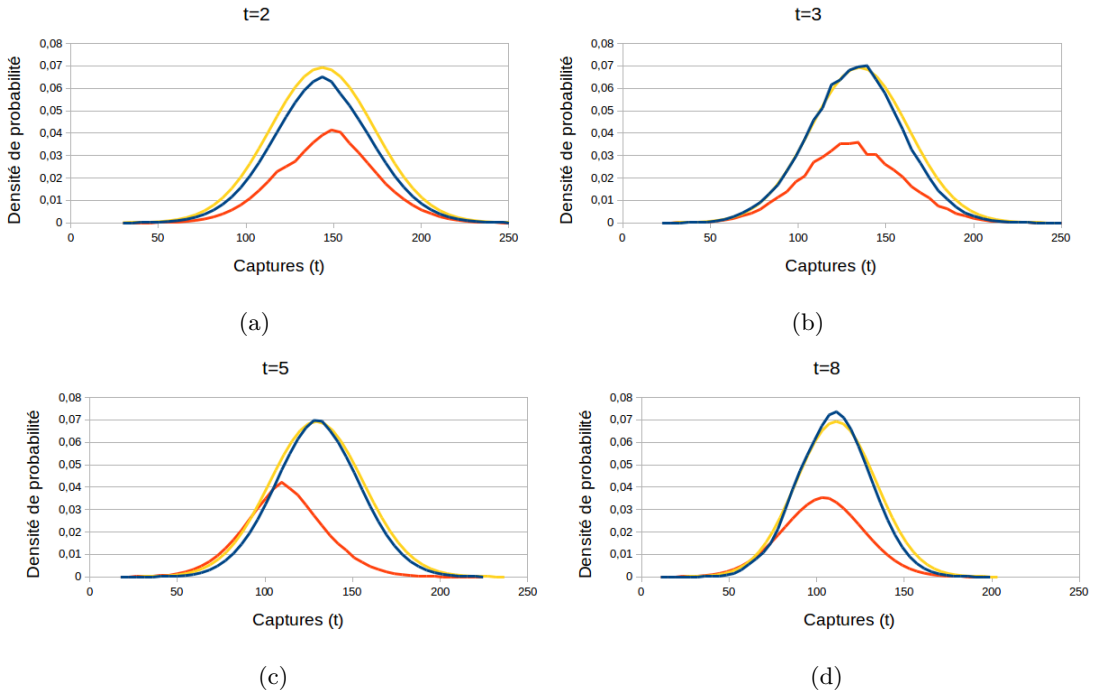
— En bleu, la loi de probabilité $Pcomputed_{C_t}$ après 50 itérations.

On remarque qu'au début, même pour $t = 2$, seule année pour laquelle nous avons conservé une estimation correcte, l'espérance de $Pcomputed_{C_t}$ initiale est légèrement décalée. Cette différence est d'autant plus grande pour les autres années. Après 50 itérations en revanche, la plupart des erreurs sont corrigées et nous retrouvons presque les lois de probabilités parfaites.

Algorithme 5 Amélioration des données incohérentes

Entrée: $P_{C_t} P_{E_t} \forall t$

Utilise DSDE avec algorithme 4 et enregistre $fitness$, $proba$ et l'ensemble C de la meilleure solution
pour $j = 0; j < nsimulation; j++$ **faire**
  pour $i = 0; i < n; i++$ **faire**
    $Pcomputed_{C_i}(C_j(i)) += proba[j]/fitness$
  fin pour
fin pour
pour $i = 0; i < n; i++$ **faire**
  Calcul les paramètres de la loi de probabilité $Pcomputed_{C_i}$ (μ, σ pour nos lois normales)
  Calcul la distance relative entre paramètres estimés et calculés
  Enregistre $ymax$ l'année avec le plus de différences
fin pour
Modifie les paramètres de $P_{C_{ymax}}$ dans le sens de $Pcomputed_{C_{ymax}}$
Boucle sur la première étape jusqu'à satisfaction d'un critère d'arrêt



(a) (b) (c) (d)

FIGURE 2.16 – Comparaison entre lois de probabilité calculées à $iteration = 0$ (rouge), calculées à $iteration = 50$ (bleue) et parfaites théoriques (jaune)

2.5 Conclusion du chapitre

Au cours de ce chapitre, nous avons présenté une étape primordiale pour toute étude via simulation, le calage du modèle. Dans notre cas, c'est un modèle de dynamique de population. Cela nous a permis, dans

un premier temps, de mettre en lumière la nature théorique du problème étudié et ses implications. Dans le cas de données d'entrée cohérentes, un ensemble continu de solutions permet de retrouver la dynamique observée, mais celles-ci peuvent avoir des réactions très différentes sur des simulations du futur. Cela implique une impossibilité d'utilisation directe des résultats du calage et la nécessité de trouver tout, ou partie significative de, l'ensemble des solutions. Cet espace étant souvent très étendu, l'utilisation de connaissances expertes est alors nécessaire pour permettre une utilisation optimale à l'avenir. Il faut donc pouvoir prendre en compte de nombreuses contraintes, pouvant être de toute nature. Les métaheuristiques nous ont donc semblé être le type de méthodes le plus approprié.

De plus, la difficulté d'observation du domaine d'étude, la biologie halieutique, peut mener à des données incohérentes entre elles et ne donnant pas de solutions. Ainsi, nous avons à la fois présenté des méthodes théoriques et directement applicables afin de permettre le calage d'un modèle de dynamique de population halieutique, adapté à la problématique de la pêche en Corse. Nous avons pu voir que, lorsque les connaissances quant à l'historique de l'exploitation sont parfaites, un calage efficace est possible et permet ainsi de connaître les caractéristiques des espèces. Dans ce cas, le problème d'équifinalité est très faible et gérable informatiquement. En revanche, la biologie marine est un domaine complexe dont les accès aux informations sont rares et sujets à caution. Des données parfaitement précises sont donc impossibles à obtenir. Il est donc nécessaire de faire appel à une expertise afin de prendre en compte la réalité biologique des solutions proposées et non pas uniquement leur qualité informatique.

Dans un deuxième temps, nous avons proposé des méthodes afin de réduire l'impact de la mauvaise qualité des données d'entrée. En effet, modifier algorithmiquement certaines données de captures identifiées comme non cohérentes permet de pallier, dans une certaine mesure, le problème de la qualité de ce type de données. Cependant, une imprécision dans les relevés d'efforts de pêche engendre un gros problème d'équifinalité gérable uniquement par expertise.

Enfin, nous avons proposé une approche théorique afin de caler les dynamiques de population d'un ensemble d'espèces en se servant du calage du stock global. Bien que cette approche top down ne soit encore que théorique, elle nous permet de montrer que plus l'apport de connaissances expertes sera important, plus nous serons à même de déduire de connaissances supplémentaires et ainsi, permettre aux deux domaines, informatique et biologie, d'être réciproquement profitable.

Le travail présenté ici n'est qu'une étape préliminaire. Il vise à être complété pour proposer des méthodes d'optimisation des stratégies de pêche afin de prévenir la destruction de certains stocks tout en permettant d'assurer la pérennité des exploitants. Cette problématique fait l'objet du chapitre suivant.

Chapitre 3

APPLICATIONS BASÉES SUR LES MÉTHODES D'OPTIMISATION

Pour rappel, l'optimisation est un processus itératif de recherche du meilleur ensemble de paramètres selon certains critères. Les objectifs, $f(x)$, donnent des valeurs numériques permettant de juger la qualité de la solution, x, proposée. Les solutions peuvent également devoir respecter des contraintes d'inégalité, $g(x)$, ou d'égalité, $h(x)$.

Dans un contexte d'étude de systèmes réels, l'optimisation se base souvent sur des simulations numériques. Elle peut alors avoir plusieurs utilités comme principalement :

— trouver le jeu de paramètres du modèle, permettant d'obtenir une simulation équivalente à une expérience, une observation réelle.

— partir d'un modèle déjà calibré et chercher à atteindre un objectif par simulations du futur.

Dans le chapitre précédent, nous nous sommes concentrés sur la phase de calage de modèle. Quand le système peut être modélisé de manière acceptable, l'optimisation s'applique sans trop de contraintes. Beyer and Sendhoff (2007) indiquent que des incertitudes peuvent être présentes sous plusieurs formes :

— des changements environnementaux ;

— des imprécisions dans les paramètres utilisés ;

— des incertitudes dans les sorties du modèle ;

— des incertitudes de faisabilité entre simulation et réalité.

Leur prise en compte demande alors l'utilisation de méthodes adaptées telles que l'optimisation robuste Beyer and Sendhoff (2007) ou l'optimisation robuste ajustable Tang and Wang (2015). En fonction des connaissances sur le système, ces incertitudes peuvent être gérées de plusieurs façons. Il est par exemple possible de se baser sur les probabilités d'occurrence des événements incertains ou utiliser les domaines de

définition des variables incertaines dans des méthodes déterministes. Une approche randomisée permet de directement incorporer les incertitudes au sein du problème d'optimisation. $f(x)$ devient alors une fonction aléatoire $\tilde{f}(x)$ calculable via simulation directe Fu (2002). L'utilisation de stratégies Monte-Carlo, d'optimisation stochastique Birge and Louveaux (2011), ou par méta-modèle est alors possible.

Dans le chapitre précédent, nous avons vu que le calage du modèle nous mène à une infinité de solutions au sein d'un espace généralement continu (pouvant ne pas l'être en fonction des contraintes et connaissances expertes) non convexe. Les optimisations que nous proposerons devront donc prendre en compte cet ensemble de solutions au sein de l'évaluation. On peut alors se poser la question de l'importance de chaque solution de calage au sein de cette évaluation.

Dans un premier temps, nous proposerons une première approche d'optimisation robuste et comparerons ces résultats aux estimations publiées et à une optimisation globale supposant des connaissances parfaites. Par la suite, nous montrerons que cette approche peut présenter un problème de pondération de l'importance des incertitudes à cause du manque d'homogénéité dans les solutions proposées par la phase de calibration. Nous proposerons alors une approche basée sur les diagrammes de Voronoï pour améliorer les connaissances quant à l'espace d'incertitudes. Cette méthode sera ensuite appliquée à l'optimisation robuste afin d'en améliorer les résultats. Nous proposerons enfin une dernière approche basée sur l'optimisation robuste ajustable afin d'aider à la résolution de la problématique posée par la calibration.

3.1 Optimisation robuste

Dans cette section, nous allons nous intéresser à l'utilisation de l'optimisation robuste pour proposer des stratégies de pêche basées sur l'effort de pêche optimal applicable, et donc la mise en place de quotas. Nous ne traiterons que des cas basés sur les données d'estimations passées afin de comparer les résultats à des données publiées Le Manacha et al. (2011), montrant ainsi l'intérêt de ces approches. Ces applications sont cependant directement applicables à des extrapolations pour le futur. En effet, les ensembles de solutions du problème de calibration peuvent être utilisé à partir de la dernière année de données pour extrapoler des préconisations pour la gestion de l'exploitation à venir.

3.1.1 Optimisation globale du passé

Cette approche consiste à partir d'une des solutions du calage en supposant qu'elle est correcte. Elle servira alors de point de comparaison optimale à ce que l'optimisation robuste peut proposer.

L'optimisation se fait sur l'ensemble des $E(t)$ afin de maximiser $f(x)$, c'est-à-dire les captures totales au cours du temps. Nous utilisons donc simplement la fonction d'évaluation suivante :

$$f(x) = \sum_{i=0}^{nbAnnees} C_i \tag{3.1}$$

Nous cherchons cependant également à obtenir de meilleurs résultats sur le plan écologique. Ainsi, nous ajoutons une contrainte permettant de s'assurer d'avoir plus de biomasse finale que dans la réalité :

$$g(x) : BFinale > BSimulee \tag{3.2}$$

Nous avons choisi cette valeur afin de pouvoir comparer nos captures simulées aux prises réelles dans des conditions de biomasse finale comparable. La valeur de bioamsse utilisée est celle déterminée lors de la phase de calibration.

Cette contrainte dépendant directement des valeurs d'effort de pêche, nous avons fait le choix de la gérer directement via une heuristique de satisfaction de contrainte spécifique plutôt que d'utiliser un algorithme spécifiquement adapté à la satisfaction de contrainte tel que le stochastic ranking Runarsson and Yao (2000). Ainsi, si cette contrainte n'est pas satisfaite, les efforts seront réduits jusqu'à obtenir une biomasse finale acceptable.

Pour réaliser cette optimisation, nous utilisons une Recherche à Voisinage Variable Générale (Siarry (2014)). Le tableau 3.3 présente les résultats sur 5 espèces d'intérêt. Nous pouvons voir que l'optimisation présente toujours de meilleurs résultats avec des cas où les captures simulées sont nettement supérieures aux captures réelles.

	Prises réelles	Prises optimisées
Dentex	8209	10972
Palinurus	11157	18934
Diplodus	10366	11774
Scorpaena	6535	8791
Physics	4184	4412

TABLE 3.1 – Comparaison entre prises réelles et simulées via optimisation

Afin de permettre une meilleure visualisation des résultats possibles, il est intéressant de montrer l'ensemble des compromis entre l'objectif économique, les captures, et l'objectif écologique, la biomasse finale. Pour cela, nous pouvons nous baser sur les méthodes d'optimisation multiobjectif.

Nous proposons donc deux objectifs à maximiser :

$$f_1(x) = \sum_{i=0}^{nbAnnees} C_i \tag{3.3}$$

$$f_2(x) = B_{finale} \tag{3.4}$$

L'utilisation de l'algorithme NSGA-II nous permet d'obtenir le front de Pareto visible en figure 3.1 pour le Dentex. On peut voir que sur une exploitation aussi longue, les résultats de captures totales varient assez peu en fonction de la biomasse finale. La mise en place d'une exploitation optimale durable les premières années permet alors une gestion efficace sur le long terme. On peut également voir une asymptote quasi-verticale sur la valeur de biomasse finale à 1200. Cette valeur correspond en réalité à la valeur de biomasse à l'équilibre du stock sans exploitation.

Il est cependant complètement irréaliste de penser que nous pourrons connaître avec exactitude la

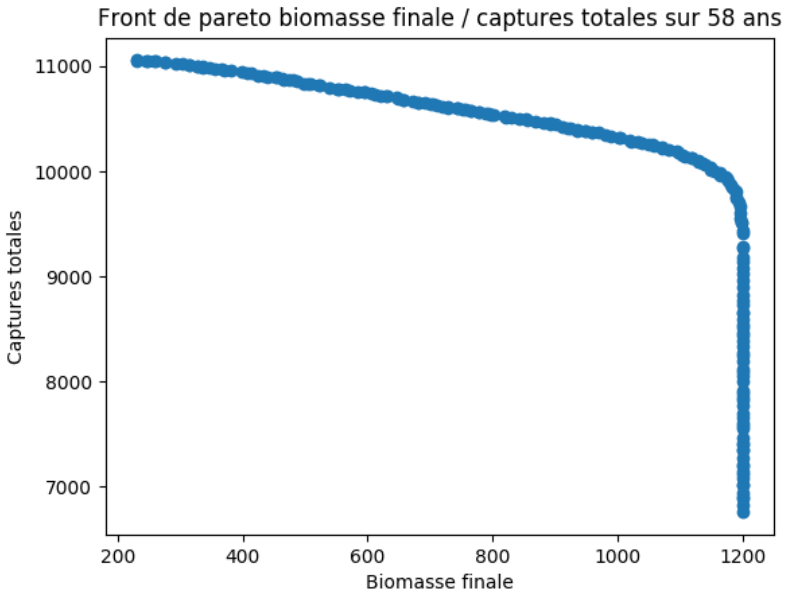


FIGURE 3.1 – Front de Pareto des captures totales en fonction de la biomasse finale, pour le Dentex

dynamique de population de chaque espèce. Ainsi, l'optimisation robuste va nous permettre de réaliser la même approche, mais sur des ensembles de dynamiques de populations possibles, c'est à dire, en prenant en compte l'espace d'incertitudes.

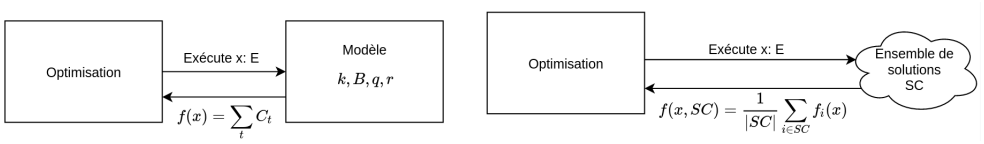
3.1.2 Première approche robuste

L'optimisation robuste va chercher à atteindre le même objectif que précédemment, mais pour toutes les dynamiques de population d'un ensemble de solutions de la phase de calage. Nous cherchons un même ensemble d'efforts de pêche à appliquer quelle que soit la dynamique de population appartenant à l'ensemble étudié. Nous devons donc adapter la fonction objectif. Une approche envisageable, classique en optimisation robuste, aurait pu être de chercher à maximiser la valeur minimum de $f(x)$ de l'ensemble, c'est-à-dire la maximisation du pire cas. Cela serait cependant simplement revenu à une optimisation classique sur la dynamique de population la plus contraignante de l'ensemble.

Ainsi, nous avons fait le choix de maximiser l'espérance de l'ensemble C via la fonction d'évaluation suivante :

$$\tilde{f}(x, SC) = \frac{1}{|SC|} \sum_{i \in SC} f_i(x) \tag{3.5}$$

Ce système est représenté par la figure 3.2. La figure 3.2a montre le cas précédent où on supposait des connaissances et un contrôle parfaits du système alors que la figure 3.2b montre le cas où on considère un ensemble de solutions possibles à la phase de calibration du modèle.



(a) Optimisation globale sans incertitudes (b) Optimisation robuste

FIGURE 3.2 – Optimisation globale et robuste

Dans cette première approche, on considérera que les solutions trouvées lors de la phase de calibration sont équiprobables et de mêmes importances. Nous reviendrons sur cette affirmation et ces implications en sections 3.2 et 3.3.

Bien sûr, nous devons également adapter la contrainte pour s'assurer de ne pas détruire le stock peu importe la vraie dynamique de population :

$$g(C) : \forall x \in C, BFinale_x > BReelle \tag{3.6}$$

En réutilisant la RVVG (section 2.3.2 et annexe 6.4.7), nous obtenons de très bons résultats. Ceux-ci ne semblent cependant pas significativement impactés par la méthode et la paramétrisation utilisées. La figure 3.3 (a) montre un exemple d'évolution du niveau de stock pour chacune des dynamiques de population du cluster. Bien sûr, des différences sont notables entre chacune, mais cela nous donne une bonne vision de l'état possible des stocks au cours du temps et la limite de stock minimum est respectée pour chaque scénario possible. La ligne rouge représente la limite de stock à ne pas dépasser selon $g(C)$. La chute de la biomasse en fin de simulation est due à un biais provoqué par la contrainte. En effet, une certaine quantité de biomasse est demandée en fin de simulation donc la solution optimale est de prendre plus tant qu'on ne dépasse pas la biomasse limite. Cette valeur est cependant facilement adaptable au niveau de stock demandé par les décideurs. Ce biais est laissé volontairement dans le but de pouvoir comparer les résultats d'optimisation avec les résultats réels dans des conditions équivalentes.

La figure 3.3 (b) elle, montre l'évolution des niveaux de stock de chaque élément du cluster, en effectuant une optimisation non robuste pour chacun. On constate que l'évolution est beaucoup moins chaotique, mais les résultats et conclusions ne semblent pas significativement différents.

La qualité des résultats est cependant très fortement impactée par le taux d'incertitude présent en entrée, c'est-à-dire dans notre cas, l'étendue de l'espace des solutions Poiron-Guidoni et al. (2018) Poiron-Guidoni et al. (2020b).

3.1.3 Analyse et amélioration de la variance

La figure 3.4 montre, pour un ensemble de clusters, l'écart moyen de ses composants par rapport au barycentre (axe x), et la différence moyenne entre les résultats des optimisations robustes et globales avec hypothèse de calage parfait. Dans chacun des cas, on considère que le cluster de solutions est centré sur la solution ayant servi pour l'optimisation globale. On peut constater que ces deux grandeurs sont presque linéaires. Il est donc important de faire des clusters les plus précis possibles, notamment via intervention

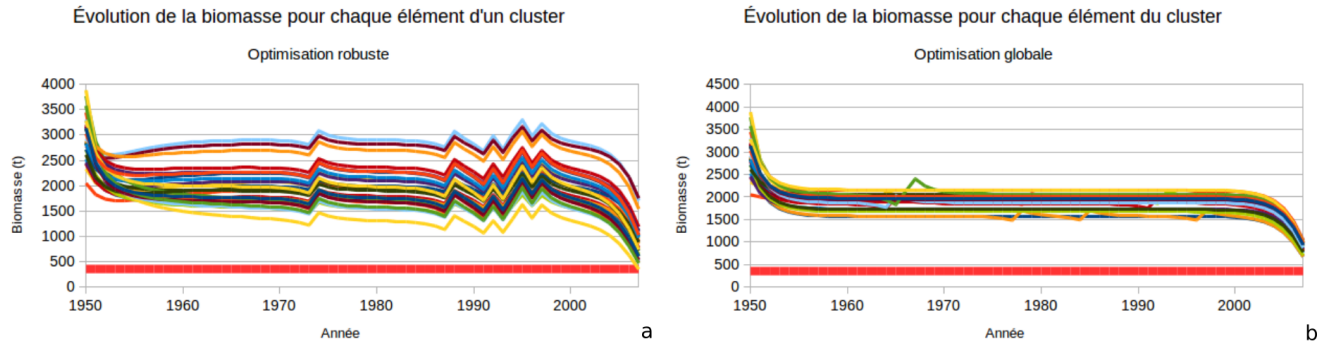


FIGURE 3.3 – Évolution de la biomasse pour chaque élément du cluster en utilisant l'optimisation robuste

humaine et utilisation de connaissances expertes.

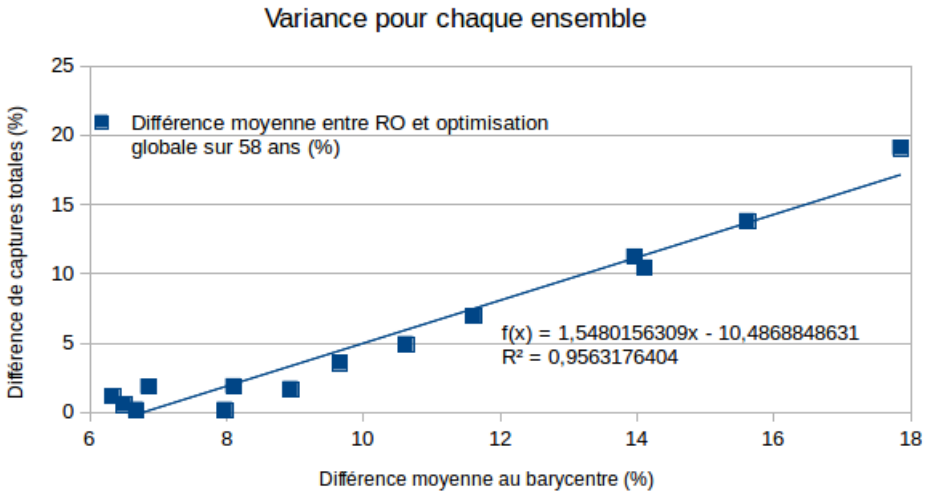


FIGURE 3.4 – Écart moyen au barycentre et différence entre optimisation robuste et globale pour chaque cluster.

Les résultats visibles en figure 3.3 montrent cependant que les différences entre les différentes solutions du cluster peuvent mener à des biomasses finales très variables. Ainsi, il peut être intéressant de chercher à réduire cet écart afin de pouvoir survenir plus facilement aux imprévus potentiels.

Pour réduire la variabilité des résultats au sein des clusters, nous allons introduire un deuxième objectif : minimiser le coefficient de variation, aussi appelé écart-type relatif Everitt (1998), $C_v = \frac{\sigma}{\mu}$. Ce qui donne dans notre cas :

$$f_2(C) = \frac{\sqrt{\frac{1}{n}\sum_{i=0}^{n}(f(x_i) - \tilde{f}(c))^2}}{\tilde{f}(c)} \tag{3.7}$$

L'optimisation multiobjectif demande cependant l'utilisation d'algorithmes adaptés. Ainsi, nous nous sommes orientés vers une variante d'algorithme génétique ayant fait ses preuves : NSGA-II Deb et al. (2002).

La figure 3.5 montre le front de pareto obtenu sur un cluster de test. Celui-ci représente l'ensemble des compromis optimaux entre les différents objectifs. L'axe des abscisses représente la valeur relative de $\tilde{f}_1(C)$, c'est-à-dire $\frac{\tilde{f}_1(C)_x}{max(\tilde{f}_1(C)_{PF})}$ pour tout point x appartenant au front de pareto PF. L'axe des ordonnées représente la valeur de $f_2(C)$. Nous sommes ainsi capables de proposer un ensemble de solutions dont le coefficient de variation peut descendre jusqu'à 8%. Celles-ci présentent cependant des prises inférieures de 30% par rapport à la meilleure solution sur $\tilde{f}_1(C)$. L'intérêt de cette méthode est alors de présenter l'ensemble des compromis possibles entre ces objectifs afin de permettre aux décideurs de choisir ce qui est le plus important pour eux.

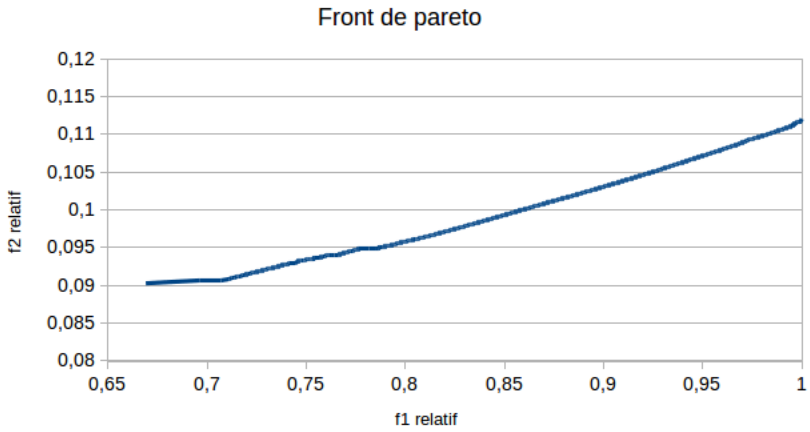


FIGURE 3.5 – Front de pareto formé par $\tilde{f}_1(C)$ and f_2

Pour l'ensemble des approches présentées ici, nous avons supposé que chaque solution de la phase de calibration avait la même importance. Cette supposition n'est cependant pas valable dans les cas où il n'a pas été possible d'effectuer une calibration avec une méthode garantissant une répartition homogène des solutions dans l'espace.

3.2 Proposition d'approche par caractérisation de l'espace des solutions au calage

3.2.1 Problème d'homogénéité

En considérant chaque solution du calage comme de même importance, on considère que celles-ci sont représentatives de l'ensemble des solutions possibles au calage et couvrent cet espace de façon uniforme. Or, quand il n'est pas possible de résoudre le problème de calage par discrétisation de l'espace des

paramètres, il peut arriver que des biais de calcul surviennent. En effet, les méthodes d'optimisation peuvent avoir tendance à être orientées dans un sens plutôt qu'un autre. Cela peut venir d'un problème de la méthode elle-même ou alors simplement d'une certaine topologie de l'espace des solutions, rendant plus facile la découverte de solutions dans une zone plutôt qu'une autre. De plus, même dans ce cas, l'espace des solutions peut présenter des plateaux, mais également des pics ou des creux, topologie qui ne doit donc pas être traitée de la même manière.

Pour illustrer cela, nous avons représenté 2000 solutions au calage sur des données complètement simulées Poiron-guidoni et al. (2020), donc pour lesquelles nous connaissons le résultat à obtenir lors de la phase de calage. Pour chacune, nous utilisons les mêmes données de captures et efforts, puis nous réalisons une estimation de densité par noyau en suivant la règle empirique de Silverman Silverman (2018).

Sur la figure 3.6, chaque point représente une solution parfaite au problème du calage. La couleur représente la densité des solutions présentes dans l'espace. Plus celle-ci est foncée, plus la zone est dense. À l'inverse, plus la couleur se rapproche du jaune, plus la densité est faible. Nous avons volontairement retiré un paramètre afin de pouvoir réaliser une représentation graphique.

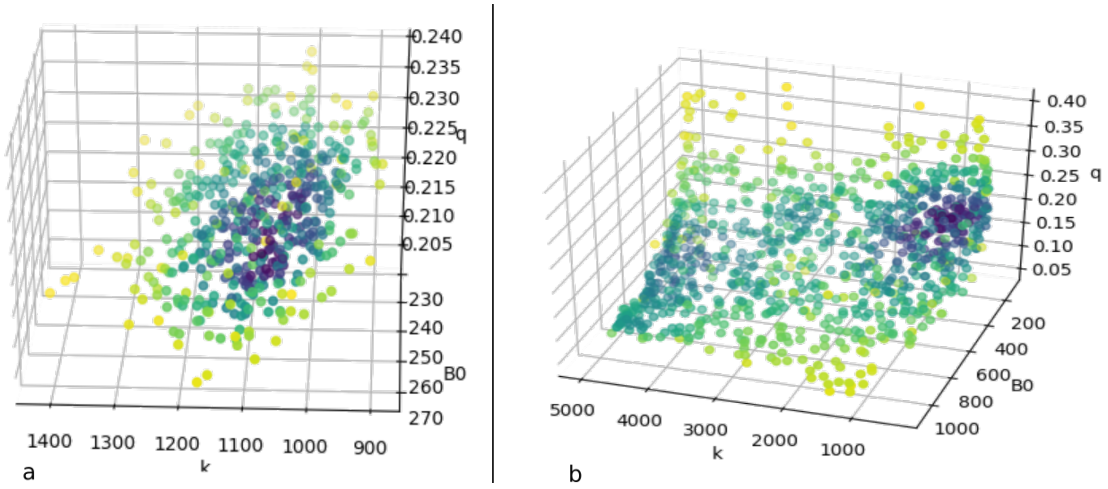


FIGURE 3.6 – Estimation de densité par noyau sur 2 tests. (a) très bonne connaissance des captures et de l'effort de pêche. (b) large intervalle de valeur pour les données d'effort de pêche

La solution à adopter est alors discutable. Faut-il conserver une pondération de l'importance des solutions équitables entre elles et considérer qu'une zone de moindre densité représente forcément une zone moins importante, car moins de solutions y sont présentes ? Ou faut-il adapter notre méthode à cette particularité, par exemple en introduisant une pondération adaptée sur l'importance de chaque solution, ou encore en introduisant des mécanismes de nichage (section 1.6) plus poussés pour forcer la réduction des fortes densités ? Difficile de répondre sans une parfaite connaissance de l'espace des solutions de la calibration.

Dans cette partie, nous proposerons donc d'étudier l'espace des solutions de la calibration de façon automatique et proposerons une méthode de pondération de l'importance des solutions en fonction.

3.2.2 Introduction aux diagrammes de Voronoï

Nous proposons de réaliser un pavage de l'espace des solutions trouvées, autour de chacune des solutions. Chaque cellule ne renfermant qu'un seul point et formant l'ensemble des points les plus proches de celui-ci et d'aucun autre. Ce pavage définira donc la zone d'influence de chaque point. On appelle ce type de pavage, un diagramme de Voronoï Voronoi (1907) Sen (2016) et chaque point est appelé germe.

La figure 3.7 montre un exemple de diagramme de Voronoï en 2 dimensions. En bleu sont représentés les germes. Les points rouges représentent les points du diagramme de Voronoï. Les traits noirs continus sont ses arêtes finies et les traits noirs en pointillés sont les arêtes qui s'étendent à l'infini.

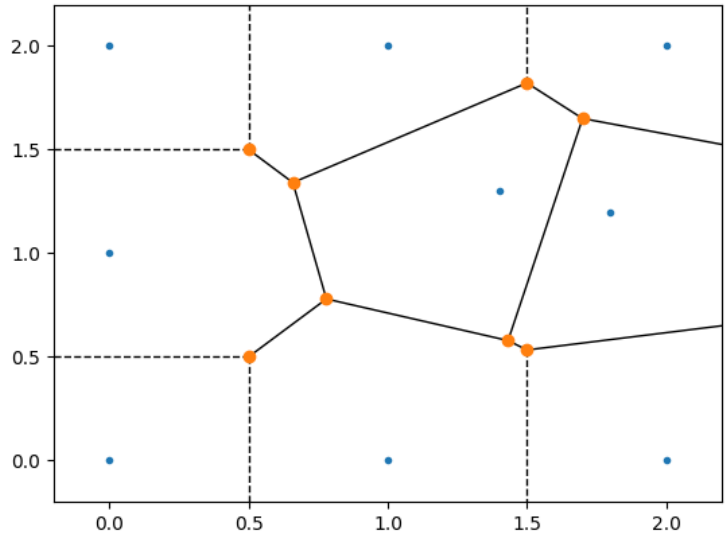


FIGURE 3.7 – Exemple de diagramme de Voronoï en dimension 2

Ce type de diagramme est généralisable en toute dimension, mais sa construction implique un passage par la triangulation de Delaunay. Une triangulation est une partition d'un objet, dans notre cas un ensemble de points, en un ensemble de simplexes. Dans le cas d'un plan (2D), une triangulation est composée de triangles. La triangulation se généralise en dimension n et s'effectue avec des $n-simplexes$. Les $n-simplexes$ sont des généralisations du triangle en dimension n. La triangulation de Delaunay Delaunay et al. (1934) Lee and Schachter (1980) d'un ensemble de points du plan est une triangulation telle qu'aucun point n'est à l'intérieur du cercle circonscrit d'un des triangles. La figure 3.8 donne la triangulation de Delaunay de l'exemple équivalent à celui de la figure précédente.

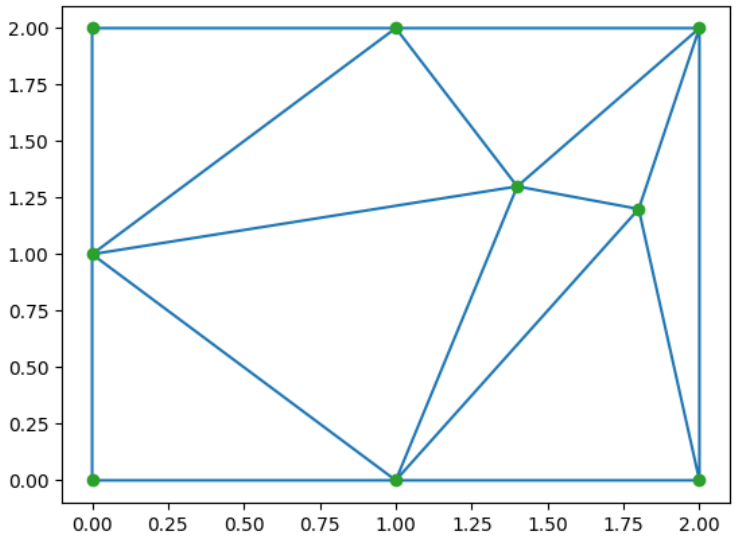


FIGURE 3.8 – Exemple de triangulation de Delaunay en dimension 2

3.2.3 Construction du diagramme de Voronoï en dimension n

La triangulation de Delaunay est réalisable en toute dimension par l'algorithme de Bowyer-Watson Bowyer (1981) Watson (1981). Elle possède un lien particulier avec le diagramme de Voronoï. En effet, les sommets du diagramme de Voronoï sont les centres des cercles circonscrits à chaque triangle et les arêtes sont sur les médiatrices des arêtes de la triangulation de Delaunay.

Les médiatrices peuvent être facilement calculées en positionnant des points au milieu des côtés des triangles. On considérera qu'une cellule ne peut pas dépasser l'enveloppe convexe pour éviter les zones de volume infini non cohérente et éviter les dépassements des intervalles de définition des paramètres. Ainsi, les arêtes infinies du diagramme de Voronoï s'arrêteront à leur intersection avec l'enveloppe convexe également. Généralement dans ce cas là, le germe en question et ses voisins font partie de l'enveloppe convexe et donc l'arrête s'arrête au niveau de la demi-arrête de Delaunay.

La généralisation en dimension n implique le calcul du centre des hypersphères circonscrites aux $n-simplexe$. Par définition, le centre de l'hypersphère circonscrite à un $n-simplexe$ est situé à égale distance de chacun des sommets. L'intersection des plans médiateurs en donne donc la position du centre.

L'algorithme 6 propose une manière de résoudre ce problème. Plaçons-nous en dimension 3 avec 4 points A B C D formant un 3-simplexe. La ligne 4 calcule le milieu de AB AC et AD. La ligne 5 calcule les vecteurs $\overrightarrow{AB}$, $\overrightarrow{AC}$ et $\overrightarrow{AD}$. Les lignes 7 à 11 permettent de remplir la matrice *var* et le vecteur *res* représentant le système d'équations à résoudre. Dans ce cas, la première ligne représente l'équation du plan médiateur de AB, la deuxième de AC et la troisième de AD. La résolution de ce système, ligne 13, permet de trouver le centre de la sphère circonscrite au 3-simplexe ABCD. La résolution de ce système

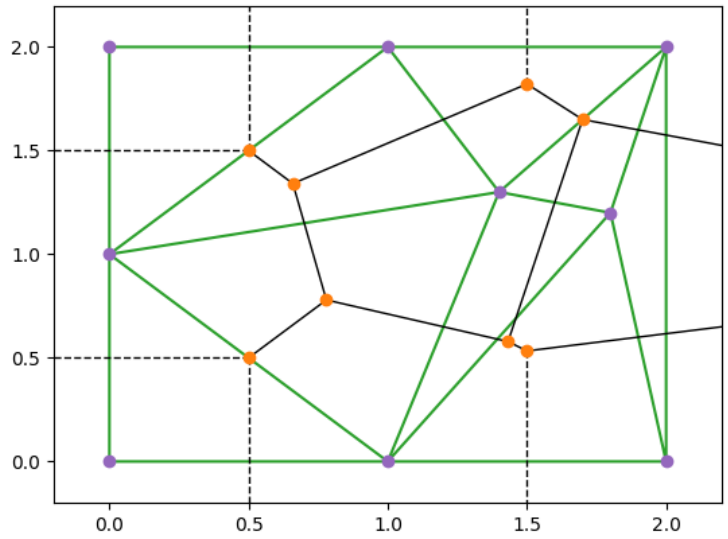


FIGURE 3.9 – Superposition de la triangulation de Delaunay et du diagramme de Voronoï en 2D

peut se faire de plusieurs manières. On peut citer les classiques : élimination de Gauss-Jordan Althoen and Mclaughlin (1987), décomposition de Cholesky Cholesky (2005), ou plus récemment par l'introduction d'aléatoire au sein d'une méthode de Krylov Peng and Vempala (2021).

La figure 3.10 montre la construction de la cellule de Voronoï du point vert clair. La cellule trouvée est délimitée en rouge. Les points violets sont ceux déterminés à partir des médiatrices. Les points oranges présents sur la délimitation représentent les centres des cercles circonscrits aux triangles de la triangulation de Delaunay dont le point vert clair fait partie. Ces points correspondent parfaitement à ceux du diagramme de Voronoï visible en figure 3.7.

On remarque cependant la présence d'un point violet inutile sur une des arêtes de la cellule de Voronoï. Dans le cas où le germe est sur l'enveloppe convexe, cette construction ne le prendra pas en compte comme un des sommets de la cellule. Il faut donc le rajouter à l'ensemble des points précédemment calculé.

Finalement, les centres des hypersphères des simplexes peuvent se trouver à l'extérieur de l'enveloppe convexe des points initiaux (figure 3.11a). Cela n'est pas dérangeant dans les diagrammes de Voronoï habituels, mais comme nous cherchons la zone d'influence de chacun des points, des points situés très loin à l'extérieur pourraient complètement fausser nos résultats en créant des zones énormes et ne correspondant à aucune réalité. Il est donc nécessaire de les ramener à l'intérieur de l'enveloppe convexe en suivant les vecteurs les ayant fait sortir.

Cependant, jusque-là, la construction de ces points ne respecte pas forcément d'ordre. Le calcul de l'enveloppe convexe de l'ensemble de ces points permet de pallier ces problèmes. On peut alors utiliser les points suivant et précédent de chacun des points extérieurs pour calculer l'intersection de leur vecteur

Algorithme 6 Détermination du centre de l'hypersphère circonscrite à un n-simplexe

```
1: constantes ndim, points[ndim + 1][ndim]
2: variables var[ndim][ndim], resultat[ndim]
3: pour i = 1; i < ndim + 1; i + + faire
4:    milieu = (point[0] + point[i])/2
5:    vecteur = point[i] − point[0]
6:    somme = 0
7:    pour j = 0; j < ndim; j + + faire
8:       var[i − 1][j] = vecteur[j]
9:       somme+ = vecteur[j] ∗ (−milieu[j])
10:   fin pour
11:   resultat[i − 1] = −somme
12: fin pour
13: centre = solve(var, resultat)
14: return centre
```

constructeur avec l'enveloppe convexe comme le montre la figure 3.11b. On obtient finalement une cellule de Voronoï parfaite dont les arêtes s'arrêtent à l'intersection avec l'enveloppe convexe de l'ensemble de points initial.

3.2.4 Détermination de fonctions d'évaluation robuste

Une fois nos cellules de Voronoï déterminées, il nous faut déterminer leur hypervolume (généralisation de l'aire 2D en dimension n). Les polytopes obtenus ne sont pas forcément réguliers. Or, tout polytope convexe peut être décomposé en simplexes de sorte que leur union reforme parfaitement le polytope original Stanley (1980) Rubin (1984). L'hypervolume du polytope peut alors être déterminé en sommant les hypervolumes des simplexes le constituant.

L'hypervolume de tout n-simplexe est donné par la relation Stein (1966) :

$$V_s = \left| \frac{1}{n!} det \begin{pmatrix} s_0 & s_1 & ... & s_n \\ 1 & 1 & ... & 1 \end{pmatrix} \right| \tag{3.8}$$

et donc l'hypervolume du polytope :

$$V_p = \sum V_s, \forall s \in S/US = P \tag{3.9}$$

On peut alors redéfinir :

$$\tilde{f}_v(C) = \sum_{i=0}^{nElements} \frac{V_{pi}}{\sum_j^{nElements} V_{pj}} f_i(x) \tag{3.10}$$

Cette nouvelle fonction d'évaluation robuste alloue autant d'importance à chacun des points de l'hypervolume V_{pi}. Or, nous avons vu en section 2.2.2 que l'espace des solutions du calage n'est pas convexe.

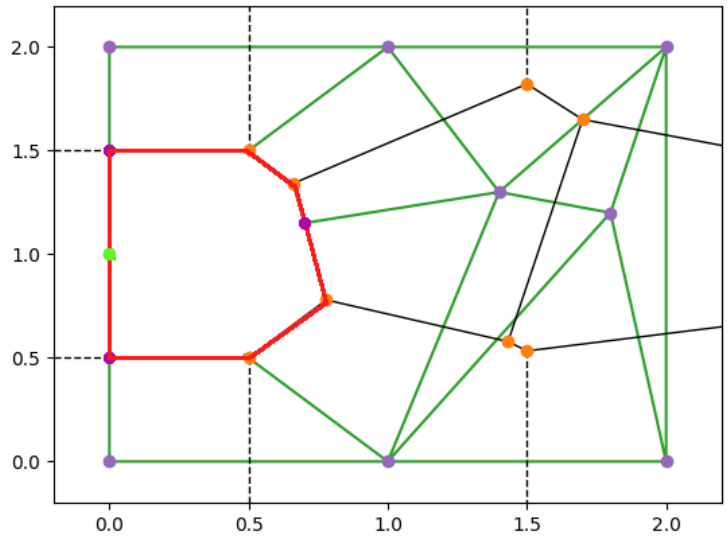


FIGURE 3.10 – Superposition de la triangulation de Delaunay et du diagramme de Voronoï en 2D, construction des régions de Voronoï

Ainsi, les points situés à l'intérieur de P_i ne sont pas forcément des solutions parfaites à la phase de calage. Il est donc discutable d'accorder la même importance à chacune des solutions constituantes P_i. Il est intéressant d'estimer l'espérance de qualité des solutions correspondantes à cet espace et de s'en servir comme pondération pour $f_i(x)$.

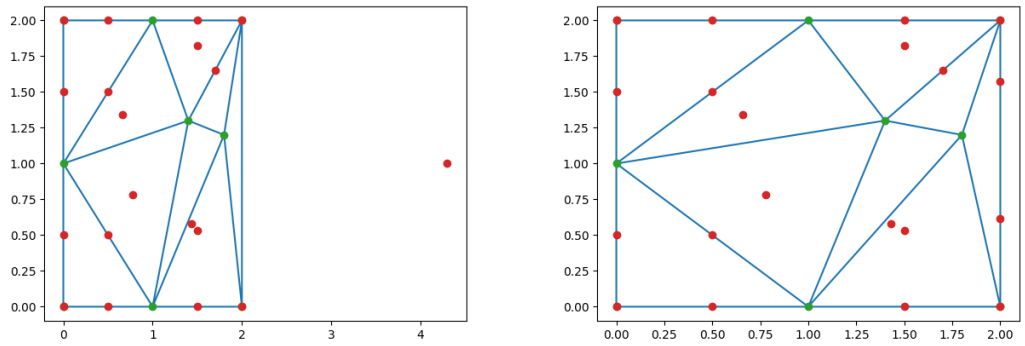
Pour cela, un grand nombre de points doit être généré à l'intérieur du polytope Rubin (1984) et chacun de ces points devra être évalué comme solution du problème de la phase de calage. Soit $f_c(x) : x \to \mathbb{R}$ la fonction d'évaluation de la phase de calage, on peut alors définir p_i le poids de la solution x_i germe du polytope P_i :

$$p_i = \int_{P_i} f_c(x)dx \tag{3.11}$$

et la fonction d'évaluation robuste :

$$\tilde{f}_c(C) = \sum_{i=0}^{nElements} \frac{V_{pi}}{\sum_j^{nElements} V_{pj}} p_i f_i(x) \tag{3.12}$$

Cette méthode permettant de pondérer l'importance de chaque point d'un espace en fonction du volume de sa cellule de Voronoï, nous allons maintenant nous en servir lors d'optimisations robustes afin de prendre en compte des caractéristiques plus précises de l'espace d'incertitudes.



(a) Point extérieur extrême (b) Recadrage du point extérieur

FIGURE 3.11 – Construction du diagramme de Voronoï en 2D à partir de la triangulation de Delaunay, gestion des points extérieur à l'enveloppe convexe

3.3 Optimisation robuste par caractérisation de l'espace d'incertitudes

Les différents calculs de triangulation de Delaunay et d'enveloppe convexe dans des espaces à plus de deux dimensions nécessitent souvent une précision extrême. Cette précision peut ne plus être garantie, notamment lorsque des points sont trop proches les uns des autres. On parle alors de superposition de facettes. La triangulation de Delaunay est connue pour éviter les triangles trop allongés, mais cette propriété peut ne plus être garantie dans certains cas, notamment en haute dimension. Les centres des hypersphères peuvent alors se trouver à des positions quasi-infinies théoriques. Même après recadrage, la forme présentée par le n-simplex peut être complètement incohérente, avec de très fortes étendues sur un paramètre et peu sur d'autres. Dans ce cas, l'estimation de la qualité du sous-espace des solutions proposées permet de complètement annuler cette solution.

En repartant de $p_i = \int_{P_i} f_c(x)dx$, nous allons chercher à ce qu'une solution parfaite au calage soit significative lors de la phase d'optimisation robuste et qu'au contraire, une solution de faible qualité de ce sous-espace ne soit pas du tout comptabilisée. Habituellement, nous utilisons (section 2.3.2), $f(x) = \sum_i^{nbAnnees}(C - Cs)^2$. Dans ces conditions, une différence entre captures, C, et captures simulées, C_s, supérieure à 1 unité de masse (tonne dans tous nos exemples), provoque directement une grande augmentation dans la fonction d'évaluation. À l'inverse, la mise au carré permet en quelque sorte de récompenser les années où les estimations sont particulièrement bonnes. Ainsi, nous proposons l'utilisation de la fonction suivante :

$$f_c(x) = 1 - f(x) \iff f(x) < 1, \ autrement \ f_c(x) = 0 \tag{3.13}$$

De plus, en repartant de $\tilde{f}_c(C) = \sum_{i=0}^{nVoronoiCells} \frac{V_{pi}}{\sum_j^{nElements} V_{pj}} p_i f_i(x)$, on peut obtenir la même

échelle de valeur que $\tilde{f}(C)$ en normalisant $\frac{V_{pi}}{\sum_j^{nElements} V_{pj}} p_i$ à 1. On obtient alors :

$$\tilde{f}_c(C) = \sum_{i=0}^{nVoronoiCells} \frac{V_{pi} p_i}{\sum_j^{nVoronoiCells} V_{pj} p_j} f_i(x) \tag{3.14}$$

Dans cette application, le tirage des solutions à évaluer au sein de chaque cellule se fait selon une répartition uniforme des valeurs des paramètres au sein d'un hyper-rectangle dont les côtés sont définis par les valeurs extrêmes de chacun des quatre paramètres du modèle de Graham-Schaefer des points extrêmes de l'enveloppe convexe de la cellule. On cherche ensuite à estimer directement les valeurs des efforts par une approche directe. Dans le cas où celle-ci n'aboutit pas, l'utilisation d'un PSO2007 permet une convergence rapide (section 2.3.2).

Les résultats sont présentés en tableau 3.2. Ceux-ci sont toujours supérieurs à ceux de l'optimisation robuste classique, mais pas forcément significatifs comme pour palinurus et physics. Cette faible différence s'explique par la contrainte restreignant fortement les possibilités d'action. L'amélioration est cependant conséquente pour les autres cas. La réduction du nombre de solutions, et le faible poids relatif accordé à celles dont la cellule de Voronoï est de faible qualité, permettent de toujours améliorer les résultats.

	Captures réelles	Optimisation globale	RO	RO_v	RO_p
Dentex	8209	10972	8101	8872	9013
Palinurus	11157	18934	17130	17406	17422
Diplodus	10366	11774	9706	10618	10744
Scorpaena	6535	8791	8068	8104	8097
Physics	4184	4412	4106	4155	4159

TABLE 3.2 – Comparaison des résultats entre les différentes méthodes d'optimisations robuste

Comme pour l'optimisation globale, la visualisation de l'ensemble des compromis possibles entre la biomasse finale et la quantité de captures peut être un bon indicateur d'aide à la décision pour une gestion de l'exploitation. Nous pouvons réutiliser l'algorithme NSGA-II en cherchant à maximiser :

$$f_1(C) = \tilde{f}_c(C) \tag{3.15}$$

$$f_2(C) = \forall x \in C, min(BFinale_x) \tag{3.16}$$

Pour le Dentex, nous obtenons les fronts de Pareto visibles en figure 3.12. On remarque dans un premier temps que l'optimisation robuste utilisant la caractérisation de l'espace par diagramme de Voronoï, propose des résultats bien meilleurs que l'optimisation robuste classique. Ceux-ci sont cependant toujours très inférieurs à ceux que peut proposer une optimisation globale. La qualité des informations à notre disposition pour la phase de calibration est donc d'une importance capitale.

On peut également voir qu'en optimisation robuste, même avec très peu de captures, il est impossible d'assurer les mêmes niveaux de biomasse qu'en optimisation globale. Le manque d'information joue donc

un rôle crucial sur les possibilités écologiques.

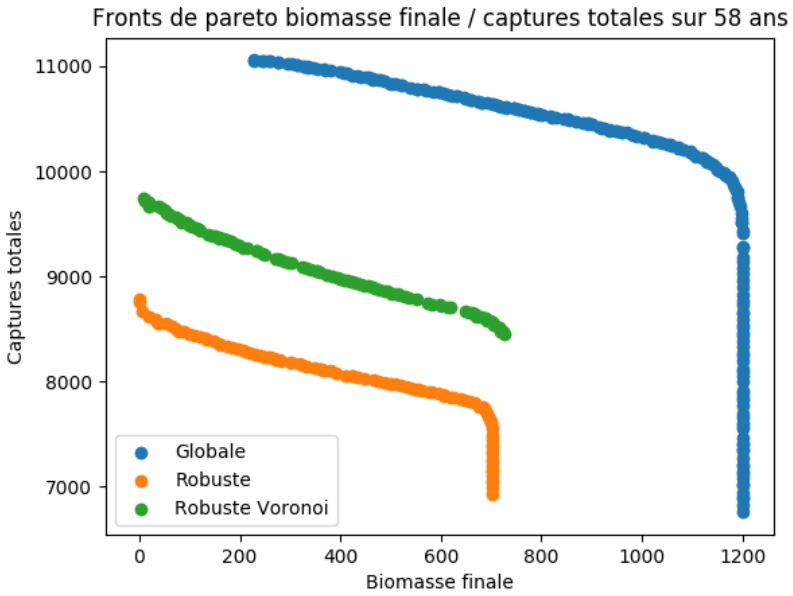


FIGURE 3.12 – Comparaison des fronts de Pareto des captures totales en fonction de la biomasse finale entre optimisation globale, robuste, et robuste avec la méthode de Voronoï, pour le Dentex.

À noter toutefois que cette représentation montre les niveaux de biomasse du pire cas, car c'est ce qui est préconisé pour les questions écologiques et également pour l'optimisation robuste. La réalité pourrait mener à des résultats bien meilleurs. Les simulations s'effectuent sur de longues durées. La découverte de nouvelles informations au cours de cette durée pourrait mener à une amélioration conséquente des connaissances et donc à une réorientation des stratégies de gestion. C'est le but de l'optimisation robuste ajustable.

3.4 Optimisation robuste ajustable

L'optimisation robuste ajustable, va permettre d'utiliser les résultats de chaque année de simulation, comme connaissance supplémentaire pour réduire les incertitudes. La figure 3.13 montre son déroulement. Une optimisation robuste est réalisée sur le cluster de solutions SC_0. L'effort correspondant à la première année, E_0, est appliqué. L'environnement réagit en produisant les captures, C_0, correspondant à la dynamique de population réelle cachée. Cette nouvelle valeur de captures peut alors être utilisée pour tester toutes les solutions, $x \in SC_0$, et construire un nouveau cluster, SC_1, constitué uniquement des x dont l'évolution soumise à E_0 donnerait des captures cohérentes. Ce processus est réitéré sur l'ensemble des années de simulation i. On a alors $[SC_{i+1}| \leq |SC_i| \forall i$, soit une amélioration des connaissances au cours du temps et donc une amélioration des résultats. Au terme de ce processus on obtient $f(x) = \sum C_i$,

correspondant aux captures totales obtenues si la dynamique était celle qui a permis à l'environnement de réagir. En itérant $\forall x \in SC_0$, on obtient la fonction d'évaluation $\tilde{F}(SC_0) = \frac{1}{|SC_0|} \sum_{\forall x \in SC_0} f(x)$

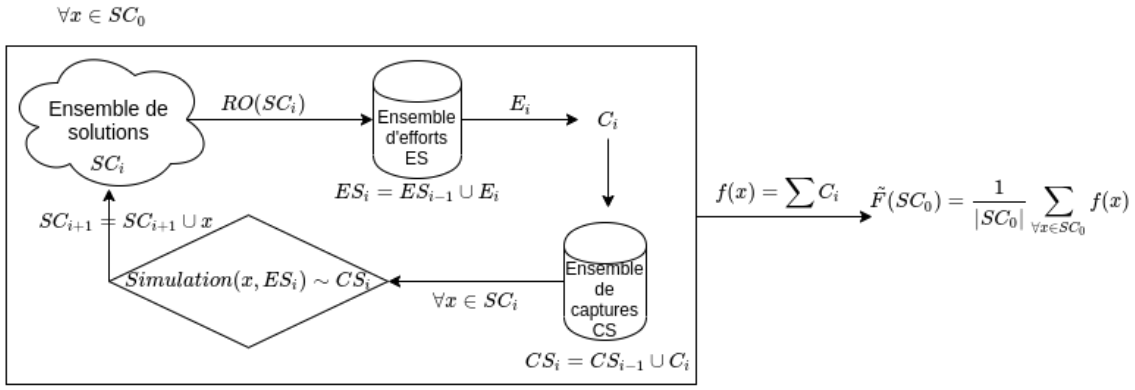


FIGURE 3.13 – Schéma général de l'optimisation robuste ajustable.

La figure 3.14 montre l'évolution de la biomasse minimum, moyenne et maximum au cours du temps lors d'un processus d'ARO. En comparant à la figure 3.3 on constate que l'ARO permet une stabilisation rapide de la biomasse, comparable à celle obtenue par un processus d'optimisation globale avec connaissance parfaite.

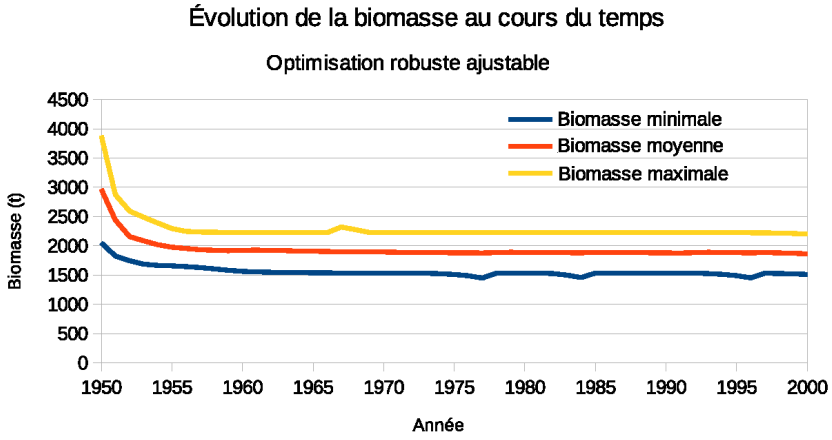


FIGURE 3.14 – Évolution de la biomasse en utilisant l'optimisation robuste ajustable

De même, les captures présentées dans le tableau 3.3 montrent que l'amélioration des connaissances obtenues au cours du processus d'ARO permet d'obtenir des résultats nettement supérieurs à ceux de l'optimisation robuste. Dans certains cas, ils se rapprochent même très fortement de ceux obtenus avec une connaissance parfaite du système.

Ces résultats sont cependant à mettre en relation avec le taux de variation au sein de SC_0. La figure

	Captures réelles	Optimisation globale	RO_p	ARO
Dentex	8209	10972	9013	9553
Palinurus	11157	18934	17422	18460
Diplodus	10366	11774	10744	10824
Scorpaena	6535	8791	8097	8578
Physics	4184	4412	4159	4273

TABLE 3.3 – Comparaison des résultats des différentes méthodes

3.15 montre que sur une longue simulation (50 ans), même un fort taux d'incertitude permet d'obtenir d'assez bons résultats : 8% de différence à l'optimal pour 18% d'incertitude. Pour seulement 10 ans de simulation, l'ARO obtient des résultats comparables à RO sur de faibles taux d'incertitude. À des taux plus élevés, l'ARO améliore ces résultats, mais devient également difficilement utilisable à partir de 15% d'incertitudes.

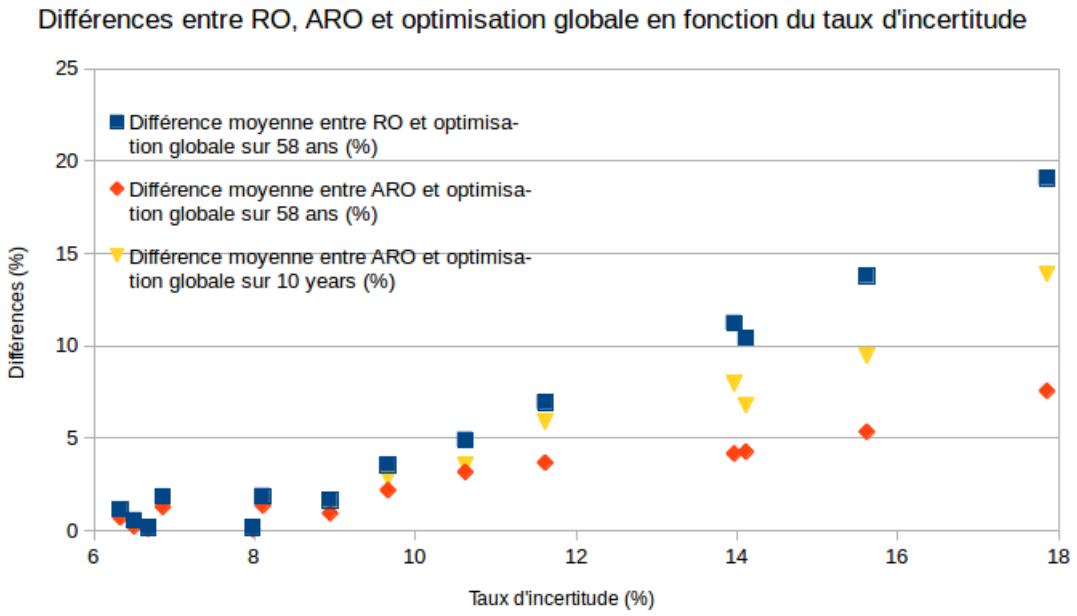


FIGURE 3.15 – Différence entre optimisation globale et robuste / robuste ajustable

3.5 Conclusion de chapitre

Dans ce chapitre, nous cherchions à proposer des stratégies de pêche permettant d'obtenir des résultats économiquement et écologiquement viables. Pour cela, nous devions prendre en compte les incertitudes engendrées par la phase de calibration de modèle. Nous avons donc testé différentes méthodes d'optimisation robuste et d'optimisation robuste ajustable, et comparé leurs résultats aux données estimées dans

la littérature.

Les approches proposées ont montré des résultats intéressants. Le processus d'optimisation permet généralement d'obtenir des résultats nettement supérieurs sur le long terme sur tous les aspects de la problématique étudiée, justifiant ainsi l'intérêt de notre étude.

De plus, une approche robuste, notamment robuste ajustable semble peu contraignante d'un point de vue des résultats économiques que les pêcheurs peuvent espérer. Elle permet de réduire les risques dus à un manque de connaissances lors de la calibration. Ces résultats sont quand même clairement dépendants de l'étendue de l'espace des solutions au calage et de sa nature. Une réduction de celui-ci, au moins via expertise, autour de valeurs significatives, permet de grandement améliorer ses résultats. Pour des applications réelles, il serait donc préférable de la privilégier à l'avenir au moins sur les espèces dont nous savons qu'elles sont déjà en situation de vulnérabilité.

D'un point de vue théorique, l'approche par caractérisation de l'espace des solutions permet d'allouer une importance cohérente à chacun des points de l'ensemble d'incertitudes d'entrée. Ce processus permet de ne pas biaiser l'optimisation vers les régions à forte densité lorsque le calage est de faible qualité. Il permet également de détecter des solutions incohérentes au vu de leur voisinage direct, réduisant ainsi la complexité de l'optimisation et donc permettant de gérer des cas plus complexes. Le temps de calcul du diagramme de Voronoï, volumes et poids peut ne pas être négligeable, notamment sur des cas où le calage est de qualité, mais celui-ci s'effectue en temps polynomial. La réduction de l'ensemble incertain engendrée permet donc une amélioration du temps de calcul des optimisations robustes sur des espaces d'incertitude large.

L'approche par optimisation semble convenir pour une étude à l'échelle de la pêcherie sous condition d'intervention humaine fiable. En revanche, la nature hautement incertaine de l'environnement étudié, la difficulté de modélisation et la nécessité d'intervention humaine pour une amélioration conséquente des résultats semblent peu cohérentes pour conseiller les exploitants directement.

Dans les approches présentées dans ce chapitre, nous avons toujours supposé un contrôle parfait de l'exploitation, c'est-à-dire que les quotas instaurés seraient parfaitement respectés. Cette hypothèse est assez peu réaliste. Pour prendre cela en compte, il devient nécessaire de modéliser le comportement individuel des pêcheurs. En effet, les incertitudes de faisabilité des solutions proposées à l'échelle des décideurs ne sont pas quantifiables directement à l'échelle des pêcheurs. Il est donc nécessaire de changer d'échelle pour mieux décrire les comportements à une granularité temporelle plus fine. Gérer un tel système par optimisation nécessiterait une énorme quantité de simulations Monte-Carlo à chaque évaluation pour quantifier la robustesse d'une solution. Cela, couplé au grand nombre de paramètres à optimiser mène à une impossibilité d'utilisation en temps de calcul raisonnable.

De plus, la plupart des incertitudes auxquelles nous avions à faire face ici venaient directement de la phase de calibration et du manque de fiabilité des données disponibles. Pour modéliser l'évolution d'un système dont les acteurs n'ont pas ou peu de connaissance sur sa dynamique, nous pouvons utiliser des processus de décision markoviens partiellement observables Monahan (1982). L'environnement étudié ici peut parfaitement être représenté sous cette forme. Les méthodes d'apprentissage par renforcement semblent alors tout indiquées Le Tuyen et al. (2018) Jaakkola et al. (1995) pour aider à résoudre ce type

de problèmes. Ainsi, les méthodes d'apprentissage par renforcement devraient permettre d'apprendre à gérer ce type d'exploitation en proposant des politiques à la fois pour les pêcheurs et les décideurs.

Ainsi, pour l'ensemble de ces raisons, nous proposons de tester et d'appliquer une approche basée sur l'apprentissage par renforcement. Cette approche sera présentée en deux chapitres. Pour commencer nous présenterons les formalismes de modélisation et une première approche basée sur les bandits manchots, puis un dernier chapitre s'intéressera aux méthodes d'apprentissage profond par renforcement.

Chapitre 4

APPRENTISSAGE AUTOMATIQUE : DESCRIPTION ET APPLICATION PAR BANDIT MANCHOT

L'apprentissage automatique (plus souvent appelé *machine learning* en anglais) est un domaine de recherche de l'intelligence artificielle. Basé sur de grandes quantités de données et sur des méthodes d'apprentissage, il a pour objectif de résoudre une tâche déterminée à l'avance. Nous pouvons distinguer 3 grands types de méthodes :

— l'apprentissage supervisé : dans ce cas l'utilisateur se base sur une grande quantité de données étiquetées (*dataset*), montre des exemples à l'algorithme à partir du *dataset* et attend une réponse. Le but est d'apprendre un modèle permettant de retrouver les étiquettes du *dataset* le plus fidèlement possible et ensuite généraliser pour d'autres entrées. L'exemple récent le plus parlant étant certainement la grande avancée en matière de reconnaissances d'image via classification supervisée. Un grand nombre de méthodes existent parmi les plus connues : les machines à vecteurs de support Boser et al. (1992), les K plus proches voisins Altman (1992), les arbres de décision Safavian and Landgrebe (1991), la classification naïve bayésienne Domingos and Pazzani (1997) ou ayant fait leurs preuves plus récemment, les réseaux de neurones artificiels LeCun et al. (2015).

— l'apprentissage non supervisé : il se base également sur une grande quantité de données, mais pour lesquelles l'utilisateur ne sait pas précisément quoi chercher. On ne peut alors pas calculer clairement un score de réussite à partir d'étiquettes. Dans ce cas on cherchera à faire une classification via similarité plutôt que, pour reprendre l'exemple précédent, de directement identifier une classe d'images à un label précis. Pour cet exemple, la plupart du temps, on utilisera des réseaux de neu-

rones à convolution comme pour le cas supervisé. Pour d'autres cas d'application, un grand nombre de méthodes existent également telles que les K-moyenne MacQueen et al. (1967), la classification hiérarchique Sibson (1973) ou encore la réduction de dimensionnalité Samet (2006).

— l'apprentissage par renforcement : ce dernier diffère beaucoup des précédents dans la mesure où, bien que certaines méthodes se basent sur des données d'observations directes de taille fixe Lange et al. (2012), il s'applique très souvent sur des modèles et simulations déjà existants afin de faire apprendre un comportement à un agent autonome au sein de cet environnement. Celui-ci devra donc interagir avec son environnement et recevra des observations à partir desquelles il devra décider de ses futures actions. La plupart du temps, elles ne sont pas déterministes mais plutôt stochastiques. Dans ce cas, l'agent apprendra à partir d'une récompense (ou pénalité quand celle-ci prend une valeur négative) qu'il recevra en fonction de ses actions et son état. Cette méthode est particulièrement efficace dans les domaines où la simulation joue un grand rôle.

Basant toutes nos études sur des simulations, l'apprentissage par renforcement semble être une méthode très adaptée à nos problématiques.

Le domaine de l'apprentissage par renforcement présente un certain lexique qu'il peut être difficile de prendre en main. L'annexe 6.1 en présente quelques définitions importantes.

L'apprentissage par renforcement est un sous-domaine de l'apprentissage automatique qui enseigne à un agent comment choisir une action dans son espace d'actions, dans un environnement particulier, afin de maximiser ses récompenses au cours du temps. On le distingue souvent des méthodes d'apprentissage supervisé et non supervisé par sa nature dynamique. L'objectif des apprentissage supervisé et non supervisé est de rechercher et d'apprendre des modèles dans les données d'entraînement, ce qui est assez statique. L'apprentissage par renforcement, consiste à développer une politique qui indique à un agent l'action à choisir à chaque étape. De plus, il ne nécessite pas de bonnes réponses directement définies. Dans l'apprentissage supervisé, la bonne réponse est donnée par les données d'entraînement. En apprentissage par renforcement, la bonne réponse n'est pas donnée explicitement : l'agent doit apprendre par essais et erreurs. La seule référence est la récompense qu'il reçoit après avoir effectué une action, qui lui indique s'il progresse ou s'il échoue. Ainsi, comme en optimisation, il nécessite une phase d'exploration. Un agent d'apprentissage par renforcement doit trouver le bon équilibre entre l'exploration de l'environnement, la recherche de nouvelles façons d'obtenir des récompenses et l'exploitation des sources de récompense qu'il a déjà découvertes. En revanche, les systèmes d'apprentissages supervisé et non supervisé tirent la réponse directement des données d'apprentissage sans avoir à explorer d'autres réponses. L'apprentissage par renforcement est un processus à décisions multiples soumises généralement à temporalité : celles-ci forment une chaîne de décision à travers le temps nécessaire pour terminer un travail spécifique. À l'inverse, l'apprentissage supervisé est un processus à décision unique : une instance, une prédiction, un résultat.

On peut définir l'apprentissage par renforcement par 4 éléments principaux :

— la politique (*policy*) : elle définit la façon dont l'agent va interagir avec son environnement. Certains algorithmes distinguent la politique de comportement (*behavior policy*), qui va agir sur l'environnement pendant la phase d'entraînement, et la politique cible (*target policy*) qui, elle, va être la

politique apprise à partir des transitions engendrées par la politique de comportement. C'est elle qui sera ensuite utilisée pour agir en situation réelle, après apprentissage. On parle alors d'algorithmes *off-policy*.

— le signal de récompense (*reward signal*) : définit l'objectif à atteindre. À chaque pas de temps, l'environnement envoie ce signal à l'agent sous forme d'un scalaire unique. Il définit ainsi ce qui est bon ou mauvais. Un exemple explicite pourrait être, en biologie, des signaux de douleurs ou de plaisirs.

— une fonction de valeur (*value function*) : définit ce qui est bien sur le long terme. Le signal de récompense détermine la désirabilité immédiate d'une situation alors que la fonction de valeur détermine la désirabilité sur le long terme.

— un modèle de l'environnement, permettant de simuler les interactions avec l'environnement réel. Il devra par exemple être capable, pour un état et une action donnés, de générer le signal de récompense et le nouvel état associé.

Buts, récompenses et épisodes L'utilisation d'un signal de récompense pour formaliser l'idée du but à atteindre est une des fonctionnalités distinctives de l'apprentissage par renforcement.

Même si cette approche peut paraître limitante de prime abord, en pratique, elle a été prouvée flexible et largement applicable.

Un épisode est défini comme l'ensemble des états, actions et récompenses entraînant le passage d'un état initial à un état final pour lequel les interactions agent-environnement s'arrêtent naturellement. Par exemple, aux échecs, du début de la partie, jusqu'à ce qu'un joueur soit échec et mat ou abandonne. On note le dernier pas de temps T, souvent appelé état terminal, et on définit le retour comme fonction de la séquence de récompense. La plus simple étant :

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{4.1}$$

Une fois l'état terminal atteint, on effectue une remise à zéro (*reset*) du système à un état initial fixe ou suivant une certaine loi de distribution indépendemment de la façon dont s'est terminé l'épisode précédent.

Il peut arriver que l'état final ne soit pas atteint naturellement et que les interactions puissent continuer à l'infini. Ce genre de tâches, appelées tâches continues, peut poser un problème de formulation car ici $T = \infty$ et la récompense peut donc potentiellement être infinie aussi.

Un autre concept nécessaire est celui de d'atténuation (*discounting* en anglais) introduisant la notion d'antériorité de la récompense avec un paramètre appelé facteur d'atténuation (*discount rate*) noté $\gamma \in [0; 1]$. On a ainsi :

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.2}$$

Politique et fonction de valeur Presque tous les algorithmes d'apprentissage par renforcement utilisent des fonctions de valeur permettant d'estimer à quel point une situation (état) est bonne. Elle

est définie en termes de future récompense possible. Bien sûr, la fonction de valeur doit être définie en respectant les possibilités d'actions, appelées politiques.

Formellement, une politique π est un mapping entre état et probabilité de sélection d'action. On a donc $\pi(a|s)$ la probabilité de choisir $A_t = a$ si $S_t = s$. Ainsi, la valeur d'un état s sous la politique π, noté $v_\pi(s)$ est la valeur de retour espérée en partant de l'état s et en suivant la politique π. Ainsi, dans un processus de décision markovien (*Markov Decision process* en anglais, MDP), on peut définir v_π comme :

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \ \forall s \in S \tag{4.3}$$

La valeur d'une action a dans l'état s sous la politique π notée $q_\pi(s, a)$ est définie comme l'espérance de retour en partant de s, en prenant l'action a puis en suivant la politique π.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a\right] \tag{4.4}$$

q_π est donc appelée fonction de valeur d'action (*action-value function*) pour la politique π.

Il est possible d'estimer ces fonctions de valeurs par expérience, par exemple en fixant une politique π et en moyennant la valeur de retour pour chaque état, s, rencontré, alors cette moyenne convergera vers la valeur de $v_\pi(s)$ quand le nombre de fois que cet état aura été rencontré tendra vers l'infini. Ce type de méthode est appelé méthode de Monte-Carlo et sera présenté en section 5.1.1. Ces méthodes présentent cependant certaines limites lorsqu'un trop grand nombre d'états est possible. Il sera en effet trop difficile de conserver des informations sur chaque état et/ou de tous les explorer. Dans ce cas, on cherchera à estimer v_π et q_π de façon paramétrique, avec moins de paramètres que d'états, en ajustant les paramètres afin de coller aux observations. Ces possibilités seront discutées en section 5.1.2.

Résoudre un problème d'apprentissage par renforcement revient donc à trouver une politique optimale π_*. On définit qu'une politique π est meilleure qu'une politique π' si et seulement si :

$$v_\pi(s) \geq v_{\pi'}(s) \ \forall s \in S \tag{4.5}$$

Notion d'environnement Dans cette section, nous introduisons le concept de processus de décision markovien (MDP) White III and White (1989) fini, formalisme à la base d'un grand nombre de problèmes de RL que nous chercherons à résoudre par la suite. Comme pour les algorithmes de bandits, ils cherchent à prédire la meilleure action possible, mais ils prennent également en compte la situation actuelle de l'agent. Ici, l'action influence non seulement la récompense obtenue, mais également l'état courant. Dans les algorithmes de bandits, nous cherchons à estimer $q_*(a)$ de chaque action a alors que dans les MDP nous estimerons $q_*(a, s)$ de chaque action a pour chaque état s ou $v_*(s)$ de chaque état par rapport aux actions optimales. Ces notions seront plus détaillées en section 4.2.

Nous pouvons définir l'agent apprenant (*learner*) et décidant *decision maker*, appelé agent, ou agent de renforcement, et ce avec quoi il agit, l'environnement. Cette interaction se fait par action de l'agent, action à laquelle l'environnement répond en générant un nouvel état, ou une nouvelle situation. À ce

moment, l'environnement fournit une récompense sous forme scalaire. Cette interaction est illustrée en figure 4.1. Ainsi, à chaque pas de temps discret, l'agent reçoit la situation actuelle de l'environnement $S_t \in S$ et effectue une action $A_t \in A(s)$. L'environnement fournit alors une récompense $R_{t+1} \in R \subset \mathbb{R}$ et un nouvel état S_{t+1}.

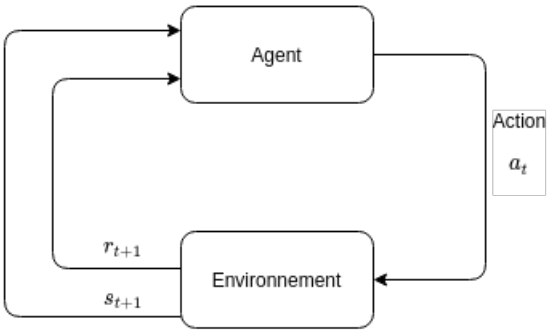


FIGURE 4.1 – Interaction entre agent et environnement dans un processus de décision markovien

Le formalisme MDP est une considérable abstraction de problèmes d'apprentissage par interactions et dirigé vers un but connu. La grande variété des états et actions entre différentes tâches peut grandement affecter les performances, particulièrement en apprentissage par renforcement.

4.1 Notions de processus de décision markoviens

Le formalisme le plus simple est certainement le processus de décision markovien fini. Dans un processus de décision markovien fini, les ensembles d'états, d'actions et de récompenses sont tous finis. Le formalisme MDP est assez abstrait et flexible et peut être appliqué à beaucoup de problèmes différents de très nombreuses manières. Par exemple, le pas de temps peut ne pas être régulier et simplement faire référence à l'intervalle de temps qui s'écoule entre deux actions. Les états peuvent également prendre toutes formes. Ainsi, en général, les actions peuvent être n'importe quelle décision que nous voulons apprendre à prendre avec efficacité.

Les connaissances de l'agent peuvent également être complètes ou partielles sur son environnement. Il peut même connaître certains mécanismes de calcul des récompenses. Ainsi, les frontières de connaissances de l'agent ne sont pas fixées par le formalisme, elles représentent juste les limites du contrôle absolu de l'agent, pas de ses connaissances.

4.1.1 Processus de décision markoviens

Les processus de décision markoviens (MDP) sont une extension des chaînes de Markov. Cette extension comporte l'addition des actions choisies par l'agent et des récompenses gagnées. On peut les caractériser par quatre éléments principaux :

— l'espace d'états S fini dénombrable ou continu ;

— l'espace d'actions A généralement fini dénombrable mais pouvant être continu ;

— la fonction de transition notée T ou $P : S \times A \times S \to [0; 1]$, que nous noterons T dans toute la suite de ce document. Elle peut être déterministe ou stochastique ;

— la fonction de récompense $R : S \times A \times S \to \mathbf{R}$ permettant de donner une valeur au fait de partir de l'état s, effectuer l'action a pour arriver dans un nouvel état s' ;

Dans le cadre de l'apprentissage par renforcement, un agent de renforcement interagit avec un environnement ϵ via une politique $\pi : S \times A \to [0; 1]$. Celle-ci permet à l'agent de choisir une action en fonction de l'état courant du système. Ainsi, le processus d'apprentissage se déroule de la façon suivante :

— l'agent reçoit l'état de l'environnement s_t et sélectionne une action a_t

— l'environnement répond à l'action en produisant un nouvel état s_{t+1} de façon probabiliste en utilisant la fonction de transition p

— une récompense r est déterminée en fonction de s_t, a_t et s_{t+1}

— la politique π est mise à jour en fonction de la récompense générée.

Pour mettre à jour les politiques, plusieurs critères peuvent être définis mais la plupart du temps nous utilisons :

$$E(\sum_{t=0}^{\infty} \gamma^t r_t) \tag{4.6}$$

ou

$$E(\sum_{t=0}^{h} r_t) \tag{4.7}$$

respectivement l'espérance des récompenses amorties à horizon infini avec le facteur d'atténuation $0 \leq \gamma < 1$, et l'espérance de la somme des récompenses pour un horizon fini fixé h.

On peut alors définir les fonctions de valeurs. La fonction de valeur des états représente le gain espéré en commençant dans l'état s et en appliquant π tout au long de la simulation. On la note $V^\pi(s)$ tel que $V^\pi : S \to \mathbf{R}$.

De même, on peut définir *l'action state-value function*, ou fonction de valeur action état, représentant les gains espérés en partant d'un état s et prenant une action a puis en suivant π jusqu'à la fin. On la note $Q^\pi(s, a)$ tel que $Q^\pi : S \times A \to \mathbf{R}$.

Dans le cas d'un critère par gain amorti on peut alors définir :

$$Q^\pi(s, a) = \sum_{s' \in S} [R(s, a, s') + \gamma V^\pi(s')] T(s, a, s') \tag{4.8}$$

d'où l'équation de Bellman Bellman (1954) :

$$V^\pi(s) = \sum_{s' \in S} [R(s, \pi(s), s') + \gamma V^\pi(s')] T(s, \pi(s), s') \tag{4.9}$$

Ceci étant dans le cas idéal où l'agent connaît parfaitement l'état courant s. Dans des problèmes réels complexes, tels que peuvent être ceux rencontrés en gestion de ressources naturelles, les connaissances

sont souvent limitées. L'état peut alors être considéré comme caché et l'agent ne doit se baser que sur des observations. On est alors dans un processus de décision markovien partiellement observable (POMDP).

4.1.2 Processus de décision markovien partiellement observable

Formellement, un Processus de décision markovien partiellement observable (POMDP) est défini par :

— S, l'ensemble des états possibles, mais qui, ici, sont cachés ;

— A, l'ensemble des actions possibles ;

— T, la fonction de transition, mais qui, cette fois, s'applique sur les états cachés ;

— R, la fonction de récompense. Généralement définie de la même façon que dans les MDP classiques, mais des variantes peuvent exister où on se basera plutôt sur les observations ou sur des préférences pour définir les récompenses Wirth et al. (2017).

— Ω l'ensemble des observations possibles ;

— $O : S \times \Omega \to [0; 1]$ la fonction d'observation qui associe la probabilité $p(\omega|s) = O(s, \omega)$ d'observer une information.

Dans les cas où on a une idée et des possibilités d'estimer la probabilité qu'un état s soit effectivement le bon, nous pouvons chercher à résoudre ces problèmes par des approches cherchant à estimer la distribution de probabilité des états sachant les observations. On peut alors parler de MDP basé sur des croyances ($belief-MDP$) Kaelbling et al. (1998). Sinon, on ne considère que les observations. Il faut alors maintenir à jour une mémoire des observations pour éviter d'être bloqué par des observations redondantes, générées par des états cachés différents. On peut alors perdre la propriété de Markov définissant que la distribution de probabilité des états futurs ne doit dépendre uniquement que de l'état présent.

Ce dernier cas est souvent incontournable quand l'apprentissage s'effectue dans un système multi-agent, chacun indépendant, et où les observations de chacun ne peuvent pas leur permettre d'estimer correctement l'état caché du système. De façon plus générale, dans le cas de systèmes multi-agents autonomes dans un PODMP, on parle de Processus de décision markovien partiellement observable décentralisé (Dec-PODMP).

4.1.3 Processus de décision markovien partiellement observable décentralisé

Comme les agents sont conçus pour des environnements de plus en plus complexes, les méthodes qui tiennent compte de l'incertitude du système présentent de grands avantages. Cette incertitude est courante dans des domaines tels que la navigation autonome, la gestion des stocks, les réseaux de capteurs et le commerce électronique. Lorsque les choix sont faits de manière décentralisée par un ensemble de décideurs, le problème peut être modélisé comme un processus de décision markovien partiellement observable décentralisé (Dec-POMDP) Oliehoek et al. (2016). Bien que le formalisme Dec-POMDP offre un cadre riche pour la prise de décision séquentielle, coopérative ou non, dans l'incertitude, la complexité de calcul du modèle présente un défi de recherche important Bernstein et al. (2002). Il s'agit d'une extension du

cadre du POMDP et d'un cas spécifique de jeu stochastique partiellement observable (POSG) Hansen et al. (2004).

Un Dec-POMDP peut être décrit de la façon suivante :

— I, l'ensemble des agents ;

— S, l'ensemble des états cachés ;

— A_i, l'ensemble des actions pour l'agent i, avec $A = \prod_i A_i$, l'ensemble des actions jointes ;

— T, la fonction de transition.

— R, la fonction de récompense globale ;

— Ω_i l'ensemble des observations possibles de l'agent i avec $\Omega = \prod_i \Omega_i$, l'ensemble des obsevations jointes

— O la fonction d'observation ;

— h définissant l'horizon, fini ou infini ;

— γ le facteur d'atténuation.

Notons que certains auteurs Katt et al. (2019) définissent plutôt D, la fonction de dynamique, agrégeant transitions et probabilités d'observation : $D(s', o|s, a)$.

À chaque instant, chaque agent prend donc une action $a_i(t)$ et obtient une observation locale et une récompense, individuelle ou jointe. On peut alors définir une politique, π_i, locale à chaque agent. Les décisions prises par chaque agent n'affectent que ses propres observations, mais ont un effet sur l'environnement commun à tous les agents.

On est alors parfaitement dans le cas d'une pêcherie, où chaque pêcheur agit indépendamment des autres, obtient des observations propres à son activité (quantités de ressources pêchées) et une récompense associée, mais où ses actions ont des répercutions sur l'ensemble de l'environnement. Dans le cas où les agents sont en compétition, on peut parler de jeu stochastique ou encore de jeu de markov.

Quand les états ne sont pas connus, mais seulement estimés via des observations, le but est alors de gérer le compromis entre exploration et approfondissement de manière optimale afin de rapidement estimer la valeur de chaque action. On peut alors penser à un problème du bandit manchot à k-bras.

4.2 Notions de bandit manchot à plusieurs bras

Le problème du bandit manchot peut être vu comme un processus de décision markovien avec un seul état. Il nous permettra d'étudier l'aspect évaluatif de l'apprentissage par renforcement sur des configurations simples qui n'incluent pas vraiment d'apprentissage pour agir sur d'autres situations que la présente. Il évite donc une grande partie de la complexité des problèmes d'apprentissage par renforcement complet, mais étudier ces problèmes et méthodes permet de mieux visualiser l'importance du processsus évaluatif.

Ainsi, pour commencer, nous étudions une version simple du problème du bandit manchot à k bras.

4.2.1 Problème du bandit manchot à k bras

Ce problème, présenté pour la première fois en 1933 Thompson (1933), et beaucoup étudié depuis Robbins (1985) Bellman (1956), peut être défini de la façon suivante : à chaque pas de temps il faut choisir entre k actions chacune générant une récompense suivant une loi de probabilité stationnaire. L'objectif étant de maximiser la récompense totale après une période de temps, par exemple 1000 itérations.

Chaque action a une récompense moyenne, appelée valeur de l'action. On a donc :

$$q_*(a) = \mathbb{E}[R_t|A_t = a] \tag{4.10}$$

avec :

— $q_*(a)$: la récompense attendue ;

— R_t : la récompense à l'instant t ;

— A_t : l'action choisit à l'instant t.

En sachant la valeur de chaque action, le problème serait trivial. Ainsi, ces valeurs ne sont pas connues et le but est de les estimer. Ainsi la valeur estimée à l'instant t est notée $Q_t(a)$ et nous cherchons à ce qu'elle soit proche de $q_*(a)$. La ou les actions qui ont la valeur estimée la plus élevée peuvent être appelées actions avides ($greedy - actions$) et choisir l'une d'ellee c'est exploiter les connaissances actuelles. Le terme *greedy* est également utilisé dans les domaines des heuristiques Pearl (1984) et représente le même concept, à savoir, choisir la solution qui semble la meilleure immédiatement. À l'inverse, choisir une action non avide ($nongreedy - action$), c'est explorer, afin d'améliorer nos estimations de ces solutions. Ainsi, l'exploitation est la chose à faire pour maximiser la récompense sur le court terme, mais l'exploration peut permettre de meilleurs résultats sur le long terme.

Ainsi, par la suite, nous nous intéresserons à la façon de gérer le compromis exploration/exploitation.

4.2.2 Méthodes basées sur les valeurs d'actions

Une façon simple d'estimer la valeur d'une action est la suivante :

$$Q_t(a) = \frac{somme\ des\ récompenses\ quand\ a\ est\ choisie}{nombre\ de\ fois\ où\ a\ est\ choisie} \tag{4.11}$$

Si le dénominateur vaut 0, on définit $Q_t(a)$ à une valeur par défaut, en général 0. À l'inverse, quand le dénominateur tend vers l'infini, $Q_t(a)$ converge vers $q_*(a)$. Ce type de méthode est appelé méthode de la moyenne de l'échantillon (*sample-average methods*).

Une règle de sélection simple est de sélectionner une des actions avec la plus grande valeur estimée Thathachar and Sastry (1985), étant ainsi une action avide. Ainsi, la méthode de sélection avide ($greedy - selection$) est notée :

$$A_t \doteq argmax_a Q_t(a) \tag{4.12}$$

Cette méthode permet une exploitation maximum des connaissances, mais donc, n'explore que très peu

l'espaces des possibilités. Ainsi, elle est souvent couplée à une petite probabilité ϵ de sélection aléatoire. On appelle cette méthode ϵ-greedy Watkins (1989).

Exemple théorique d'un bandit à 10 bras La figure 4.2 montre la répartition des récompenses en fonction des actions sur un test de bandit manchot à 10 bras. Chacune suit une loi normale centrée sur $q_*(a)$ et de variance 1. Pour résoudre ce problème d'apprentissage par renforcement, nous utilisons les solutions proposées précédemment. Ainsi, la figure 4.3 montre les résultats en utilisant une méthode de sélection ϵ-greedy. Chaque courbe représentant les résultats pour une valeur différente d'ϵ.

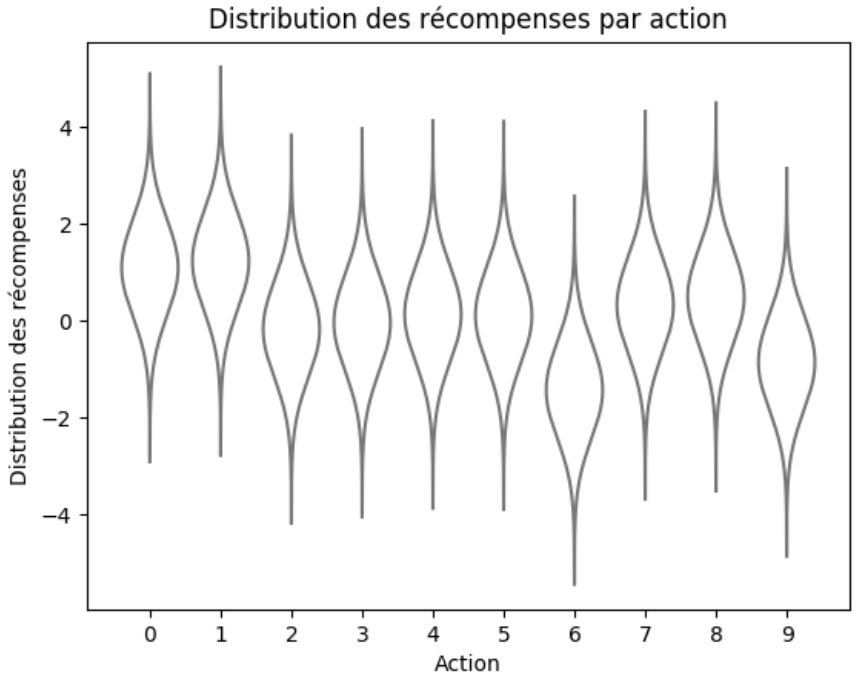
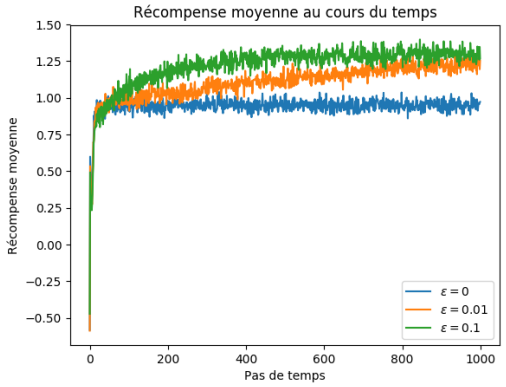


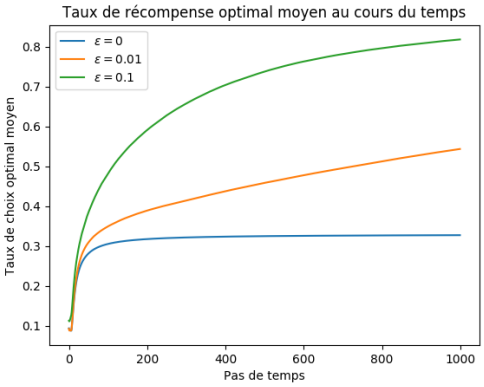
FIGURE 4.2 – Exemple de problème du bandit à 10 bras. La valeur $q_*(a)$ de chaque action suit une loi normale de variance 1

Cette méthode est particulièrement importante, car elle est utilisée dans la majorité des processus d'apprentissage par renforcement. L'avantage de la méthode ϵ-greedy n'est valable que pour une certaine variance dans la fonction d'évaluation. En effet, si la variance était égale à 0, une simple méthode avide aurait été plus efficace. Dans le cas où la variance aurait été plus élevée par contre, celle-ci aurait été plus efficace mais aurait pris beaucoup plus de temps à converger.

Quand un grand nombre d'itérations est effectué, le calcul de la moyenne (espérance) peut poser un problème de temps de calcul et d'espace mémoire. En effet, la méthode triviale nécessite le stockage de toutes les valeurs de récompenses précédentes. Pour palier cela, une formule plus efficace peut être



(a) Récompense moyenne au cours du temps



(b) Taux de choix de l'action optimale au cours du temps

FIGURE 4.3 – Résultats en faisant varier ϵ

utilisée :

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i \tag{4.13}$$

$$Q_{n+1} = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \tag{4.14}$$

$$Q_{n+1} = \frac{1}{n}(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i) \tag{4.15}$$

$$Q_{n+1} = \frac{1}{n}(n-1)Q_n \tag{4.16}$$

$$Q_{n+1} = \frac{1}{n}(R_n + nQ_n - Q_n) \tag{4.17}$$

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n] \tag{4.18}$$

Ainsi, Q_{n+1} ne dépend plus que de Q_n et de R_n.

On peut ainsi proposer l'algorithme 7.

La méthode proposée jusque-là peut se montrer efficace en environnement stationnaire, c'est-à-dire, un environnement où les probabilités de récompense ne changent pas au cours du temps. En revanche, en environnement non stationnaire, il peut être intéressant de pondérer les récompenses pour donner plus de poids aux plus récentes.

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n] \tag{4.19}$$

Algorithme 7 Un algorithme de bandit simple

initialise $\forall a \in A, Q(a), N(a) = 0$
repeat
  si $rand() < \epsilon$ **alors**
    $a = random\ action \in A$
  sinon
    $a = argmax_a Q(a)$
  fin si
  $R = bandit(a)$
  $N(a) = N(a) + 1$
  $Q(a) = Q(a) + \frac{1}{N(a)}[R - Q(a)]$
until forever

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-1} R_i \tag{4.20}$$

α constant $\in]0; 1]$

D'autres méthodes de gestion de ce problème sont détaillées en annexe 6.2.1.

Une variante de ce problème permet de considérer un état, appelé contexte, influençant les récompenses. On parle alors de bandit manchot contextuel et on a alors une version s'approchant des problèmes d'apprentissage par renforcement à proprement parler.

4.2.3 Notions de bandit manchot contextuel

Le problème du bandit à plusieurs bras demande de produire une action en n'utilisant aucune information sur l'état de l'environnement (contexte). Par exemple, si un bandit à plusieurs bras est utilisé pour choisir d'afficher des publicités à l'utilisateur d'un site web, la même décision aléatoire sera prise même en connaissant les préférences de l'utilisateur. Le bandit contextuel étend le modèle en rendant la décision conditionnelle à l'état de l'environnement.

La figure 4.4 montre les différences entre un problème de bandit manchot, un problème de bandit manchot contextuel et un problème d'apprentissage par renforcement. Dans le premier cas, aucun état n'est considéré et l'apprentissage ne se fait que sur la valeur des actions. Dans le bandit contextuel, l'agent doit prendre en compte un état courant car celui-ci agit sur l'environnement et donc sur la récompense. Dans un problème d'apprentissage par renforcement, la principale différence est que la récompense n'est pas forcément délivrée pour chaque action. Celle-ci peut donc agir sur l'état mais la récompense peut dépendre d'un ensemble d'action.

Le bandit contextuel peut être un bon moyen de représenter certains problèmes où le but est de rapidement estimer l'état du système. Par exemple, en considérant que l'agent connaît la fonction de récompense en fonction de l'état réel du système, celui-ci peut chercher à gérer le compromis entre exploration et approfondissement afin d'estimer au mieux l'état du système et donc maximiser ses récompenses au cours du temps.

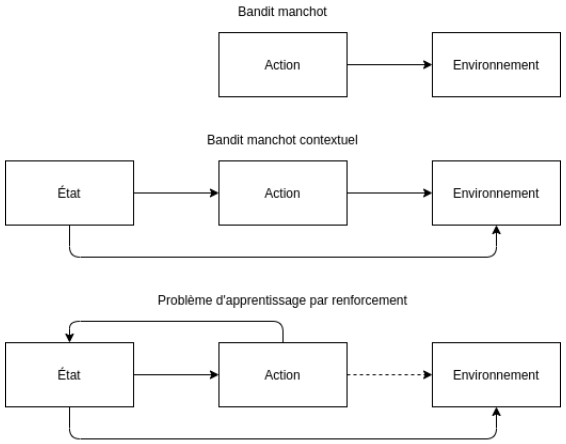


FIGURE 4.4 – Différences entre problèmes du bandit manchot et apprentissage par renforcement

On peut penser à l'application où un pêcheur doit choisir où pêcher. Dans un premier temps, il devra explorer pour estimer la qualité de chaque zone. Les pêches étant incertaines, il devra maintenir et mettre à jour des estimations de lois de probabilité de ses prises, et donc gains, par zone. De plus, son activité, et celle des autres pêcheurs influenceront l'état réel de chaque zone au cours du temps. On est alors en environnement non stationnaire et la temporalité des observations devra donc être prise en compte dans l'estimation des lois de probabilités.

Nous allons nous intéresser à cette application. Dans un premier temps, nous définirons l'environnement à modéliser et les différentes étapes de sa conception et de son implémentation. Une fois le cadre formel défini et les environnements de simulation générés, nous pourrons définir les problématiques d'apprentissage et comment les résoudre.

4.3 Modélisation d'une pêcherie sous forme de processus de décision markovien

Nous allons chercher à représenter une activité de pêche aux petits métiers proche de ce que nous avons en Corse. À savoir :

— un ensemble de pêcheurs exploitant divers stocks ;

— des sorties ne durant que la journée ;

— une grande diversité des prix et de l'importance des espèces dans l'économie ;

— des gestionnaires et décideurs, cherchant à étudier les espèces et améliorer leur exploitation pour atteindre un compromis économie/écologie durable.

Nos échanges avec les gestionnaires et exploitants des différents projets menés en collaborations, nous ont permis d'avoir les informations suivantes :

— environ 120 sorties par an et par pêcheur sont enregistrées.

— ils ne se concentrent que rarement sur une espèce spécifique.

— les prix de vente sont très variables d'une espèce à l'autre.

— elles peuvent donc avoir des importances très différentes dans l'économie locale.

La figure 4.5 montre le prix moyen par Kg des espèces principalement exploitées en Corse. Les espèces manquantes sont moyennées dans "others". La répartition des prix n'est pas uniforme. La plupart des espèces ont un prix de vente faible alors que quelques-unes sont nettement plus intéressantes commercialement.

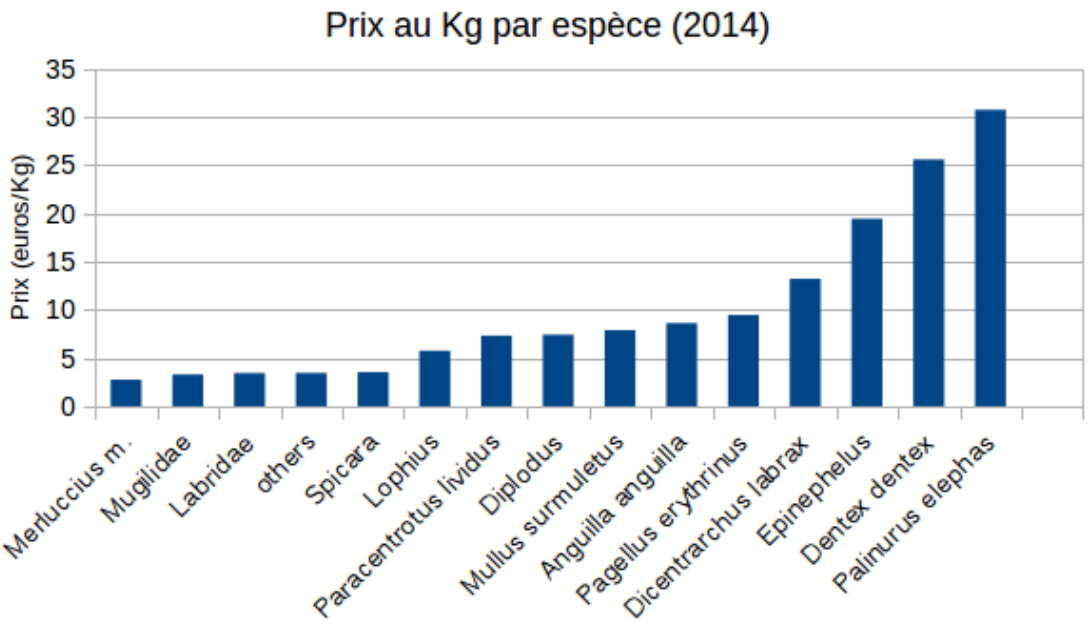


FIGURE 4.5 – Prix (euros/Kg) de chaque espèce en 2014 tiré de Sea Around Us (2020)

En couplant cela avec les captures, on obtient la répartition de l'importance économique de chaque espèce présentée en figure 4.6. On peut remarquer 3 faits marquants :

— les 2 espèces dont les prix sont les plus élevés (Dentex dentex et Palinurus elephas) sont les 2 individuelles dont l'importance économique est la plus grande.

— la classe "others" est celle avec la plus grande importance économique, ce qui signifie que malgré leur faible prix, l'économie de la pêche Corse est majoritairement constitué d'un ensemble d'espèces de moindre importance.

— les espèces arrivant 3ème et 4ème dans l'ordre des prix (Epinephelus et Dicentrachus labrax) ne représentent qu'une très faible proportion de l'importance économique. Prix et importance économique ne sont donc pas forcément corrélés.

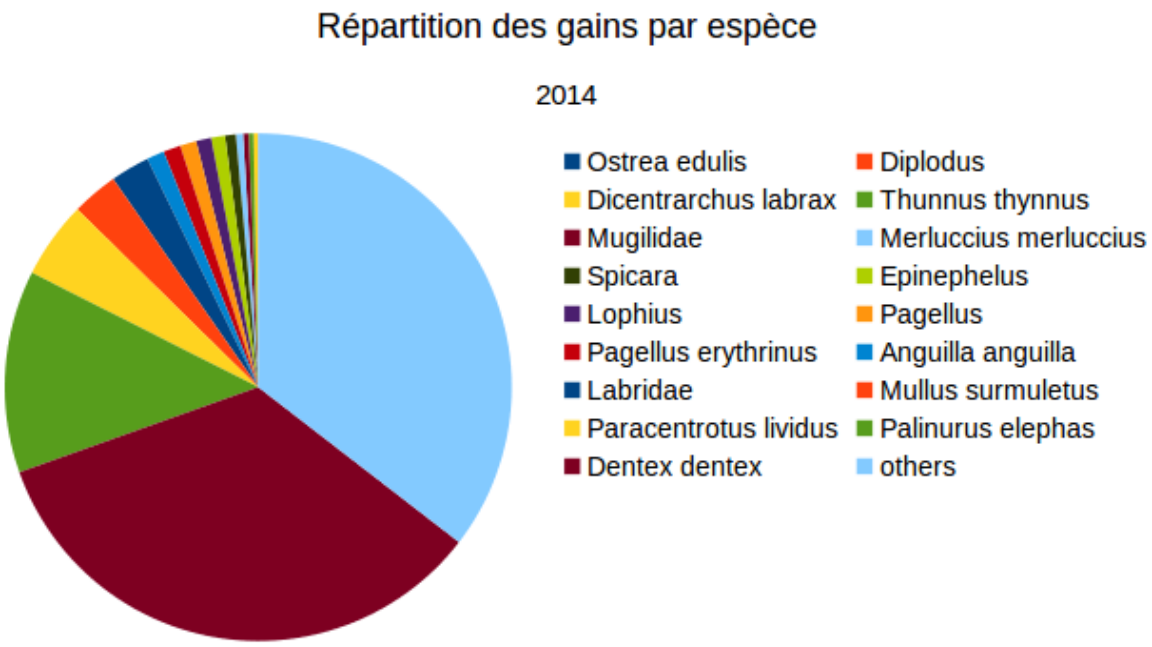


FIGURE 4.6 – Importance économique de chaque espèce

Deux entités agiront sur l'environnement, les décideurs et les pêcheurs. Les premiers devront définir le niveau d'exploitation à appliquer sur chaque espèce. Celui-ci prendra la forme de quotas que les pêcheurs devront respecter. Les pêcheurs, eux, chercheront à maximiser leurs gains tout en respectant les quotas. Chacun prenant ses propres décisions de manière décentralisée.

4.3.1 Formalisation de l'environnement

Dans cette partie nous allons formaliser notre vision de l'environnement en général et détaillerons les spécificités pour les applications basées sur les bandits. Ce type d'environnement nous servira pour l'ensemble des expériences d'apprentissage.

Le problème sera basé sur un système multi-agent, chaque agent représentant un bateau interagissant avec son environnement et avec peu voire pas de connaissance des actions des autres. Chacun ne sera guidé que par ses observations, éventuellement des "on-dit", représentés par les observations probabilistes que chacun aura des activités des autres, et les recommandations d'exploitation des décideurs.

La figure 4.7 montre une simplification du déroulement d'une simulation, dans le cas d'utilisation de l'estimation d'un contexte, appelée croyance en référence aux *belief-MDP*. Elle suit plusieurs étapes.

1. Les décideurs calculent les quotas à appliquer en fonction de leurs croyances et les transmettent aux pêcheurs. Dans le cas d'un problème de bandit manchot, nous utiliserons les valeurs optimales théorique en nous basant sur les variables cachées. Cela nous permettra de montrer les limites de l'approche même en cas de gestion optimale théorique de la part des décideurs.

2. Chaque pêcheur détermine dans quelle zone pêcher.
3. Les efforts associés sont appliqués dans les zones correspondantes :
   (a) Calcul des captures correspondantes à chaque pêcheur pour chaque poisson.
   (b) La biomasse de chaque poisson est mise à jour
4. Les observations brutes sûres sont transmises au pêcheur correspondant.
5. Les observations brutes incertaines sont transmises suivant une certaine probabilité.
6. Chaque pêcheur met à jour ses croyances b_x quant à la zone x dans laquelle il a pêché en suivant $\tau(b_x, o, a)$.
7. Si on est en fin d'année, les décideurs mettent à jour leurs croyances et retour à l'étape (1). Sinon, retour à l'étape (2).

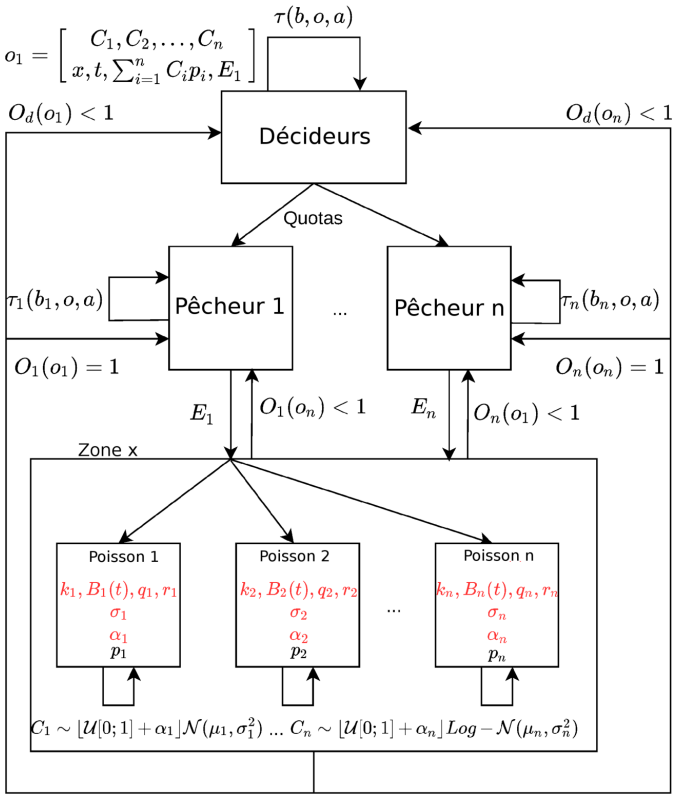


FIGURE 4.7 – Système global, en rouge les paramètres cachés.

L'environnement est constitué de m zones que n pêcheurs exploiteront. On considérera l espèces dans toute la pêcherie. Chaque zone sera constituée de plusieurs espèces, potentiellement toutes, chacune avec une portion de la biomasse totale de l'espèce. Chaque espèce possède un prix de vente, p, indépendant de la zone de pêche et fixe au cours du temps. À l'avenir, nous espérons inclure un modèle économique plus pertinent.

Pour toutes les applications futures, l'espace des actions de chaque pêcheur sera le même, à savoir un vecteur de m réels, chacun représentant la probabilité d'aller pêcher dans une zone : $A_i : \mathbf{R}^m$

Chaque pêcheur exercera un effort E_i dans la zone qu'il a sélectionnée. Les dynamiques des espèces suivront le modèle de Graham-Schaefer à la particularité près que la fonction de capture sera stochastique pour mieux correspondre à la réalité du métier. Les différents paramètres de chaque espèce représenteront les états cachés, parfaitement inconnus des pêcheurs.

Le modèle de Graham-Schaefer modélise les prises sur un intervalle de temps relativement important et pour un ensemble de pêcheurs. Il est donc pertinent de considérer que la loi de probabilité des captures totales est en fait la somme d'un très grand nombre de lois de probabilité moins importantes. Ainsi, d'après le théorème central limite, on peut estimer que celle-ci suit une loi normale centrée sur $\mu = C(t) = qE_{total}B(t)$.

Dans un premier temps et faute d'information, nous avions fait l'hypothèse, facilement modifiable si des informations contradictoires venaient à apparaître, que les captures individuelles par unité de temps suivent également une loi normale.

Importance du pas de temps Une première proposition a été de définir les prises pour chaque pêcheur par unité de temps en suivant la loi normale équivalente des prises de l'espèce avec $\mu_i = \mu/n, \sigma_i = \sigma/\sqrt{n}$ avec n, le nombre de sorties correspondantes à l'effort unitaire. Ce qui donnerait dans notre cas :

$$\mu_i = \frac{qE_{unitaire}B(t)}{\frac{E_{unitaire}}{E_{individuel}}} = qE_{individuel}B(t) \tag{4.21}$$

$$\sigma_i = \frac{\sigma}{\sqrt{\frac{E_{unitaire}}{E_{individuel}}}} = \frac{\sigma}{\sqrt{\frac{1}{E_{individuel}}}} = \sigma\sqrt{E_{individuel}} \tag{4.22}$$

Or les prises ne peuvent pas être négatives en réalité, ce qui est possible avec cette loi de probabilité. Se contenter de ramener les résultats négatifs à 0 peut grandement influencer la validité de la loi de probabilité. D'où l'importance du pas de temps. Il doit être assez grand pour que $P(C < 0) \sim 0$ afin d'éviter ce phénomène le plus possible et ne pas affecter la loi totale. Mais il doit également être assez petit pour pouvoir considérer le théorème central limite applicable. Empiriquement, poser $\Delta t = semaine$ semblait être un bon compromis, mais nous reviendrons sur ce point par la suite.

De plus, pour ne pas biaiser les résultats, il ne faut pas que l'ordre des agents, c'est-à-dire lequel va exécuter son action en premier, influence leur résultat individuel Ferber (1997). Nous allons donc considérer que les pêches se font en parallèle : chaque pêcheur agit sur la biomasse de l'instant t et celle-ci est mise à jour une fois que tous les pêcheurs ont exécuté leurs actions. Cela revient à considérer $\mathbb{E}(B(t+1)) = B(t) + r(1 - \frac{B(t)}{k})B(t) - qB(t)\sum_{i=0}^{nPeche} E_i$ avec $\mathbb{E}$ l'espérance due aux lois de probabilité.

Amélioration de la précision Ces compromis semblent cohérents, mais nous nous sommes rendu compte qu'ils conduisaient à peu de données observées et une variabilité des observations moindre qu'en réalité. Nous avons donc décidé de légèrement changer notre façon de voir les choses. En effet, les prises

se mesurent en unité de masse, mais cela est équivalent au produit du nombre de captures par leur masse. Or, les données du projet MoonFish nous permettent de déterminer des estimations des lois de probabilité des prises par espèce. Bien que celles-ci ne soient pas assez précises pour une utilisation directe, elles nous donnent d'importantes indications sur la tendance de cette variable aléatoire. Nous pouvons donc les utiliser dans nos simulations et apprentissages.

En séparant les données de prises par espèce, on se rend compte qu'il y a énormément de valeurs à 0. En effet, comme la pêche Corse n'est pas concentrée sur une espèce en particulier, mais touche beaucoup d'espèces distinctes, il est très rare d'avoir des prises de toutes les espèces exploitées en même temps. Ce grand nombre de 0 a faussé le calage de lois de probabilité classiques. Nous avons donc fait le choix de les retirer temporairement des données tout en gardant en mémoire la proportion qu'ils représentaient. Pour continuer sur des lois de probabilité positives classiques, nous avons enlevé 1 à chaque donnée permettant d'obtenir de nouvelles valeurs à 0. On se rend compte alors que les prises de chaque espèce semblent suivre une loi log-normale. Celle-ci étant cependant une loi continue, les résultats pouvaient être légèrement différents des données, mais en ramenant toutes les probabilités d'un intervalle unitaire à sa valeur basse (équation 4.23), les résultats étaient plus que concluant.

$$\forall x \in \mathbf{N} : p(x) = \int_{\lfloor x \rfloor}^{\lceil x \rceil} Log - \mathcal{N}(\mu_x, \sigma_x^2) \tag{4.23}$$

On définit alors α, la proportion de valeurs différentes de 0, et β le décalage à effectuer pour retrouver les données réelles. En pratique on utilisera toujours $\beta = 1$.

En posant $\mathcal{X} \sim Log - \mathcal{N}(\mu, \sigma^2)$ on a alors la loi des captures unitaires par sortie C_u :

$$C_u \sim \lfloor \mathcal{U}(0,1) + \alpha \rfloor \lfloor \mathcal{X} + \beta \rfloor \tag{4.24}$$

De plus, d'après Froese et al. (2014), la probabilité de trouver des poissons d'une certaine taille, masse, dépend directement d'une loi normale de paramètres variables pour chaque espèce. Ainsi, les captures de chaque espèce dépendent non seulement des paramètres de cette loi, mais également de l'engin utilisé (un filet trop gros ne permettra pas de pêcher les petits poissons par exemple). Dans cette étude, nous ne prenons pas en compte les engins, car nous manquons d'informations. Nous considérerons donc que la probabilité de capturer des individus est équivalente peu importe son poids. Pour ne pas biaiser l'apprentissage en utilisant les paramètres correspondant à nos espèces, nous utiliserons des paramètres générés aléatoirement dans une plage de valeurs cohérentes, déterminée via nos analyses. Ainsi, la loi de probabilité des masses prélevées par sortie est donnée par la relation suivante :

$$M_u \sim \mathcal{N}(\mu_m, \sigma_m^2) \times C_u \tag{4.25}$$

D'après le modèle de Graham-Schaefer (voir section 2.1.1) on considère l'espérance massique des captures totales sur une année t : $\mathbb{E}(M_{total}) = qE_{total}B(t)$. On a donc $\mathbb{E}(M_u) = \mathbb{E}(M_{total}) \times \frac{E_{individuel}}{E_{total}}$ Ainsi, nous devons déterminer σ_x^2 en fonction de $E_{individuel}$.

Sachant que l'espérance d'une loi log-normale est donnée par $\mathbb{E}(\mathcal{X}) = e^{\mu_x + \sigma_x^2/2}$, on peut approximer

$\mathbb{E}(C_u)$ par :

$$\mathbb{E}(C_u) = \alpha(\mathbb{E}\mathcal{X} + \beta)) \tag{4.26}$$

$$\mathbb{E}(C_u) = \alpha(e^{\mu_x + \sigma_x^2/2} + \beta) \tag{4.27}$$

Soit après résolution :

$$\sigma_x^2 = 2\left(ln\left(\frac{\mathbb{E}(C_u) - \alpha\beta}{\alpha}\right) - \mu_x\right) \tag{4.28}$$

De plus, d'après équation 4.25 :

$$\mathbb{E}(M_u) = \mathbb{E}\mathcal{N}(\mu_m, \sigma_m^2)\mathbb{E}(C_u) \tag{4.29}$$

$$\mathbb{E}(M_u) = \mu_m\mathbb{E}(C_u) \tag{4.30}$$

Donc :

$$\mathbb{E}(C_u) = \frac{\mathbb{E}(M_{total})E_{individuel}}{\mu_m E_{total}} \tag{4.31}$$

$$\sigma_x^2 = 2\left(ln\left(\frac{\mathbb{E}(M_{total}) \times \frac{E_{individuel}}{\mu_m E_{total}} - \alpha\beta)}{\alpha}\right) - \mu_x\right) \tag{4.32}$$

L'utilisation de plusieurs lois de probabilité différentes permet à l'agent apprenant de s'adapter à toutes situations réelles éventuelles. Les lois de probabilité finales utilisées sont données dans le tableau 4.1 :

Loi	σ^2	μ	Captures
Normal	$[0.05\,;0.15]\mu$	$\frac{qEB(t)}{\alpha}$	$\lfloor\mathcal{U}[0;1] + \alpha\rfloor\mathcal{N}(\mu, \sigma^2)$
Log-$\mathcal{N}$	$[0.5\,;2]$	$ln(\frac{qEB(t)}{\alpha}) - \frac{\sigma^2}{2}$	$\lfloor\mathcal{U}[0;1] + \alpha\rfloor Log - \mathcal{N}(\mu, \sigma^2)$
	β_β	α_β	
Beta	$[0.5\,;10]$	$\frac{\frac{qEB(t)}{\alpha C_{max}}\beta_\beta}{1-\frac{qEB(t)}{\alpha C_{max}}}$	$\lfloor\mathcal{U}[0;1] + \alpha\rfloor\beta(\alpha_\beta, \beta_\beta)C_{max}$

TABLE 4.1 – Paramètres des lois de probabilité des captures

Transitions et observations Les transitions des états cachés seront simplement l'évolution classique des populations représentées par le modèle de Graham-Schaefer, soumises à une activité de pêche.

Nous allons à présent parler de ce que nous appelons l'espace des observations brutes. En effet, l'espace des observations pourra varier en fonction des applications. En revanche, celui-ci sera toujours basé sur des observations brutes, générés par l'environnement lors de la transition. L'espace des observations brutes peut être décomposé en 2 parties, l'ensemble des observations brutes sûres, correspondant à ce que le pêcheur voit de lui-même à chaque pêche, et l'ensemble des observations brutes probabilistes, correspondant à ce qu'il peut observer ou entendre dire. Les observations brutes probabilistes surviendront avec une probabilité que nous donnerons explicitement lors de chaque application. Elles sont là pour

prendre en compte les "on-dit" et ce que le pêcheur pourrait voir des autres par hasard dans la réalité. Ainsi, elles auront exactement la même forme que les observations brutes sûres, à savoir :

— $\mathbf{R}^l$, les captures de chaque espèce de la pêcherie (constante = 0 si l'espèce n'est pas présente dans la zone) ;

— l'identifiant de la zone où la pêche a eu lieu, permettant de gérer un historique dans certains cas ;

— un indicateur temporel, permettant de prendre en compte le temps écoulé depuis le relevé. Cela permettra, en plus de gérer l'historique, de prendre en compte la dégradation de l'information au cours du temps. En effet, les espèces étant exploitées, nous sommes dans un environnement non stationnaire Papoudakis et al. (2019). Une information plus ancienne doit donc être considérée moins fiable qu'une information récente.

— les gains engendrés en euros.

— l'effort de pêche, même si on le considérera fixe par intervalles de temps pour chaque pêcheur, le stocker permettra de comparer les résultats des différents pêcheurs dans certains cas.

4.3.2 Génération automatique des environnements

De par la complexité de l'environnement, une simple génération par répartition uniforme n'est pas envisageable. De plus, même en respectant la répartition des prix par espèce, il est très difficile de mathématiquement définir des règles à appliquer pour obtenir une répartition des gains cohérente selon la figure 4.6. De plus, une part d'aléatoire doit, dans tous les cas, être maintenue afin que les agents apprennent à gérer des situations variées et ainsi éviter le sur-apprentissage. Il semble donc particulièrement approprié d'effectuer la génération des environnements par optimisation nous permettant ainsi de nous affranchir d'une certaine complexité mathématique, mais également de maintenir une part d'aléatoire nécessaire au bon déroulement de l'apprentissage.

Déroulement général Bien qu'un grand nombre d'espèces ne poserait pas de problèmes à l'optimisation, nous avons fait le choix de limiter ce nombre à 10 (9 espèces individuelles et une dernière représentant l'agrégation de toutes les autres) pour ne pas surcharger les espaces d'états de l'apprentissage. De plus, nous nous intéressons à 6 espèces d'intérêt primordial, ce nombre semble donc être un compromis acceptable entre la quantité minimale dont nous avons besoin et l'évolutivité potentielle de la situation.

Nous allons donc, dans un premier temps, générer aléatoirement des dynamiques pour chacune de ces espèces dans leur globalité, c'est-à-dire, indépendamment des zones. En réalité, les espèces sont réparties dans des zones, mais les quotas ne différencient les prises par zone que dans les cas des réserves. Elles font donc toutes partie du stock global de cette espèce.

Pour diviser le stock global en plusieurs zones, nous devons nous référer au modèle de Graham-Schaefer. Le but étant, à exploitation équivalente, de conserver la dynamique globale.

$$\frac{dB(t)}{dt} = r\left(1 - \frac{B(t)}{k}\right)B(t) - qEB(t) \tag{4.33}$$

L'étude de cette fonction montre qu'en divisant la biomasse $B(t)$, il faut également diviser la biomasse à l'équilibre k. Notons $B_i(t)$ la biomasse présente dans la zone i tel que $\sum_{i=0}^{n} B_i(t) = B(t)$:

$$\frac{dB_i(t)}{dt} = r\left(1 - \frac{B_i(t)}{k \times \frac{B_i(t)}{B(t)}}\right) B_i(t) - qEB_i(t) \tag{4.34}$$

$$\frac{dB(t)}{dt} = \frac{d\sum_{i=0}^{n} B_i(t)}{dt} \tag{4.35}$$

$$\frac{dB(t)}{dt} = r\left(1 - \frac{\sum_{i=0}^{n} B_i(t)}{k \times \frac{\sum_{i=0}^{n} B_i(t)}{B(t)}}\right) \sum_{i=0}^{n} B_i(t) - qE\sum_{i=0}^{n} B_i(t) \tag{4.36}$$

En simplifiant on retrouve bien l'équation 4.33

Bien qu'il soit plus simple de déterminer les prises optimales par espèce avant la répartition dans les zones, le fait que l'effort de pêche s'applique sur toutes les espèces de la zone sélectionnée rend cela impossible. En effet, il peut être impossible d'atteindre un niveau d'exploitation optimal si l'exploitation de certaines zones entraîne la surexploitation, voir la destruction d'autres espèces. La répartition doit donc soit être un préalable à l'optimisation, soit en être un paramètre.

Nous devons alors distinguer 2 parties :

1. la détermination de l'exploitation optimale durable en fonction de l'environnement ;
2. la détermination des paramètres de l'environnement. Ce sont les paramètres qui permettent, à partir des niveaux d'exploitation, de juger si l'environnement est cohérent avec les observations réelles.

En effet, les observations réelles ont été faites alors que l'exploitation était déjà effective. Bien qu'elle ne soit pas nécessairement optimale, les pêcheurs et décideurs connaissant leur métier, elle n'est pas non plus aléatoire. Nous ne pouvons donc pas juger un environnement par une exploitation aléatoire. C'est pourquoi, pour chaque environnement proposé, son exploitation optimale durable doit être déterminée au moins pour quelques années avant de pouvoir l'évaluer. Une fois qu'un environnement sera jugé comme cohérent, nous pourrons alors optimiser son exploitation sur un horizon h plus long et stocker ces informations pour évaluation (reward) des processus d'apprentissage.

Une fois l'exploitation optimale durable déterminée, le but va être, à partir de l'ensemble des captures obtenues durant l'année, de déterminer les meilleurs prix de chaque espèce afin d'obtenir la répartition économique souhaitée. Cela peut ne pas être possible avec les dynamiques / répartitions proposées. Si cette optimisation échoue, sa meilleure évaluation servira d'évaluation à une optimisation de niveau supérieur. Celle-ci devra modifier la répartition des espèces au sein des zones avant de recommencer. De même, si celle-ci échoue, les dynamiques seront modifiées.

Détermination de l'exploitation optimale durable théorique Connaissant la dynamique globale de chaque espèce indépendamment des zones, nous pouvons facilement calculer le Most Sustainable Yield (MSY) c'est-à-dire le niveau d'exploitation optimale par espèce dans la pêcherie.

$$E_{msy} = r/(2q) \Rightarrow C_{msy} = rk/4 \tag{4.37}$$

Il est cependant à noter que C_{msy} ne représente pas les captures qui seront obtenues à l'instant t mais les captures qui seront obtenus une fois atteint le niveau d'exploitation optimale durable. Cela est illustré par la figure 4.8 représentant l'évolution des biomasses et captures soumises à E_{msy} au cours du temps. $Binferieur$ et $Cinferieur$ représentent les biomasses et captures quand on commence à une biomasse inférieure au niveau de stock optimal durable ($Bmsy$). De même, $Bsuperieur$ et $Csuperieur$ représentent les biomasses et captures quand on commence à une biomasse supérieure à ce niveau. Dans les 2 cas, il y a convergence finale vers $B_{msy} = 500$ et $C_{msy} = 125$.

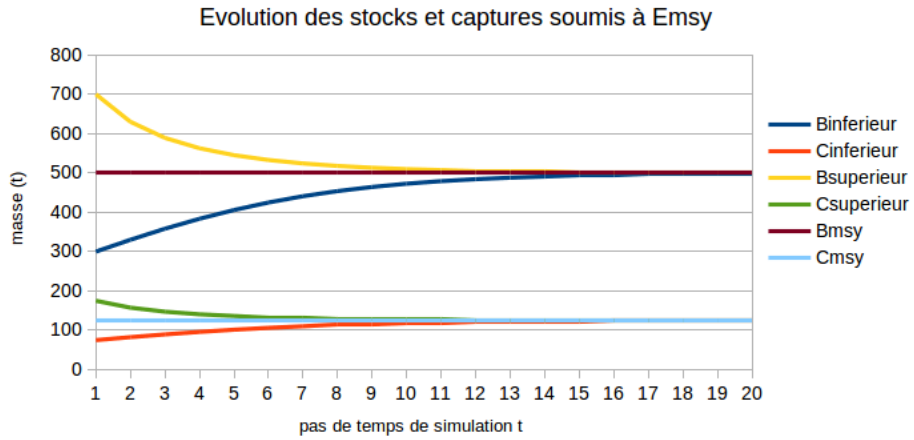


FIGURE 4.8 – Évolution des stocks soumis à E_{msy}

Comme expliqué précédemment, la répartition des stocks dans les différentes zones peut ne pas permettre d'obtenir ce niveau d'exploitation optimal. Cette optimisation a pour but de s'en rapprocher le plus possible. Pour cela, nous allons déterminer les efforts à appliquer par zone. Pour juger de la qualité des résultats, deux solutions s'offrent à nous :

— calculer la somme des efforts équivalents par espèce ($E = \sum_i E_{equivalent_i} = \sum_i E_i * B_i(t)/B(t)$, avec i, l'identifiant de la zone) et le comparer à E_{msy} pour chaque espèce ;

— calculer les captures engendrées par les efforts et les comparer aux captures espérées avec E_{msy} à l'instant t.

Les deux options sont absolument équivalentes, mais la 2ème est plus simple algorithmiquement et permet d'obtenir des nombres plus grands, accentuant ainsi la mesure des erreurs et facilitant donc

l'optimisation. Nous utiliserons donc la fonction d'évaluation à minimiser suivante :

$$f(x) = \sum_{i=0}^{nespece} \left(C_{imsy} - \sum_{j=0}^{nzone} C_{i,j} \right)^2 \tag{4.38}$$

Détermination des paramètres de l'environnement Ici, nous allons chercher à déterminer les prix, compris entre 1 et 30 euros/Kg, de chaque espèce afin de respecter les conditions de cohérence de l'environnement. Celles-ci prendront la forme des contraintes suivantes déterminées empiriquement à partir des données présentées en figures 4.5 et 4.6 :

— $g1$: au moins une espèce a un prix élevé ($p_i > 3\overline{p}$) avec $\overline{p}$ la moyenne des prix des espèces exploitées, et représente un gros pourcentage des gains de l'exploitation (au moins 20%) ;

— $g2$: au moins une espèce a un prix élevé ($p_i > 20$) et ne représente qu'un faible pourcentage des gains, maximum 10% ;

— $g3$: la classe *others* regroupant toutes les espèces de moindre importance, doit avoir un prix faible $p_{others} \leq \frac{1}{2}\overline{p}$ mais être la classe rapportant le plus ;

— $g4$: l'écart type des prix est élevé.

Elles sont représentées mathématiquement par :

$$g_1 : \exists i | p_i > 3 \times \overline{p} \wedge \frac{Ctotal_i \times p_i}{\sum_{j=0}^{n} Ctotal_j \times p_j} > \frac{2}{n} \tag{4.39}$$

$$g_2 : \exists i | p_i > 3 \times \overline{p} \wedge \frac{Ctotal_i \times p_i}{\sum_{j=0}^{n} Ctotal_j \times p_j} < \frac{1}{n} \tag{4.40}$$

$$g_3 : \exists i | p_i < \frac{1}{2}\overline{p} \wedge \frac{Ctotal_i \times p_i}{\sum_{j=0}^{n} Ctotal_j \times p_j} > 0.35 \tag{4.41}$$

$$g_4 : \sqrt{\frac{1}{n}\sum_{i=0}^{n}(p_i - \overline{p})^2} > 5 \tag{4.42}$$

Une solution respectant toutes ces contraintes sera considérée acceptable et pourra être enregistrée pour l'apprentissage. Mais en plus de cela, les différentes solutions seront évaluées sur un indicateur de la répartition de l'espérance des gains par espèce.

$$f : \sqrt{\frac{1}{n}\sum_{i=0}^{n}\left(\frac{Ctotal_i \times p_i}{\sum_{j=0}^{n} Ctotal_j \times p_j}\right)^2} \tag{4.43}$$

La solution respectant les contraintes et ayant la meilleure évaluation sera alors sauvegardée. La valeur de f sera également stockée afin de comparer les résultats des apprentissages en fonction de la difficulté. De même, les niveaux de biomasses théoriques optimaux seront stockés et pourront être comparés aux biomasses initiales afin de définir si une pêcherie est en difficulté à l'instant initial.

L'optimisation de la section 3.1.1 sera finalement réutilisée afin de calculer les captures théoriques et les gains optimaux sur l'horizon h. Ces informations seront réutilisées lors du calcul des récompenses des différentes applications via apprentissage.

Une fois suffisamment d'environnements générés, nous pouvons nous intéresser aux expériences d'apprentissage.

4.4 Modélisation de notre système sous forme de bandit manchot contextuel

4.4.1 Déroulement d'une simulation

Une simulation représente 10 ans de l'exploitation d'un environnement. Cette durée est fixée pour laisser le temps à de mauvaise stratégie d'impacter significativement la ressource ou au contraire, à une bonne de l'exploiter de manière optimale.

Le déroulement est visible en algorithme 8.

Pour commencer l'environnement est initialisé. Cela se fait en sélectionnant un environnement aléatoire parmi tous ceux générés avec la procédure présentée précédemment. Toutes les espèces et leurs lois de probabilité associées sont initialisées et $t = 0$.

Pour chaque année, à chaque pas de temps, $nbExploitationDays = 150$, chaque pêcheur, $nbFishermen = 50$, doit déterminer l'action à effectuer (ligne 5), $a_k = \pi_k(b_k$.

Une fois toutes les actions déterminées, un pas est effectué en fonction de l'ensemble des actions. Générant ainsi les récompenses ainsi que l'ensemble des observations pour chaque pêcheur (ligne 7). Pour ne pas surcharger l'algorithme, nous ne détaillons pas la gestion de la probabilité que chaque pêcheur a d'observer le comportement des autres. Elle est gérée par la fonction $step(action)$ ligne 7. Celle-ci peut être faite par simple boucle sur l'ensemble des observations des pêcheurs et par tirage aléatoire en fonction de la probabilité d'observation. Pour gagner en complexité algorithmique, on peut plutôt proposer l'utilisation d'un premier tirage par loi binomiale déterminant le nombre d'observations extérieures *réussies* puis un tirage aléatoire déterminant lesquelles.

Chaque pêcheur met ensuite à jour ses connaissances du contexte suivant $\tau_k(b_k, o_k)$ (lignes 8 à 11) ainsi que ses variables internes telles que ses gains annuels ou ses prises totales de chaque espèce.

À la fin de chaque année, de nouveaux quotas à appliquer pour l'année suivante (ligne 15) sont déterminés.

Dans le chapitre précédent, nous avons vu que l'optimisation pouvait permettre une bonne gestion des décideurs et avions conclu que la principale limitation de ces méthodes venait de la considération de l'échelle des pêcheurs. Nous allons donc dans un premier temps nous intéresser au point de vue des pêcheurs et à leur prise de décision journalière. Leur point de vue peut être représenté par un problème du bandit manchot.

Algorithme 8 Simulation bandit

Initialisation de l'environnement
pour $i \in [0; nbYear]$ **faire**
  pour $j \in nbExploitationDays$ **faire**
    pour $k \in nbFishermen$ **faire**
      $action_k = determineAction(contexte_k, t)$
    fin pour
    $observations, reward = step(action)$
    pour $k \in nbFishermen$ **faire**
      mise à jour $contexte_k = \tau_k(contexte_k, observations_k)$
      mise à jour des variables de $fisherman_k$ (nombre total de prises par espèces, gains totaux, ...)
    fin pour
    $t = t + 1$
  fin pour
  $t = i \times 365$
  $quotas = determineFishingQuotas(t)$
fin pour

4.4.2 Description et objectifs

En simplifiant à l'extrême, le comportement d'un pêcheur peut se résumer à choisir de pêcher ou non et dans quelle zone pour chaque intervalle de temps. Ce comportement peut être représenté par la figure 4.9.

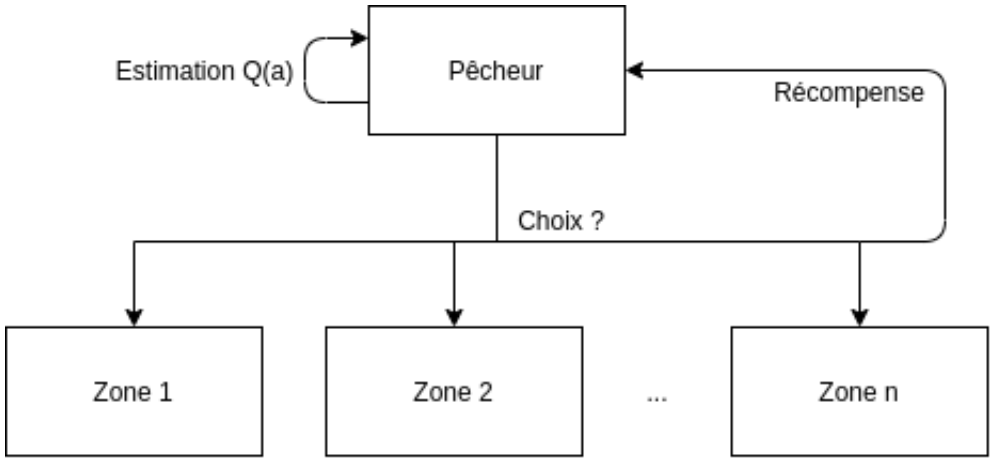


FIGURE 4.9 – Point de vue d'un pêcheur selon un bandit manchot

On peut alors considérer cela comme un problème du bandit manchot à k bras où chaque zone de pêche est une action possible. Une dernière action est ajoutée pour la non pêche. Ainsi, à chaque pas de temps, il devra choisir une action et obtiendra une récompense.

En considérant le système modélisé par la figure 4.7, on peut complexifier le système et considérer un bandit manchot contextuel (figure 4.10).

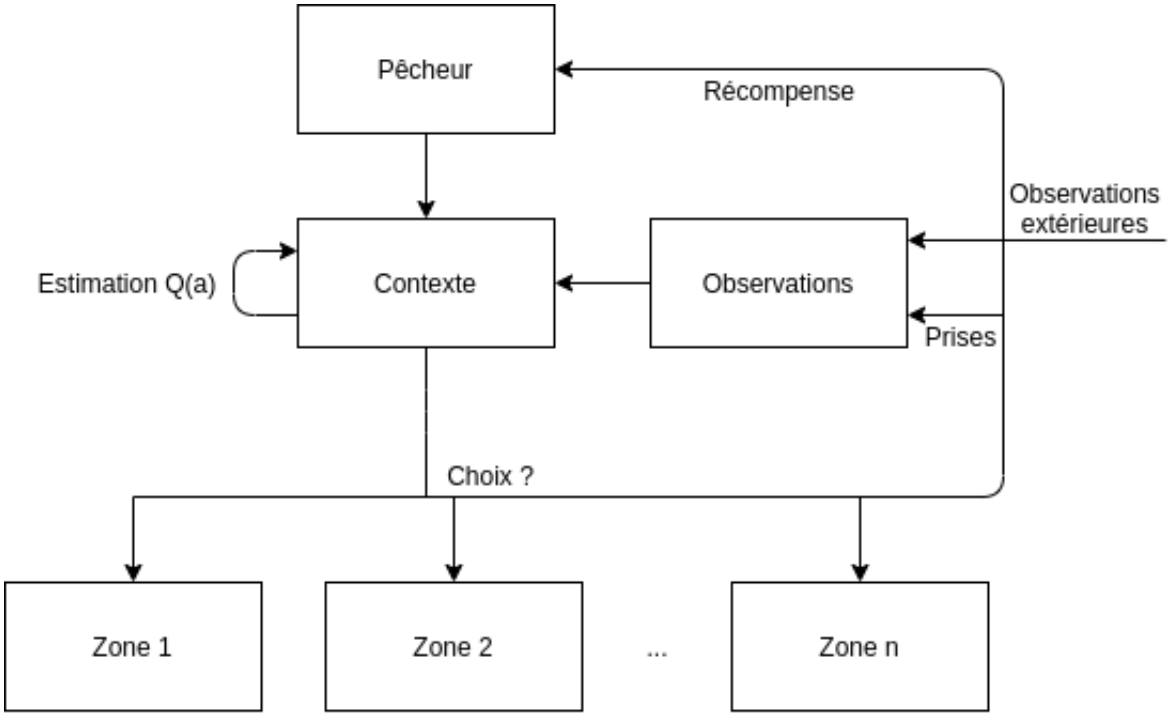


FIGURE 4.10 – Point de vue d'un pêcheur selon un bandit manchot contextuel

Dans ce cas, à chaque pas de temps, chaque pêcheur devra choisir une action. Celle-ci lui donnera une récompense, mais également une observation correspondant aux prises de chaque espèce. Chacun aura également une certaine probabilité d'observer les actions et résultats des autres, concept représenté par les observations extérieurs sur la figure 4.10. Ces observations permettront une estimation du contexte, c'est-à-dire dans ce cas, ce que l'on peut espérer capturer par zone et par espèce. Le pêcheur a également connaissance des prix de vente. Ainsi, il peut estimer ses gains pour chaque action.

Nous pouvons également prendre en compte la notion de quota. Dans ce cas, le pêcheur devra maintenir à jour un historique de ses captures pour ne pas dépasser un quota annuel fixé par les décideurs.

La récompense engendrée par une action est alors proportionnelle aux gains correspondant aux prises engendrées si aucun quota n'est dépassé, une pénalité dans le cas contraire. À première vue, sa valeur n'a que peu d'importance dans ce contexte, car le but va être d'estimer la meilleure action à chaque instant. Elle doit donc simplement être inférieure à toute autre valeur de récompense possible. Or, par la suite nous allons tester l'algorithme *Upper Confidence Bounds* (UCB, annexe 6.2.1). Celui-ci utilise le calcul d'un biais, correspondant en quelque sorte à l'incertitude sur les estimations, pour modifier la valeur $Q(a)$ estimée et permettre une certaine exploration. Il sera donc plus facile à utiliser, et plus efficace, si les bornes des récompenses ne sont pas variables avec les caractéristiques de l'environnement.

On utilise alors :

$$recompense = \frac{Gains}{GainsOptimaux} \iff \nexists i,\ captures_i > quota_i \tag{4.44}$$

$$recompense = -1 \; sinon \tag{4.45}$$

L'objectif de cette expérience est de tester l'efficacité de différentes méthodes d'exploration de l'espace d'actions. Cela permettra de montrer la vitesse et la qualité de l'acquisition des connaissances en fonction de différentes méthodes et paramétrage de l'environnement.

4.4.3 Estimation des récompenses

La principale difficulté vient de l'estimation des récompenses pour les pêcheurs. La plupart des méthodes estiment l'espérance de récompenses grâce aux récompenses précédemment observées. Ici, le contexte fait qu'une action qui s'avère bénéfique habituellement peut devenir négative lors d'un dépassement de quotas. Il est alors nécessaire de maintenir une base de connaissance des captures espérées en fonction des actions plutôt que des récompenses obtenues.

De plus, celles-ci dépendent de l'état des stocks à chaque instant. L'environnement est donc non stationnaire. Il est donc nécessaire de prendre en compte la temporalité dans nos estimations. Ici, les agents, pêcheurs, ont connaissance de la fonction de calcul de la récompense. Leur but est alors d'estimer le contexte plutôt que la distribution de probabilité de récompense de chaque action.

Pour gérer la temporalité on introduit donc un facteur d'actualisation (*discount factor* en anglais), γ, pour lequel différentes valeurs seront testées.

On a alors :

$$\mathbb{E}(C_{ij}) = \frac{1}{\sum_{obs} \gamma^{t-tobs}} \sum_{obs} \gamma^{t-tobs} Cobs_{ij} \tag{4.46}$$

Par la suite, à chaque fois que nous ferons référence à γ nous parlerons de sa valeur en termes de $\gamma = \sqrt[365]{(x)}$. En effet, le pas de temps de simulation étant fixé à un jour, une valeur $\gamma = \sqrt[365]{x}$ signifie que la donnée observée aujourd'hui a $\frac{1}{x}$ fois plus d'importance qu'une donnée observée l'année dernière, à la considération des années bissextiles près. Cette notation rend la représentation de la réalité de ce paramètre plus simple à appréhender que sa valeur numérique du fait de la faible variation que celle-ci représente dans les différents tests.

Une autre méthode pourrait être de se baser sur l'échantillonnage de Thompson Thompson (1933). L'échantillonnage de Thompson, généralement utilisé sur des problèmes stationnaires, mais dont des versions non stationnaires existent Russo et al. (2017), maintient à jour des lois de probabilité du contexte et de la récompense. Il les utilise ensuite à chaque itération pour estimer l'espérance de chaque action et sélectionne celle dont l'espérance est la plus élevée. Il se base généralement sur des méthodes bayésiennes d'estimation des lois de probabilité qui nécessitent des connaissances a priori sur les lois réelles. Dans notre cas, il est impossible de savoir quel type de loi de probabilité suivent les captures de chaque espèce en réalité.

Dans les environnements générés nous utilisons 3 types de lois de probabilité différents (tableau 4.1). Le but va alors être d'estimer quel type de loi représente le mieux chaque espèce et avec quels paramètres. Pour cela nous pouvons utiliser des méthodes basées sur le maximum de vraisemblance

(*maximum likelihood estimation*, MLE, en anglais) Rossi (2018).

4.4.4 Maximum de vraisemblance

Cette méthode permet d'estimer les paramètres d'une loi de probabilité. En supposant qu'un type de loi donné correspond aux données observées, le but est de trouver l'ensemble de paramètres qui maximise la fonction de vraisemblance. C'est-à-dire, l'ensemble de paramètres pour lequel il est le plus probable que l'ensemble des données observées ait été tiré.

On peut alors estimer le jeu de paramètres optimal grâce à des méthodes d'optimisation en utilisant la fonction d'évaluation suivante à minimiser avec θ les paramètres de la loi de probabilité et *obs* l'ensemble des données observées :

$$f(\theta) = -\frac{1}{|obs|} \sum_{obs} log(p(obs|\theta)) \tag{4.47}$$

Il est alors possible d'utiliser des algorithmes à base de gradient comme les régions de confiance Rosen et al. (2012) ou l'algorithme de Levenberg-Marquardt Li et al. (2017) pour obtenir une convergence rapide. Dans le cadre de notre étude, il s'est avéré que l'algorithme de Levenberg-Marquardt pouvait s'avérer instable, et présentait de moins bons résultats et temps de convergence.

La loi de probabilité dont l'estimation des paramètres via MLE donne les meilleurs résultats sera donc considérée comme valide jusqu'à la prochaine estimation.

4.4.5 Tests d'algorithmes

Nous avons donc comparé les gains obtenus par différents algorithmes de bandits, sur ce problème. La figure 4.11 montre la somme des gains de tous les pêcheurs au cours du temps (en année) pour chaque algorithme en utilisant les mêmes hyperparamètres d'environnement γ et la probabilité d'observation des actions et résultats des autres pêcheurs P. On peut voir que l'échantillonnage de Thompson utilisant l'estimation par maximum de vraisemblance présente de bien meilleurs résultats que les autres. L'algorithme UCB présente des gains comparables sur les années finales mais il prend beaucoup plus de temps à converger.

Il peut alors être intéressant de comparer les résultats pour différentes valeurs des hyperparamètres. Le tableau 4.2 montre que P n'a que peu d'influence sur les gains totaux. Il influence cependant grandement la variabilité des gains en fonction des années. La valeur d'écart minimale, donc le cas de figure où les gains sont les plus stables, est obtenue dans le cas où tous les pêcheurs coopèrent entre eux, $P = 1$, et où on ne considère que très faiblement le passé dans nos estimations du futur $\gamma = \sqrt[365]{0,1}$. Dans ce cas, on peut supposer que la quantité d'observation du présent est suffisante pour estimer correctement les lois de probabilité rapidement en ne prenant que très peu en compte le passé. On remarque également que pour $\gamma = \sqrt[365]{0,5}$, les gains sont optimaux quand P est faible et très faible. L'écart moyen est également optimal à $P = 0.1$ mais il est le pire à $P = 0.01$. Il semble donc qu'il soit impossible de déterminer une valeur de γ optimale pour tous les cas.

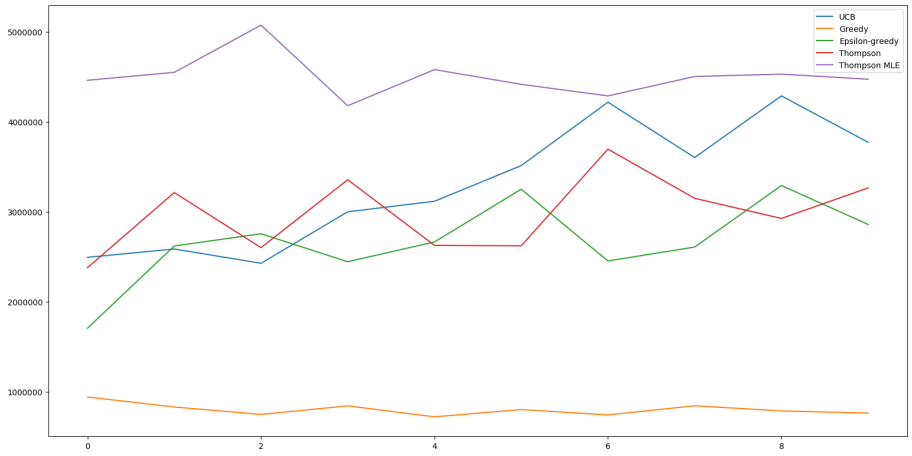


FIGURE 4.11 – Comparaison des gains pour différents algorithmes

	$P = 0,01$		$P = 0,1$		$P = 1$	
γ^{365}	Gains	Écart moyen	Gains	Écart Moyen	Gains	Écart Moyen
0,1	4,493 E06	1,57 E05	4,406 E06	1,67 E05	4,381 E06	**1,15 E05**
0,25	4,365 E06	**1,34 E05**	4,423 E06	1,48 E05	**4,406 E06**	1,70 E05
0,5	**4,528 E06**	2,31 E05	**4,505 E06**	**1,42 E05**	4,244 E06	1,92 E05
1	4,417 E06	2,22 E05	4,348 E06	2,73 E05	4,399 E06	2,25 E05

TABLE 4.2 – Gains et écart moyen entre les années en fonction de γ et P

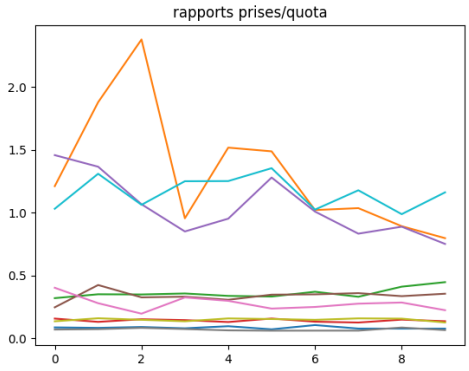
Il se pose également la question du respect des quotas. La figure 4.12 montre l'évolution des rapports entre les prises et le quota de chaque espèce au cours du temps pour $\gamma = \sqrt[365]{0.5}$ et différentes valeurs de P. Chaque courbe représente une espèce. On peut tout d'abord remarquer que 3 espèces sont particulièrement prises par rapport aux quotas fixés pour elles. Avec un taux d'observation faible et très faible, $P = 0.1$ et $P = 0.01$, il semble qu'il soit difficile de faire respecter les quotas pour toutes les espèces. En revanche, quand tous les pêcheurs mettent en commun leurs connaissances, $P = 1$, il semble qu'il n'y ait besoin que d'une à deux années pour que les quotas deviennent parfaitement respectés ou presque.

On peut alors analyser la qualité des estimations des lois de probabilité. Pour cela, on peut comparer le MLE des lois estimées avec le MLE des lois réelles pour un ensemble de nouvelles données générées à partir de la vraie loi de probabilité. On utilise donc l'équation :

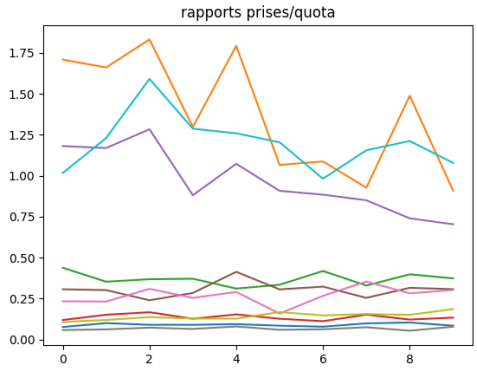
$$qualiteEstimation = \frac{MLE_{loiReelle}}{MLE_{loiEstimee}} \tag{4.48}$$

Dans ce cas, la valeur optimale est 1.

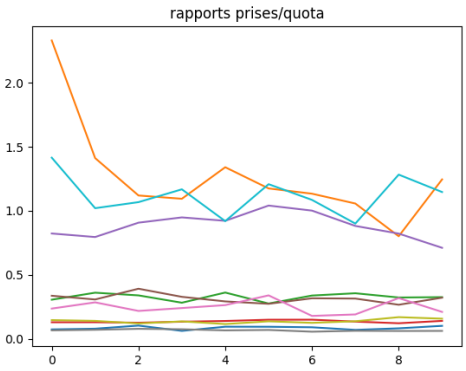
La figure 4.13 montre le cas de l'estimation pour une espèce qui est quasi stationnaire. On peut



(a) $P = 0.01$



(b) $P = 0.1$



(c) $P = 1$

FIGURE 4.12 – Rapports entre prises et quota de chaque espèce au cours du temps pour $\gamma = \sqrt[365]{0.5}$ et différentes valeurs de P

observer que la valeur de γ n'a pas d'influence sur la qualité de l'estimation. En effet, dans le cas où l'espèce évolue peu, donner une grande importance au passé peut ne pas être néfaste.

En revanche, sur la figure 4.14, γ a une importance. On peut dans un premier temps remarquer que pour une espèce dont l'évolution est non stationnaire, la probabilité d'observation P, et donc le nombre d'observations, a une importance non négligeable. En effet, pour $P = 1$, figure 4.14c, et $P = 0.1$, figure 4.14b, les estimations sont de bien meilleure qualité que pour $P = 0.01$, figure 4.14a. On remarque de plus qu'une forte valeur de γ mène a de bien moins bonnes estimations, notamment lorsque la probabilité d'observation est faible.

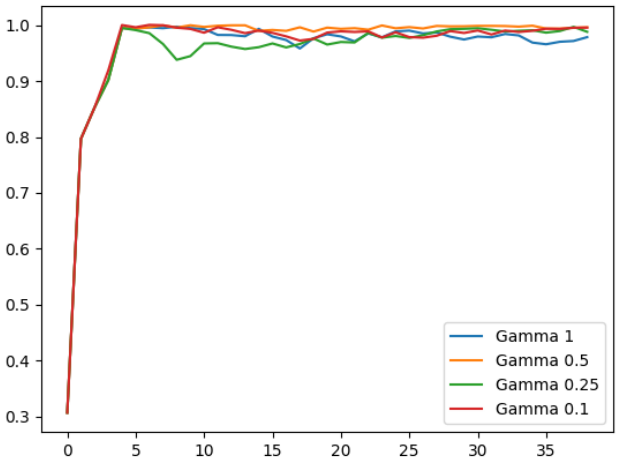
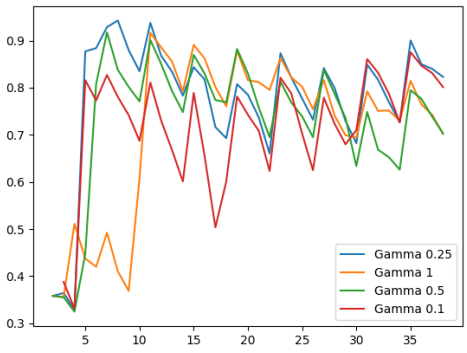
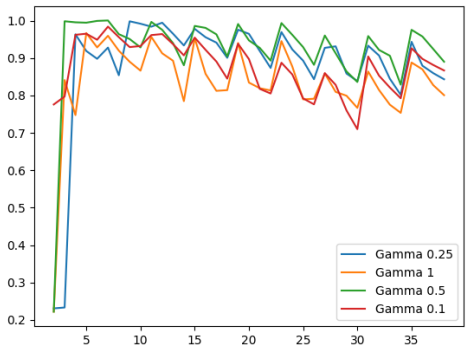


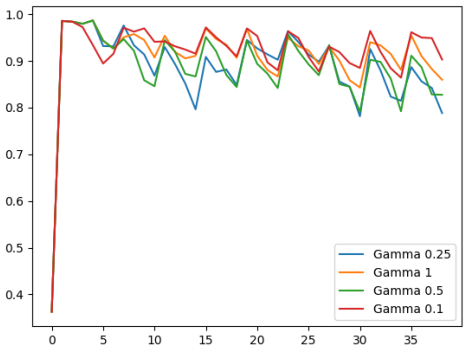
FIGURE 4.13 – Comparaison de la convergence de l'estimation de la loi de probabilité en fonction de γ pour $P = 0.1$ sur une espèce quasi stationnaire



(a) $P = 0.01$



(b) $P = 0.1$



(c) $P = 1$

FIGURE 4.14 – Comparaison de la convergence de l'estimation de la loi de probabilité en fonction de γ pour différentes valeurs de P sur une espèce à évolution non stationnaire

Il semble donc qu'il soit impossible de trouver une valeur optimale pour γ. Une valeur de $\gamma = \sqrt[365]{0.5}$ semble être un compromis acceptable notamment pour de faibles probabilités d'observation, ce qui semble cohérent avec la réalité.

Cependant, même avec de faibles probabilités d'observation, il est possible d'estimer relativement efficacement les lois de probabilité des captures. La connaissance de chaque pêcheur peut donc suffire pour une exploitation personnelle quasi optimale. Cependant, nous avons vu que certains quotas sont quand même légèrement dépassés, surtout les premières années. De plus, d'autres espèces sont sous-exploitées. Cela met en lumière la présence d'une incertitude de faisabilité quant à l'estimation de l'exploitation optimale via optimisation et donc la nécessité d'utiliser des modèles prenant en compte une échelle plus fine comme nous le faisons via les méthodes de bandits.

4.5 Conclusion du chapitre

Dans ce chapitre, nous avons détaillé différentes variantes des processus de décision markoviens afin de représenter la prise de décision dans un environnement partiellement observable.

Les différents formalismes des processus de décision markoviens permettent une représentation efficace des processus à l'échelle de la personne, dans notre cas, l'action individuelle de pêche de chacun. En diminuant l'échelle temporelle de simulation pour arriver à la prise en compte de comportements individuels, à transitions stochastiques, dans un système complexe peu connu, les méthodes d'optimisations trouvent rapidement leur limite. Cependant, cette problématique semble particulièrement correspondre aux problèmes de bandit manchots contextuels. Ainsi, par la suite, nous avons développé différentes méthodes permettant la prise de décision pour les problèmes formalisés de cette façon.

Plusieurs algorithmes ont été testés. Ces expériences ont montré l'impact des incertitudes de faisabilité des solutions théoriquement optimales à l'échelle des décideurs, mais irréalisable à l'échelle des pêcheurs. Elles nous ont également permis de montrer qu'il est possible de rapidement estimer les lois de probabilité des captures par espèce et par zone à partir d'observations, afin de proposer des outils de gestions efficaces.

Ce formalisme est cependant limité, par exemple, les décideurs peuvent difficilement être représenté de cette façon. Dans l'application présentée ici, les quotas optimaux ont été déterminés à partir des variables cachées de l'environnement. La représentation du contexte atteint également rapidement ces limites. Elle peut être suffisante dans le cas d'applications simples comme nous l'avons présenté ici, mais la complexification du modèle et de l'environnement, nécessaire à l'utilisation en situation réelle, forcera à la complexification des méthodes. C'est pourquoi, dans le chapitre suivant nous allons nous intéresser à des méthodes plus complexes d'apprentissage par renforcement. Nous montrerons que les méthodes tabulaires ne sont pas suffisantes pour une utilisation efficace et nous orienterons donc vers l'apprentissage profond par renforcement.

Chapitre 5

APPRENTISSAGE PROFOND PAR RENFORCEMENT : DESCRIPTION ET APPLICATIONS

L'apprentissage par renforcement diffère beaucoup des méthodes d'apprentissage supervisé et non supervisé dans la mesure où, bien que certaines méthodes se basent sur des données d'observations directes Kang et al. (2019), il s'applique très souvent sur des modèles et simulations afin de faire apprendre un comportement à un agent autonome au sein de cet environnement. L'agent apprenant devra donc interagir avec son environnement et recevra des observations. La plupart du temps celles-ci ne sont pas déterministes mais plutôt stochastiques, à partir desquelles il devra décider de ses futures actions. Dans ce cas là, l'agent apprendra à partir de récompenses (ou pénalités quand celles-ci prennent des valeurs négatives) qu'il recevra en fonction de ses actions et son état. Cette méthode est particulièrement efficace dans les domaines où la simulation joue un grand rôle.

Ainsi, il a naturellement été appliqué avec succès à un grand nombre de jeux-vidéo tel que les jeux ATARI Mnih et al. (2015), ou plus récemment sur Dota2 Berner et al. (2019) et a été popularisé par ses performances inattendues sur le légendaire jeu de GO Silver et al. (2016). Pour les applications du monde réel cependant, il semble peu utilisé en dehors du domaine médical Mahmud et al. (2018), des systèmes de conduite autonome Talpaert et al. (2019) ou des télécommunications Luong et al. (2019). Mais d'autres problèmes réels, tels que la gestion des ressources marines, semblent adaptés à son utilisation.

Basant toutes nos études sur des simulations, l'apprentissage par renforcement a naturellement été une méthode à envisager pour nos problématiques.

L'optimisation peut s'appliquer à tout problème formulable sous forme d'objectifs et de contraintes. Cependant, en situation d'observabilité partielle du système, la formalisation du problème peut s'avérer

très difficile, voire impossible, et conduire à un problème de dimensions, et donc de temps de calcul trop important. Ainsi, récemment, l'optimisation a été utilisée dans un contexte de pêcherie quasi-exclusivement dans des cas de fortes connaissances sur le système. On peut par exemple citer des calages du modèle Gadget Begley and Howell (2004) avec plusieurs métaheuristiques différentes Penas et al. (2019). Elle a aussi permis d'étudier la saisonnalité de certains événements de dynamique de population, montrant ainsi que certains résultats ne peuvent pas arriver sans cette prise en compte Ni and Sandal (2019). Les auteurs se limitent cependant à une pêcherie mono-spécifique, permettant ainsi plus facilement l'acquisition de données pour une modélisation multi-age précise. L'impact de la migration entre différentes zones d'exploitation peut également être étudié Voss et al. (2018). Finalement, Akinbulire et al. (2018) présentent une application de l'optimisation multi-objectif pour déterminer les compromis entre les coûts et la probabilité de détection de la pêche illicite, non déclarée et non réglementée. Toutes ces applications présentes des représentations relativement simples de l'état du système à un instant donné. Dans un contexte comparable, Akinbulire et al. (2017) proposent l'utilisation de RL pour aider à la détection et la gestion de comportements de pêche illégale. Dans ce cas là, la difficulté de représentation des espaces d'états et d'actions a entraîné l'utilisation d'un système d'inférence flou Jang (1993) et RL.

Dans notre contexte, l'optimisation semble donc particulièrement adaptée lorsque le système est suffisamment connu, ou simplifié, pour permettre une formalisation du problème supposant un contrôle parfait et des transitions d'état déterministes. L'optimisation robuste ou l'optimisation robuste ajustable, permettent de pallier certains problèmes d'incertitude quant à la calibration, comme nous l'avons vu dans les chapitres précédents. En revanche, une observabilité trop partielle du système, et donc des incertitudes sur l'ensemble des paramètres, peut facilement mener à une explosion de la dimensionnalité et/ou des problèmes d'équifinalité insolvables. Dans ce cas, les différentes formes de MDP permettant de représenter des systèmes partiellement observables et soumis à transitions d'états suivant des lois de probabilité cachées. Ainsi, l'apprentissage par renforcement semble parfaitement adapté.

Dans un premier temps, nous nous intéresserons aux différentes méthodes tabulaires d'apprentissage par renforcement et aux méthodes d'apprentissage profond par renforcement. Nous réaliserons ensuite différentes expériences d'apprentissage en nous basant sur les environnements générés dans le chapitre précédent. Nous montrerons alors les résultats des apprentissages en tant que tel mais étudierons également l'impact économique et écologique des différentes stratégies qu'ils peuvent proposer, notamment en situation de difficulté à faire respecter les quotas instaurés.

5.1 Méthodes d'apprentissage par renforcement

Le passage des problèmes formalisés sous forme du bandit manchot à l'apprentissage profond par renforcement implique, au préalable, la considération des méthodes tabulaires d'apprentissage par renforcement, à la base de ces dernières. Nous commencerons donc par les survoler rapidement avant de développer les méthodes que nous utiliserons par la suite : les méthodes par approximations et plus précisément, celles basées sur les réseaux de neurones profonds.

5.1.1 Méthodes tabulaires

Ces méthodes sont utilisables dans le cas où les espaces d'actions et d'états sont suffisamment petits pour que la fonction de valeur soit représentable sous forme de tableau. Dans ce cas, les méthodes peuvent aller jusqu'à trouver l'exacte politique optimale, contrastant ainsi avec les méthodes par approximations qui seront décrites en section 5.1.2.

On distingue trois grands types de méthodes :

— la programmation dynamique Bellman and Dreyfus (2015) Minsky (1967) : bien développée mathématiquement, mais nécessitant un modèle complet et précis de l'environnement.

— les méthodes de Monte-Carlo, conceptuellement plus simples, mais peu adaptées pour des calculs pas à pas incrémentaux.

— les méthodes de différences temporelles (*Temporal-difference methods*) qui ne nécessitent pas forcément de connaissances sur le modèle et sont très incrémentales, mais complexes à analyser.

Bien que, par la suite, nous n'utiliserons pas ces méthodes directement car nous aurons des espaces d'états trop grands, en parler brièvement nous permettra d'introduire les principaux concepts des méthodes d'apprentissage par renforcement.

Programmation dynamique (DP)

L'évaluation de la politique fait référence au calcul, généralement itératif, des fonctions de valeur pour une politique donnée. L'amélioration de la politique fait référence au calcul d'une politique améliorée étant donné la fonction de valeur pour cette politique. En combinant ces deux notions, nous obtenons l'itération de la politique et l'itération de la valeur, les deux méthodes de DP les plus populaires. On peut mieux comprendre les méthodes DP, et en fait même presque toutes les méthodes d'apprentissage par renforcement, en les considérant comme une itération de politique généralisée (GPI). La GPI est l'idée générale de deux processus interactifs tournant autour d'une politique approximative et d'une fonction de valeur approximative. Un processus prend la politique comme donnée et effectue une certaine forme d'évaluation de la politique, en changeant la fonction de valeur pour qu'elle ressemble davantage à la véritable fonction de valeur de la politique. L'autre processus prend la fonction de valeur comme donnée et effectue une certaine forme d'amélioration de la politique, en changeant la politique pour l'améliorer, en supposant que la fonction de valeur est sa fonction de valeur. Bien que chaque processus modifie la base de l'autre, ils travaillent ensemble pour trouver une solution conjointe, une politique et une fonction de valeur, qui ne soit pas modifiée par l'un ou l'autre processus et, par conséquent, est optimale.

Toutes les méthodes de DP mettent à jour les estimations des valeurs des états sur la base des estimations des valeurs des états précédents. C'est ce qui est appelé *bootstrapping*, et est réutilisé dans un très grand nombre de méthodes d'apprentissage par renforcement.

Les méthodes de DP, bien que souvent efficaces présentent le désavantage de nécessiter une connaissance parfaite du modèle et des transitions d'état de l'environnement. Cette condition est parfaitement inenvisageable dans notre contexte d'étude d'environnements incertains et non stationnaires. Nous allons

donc maintenant nous intéresser à des méthodes permettant de s'affranchir de ces notions, les méthodes de Monte-Carlo.

Méthodes de Monte-Carlo

Les méthodes de Monte-Carlo apprennent les fonctions de valeur et les politiques optimales à partir de l'expérience sous la forme d'épisodes d'échantillonnage. Cela leur confère au moins trois types d'avantages par rapport aux méthodes DP :

— elles peuvent être utilisées pour apprendre directement à partir de l'environnement, sans modèle ou dynamique de l'environnement connu.

— elles peuvent également être utilisées par simulation ou modèle simplifié.

— elles peuvent être utilisées sur des sous-ensembles du problème initial, des régions d'intérêt, sans évaluer le reste de l'ensemble des états.

Elles sont de plus moins impactées par une éventuelle violation de la propriété de Markov car elles ne mettent pas à jour les fonctions de valeur par rapport aux états suivants.

Les méthodes de Monte-Carlo offrent un processus alternatif d'évaluation des politiques. Plutôt que d'utiliser un modèle pour calculer la valeur de chaque état, elles font simplement la moyenne de nombreux rendements qui commencent dans l'état. Comme la valeur d'un état est le rendement attendu, cette moyenne peut devenir une bonne approximation de la fonction de valeur.

Maintenir une exploration suffisante est un problème dans les méthodes de contrôle de Monte-Carlo. Il ne suffit pas de sélectionner les actions actuellement estimées comme étant les meilleures, car alors aucun retour ne sera obtenu pour les actions alternatives, et il se peut que l'on n'apprenne jamais qu'elles sont réellement meilleures.

Dans les méthodes on-policy, l'agent va toujours explorer et chercher la meilleure politique. Dans les méthodes off-policy, l'agent explore mais apprend une politique déterministe optimale qui peut être sans rapport avec la politique suivie. Celles-ci font référence au fait d'apprendre la fonction de valeur d'une politique cible (*target-policy*) à partir de données engendrées par une autre politique : la politique comportementale (*behavior-policy*).

Puisque les méthodes de Monte-Carlo fonctionnent par expérience, elles peuvent être utilisées pour l'apprentissage direct sans modèle. Elles ne font pas de *bootstraping*, c'est-à-dire qu'elles ne mettent pas à jour leurs estimations de valeur sur la base d'autres estimations de valeur. Ces deux différences ne sont pas étroitement liées et peuvent être séparées.

Les méthodes de différence temporelle utilisent les avantages de ces deux types de méthodes.

Méthodes de différence temporelle (TD)

TD est une combinaison de DP et MC. Les méthodes de TD peuvent apprendre directement à partir d'expérience sans connaissance sur le modèle. Comme pour la DP, elles peuvent mettre à jour certaines estimations à partir de celles apprises précédemment.

Comme les méthodes de Monte-Carlo, TD utilise des expériences pour résoudre le problème de prédiction, c'est-à-dire, estimer la fonction de valeur. En prenant quelques expériences qui suivent la politique π, elles mettent à jour V de v_π pour les états non terminaux S_t de cette expérience. MC attend que la valeur de retour soit connue et l'utilise comme cible pour $V(S_t)$.

Un des avantages évident par rapport aux méthodes de DP est que celles-ci ne nécessitent pas forcément de connaissances sur le modèle de l'environnement, les récompenses et les probabilités de transition. Par rapport aux méthodes de Monte-Carlo, celles-ci sont naturellement implémentées de façon on-line et complètement incrémentales. De plus, certaines applications nécessitent de très longs épisodes (qui peuvent même être infinis dans le cas de tâches continues), ce qui peut être problématique et long avec les méthodes de MC.

Le Q-learning (algorithme 9), est un algorithme de différence temporelle off-policy très connu aujourd'hui de par ses extensions à l'apprentissage profond par renforcement (*deep reinforcement learning*). Il est cependant historiquement utilisé comme méthode tabulaire. Il se base sur la relation maîtresse suivante :

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{5.1}$$

Dans ce cas, l'apprentissage de la fonction Q approxime directement q_* *l'action-value function* optimale, indépendemment de la politique actuellement suivie. Cette méthode simplifie drastiquement les analyses de l'algorithme et permet une convergence Watkins and Dayan (1992) rapide. La politique a toujours un effet en déterminant quelle paire action-état sera visitée et mise à jour.

Algorithme 9 Q-learning : Off-policy TD-control

Initialise $Q(s, a)$ $\forall s \in S, a \in A(s)$, arbitrairement et $Q(etats\ terminaux) = 0$
repeat
  Initialise s
  repeat
    choisir a à partir de s en utilisant la politique dérivée de Q (ϵ-greedy par exemple)
    $r, s' =$pas_de_simulation(a)
    $Q(s, a) = Q(s, a) + \alpha[r + \gamma max_a Q(s', a) - Q(s, a)]$
    $s = s'$
  until s est terminal
until $nEpisode$

Un grand nombre d'autres méthodes de différence temporelle existent. L'annexe 6.2.4 en détaille les principales.

Toutes les méthodes que nous venons de voir ont fait leurs preuves. Elles s'appliquent cependant uniquement à des problèmes de taille relativement faible Bhowmik (2010), permettant un stockage optimal de fonction de valeur et une exploration relativement complète des transitions d'états. Ce n'est malheureusement pas toujours le cas.

Nous travaillons dans un contexte d'observabilité partielle basé sur un environnement dynamique à transitions stochastiques. Le nombre d'états semble déjà très important et ceux-ci devront être estimés à partir des observations. Il devient alors impossible de stocker, et explorer, l'ensemble de ces possibilités

dans un processus tabulaire.

Il est alors nécessaire de définir des fonctions permettant d'approximer efficacement les fonctions de valeurs au fur et à mesure de l'apprentissage. On parle alors de méthodes par approximation.

5.1.2 Méthodes par approximation

Le problème des grands espaces d'états n'est pas seulement un problème de manque de mémoire pour les grandes tables, mais également le temps et les données nécessaires pour les remplir avec précision. Dans beaucoup de nos tâches utilisant les méthodes par approximation, presque tous les états rencontrés n'auront jamais été vus auparavant. Étendre l'apprentissage par renforcement à l'estimation de fonction peut aussi permettre de prendre en compte des problèmes partiellement observables, c'est-à-dire, où l'agent n'a pas connaissance de l'ensemble des états. Pour prendre des décisions judicieuses dans de tels états, il est nécessaire de généraliser à partir des états et transitions rencontrés précédemment qui sont, dans un certain sens, similaires à l'état actuel. La question clef ici est donc celle de la généralisation.

On-policy Prediction avec approximations

La nouveauté dans cette partie est que les approximations ne sont pas représentées par une table, mais par des fonctions paramétriques formées par un vecteur $w \in \mathbb{R}^d$. Ainsi, la notation de la valeur approximative d'un état s sera notée : $\hat{v}(s, w) \approx v_\pi(s)$. En général $\hat{v}$ est une fonction calculée par un réseau de neurones artificiel à multiples couches avec w le vecteur des poids de connexion entre les couches.

$\hat{v}$ peut également être la fonction calculée par un arbre de décision où w est l'ensemble des nombres définissant les points de séparation et les valeurs des feuilles. Le nombre de poids (la dimension de w) est beaucoup moins important que le nombre d'états possible. Ainsi, changer un poids change les estimations de beaucoup d'états, réduisant ainsi la complexité et permettant de gérer des problèmes bien plus complexes.

Il est possible de voir chaque *update* comme un exemple d'entraînement conventionnel en apprentissage supervisé. Ainsi, en principe, on peut utiliser n'importe quel outil d'apprentissage supervisé comme les réseaux de neurones, arbres de décisions et différents types de régressions multivariées. Toutes les méthodes par approximation ne sont cependant pas forcément adaptées à l'apprentissage par renforcement. En RL, il est important que l'apprentissage puisse s'effectuer via interaction de l'agent avec son environnement, ce qui requiert des capacités d'apprentissage au fur et à mesure de l'acquisition des données. De plus, les fonctions d'approximation doivent pouvoir prendre en compte la non-stationnarité de l'environnement.

Pour pouvoir ici estimer la qualité d'une politique, nous devons définir quels états nous intéressent le plus via une distribution d'états pondérés (*state weighting*), $\mu(s) \geq 0, \sum_s \mu(s) = 1$. Ainsi, nous pouvons calculer l'erreur de chaque état par rapport à la valeur réelle, c'est-à-dire généralement le carré de la différence entre $\hat{v}(s, w)$ et $v_\pi(s)$. Avec les poids, on obtient une fonction d'objectif, la moyenne des carrés

des valeurs d'erreur (*Mean Squared Value Error*), $\overline{VE}$:

$$\overline{VE}(w) = \sum_{s \in S} \mu(s)[v_\pi(s) - \hat{v}(s,w)]^2 \tag{5.2}$$

Ainsi, la racine carré de cette valeur donne une mesure de la différence entre la valeur approximative et les vraies valeurs. Souvent, $\mu(s)$ est choisi comme une fraction du temps passé dans s. Sous un entraînement $on - policy$, cela est appelé *on-policy distribution*.

Stochastic-gradient et méthodes de semi-gradient La méthode de descente de gradient stochastique (SGD) permet d'ajuster les poids du vecteur w afin d'essayer de minimiser $\overline{VE}$. Il réalise cela par une petite variation dans la direction qui réduit le plus l'erreur :

$$w_{t+1} = w_t - \frac{1}{2}\alpha\nabla[v_\pi(S_t) - \hat{v}(S_t, w_t)]^2 \tag{5.3}$$

$$w_{t+1} = w_t + \alpha[v_{pi}(S_t) - \hat{v}(S_t, w_t)]\nabla\hat{v}(S_t, w_t) \tag{5.4}$$

α étant un paramètre $step - size$ positif décroissant au cours du temps, et $\nabla f(w)$ représente, pour toute expression scalaire $f(w)$, le vecteur de dérivées partielles :

$$\nabla f(w) = \left(\frac{\partial f(w)}{\partial w_1}, \frac{\partial f(w)}{\partial w_2}, ..., \frac{\partial f(w)}{\partial w_d}\right)^T \tag{5.5}$$

On pourrait naïvement penser qu'il serait plus efficace de complètement éliminer l'erreur sur chaque exemple rencontré. Cependant, il faut tenir compte que, du fait du moindre nombre de dimensions du vecteur w par rapport au nombre d'états, il est bien souvent impossible d'obtenir une politique optimale pour tout exemple. On cherche plutôt une valeur d'équilibre, permettant de minimiser l'erreur sur chaque exemple.

Un grand nombre de méthodes d'optimisation basées sur les gradients ont été proposées et peuvent être utilisées pour optimiser les poids du vecteur w. La section 1.2 en présente les principales.

Méthodes linéaires Un des cas les plus importants des fonctions d'approximation est quand on considère que $\hat{v}(., w)$ est une fonction linéaire du vecteur w. Ainsi, $\forall s \in S \; \exists \, x(s) = (x_1(s), x_2(s), ..., x_d(s))^T \in \mathbb{R}^d$ appelé le *feature vector* avec le même nombre de composantes que w. On a :

$$\hat{v}(s,w) = w^T x(s) = \sum_{i=1}^{d} w_i x_i(s) \tag{5.6}$$

Lors de l'utilisation de descente de gradient stochastique, le gradient est très facilement calculable et vaut :

$$\nabla\hat{v}(s,w) = x(s) \tag{5.7}$$

Approximation de fonctions non linéaires : réseau de neurones artificiels

Le réseau de neurones artificiel est l'élément central des méthodes d'apprentissage profond et donc de l'apprentissage profond par renforcement, constituant ainsi la base des méthodes récentes d'apprentissage par renforcement basées sur les approximations.

Un réseau de neurones artificiel (*artificial neural network*, ANN) est composé d'un ensemble de neurones répartis en couche. Celui-ci débute par une couche d'entrée représentant les états, s'ensuit un ensemble de couches appelées les couches cachées puis une couche de sortie, représentant les résultats possibles, les actions dans un contexte de MDP. Les couches cachées sont composées de neurones, chacun effectuant un calcul linéaire fonction des poids, w, et des valeurs précédentes. De plus, entre chaque couche, une fonction non linéaire est appliquée, appelée la fonction d'activation.

Les couches doivent être entraînées, c'est-à-dire que les poids de chaque neurone seront modifiés afin de minimiser ou de maximiser une fonction d'évaluation, généralement appelée le *loss*. On utilise généralement des méthodes basées sur la descente de gradient stochastique vu précédemment, comme Kingma and Ba (2014) Zhang (2018) Tato and Nkambou (2018).

Dans les ANNs, le sur-apprentissage (overfiting) est un problème récurent du fait du grand nombre de poids. Cela consiste, comme son nom l'indique, à trop apprendre des données présentes et ainsi, de ne plus être capable de s'en servir pour généraliser, c'est-à-dire, s'en servir sur d'autres cas.

Méthodes Off-policy avec approximations

Le Q-learning tabulaire peut laisser penser que l'apprentissage off-policy est facile et peut être généralisé sans problème. Le passage à des méthodes par approximation peut cependant provoquer de nouveaux challenges comme la divergence.

En combinant fonction d'approximation, *bootstraping* et entraînement off-policy. L'ensemble de ces trois éléments est appelé *deadly triad* Sutton and Barto (1998). Combiner deux de ces éléments peut mener à une instabilité, mais elle peut être évitée. L'utilisation de ces trois éléments combinés est cependant à éviter. Le bootstrapping peut ne pas être utilisé contre un coût en temps de calcul et données. L'apprentissage off-policy peut souvent également être évité. Pour des apprentissages hors modèle (*model-free*) par exemple, Sarsa est utilisable à la place de Q-learning. Le Q-learning profond (*deep Q-learning*) propose cependant une méthode permettant d'éviter la divergence : la répétition d'expérience (*experience replay*) ce qui le rend utilisable dans la plupart des cas.

Deep Q-learning Le deep Q-learning Mnih et al. (2015) est l'extension à base de fonction d'approximation du Q-learning. Pour rappel, cette variante de TD se base sur la notion de Q-fonction, c'est-à-dire la fonction de valeur état-action de la politique π. $Q^\pi(s, a)$ mesure l'espérance de retour que l'on obtiendrait en partant de l'état s et en effectuant l'action a puis en suivant π. Dans les méthodes tabulaires, cette fonction est stockée sous forme de tableau contenant les valeurs de chaque combinaison possible de s et a. Quand les espaces d'états et d'actions sont trop importants, cette représentation devient impossible.

Deep Q-learning propose d'entraîner une fonction d'approximation de paramètres θ, prenant la forme d'un réseau de neurones. Cette fonction permet alors d'estimer les valeurs Q, c'est-à-dire $Q(s, a, \theta) \sim$

$Q^*(s,a)$ en minimisant une fonction de perte (*loss*) à chaque étape i :

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim p(.)}[(y_i - Q(s,a,\theta_i))^2] \tag{5.8}$$

$$y_i = r + \gamma max_{a'} Q(s', a', \theta_{i-1}) \tag{5.9}$$

avec y_i la cible d'erreur temporelle (*TD target*) et $y_i - Q$ l'erreur de différence temporelle (*TD error*). p représente la distribution de comportement, c'est-à-dire la distribution de transitions s, a, r, s' collectées sur l'environnement. Les paramètres de l'itération précédente, θ_{i-1} sont fixes. En pratique, on considère souvent plusieurs itérations précédentes Hernandez-Garcia and Sutton (2019).

Deep Q-learning apprend la politique gloutonne :

$$a = max_a(Q(s,a;\theta)) \tag{5.10}$$

Elle utilise une politique comportementale différente pour interagir avec l'environnement et collecter des données. La politique comportementale est très souvent basée sur une politique ϵ-greedy, sélectionnant l'action gloutonne avec une probabilité $1-\epsilon$ et une action aléatoire avec une probabilité ϵ. ϵ décroissant au cours des itérations. Ceci permet une bonne couverture de l'espace état-action.

Pour éviter les problèmes d'instabilité et de divergence des méthodes off-policy, le deep q-learning utilise la répétition d'expérience (*experience replay*). À chaque pas de temps, les données collectées, c'est-à-dire les états, actions, transitions d'états, et récompenses, sont stockées dans un tampon circulaire appelé *replay buffer*. Durant la phase d'entraînement, la fonction de perte n'est donc pas calculée uniquement à partir de la dernière transition, mais plutôt à partir d'un mini-batch de transitions échantillonnées à partir du buffer. Cela a l'avantage d'augmenter l'efficacité des données en les réutilisant dans plusieurs mises à jour, mais surtout en améliorant la stabilité en utilisant des transitions non corrélées dans le batch.

Double Deep Q-network Au début de l'apprentissage, il est nécessaire d'estimer la valeur de $Q(s,a)$ puis de la mettre à jour. Mais elle peut être fortement bruitée, ne permettant jamais d'être complètement sûr que l'action avec la meilleure Q-valeur est effectivement la meilleure action possible. Quand cela arrive, l'agent suivant la politique *optimale* continuera à choisir cette action. Ce problème est appelé surestimation de la valeur d'action et mène à un apprentissage chaotique et compliqué.

Double Q-learning proposé dans Hasselt (2010) utilise deux différentes fonctions d'estimation de valeur d'action Q et Q'. Même si elles sont bruitées, les bruits peuvent être vus comme une distribution uniforme et donc cet algorithme résout le problème de surestimation de la valeur d'action. Hasselt (2010) utilise la fonction de mise à jour suivante :

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R_{t+1} + \gamma Q'(s_{t+1}, a) - Q(s_t, a_t)) \tag{5.11}$$

$$a = max_a Q(s_{t+1}, a) \tag{5.12}$$

Dans double deep Q-network de Van Hasselt et al. (2016), deux réseaux de neurones sont utilisés : *Deep Q-network*, Q_q, et *Target network*, Q_t. L'action est choisie en fonction de la Q valeur de Q_q :

$$a = max_a Q_q(s_{t+1}, a) \tag{5.13}$$

Le *Q-network* est alors mis à jour :

$$Q_q(s_t, a) = R_{t+1} + \gamma Q_t(s_{t+1}, a) \tag{5.14}$$

Après un certain nombre d'itérations, paramètre de l'algorithme, on met à jour les paramètres du réseau cible :

$$Q_t(s, a) = Q_q(s, a) \tag{5.15}$$

Dueling DQN Comme les méthodes basées sur l'actor-critic que nous verrons plus en détails en section 5.1.2, le dueling-DQN de Wang et al. (2016) se base sur la notion d'avantage. L'avantage est défini comme :

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{5.16}$$

Le principal attrait de cette méthode est que dans certains cas, il n'est pas nécessaire de connaitre la valeur de chaque action à chaque pas de temps. C'est le cas par exemple quand une conséquence est devenue inévitable peu importe l'action choisie. En séparant explicitement deux estimateurs, cette architecture peut apprendre quels états sont les meilleurs sans avoir à apprendre les effets de chaque action à chaque état. Cette méthode est alors particulièrement intéressante dans les cas où il existe des actions qui n'ont pas ou peu d'impact sur l'environnement sous certaines circonstances (état courant s_t).

La Q-valeur est alors donnée par :

$$Q(s, a, \theta) = V(s, \theta) + A(s, a, \theta) - max_{a' \in |\mathcal{A}|} A(s, a, \theta) \tag{5.17}$$

ce qui forcera la valeur Q de l'action maximisante à être égale à V, résolvant ainsi le problème d'identifiabilité. En effet, utiliser simplement $Q(s, a, \theta) = V(s, \theta) + A(s, a, \theta)$ empêche de retrouver V et A à partir de Q, ce qui impacte les performances Wang et al. (2016).

L'utilisation de la moyenne des valeurs d'actions peut également être proposée :

$$Q(s, a, \theta) = V(s, \theta) + A(s, a, \theta) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a, \theta) \tag{5.18}$$

Enfin, le choix de l'action optimale a^* est généralement basé sur :

$$a^* = argmax_{a' \in \mathcal{A}} Q(s, a, \theta) \tag{5.19}$$

Categorical DQN Le *categorical* DQN Bellemare et al. (2017) utilise une forme de DQN classique sauf qu'au lieu d'approximer les valeurs de récompense futures attendues, il génère les distributions complètes discrétisées des résultats possibles. La principale motivation derrière ce changement est que les distributions peuvent avoir plusieurs pics. Le fait d'en faire la moyenne en une seule valeur attendue peut être inapproprié et conduire à des résultats imparfaits.

Noisy DQN Cette variante de DQN présentée dans Fortunato et al. (2017a) propose de réaliser la phase d'exploration d'une manière différente qu'avec une probabilité décroissante au cours du temps comme ϵ-greedy. Pour cela, il introduit du bruit en sortie du réseau afin de biaiser les valeurs d'actions et explorer d'autres trajectoires quand du bruit est présent.

Pour réaliser cela, la structure du réseau est modifiée.Classiquement on a $y = wx + b$ avec $x \in R^p$, la couche d'entrée, $w \in \mathbb{R}^{q \times p}$ la matrice de poids et $b \in \mathbb{R}^q$ les biais. Mais dans un noisy-DQN on a :

$$y = (\mu^w + \sigma^w \odot \epsilon^w)x + \mu^b + \sigma^b \odot \epsilon^b \tag{5.20}$$

Où $\odot$ signifie multiplication élément par élément, $(\mu^w + \sigma^w \odot \epsilon^w)$ et $\mu^b + \sigma^b \odot \epsilon^b$ remplacent w et b. Les paramètres $\mu^w \in \mathbb{R}^{q \times p}$, $\mu^b \in \mathbb{R}^q$, $q^w \in \mathbb{R}^{q \times p}$ et $\sigma^b \in \mathbb{R}^q$ sont apprenables alors que $\epsilon^w \in \mathbb{R}^{q \times p}$ et $\epsilon^b \in \mathbb{R}^q$ sont du bruit aléatoire variable.

Dans la première version Fortunato et al. (2017a), les valeurs des μ sont initialisées suivant une distribution uniforme $\mu_{i,j} = \mathcal{U}[-\sqrt{\frac{3}{p}}, \sqrt{\frac{3}{p}}] \forall i, j$ avec p le nombre de variables d'entrée. Les σ sont initialisés à $\sigma_{i,j} = 0.017 \forall i, j$. Cette initialisation s'inspire de valeurs éprouvées en apprentissage supervisé Fortunato et al. (2017b)

Les méthodes basées sur les DQN ont été très utilisées au début de l'apprentissage par renforcement profond. Bien qu'elles restent encore utilisées, les méthodes basées sur les gradients de politiques leur sont souvent préférées. Bien qu'il ne semble pas encore y avoir de consensus, celles-ci présentent souvent, mais pas toujours, de meilleurs résultats. Nous nous devons donc de les tester.

Méthode par gradient de politique

Jusque-là nous avons considéré des méthodes pour estimer des fonctions de valeur des actions puis les utiliser pour sélectionner les meilleures actions suivant une politique. Ici, nous considérons des méthodes qui, au contraire, apprennent une politique paramétrée qui peut sélectionner des actions sans consulter une fonction de valeur. Une fonction de valeur peut toujours être utilisée pour apprendre les paramètres de la politique, mais elle n'est pas nécessaire pour la sélection des actions.

Soit $\theta \in R^d$ le vecteur de paramètres de la politique. $\pi(a|s, \theta) = Pr\{A_t = a|S_t, \theta_t = \theta\}$ représente la probabilité que l'action a soit prise au temps t pour un environnement dans l'état s au temps t avec des paramètres θ. L'apprentissage se fait grâce au gradient d'une mesure de performance notée $J(\theta)$. On a alors la mise à jour suivante :

$$\theta_{t+1} = \theta_t + \alpha \nabla \hat{J}(\theta_t) \tag{5.21}$$

avec $\nabla \hat{J}(\theta_t)$ l'estimation stochastique dont l'espérance approxime le gradient de la mesure de performance

de θ_t.

Toute méthode suivant ce schéma peut être appelée méthode par gradient de politique (*policy gradient method*). La politique peut alors être paramétrée de n'importe quelle façon tant que $\pi(a|s,\theta)$ est différentiable, c'est-à-dire, tant que $\nabla_\theta \pi(a|s,\theta)$ existe et a toujours une valeur finie. En pratique, pour assurer l'exploration, il est nécessaire que la politique ne soit jamais totalement déterministe.

Pour des espaces d'actions discrets, une paramétrisation naturelle est d'utiliser une fonction de préférence notée $h(s,a,\theta) \in \mathbb{R}$ pour chaque paire action-état. L'action avec la préférence la plus haute donnera la meilleure probabilité suivant par exemple une distribution softmax exponentielle :

$$\pi(a|s,\theta) = \frac{exp(h(s,a,\theta))}{\sum_b exp(h(s,b,\theta))} \tag{5.22}$$

Pour des actions continues, plutôt que d'apprendre des probabilités pour toutes les actions (discrétisation de l'espace), il est possible d'apprendre des distributions de probabilités. Par exemple, l'espace d'action peut être les nombres réels avec actions choisies selon une loi normale $p(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(x-\mu)^2}{2\sigma^2})$. La politique peut alors être définie comme la densité de probabilité normale d'une action à valeur scalaire réelle avec les paramètres de moyenne, μ, et écart-type, σ, donnée par une fonction dépendante de l'état :

$$\pi(a|s,\theta) = \frac{1}{\sigma(s,\theta)\sqrt{2\pi}} exp\left(-\frac{(a-\mu(s,\theta))^2}{2\sigma(s,\theta)^2}\right) \tag{5.23}$$

avec $\mu : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$ et $\sigma : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}^+$, des approximations de fonctions paramétrées. On peut alors séparer les paramètres de θ en deux parties, $\theta = [\theta_\mu, \theta_\sigma]^T$ pour estimer chacun de ces paramètres. À cause des caractéristiques de ces variables, il sera alors naturel d'utiliser une fonction linéaire pour μ et l'exponentielle d'une fonction linéaire pour σ :

$$\mu(s,\theta) = \theta_\mu^T x_\mu(s) et \sigma(s,\theta) = exp(\theta_\sigma^T x_\sigma(s)) \tag{5.24}$$

avec $x(s)$ le vecteur d'état.

Actuellement, les paramètres θ utilisés sont très souvent les poids des connexions entre chaque neurone d'un réseau de neurones profond. Dans ce cas-là, on a :

$$h(s,a,\theta) = \theta^T x(s,a) \tag{5.25}$$

Un avantage immédiat de la sélection des actions selon la distribution softmax est que la politique approximative peut se rapprocher d'une politique déterministe, alors qu'avec la sélection d'actions ϵ-greedy il y a toujours une probabilité ϵ de sélectionner une action aléatoire. L'avantage le plus simple est certainement que la paramétrisation de la politique est finalement une fonction plus simple à approximer. En plus des avantages pratiques de la paramétrisation de la politique par rapport à la sélection d'actions ϵ-greedy, il y a aussi un avantage théorique important. Avec la paramétrisation continue de la politique, les probabilités d'action changent doucement en fonction du paramètre appris, alors que dans la sélection ϵ-greedy, les probabilités d'action peuvent changer drastiquement pour un changement arbitrairement

petit dans les valeurs d'action estimées, si ce changement résulte en une action différente ayant la valeur maximale. En grande partie à cause de cela, des garanties de convergence sont disponibles pour les méthodes policy-gradient uniquement pour les méthodes de valeur d'action Mei et al. (2020) Wang et al. (2019a).

Un grand nombre de méthodes ont été proposées. Par exemple, REINFORCE Williams (1992) ou REINFORCE avec Baseline Mnih and Rezende (2016), une méthode Monte-Carlo. Ici nous nous intéresserons particulièrement aux méthodes basées sur l'actor-critic.

Actor-critic Les méthodes dites actor-critic apprennent des approximations à la fois de la politique et des fonctions de valeur, où l'"acteur" est une référence à la politique apprise, et "critique" fait référence à la fonction de valeur apprise, généralement une fonction d'un couple état-valeur. Les méthodes comme REINFORCE avec baseline apprennent ces deux composants mais la fonction de valeur n'est utilisée que comme baseline, pas comme critique ni utilisée pour du *bootstrapping*. Celui-ci peut introduire des biais sur la représentation des états mais qui sont finalement bénéfiques car ils réduisent la variance. Afin de bénéficier de ces avantages dans le cas des *policy-gradient methods*, nous utilisons des méthodes *actor-critic* avec un critique basé sur le *bootstrapping*.

Dans un premier temps, on peut considérer une méthode one-step. Comme elle sont parfaitement incrémentales, elles sont particulièrement simples à utiliser et évitent la complexité de compréhension des cas avec traces d'éligibilité. Dans ce cas, les mises à jour de θ se font à partir des retours d'une étape et de la fonction de valeur d'état comme base :

$$\theta_{t+1} = \theta_t + \alpha(G_{t:t+1} - \hat{v}(S_t, w))\frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{5.26}$$

$$\theta_{t+1} = \theta_t + \alpha\gamma_t\frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{5.27}$$

Dans ce cas, l'apprentissage classique de la fonction de valeur se base sur un semi-gradient TD(0). Les généralisations aux méthodes à n étapes, puis à un algorithme à λ retour, sont simples. Le retour à une étape est simplement remplacé par $G^{\lambda}_{t:t+k}$ et G^{k}_{t} respectivement. De plus, les méthodes à base de traces d'éligibilité utilisent des traces d'éligibilité séparées pour l'acteur et le critique.

Advantage Actor-critic (A2C) Cette variante d'actor-critic Mnih et al. (2016) se base sur la notion d'avantage. Utiliser la fonction de valeur d'état, V, comme base et la fonction de valeur de l'action, Q, permet de définir la fonction d'avantage :

$$A(s_t, a_t) = Q_w(s_t, a_t) - V_v(s_t) \tag{5.28}$$

Cette fonction décrit dans quelle mesure il est préférable de prendre une action spécifique par rapport à l'action générale moyenne à l'état donné. Pour continuer à n'apprendre que la fonction de valeur d'état,

on peut utiliser la relation suivante basée sur l'équation d'optimalité de Bellman Bellman (1954) :

$$Q(s_t, a_t) = \mathbb{E}[r_{t+1} + \gamma V(s_{t+1})] \tag{5.29}$$

et donc :

$$A(s_t, a_t) = r_{t+1} + \gamma V_v(s_{t+1} - V_v(s_t)) \tag{5.30}$$

Le gradient de politique peut alors être réécrit :

$$\nabla_\theta J(\theta) \sim \sum_{t=0}^{T-1} \nabla_\theta log \pi_\theta(a_t|s_t) A(s_t, a_t) \tag{5.31}$$

Asynchronous Advantage Actor-Critic (A3C) Cet algorithme utilise plusieurs agents, chacun ayant ses propres paramètres et une copie de l'environnement. Ils interagissent avec leurs environnements respectifs de manière asynchrone, en apprenant à chaque interaction. Chaque agent est contrôlé par un réseau global. Au fur et à mesure que chaque agent acquiert plus de connaissances, il contribue à la connaissance totale du réseau global. Ces expériences uniques sont ensuite utilisées pour mettre à jour le réseau de neurones global qui est partagé par tous les agents. Ce réseau influence toutes les actions des agents, et chaque nouvelle expérience de chaque agent améliore plus rapidement le réseau global. Comme il y a plusieurs instances de cet agent, l'entraînement sera beaucoup plus rapide et meilleur. La présence d'un réseau global permet à chaque agent de disposer de données d'entraînement plus diversifiées. Cette configuration imite l'environnement réel dans lequel vivent les humains, car chaque humain acquiert des connaissances grâce aux expériences d'autres humains, ce qui permet à l'ensemble du "réseau global" de s'améliorer.

Soft Actor-Critic (SAC) Dans cette variante de Haarnoja et al. (2018), trois réseaux de neurones sont utilisés. Comme précédemment, deux sont consacrés à l'apprentissage de la fonction de valeur, V paramétrée par ψ et de la politique paramétrée par θ. Un dernier est utilisé pour apprendre la soft Q fonction, paramétrée par ϕ. On a alors :

$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\phi(s_t, a_t) + log \pi_\theta(a_t|s_t)) \tag{5.32}$$

$$\nabla_\phi J_Q(\psi) = \nabla_\psi Q_\psi(a_t, s_t)(Q_\psi(s_t, a_t) - r(s_t, a_t) - \gamma V_\phi(s_{t+1})) \tag{5.33}$$

$$\nabla_\theta J_\pi(\theta) = \nabla_\theta log \pi_\theta(a_t|s_t) + (\nabla_{a_t} log \pi_\theta(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\theta f_\theta(\epsilon_t, s_t) \tag{5.34}$$

avec $a_t = f_\theta(\epsilon_t, s_t)$ assurant que la politique soit différenciable, ϵ_t est un vecteur de bruit généralement basé sur une distribution normale Haarnoja et al. (2018)

Twin-Delayed Deep Deterministic Policy Gradient Agents (TD3) Dans les algorithmes basés sur la fonction de valeur comme en deep Q-learning, les erreurs d'approximation de fonction peuvent mener à la surestimation de la valeur estimée et à des politiques sous-optimales. Ce problème peut persister dans les méthodes à base d'actor-critic Fujimoto et al. (2018). Cette méthode se base alors sur du double Q-learning et utilise la valeur minimale de la paire de critiques pour éviter les surestimations. De plus, l'acteur est mis à jour moins fréquemment que les Q fonctions, généralement toutes les 2 itérations, et du bruit est ajouté afin d'éviter de trop exploiter les actions avec une Q valeur élevée.

Cet algorithme semble particulièrement efficace sur les problèmes à action continue.

Maintenant que nous avons fait un tour d'horizon des différents types d'algorithmes utilisable, nous pouvons réutiliser les environnements générés dans le chapitre 4.3.2 pour réaliser nos expériences d'apprentissage.

5.2 Cas d'application

Tous nos tests d'apprentissage se feront à partir de réseaux de neurones. Dans cette partie, nous détaillerons donc les différentes applications proposées. Pour chacune, nous parlerons de l'intérêt de les tester, des couches d'entrée et de sortie et de la façon de les initialiser, des fonctions de récompenses utilisées, et enfin de leurs résultats.

Pour commencer, il nous faut définir le déroulement d'un épisode.

5.2.1 Déroulement d'un épisode

Cela se déroule de manière très similaire aux simulations du problème du bandits présenté en chapitre 4.4.

Un épisode représente une simulation de 10 ans de l'exploitation d'un environnement. Le déroulement est visible en algorithme 10. Pour commencer l'environnement est initialisé. Cela se fait en sélectionnant un environnement aléatoire parmi tous ceux générés avec la procédure présentée précédemment. Toutes les espèces et leurs lois de probabilité associées sont initialisées et $t = 0$.

La première année de simulation est toujours complètement aléatoire (lignes 8 et 9) pour les méthodes à base d'apprentissage par renforcement. Pour notre expérience à base de bandit contextuel, cette phase n'est pas considérée. Cela permet d'initialiser les connaissances et croyances comme si nous commencions à gérer un environnement réel dans lequel les exploitants ont déjà leurs expériences et croyances. Pour les autres années, à chaque pas de temps, $nbExploitationDays = 150$, chaque pêcheur, $nbFishermen = 50$, doit déterminer l'action à effectuer (ligne 11), $a_k = \pi_k(b_k$. En fonction des applications, ils pourront tous avoir la même politique ou non. Cela sera précisé lors de chaque application.

Une fois toutes les actions déterminées, un pas est effectué en fonction de l'ensemble des actions. Générant ainsi les récompenses ainsi que l'ensemble des observations pour chaque pêcheur (ligne 14). Pour ne pas surcharger l'algorithme, nous ne détaillons pas la gestion de la probabilité que chaque pêcheur a d'observer le comportement des autres. Elle est gérée par la fonction $step(action)$ ligne 14. Celle-ci peut être faite par simple boucle sur l'ensemble des observations des pêcheurs et par tirage

aléatoire en fonction de la probabilité d'observation. Pour gagner en complexité algorithmique, on peut plutôt proposer l'utilisation d'un premier tirage par loi binomiale déterminant le nombre d'observations extérieures *réussies* puis un tirage aléatoire déterminant lesquelles.

Chaque pêcheur met ensuite à jour ses croyances suivant $\tau_k(b_k, o_k)$ (lignes 15 à 18) ainsi que ses variables internes telles que ses gains annuels ou ses prises totales de chaque espèce. Les observations des décideurs sont également ajoutées à leur historique d'observations (lignes 19 à 21).

À la fin de chaque jour d'exploitation, si certains algorithmes de décisions sont basés étape (*step-based*), leur phase d'apprentissage est exécutée (lignes 22 à 26).

À la fin de chaque année, les décideurs mettent à jour leurs croyances (ligne 30) et déterminent les quotas à appliquer pour l'année suivante (ligne 35). Comme pour les pêcheurs, si leur algorithme de décision est basé étape, les étapes correspondantes sont exécutées (ligne 32 et 33).

À la fin de chaque épisode, les étapes d'apprentissage des différents algorithmes sont exécutées avant de passer à l'épisode suivant (ligne 37).

Dans le chapitre précédent, nous avons vu que le problème du bandit manchot contextuel pouvait permettre une représentation efficace des pêcheurs mais arrivait rapidement à ses limites. Ainsi, dans un premier temps, nous allons nous intéresser à différentes méthodes d'apprentissage basée sur les croyances pour la gestion des actions des pêcheurs.

5.2.2 Apprentissage des pêcheurs basé sur les croyances

Quelle que soit l'application, chaque agent (pêcheur) sera caractérisé par les informations suivantes :

— Argent nécessaire par an (agrégeant les coûts fixe d'entretien / salaires / etc.), A ;

— Coûts fixes par sortie, Cs ;

— Coût du carburant par unité de distance parcourue Cc ;

En plus de cela, chaque année, la pêcherie mettra en place des règles d'exploitation par espèce que chaque pêcheur devra respecter.

Cette application est certainement celle qui se rapproche le plus de la réalité. Elle permet de représenter un pêcheur cohérent avec ce qu'il a pu observer / entendre dans le passé. Au fur et à mesure des pêches, il aura plus ou moins confiance en la rentabilité d'une zone (espérance de gains, μ, et risques, fonction de σ ainsi que du nombre et la temporalité des observations) et orientera ses choix en fonction. Il ne se souviendra donc que du traitement des informations qu'il en a fait plutôt que des informations brutes.

Le but de ces applications est d'estimer l'état réel du système, à partir des observations, en maintenant à jour des lois de probabilité des états sachant les observations. On définit alors $\tau(b, a, o) \to b'$, la fonction de mise à jour des croyances. Quand les dynamiques D sont connues, la probabilité d'un nouvel état s' peut être calculée par la formule de Bayes :

$$b'(s') = \tau(b, a, o)(s') \propto \sum_s D(s', o|s, a)b(s) \tag{5.35}$$

Algorithme 10 Episode

Initialisation de l'environnement
pour $i \in [0; nbYear]$ **faire**
  si Première année **alors**
    Initialise une haute valeur de quota
  fin si
  pour $j \in nbExploitationDays$ **faire**
    pour $k \in nbFishermen$ **faire**
      si First year **alors**
        $action_k$ = selection aléatoire
      sinon
        $action_k = determineAction(beliefs_k, t)$
      fin si
    fin pour
    $observations, reward = step(action)$
    pour $k \in nbFishermen$ **faire**
      mise à jour $beliefs_k = \tau_k(beliefs_k, observations_k)$
      mise à jour des variables de $fisherman_k$ (nombre total de prises par espèces, gains totaux, ...)
    fin pour
    si $rand() < pObservationDecideur$ **alors**
      ajoute $observations_d$ aux $observations$ des décideurs
    fin si
    pour chaque algorithme d'apprentissage pour pêcheur, FLA_l **faire**
      si FLA_l est step-based **alors**
        exécute la procédure d'apprentissage FLA_l
      fin si
    fin pour
    $t = t + 1$
  fin pour
  $t = i \times 365$
  Met à jour les croyances des décideurs $beliefs_d = \tau_d(beliefs_d, observations_d)$
  si $i > 1$ **alors**
    calcul de la récompense
    exécute la procédure d'apprentissage des algorithmes step-based
  fin si
  $quotas = determineFishingQuotas(beliefs_d, t)$
fin pour
exécute la procédure d'apprentissage des algorithmes basés épisode

D n'est cependant que rarement connue. Il est donc nécessaire d'estimer une distribution de probabilité $p(D)$ que l'on peut représenter par une loi de Dirichlet Katt et al. (2019) pour chaque paire (s, a). De manière plus générale, chaque transition (s', o, s, a) est associée à un compteur $\mathbf{X}_{s\ a}^{s'\ o}$ et l'ensemble des compteurs $\mathbf{X}$ décrit la loi de probabilité des dynamiques D, $p(D_{sa}) = \mathbf{X}_{sa}$.

On peut alors définir la probabilité d'obtenir le nouvel état s' et les observations o sachant l'état courant s, l'ensemble des compteurs $\mathbf{X}$, et l'action a :

$$p(s', o|s, \mathbf{X}, a) = \frac{\mathbf{X}_{s\ a}^{s'\ o}}{\sum\limits_{s'\ o} \mathbf{X}_{s\ a}^{s'\ o}} \tag{5.36}$$

Finalement, chaque agent pourra être caractérisé par les lois de probabilité qu'il maintiendra à jour. Il cherchera à estimer les lois de probabilité de captures de chaque espèce pour chaque zone afin de maximiser son rendement et de respecter les quotas.

Croyances Dans cette application, nous utiliserons l'espace des observations brutes présenté en section 4.3.1. Celles-ci seront traitées afin d'obtenir les croyances qui nous serviront pour la couche d'entrée. On définit alors les croyances, b, comme étant l'ensemble des espérances de prises et écarts-types de chaque espèce par zone. On ajoutera aussi un indice de confiance, Ic par zone, proportionnel au nombre d'observations en fonction du temps. Soit sous forme matricielle :

$$b = \begin{bmatrix} \mu_{0,0} & \sigma_{0,0} & \cdots & \mu_{0,ns} & \sigma_{0,ne} & \mathbb{E}[e_0] & Ic_0 \\ \mu_{1,0} & \sigma_{1,0} & \cdots & \mu_{1,ns} & \sigma_{1,ne} & \mathbb{E}[e_1] & Ic_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mu_{na,0} & \sigma_{na,0} & \cdots & \mu_{na,ns} & \sigma_{na,ns} & \mathbb{E}[e_{na}] & Ic_{na} \end{bmatrix} \tag{5.37}$$

Les calculs des différents constituants sont détaillés en section 5.2.2.

Informations sur l'activité Les décisions vont bien évidemment dépendre des pêcheurs. Il faut alors stocker les différentes variables le caractérisant : A, C_s, C_c.

De plus, pour prendre en compte la notion de quotas ou de quantité d'exploitation, l'historique des captures par espèce ainsi que les gains totaux pour l'année en cours doivent également être utilisés. Avec $o_{bp,tannee}$ les observations brutes personnelles de l'année en cours, on a alors :

$$\forall i \in [0, l], C_{annee,i} = \sum_{o \in o_{bp,tannee}} C_i(o) \tag{5.38}$$

$$G = \sum_{o \in o_{bp,tannee}} G(o) \tag{5.39}$$

Les coûts en carburant dépendant de la distance entre le port d'attache et chaque zone, cette distance est également stockée, notée Dz.

Enfin, en fonction du type d'action des décideurs, il faudra également prendre en compte soit le quota, soit l'indicateur du niveau d'exploitation de chaque espèce. Dans les deux cas, cela prendra la forme d'un vecteur, noté Dd.

Représentation de la couche d'entrée Les réseaux de neurones gèrent efficacement des vecteurs de nombres réels en entrée, toutes les informations devront être mises sous ce format. La couche d'entrée sera alors la concaténation de la matrice de croyance aplatie, A, C_s, C_c, des vecteurs C_{annee}, Dz et Dd. Soit :

$$Input = flatten(b) + A + C_s + C_c + C_{annee} + Dz + Dd \tag{5.40}$$

Traitement, $\tau(b, o) \rightarrow b'$, des observations brutes

Après chaque observation, o, la fonction de mise à jour des croyances, $\tau(b, o)$ est exécutée. Celle-ci va recalculer les différentes valeurs $\mu_{i,j}, \sigma_{i,j}, Ic_i, \forall j \in [0, ne]$ avec ne, le nombre d'espèces et i, l'identifiant de la zone de pêche. Tous les coefficients Ic seront également mis à jour, car représentant la fiabilité des estimations.

Calcul de $\mu_{i,j}$ L'environnement étant non stationnaire, nous avons décidé de prendre en compte la temporalité par un facteur d'actualisation (*discount factor* en anglais), γ_c. Il est difficile de lui déterminer une valeur optimale. Le modèle de Graham-Schaefer étant particulièrement stable, la variation de biomasse et des captures d'un jour à l'autre n'est pas très importante. L'évolution des paramètres des lois de probabilité caractérisant les captures est donc un processus lent. En section 4.4, nous avons vu que la valeur optimale de l'importance donnée au passé est extrêmement dépendante de la variabilité des espèces et du nombre d'observations. Les conclusions de nos tests ont montré que donner 2 fois plus d'importance à l'observation actuelle qu'à une observation datée d'un an semblait être un bon compromis. Nous réutiliserons donc cette valeur ici en fixant $\gamma_c = \sqrt[365]{0.5}$.

$$\mu_{i,j} = \frac{1}{\sum\limits_{t_{releve} \in o,i} \gamma_c^{t-t_{releve}}} \sum_{t_{releve} \in o,i} C_j(t_{releve}) \gamma_c^{t-t_{releve}} \tag{5.41}$$

avec t le pas de temps actuel ; t_{releve} le pas de temps du relevé correspondant ; $C_j(t_{releve}$ les captures de l'espèce j à l'instant t_{releve}. Après simplification algorithmique, on obtient la formule itérative suivante en fonction de la dernière espérance calculée, $\mu_{p,j}$, pour laquelle on aura stocké son coefficient de pondération $cp = \sum\limits_{t_{releve} \in o_p,i} \gamma_c^{t-t_{releve}}$ au temps tp :

$$\mu_{i,j} = \frac{\mu_{p,j} \times cp \times \gamma_c^{t-tp} + C_j(t)}{1 + cp \times \gamma_c^{t-tp}} \tag{5.42}$$

Calcul de $\sigma_{i,j}$ Pour le calcul de σ pondéré, la variation de $\mu_{i,j}$ rend impossible l'utilisation de méthodes itératives. On utilisera donc le formule classique :

$$\sigma_{i,j} = \sqrt{\frac{\sum\limits_{t_{releve} \in o,i} \gamma_c^{t-t_{releve}} (C_j(t_{releve}) - \mu_{i,j})^2}{\sum\limits_{t_{releve} \in o,i} \gamma_c^{t-t_{releve}}}} \tag{5.43}$$

Calcul de Ic Ic doit représenter la fiabilité des estimations. Il doit donc prendre en compte à la fois le nombre de données nous ayant servi à réaliser les estimations, mais également la temporalité de celles-ci. Il est donc important de mettre à jour tous les coefficients Ic à chaque pas de temps. On pose alors :

$$Ic_i = \sum_{t_{releve} \in o,i} \gamma_c^{t-t_{releve}} \tag{5.44}$$

Soit de manière itérative, pour la zone sélectionnée à l'instant t :

$$Ic_i(t) = 1 + Ic_i(t-1) \times \gamma_c \tag{5.45}$$

et pour les autres :

$$Ic_i(t) = Ic_i(t-1) \times \gamma_c \tag{5.46}$$

On peut donc simplifier les expressions précédentes pour i la zone sélectionnée à l'instant t :

$$\mu_{i,j} = \frac{\mu_{p,j} \times (Ic_i(t) - 1) + C_j(t)}{Ic_i(t)} \tag{5.47}$$

$$\sigma_{i,j} = \sqrt{\frac{\sum\limits_{t_{releve} \in o,i} \gamma_c^{t-t_{releve}} (C_j(t_{releve}) - \mu_{i,j})^2}{Ic_i(t)}} \tag{5.48}$$

Récompenses et algorithmes à tester

Le but de chaque pêcheur est de maximiser ses gains. La fonction de récompense instantanée est donc le rapport des gains par l'argent nécessaire par an :

$$\frac{1}{\mathbb{E}[e_y]} \sum_{i=0}^{nf} e_i \tag{5.49}$$

À cela, il faut également ajouter la prise en compte des objectifs définis par les décideurs. Dans le cas où les décideurs instaurent des quotas, on peut directement appliquer un malus $r = -1$ si le quota est dépassé. Si les décisions se font par indicateurs du niveau d'exploitation, nous pouvons calculer le niveau de capture optimal caché C_{msy} de chaque espèce et nous en servir pour juger de la qualité de l'exploitation. Si les captures totales d'une espèce sont supérieures aux captures optimales alors que cette espèce doit

être préservée, $r = -1$. Pour les espèces à niveau d'exploitation normal, un dépassement de C_{msy} est autorisé à condition que ce ne soit pas pour s'enrichir plus que nécessaire : $r = -1$ *ssi* $G_{annee} > A$.

Ainsi, pour cette application, il semble que nous pourrions utiliser un algorithme classique tel que le Deep Q-learning ou encore le plus récent Quantile Regression DQ-learning Dabney et al. (2018). Les méthodes de type actor-critic Konda and Tsitsiklis (2000) Grondman et al. (2012) A2C/A3C Babaeizadeh et al. (2016) Mnih et al. (2016) semblent également très bien réagir sur de nombreux problèmes. Enfin, l'algorithme Proximal Policy Optimization Schulman et al. (2017) Chen et al. (2018) (PPO) a également récemment montré des résultats comparables, voire meilleurs, qu'A2C. Nous devrons donc tester l'ensemble de ces algorithmes.

Entraînement des pêcheurs se basant sur les croyances

Les récompenses des pêcheurs dépendent très fortement des choix des décideurs et inversement. Ayant accès aux paramètres cachés du système, nous avons fait le choix d'entraîner ces deux entités de manière indépendante. Ainsi dans cette partie nous présenterons les résultats de l'entraînement des pêcheurs en utilisant les quotas optimaux calculables à partir des variables cachées.

Étant dans un système multi-agent, chaque agent jouant le rôle d'un pêcheur, nous pouvons entraîner plusieurs politiques en même temps. Cela permet également d'assurer une certaine variabilité dans les choix des différents agents, cohérents avec la réalité.

Nous avons fait le choix de tester 7 algorithmes différents : (1) DQN ; (2) Dueling DQN ; (3) Distributional DQN ; (4) Noisy DQN ; (5) A2C ; (6) SAC ; (7) PPO. Chacune des politiques ainsi créée gère 5 agents différents, n'ayant pas d'interactions entre eux, autres que par les observations de l'environnement. À cela on ajoute une politique avide gérant 5 agents et sélectionnant toujours la zone avec l'espérance de gain la plus élevée en se basant sur les croyances. Une politique de choix aléatoire gère également 10 agents. Ces deux politiques permettront surtout de comparer les résultats aux politiques entraînées.

Le nombre maximum d'épisodes est fixé à 10000. L'exploration étant assurée par la première année de choix aléatoire et par les observations extérieures, les algorithmes basés sur une politique ϵ-greedy utilisent un ϵ faible, décroissant sur l'intervalle $[0.2, 0.01]$. On fixe également la probabilité d'observation des pêcheurs à 1%.

La figure 5.1 montre l'évolution de la récompense moyenne par an de simulation au cours de l'entrainement pour chaque algorithme. On constate que les algorithmes SAC et PPO convergent beaucoup plus vite que les autres et vers de meilleures politiques. En environ 500 épisodes, ils apprennent à respecter les quotas (récompense > 0) et convergent rapidement vers une politique permettant d'obtenir entre 80 et 85% des gains de la stratégie optimale théorique.

À terme, tous les algorithmes présentent de bons résultats, mais les DQN-based algorithm convergent plus lentement et vers de moins bonnes politiques.

Il est intéressant d'analyser les gains par an en fonction des politiques. La figure 5.2 montre les résultats par rapport au gain de la stratégie optimale théorique. On remarque que la politique obtenue via DQN rapporte à peine plus qu'une politique aléatoire. Celle-ci n'est cependant pas soumise aux quotas. La politique avide en revanche est celle qui rapporte le plus, voir même plus que les gains optimaux durables

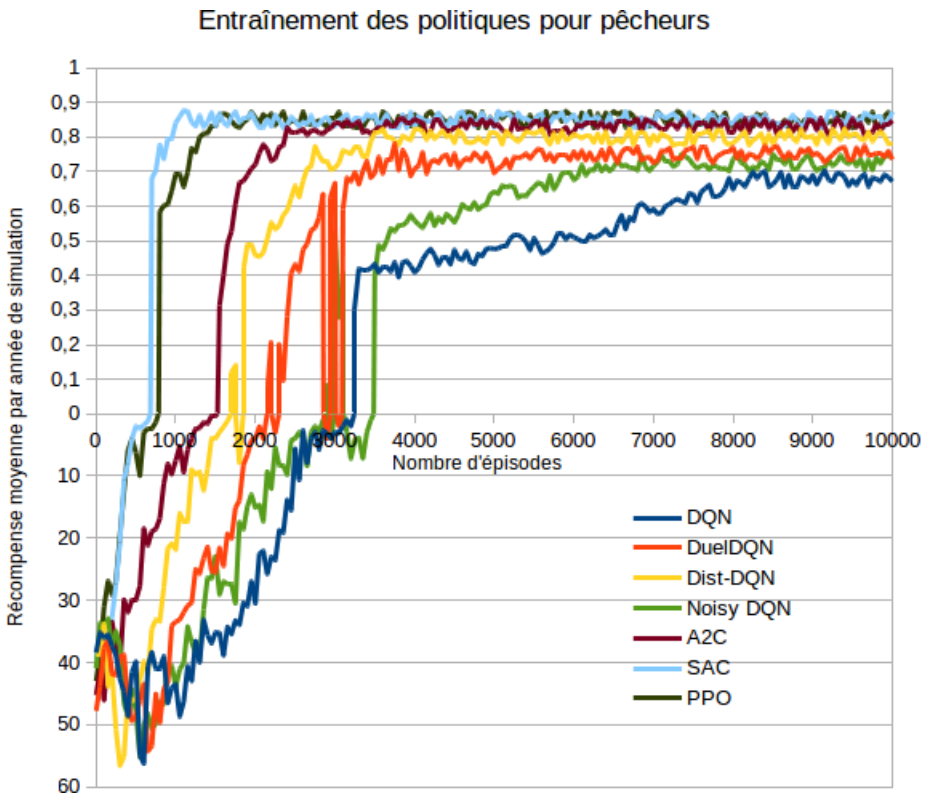


FIGURE 5.1 – Évolution de la récompense moyenne par année de simulation

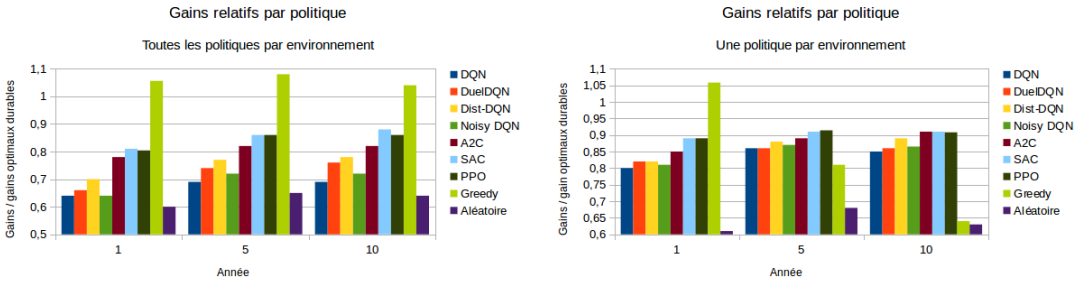
quand toutes les stratégies sont exécutées dans la même simulation (figure 5.2a). Elle est cependant valable ici, car elle ne représente que 10% des agents. En effet, la figure 5.2b montre que la politique avide mène très rapidement à des gains plus faibles à cause de la destruction des stocks. Grâce à ces deux figures, on remarque également que les gains de toutes les stratégies sont supérieurs quand on les applique à tous les pêcheurs. Cela s'explique par l'impact du non respect des quotas de la politique avide sur les stocks Poiron-Guidoni et al. (2020a).

De plus, il semble que la proportion d'observations extérieures n'impacte que très peu les résultats. Une valeur plus importante permet d'améliorer les gains de la première année, mais ceux-ci se stabilisent rapidement.

5.2.3 Apprentissage des décideurs basé sur les croyances

Une autre entité a également un rôle dans l'exploitation marine. Il s'agit des décideurs faisant des lois et des quotas quant à l'exploitation des espèces. La complexité de leur comportement n'est bien sûr pas parfaitement représentable ici. On peut simplifier leur rôle en attribuant un niveau d'exploitation à chaque espèce.

Pour cela, les décideurs se baseront sur certaines observations probabilistes. En effet, via observa-



(a) Gains pour chaque politique par année de simulation, toutes dans la même simulation

(b) Gains pour chaque politique par année de simulation, une politique par simulation

FIGURE 5.2 – Gain relatif des pêcheurs

tion des marchés, campagnes de mesures, embarquements auprès des exploitants etc., les décideurs réels peuvent obtenir certaines informations quant à l'exploitation. Ainsi, lors de la transition engendrée par une activité de pêche, il y aura une certaine probabilité (variable selon les tests) que cette pêche soit observée par les décideurs.

Le but final sera de déterminer un coefficient d'exploitation optimal ou un quota de pêche pour chaque espèce. Ces coefficients seront transmis à chaque pêcheur. Ceux-ci doivent donc en tenir compte pour ne pas recevoir de pénalités, soit purement économiques soit directement sous forme de récompenses négatives.

La détermination de quotas individuels peut être très difficile, car nécessite une vraie connaissance quant aux dynamiques de chaque espèce plutôt que de simplement connaître les tendances évolutives. Elle aurait cependant l'avantage de permettre une évaluation directe de la fonction de récompense intrinsèque des pêcheurs. À l'inverse, la détermination de niveau d'exploitation nécessiterait certainement de moins bonnes estimations, mais augmenterait la difficulté d'interprétation des informations pour les pêcheurs.

À l'inverse des pêcheurs, les décideurs n'agiront qu'une fois par an.

Comme expliqué précédemment, le but des décideurs est de déterminer, soit des quotas, soit des indicateurs du niveau d'exploitation pour chaque espèce. L'approche par indicateur semble à première vue sous-optimale, car elle ajoute une abstraction à l'apprentissage des pêcheurs. En effet, avoir un indicateur relatif entraîne forcément une interprétation difficile pour l'apprentissage pêcheur. Elle semble cependant plus simple à mettre en place à cause de la difficulté à déterminer des quotas réels à partir de simples observations soumises à l'incertitude des captures.

Dans les deux cas, ils auront un ensemble d'actions indépendantes à prendre, une par espèce. La couche de sortie ne sera donc toujours composée que d'un seul neurone à valeur réelle.

Apprentissage supervisé et apprentissage par transfert Les variables cachées sont par définition inconnues des différents acteurs au cours des simulations, mais rien ne nous empêche de nous en servir pour comparer les solutions proposées aux solutions optimales. Ainsi, nous pourrions ici complètement découpler l'apprentissage des décideurs de celui des pêcheurs.

En générant des comportements de pêcheurs plus ou moins optimaux via simulations, nous pouvons générer des observations pour un grand nombre d'environnements plus rapidement que dans un contexte d'apprentissage. De plus, l'apprentissage ne souffrira pas de tout le temps où les pêcheurs apprendront à respecter les objectifs écologiques des décideurs.

Nous pourrons donc utiliser un réseau à propagation avant *feed forward* classique avec pour objectif de déterminer le quota par espèce à partir des observations brutes de chacune. Les solutions proposées pourront alors être comparées à C_{msy}, déterminées via les paramètres cachés, au sein de la fonction de coût pour effectuer un premier apprentissage supervisé.

Une fois cet apprentissage réalisé, il est important de l'affiner au cours de simulations dépendantes des pêcheurs. C'est la phase d'apprentissage par transfert. Ceci permet au réseau de potentiellement affiner ses recommandations en fonction des comportements appris des pêcheurs.

Bien que cette séparation puisse sembler lourde de prime abord, nous pensons que c'est le moyen le plus efficace d'obtenir des quotas cohérents. En effet, juger la qualité des quotas serait très difficile pendant la phase d'apprentissage des pêcheurs où ceux-ci ne les respecteront que peu. Avoir confiance en leur qualité et ne reprendre l'apprentissage que lorsque les quotas seront relativement bien appliqués par les pêcheurs pourra grandement simplifier le problème.

Utilisation de croyances

À la différence des pêcheurs, la matrice de croyance *bd* doit tenir compte de l'effort de pêche appliqué par sortie. De plus ici, on va chercher à représenter l'évolution de chaque espèce au cours du temps. Δt doit être suffisamment important pour que le nombre de données permette des résultats significatifs statistiquement, problématique déjà étudiée en section 4.4.

On définit alors :

$$\mu_{i,j}(t) = \frac{1}{\sum\limits_{t_{data} \in o,i,j,\Delta t} \gamma_b^{t-t_{data}}} \sum_{t_{data} \in o,i,j,\Delta t} \frac{C_j(t_{data})}{E_f} \gamma_b^{t-t_{data}} \tag{5.50}$$

avec E_f l'effort que le bateau correspondant a fourni et $o, i, j, \Delta t$ l'ensemble des observations dans la zone i pour l'espèce j sur l'intervalle de temps $t - \Delta t$. On a alors :

$$bd(t) = \begin{bmatrix} \mu_{0,0}(t) & \mu_{0,1}(t) & \cdots & \mu_{0,ne}(t) & Ic_0 \\ \mu_{1,0}(t) & \mu_{1,0}(t) & \cdots & \mu_{1,ne}(t) & Ic_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mu_{m,0}(t) & \mu_{m,0}(t) & \cdots & \mu_{m,ne}(t) & Ic_m \end{bmatrix}$$

Ici, deux choix s'offrent à nous.

1. Utiliser ces informations en entrée d'un réseau de neurones et l'entraîner à déterminer lui-même les objectifs à atteindre. Cela étant le plus cohérent avec la structure du Deep Hierarchical Reinforcement learning.
2. Utiliser une méthode d'optimisation basée sur le modèle et les données.

Dans le premier cas, on utilisera simplement l'ensemble des $CPUE$ par intervalle de temps et le pas de temps associé en entrée afin de récupérer l'indicateur de niveau d'exploitation en sortie.

Dans le cas de l'optimisation, on modélisera l'évolution de la population au cours du temps en comparant les $CPUE_{simulees}$ aux $CPUE_{calculees}$. Soit la fonction d'évaluation suivante :

$$f : \sum_{\forall t} (CPUE_{simulees}(t) - CPUE_{calculees}(t))^2 \tag{5.51}$$

Sous la contrainte :

$$g : \forall t, \ \frac{|E(t+1) - E(t)|}{E(t)} < 0.2 \tag{5.52}$$

À la suite de cela, on obtiendra les différentes variables cachées de la dynamique de population que nous pourrons mettre en entrée d'un réseau de neurones pour obtenir le niveau d'exploitation.

Sinon, les différentes variables cachées étant estimées, nous pouvons déterminer le niveau de biomasse actuelle par rapport à la biomasse au MSY (voir section 4.3.2) et nous en servir pour générer un arbre de décision déterministe. Par exemple :

— Si $B_{msy} < 0.8B_{msy} \Rightarrow$ niveau d'exploitation faible ;

— Si $0.8B_{msy} < B(t) < 1.2B_{msy} \Rightarrow$ niveau d'exploitation normal ;

— Sinon, niveau d'exploitation élevé.

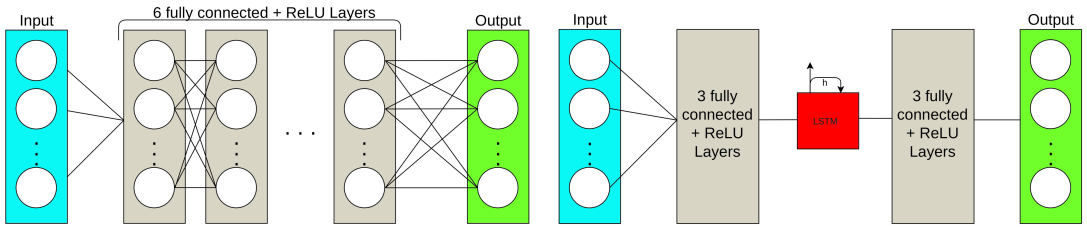
Récompenses et algorithmes à tester

Les pêcheurs prenant déjà en compte l'économie dans leur fonction de récompense intrinsèque, ici nous ne nous baserons que sur l'écologie. Nous proposons donc l'utilisation de :

$$r = \begin{cases} 1 & \text{Si nouvel objectif=Maintien} \\ 0 & \text{Si nouvel objectif=Ancien objectif} \\ -1 & \text{Sinon} \end{cases} \tag{5.53}$$

Le problème de l'apprentissage des décideurs est relativement classique. Un grand nombre de méthodes d'apprentissage pourront donc être testées. Les plus prometteuses semblent, encore une fois, être Deep Q-learning, Quantile Regression DQ-learning, type actor-critic A2C/A3C.

Finalement, la figure 5.3 montre le DNN pour chaque application. Dans 5.3a, le réseau d'applications basé sur les croyances avec 6 couches cachées entièrement contiguës avec fonction d'activation ReLU est utilisé pour les applications des pêcheurs et des décideurs. Dans 5.3b, on peut voir que pour les applications basées sur des observations, une couche LSTM est ajoutée au milieu des couches cachées pour traiter la temporalité. Pour les applications de décideurs, la couche de sortie est seulement un neurone avec une valeur réelle représentant le quota à appliquer pour l'espèce correspondante. La couche de sortie des applications destinées aux pêcheurs comporte un neurone par zone de pêche et un autre pour le choix "pas de pêche".



(a) Réseau pour les applications basées sur les croyances

(b) Réseau pour applications basées sur les observations

FIGURE 5.3 – Modèles de réseau de neurones

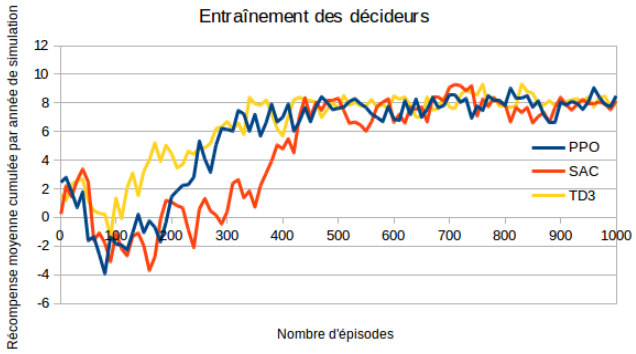


FIGURE 5.4 – Évolution de la récompense cumulée moyenne par année de simulation

Entraînement des décideurs se basant sur les croyances

Pour cette application, nous utiliserons la politique entraînée par l'algorithme SAC pour diriger les pêcheurs. Nous entraînerons donc 3 algorithmes, SAC, PPO et TD3 sur 1000 épisodes de 10 ans de simulations.

Cette fois, à cause de la nature de l'application, nous sommes obligés de les entraîner chacun indépendamment. La figure 5.4 montre l'évolution de la récompense cumulée moyenne par an pour chaque algorithme au cours de l'entraînement pour un taux d'observation des décideurs fixé à 10%. Les 3 algorithmes permettent d'atteindre des politiques ayant des résultats comparables, mais TD3 semble converger plus rapidement.

Les quotas proposés sont souvent proches de l'optimum théorique, mais ils peuvent encore s'avérer variables ce qui mène à des gains instables pour les pêcheurs. Ceux-ci sont également légèrement inférieurs à ceux proposés par l'application précédente.

De plus on constate qu'ici, le taux d'observation des décideurs à une très grande importance sur les résultats. Le tableau 5.1 représente le taux de variation moyen par rapport au quota optimal théorique,σ_{qopt}, l'année à partir de laquelle les quotas sont acceptables, y_{qAcc}, les gains des pêcheurs par rapport à l'optimal E_f/E_{opt}, et la variation de gains des pêcheurs au cours des années de simulations, $\sigma_{E_f/E_{opt}}$ en fonction du taux d'observation des décideurs, $O_d(o_i)$. On constate qu'il faut au moins 5 voire 10% d'ob-

$O_d(o_i)$	σ_{qopt} (%)	y_{qAcc}	E_f/E_{opt}	$\sigma_{E_f/E_{opt}}$
0.01	18.77	5	0.72	0.087
0.05	14.58	2	0.81	0.064
0.1	8.31	2	0.84	0.054
0.2	7.24	1	0.85	0.047
0.5	4.93	1	0.86	0.045

TABLE 5.1 – Résultats des décideurs en fonction du taux d'observation.

servations pour obtenir des résultats acceptables en seulement quelques années de gestion. En revanche, même un très haut taux d'observation ne permet pas de déterminer les quotas optimaux de façon fiable et les gains des pêcheurs restent assez variables.

5.2.4 Apprentissages basés sur les observations brutes

Ces méthodes d'apprentissage devront directement traiter les données des observations brutes sans aucun prétraitement.

Deux options s'offrent à nous :

— utiliser un réseau de neurones à propagation avant classique pour lequel on utilisera l'ensemble des observations brutes sur la couche d'entrée ;

— utiliser un réseau de neurones récurent de type LSTM (long-short term memory) Gers et al. (1999).

Avantages et inconvénients

La première approche présente l'avantage d'être plus facilement utilisable. En effet, avec un tel réseau, nous pourrions facilement n'utiliser qu'un réseau apprenant pour l'ensemble des agents. Il leur est cependant difficile de gérer le nombre d'observations optimal à conserver. En effet, un grand nombre d'observations devrait être optimal pour prendre en compte les variations sur le long terme et l'estimation des paramètres cachés au sein du réseau. Cela entraînerait cependant une explosion de la dimensionnalité de la couche d'entrée, rendant l'entraînement très difficile.

Ce problème est également présent dans le cas des réseaux de neurones récurrents (RNN). En revanche, les LSTM semblent résoudre ce problème au moins en partie Chung et al. (2014).

Cette méthode peut également poser un problème de prise en compte de la temporalité. En effet, la pêche n'est pas effective toute l'année. Le passage d'une saison de pêche à l'autre implique un grand saut dans la temporalité qui doit généralement être géré par la variable temporelle que nous mettons en entrée. Les LSTM sont fait pour gérer la notion de temporalité mais, à notre connaissance, ils sont toujours utilisés avec un pas de temps fixe. L'utilisation de cette seule variable temporelle, à cause de la création incontrôlée des états cachés, pourrait donc ne pas être suffisante. Cela pourrait alors impliquer de devoir ajouter de fausses entrées vides pendant le reste de l'année.

Ces deux approches seront donc à tester.

Couche d'entrée

Pêcheurs Dans le cas de l'utilisation d'un réseau feed-forward, la couche d'entrée se constituera donc, pour chaque pas de temps enregistré, de :

— l'ensemble des observations brutes (voir section 4.3.1) ;

— les différentes variables caractérisant l'activité déjà présentée en section 5.2.2 ;

— t, le pas de temps actuel de la simulation.

Cette structure sera également utilisée dans le cas de réseau récurent ou LSTM avec uniquement un pas de temps et le retour du traitement précédent.

Décideurs Ici nous allons traiter chaque espèce indépendamment les unes des autres. Ainsi la couche d'entrée n'aura besoin que des observations relatives à l'espèce actuellement traitée. On a donc uniquement un ensemble d'observations de la forme :

— nombre de captures ;

— identifiant de la zone, géré par *one-hot encoding* ;

— effort de pêche associé ;

— pas de temps de la simulation ;

À cela, on ajoute uniquement le pas de temps de simulation actuel. Comme pour l'apprentissage des pêcheurs, la question de l'utilisation de réseau récurent de type LSTM se pose, avec les mêmes avantages et inconvénients.

Importance du one-hot encoding Précédemment, nous avons dit que les observations brutes conservent l'identifiant de la zone dans lesquelles elles ont été faites. Cela ne pose aucun problème de traitement habituellement mais dans le cas de sa présence en entrée d'un réseau de neurones, des précautions s'imposent. En effet, bien qu'ayant une valeur numérique, l'identifiant reste un identifiant, c'est-à-dire, une variable qualitative et non quantitative. Ainsi, l'utilisation du numéro de zone 10 par exemple, donnerait 10 fois plus de poids à celui-ci au sein du traitement par le réseau de neurones que la zone avec l'identifiant 1. C'est un problème bien connu en traitement du langage naturel (*Natural Language Processing (NLP)*) où l'ensemble des mots est généralement contenu dans un dictionnaire, les identifiant chacun par un numéro par exemple.

Pour résoudre ce problème, les variables qualitatives doivent être gérées via la méthode du one-hot enoding Rodríguez et al. (2018). Cela consiste à utiliser un vecteur binaire constitué uniquement de 0 sauf un seul 1 à la position représentant l'identifiant voulu, plutôt que d'utiliser directement une valeur numérique. Par exemple, pour un identifiant pouvant prendre 10 valeurs, la 8ème sera encodée par le vecteur $[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]$.

Une variante appelée neural network Embeding Rodríguez et al. (2018) semble également remplir ce rôle dans certains cas, mais elle semble plus adaptée à la gestion de couples de variables.

Récompense, couche de sortie et algorithmes à tester

Les récompenses seront calculées exactement de la même façon que présentée en section 5.2.2. De même, la couche de sortie ne sera constituée que d'un neurone par zone, déterminant dans quelle zone aller pêcher.

Comme précédemment, les Deep Q-learning, quantile-regression DQL et actor-critic semblent être de bons candidats à la résolution de ce problème. Dans le cas de réseau récurent de type LSTM, on pourrait se baser sur l'architecture proposée par Le Tuyen et al. (2018) et le hDRQNv1/2. L'algorithme PPO a également déjà été utilisé avec des LSTM August and Hernández-Lobato (2018), il semble donc intéressant également.

Résultats des apprentissages basés sur les observations

L'utilisation directe des observations via un réseau LSTM mène à de grosses difficultés. L'apprentissage des pêcheurs mène à un respect des quotas optimal. Les gains sont cependant toujours inférieurs à ceux obtenus avec une politique basée sur les croyances. Ils sont quand même meilleurs que ceux d'une stratégie aléatoire donc il semble que l'apprentissage fonctionne, mais que cela ne soit pas optimal pour cette application.

Pour les décideurs, même après 10000 épisodes, les quotas proposés restent très éloignés des optimaux et mènent rarement à de bons résultats pour les pêcheurs.

Ces mauvais résultats sont certainement dus au nombre très important de pas de temps de simulation à gérer. À l'avenir cette application devra être repensée et le réseau certainement complexifié si nous voulons obtenir des résultats acceptables avec celle-ci.

5.3 Analyse des politiques proposées

5.3.1 Méthodologie de tests

Nous allons chercher à étudier l'impact de plusieurs variables sur les résultats économiques et écologiques : le niveau d'exploitation initial, le taux d'avidité (qui sera abrégé GR pour *Greedy Rate*), $GR \in [0, 0.2]$, représentant le ratio entre pêcheurs avides et pêcheurs respectant les recommandations, et l'effort que la flotille peut fournir par rapport à l'effort optimal durable $E/EMSY \in [0.5, 1.5]$. Ainsi, après la phase d'entraînement, nous générons 2000 nouveaux environnements de chaque niveau d'exploitation initial. De plus, nous définissons l'attractivité économique EA_i, d'une zone a :

$$att_a = \frac{\mathbb{E}[e_a]}{\sum_i^{nbArea} \mathbb{E}[e_i]} \tag{5.54}$$

avec $\mathbb{E}[e_a]$ l'espérance de gain provoquée par l'exploitation la zone a, en suivant les croyances du pêcheur concerné.

Ainsi, la politique avide π_g est définie telle que :

$$\pi_g : \forall a \in areaSet,\ P(A = a) = att_a \tag{5.55}$$

La politique finale utilisée est donc l'utilisation de π_{PPO} avec une probabilité $P_{\pi_{PPO}} = 1 - GR$ ou l'utilisation de π_g avec une probabilité $P_{\pi_g} = GR$.

5.3.2 Exploitation sélective

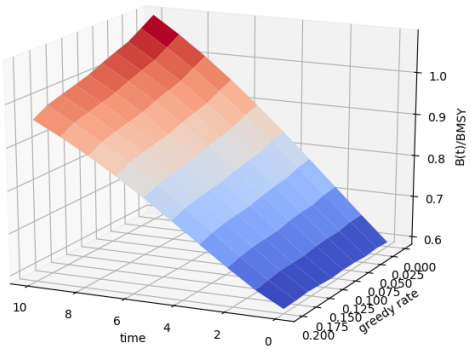
surexploitation Dans un contexte initial de surexploitation, le principal objectif va être d'au minimum maintenir les niveaux de stocks, voire de permettre un restockage. Nous nous sommes donc particulièrement intéressé à l'évolution de la biomasse par rapport à sa quantité au MSY, en fonction de la quantité d'effort applicable par la pêcherie, EE et le taux d'avidité GR.

À partir de la figure 5.5a, nous pouvons constater qu'en ayant une capacité d'effort limitée à 0.5MSY, même avec $GR = 20\%$, la biomasse atteint un niveau de stock très proche du MSY après 10 ans. Pour un effort limité au MSY, figure 5.5b, on constate que les niveaux de stocks atteint sont bien inférieurs, mais que toutes les espèces évoluent dans le bon sens. Un effort maximum applicable de 1.2MSY, figure 5.5c semble être la limite à partir de laquelle certaines espèces sont surexploitées. En effet, on constate une diminution des stocks après la première année et avec du comportement avide, ils peinent à retrouver leur niveau initial. Enfin, pour un effort applicable de 1.5MSY, figure 5.5d, une faible proportion, inférieure à 10%, et GR est suffisante pour accentuer la surexploitation.

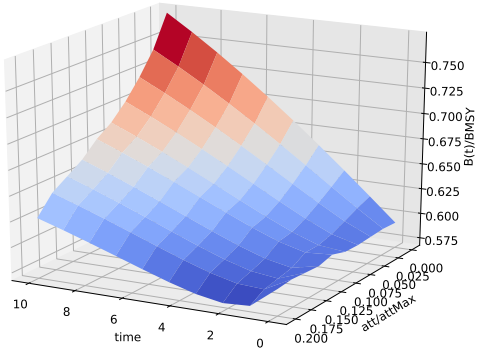
Les stratégies permettant un restockage rapide ne sont cependant pas viables économiquement. En effet, en figure 5.6a, nous pouvons observer que les gains par espèce, pour une exploitation avec un effort applicable de 0.5MSY et sans comportement avide, augmentent. Cependant, même après 10 ans, ils restent entre 50 et 65% des gains optimaux. Cette stratégie n'est donc certainement pas envisageable, mais elle pose le cadre écologique théorique optimal. Pour un effort applicable de $EMSY$ cependant, les gains croissent rapidement et finissent par atteindre entre 70 et 95% de GMSY.

L'augmentation de GR, bien que ne provoquant que peu de pertes pour les exploitants respectueux de l'environnement, provoque une grande variation dans l'homogénéité de l'attractivité des différentes espèces. En effet, on peut constater en figure 5.7a que pour $GR = 0$, l'effort appliqué à chaque espèce est lié à son attractivité économique, mais peu au temps. En revanche, un $GR = 0.2$ provoque d'importantes variations d'effort appliqué en fonction de l'attractivité au cours du temps. L'exploitation serait donc bien plus difficile à contrôler, car très variable.

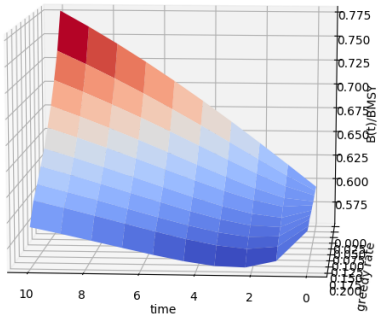
Sous-exploitation En cas de sous-exploitation, un effort applicable de $EMSY$ semble être quasi optimal écologiquement comme économiquement même en présence de comportements avides. En effet, sur la figure 5.8a on observe une diminution de la biomasse pendant les 10 années de simulation, jusqu'à atteindre une asymptote à $B(t) = BMSY$. Pour des niveaux de GR élevés, on observe un léger abus et une biomasse passant en dessous de $BMSY$, un effort applicable de $0.9EMSY$ permet de garantir $B(t = 10) > BMSY$. D'un point de vue économique, figure 5.8b, on remarque que l'augmentation du GR



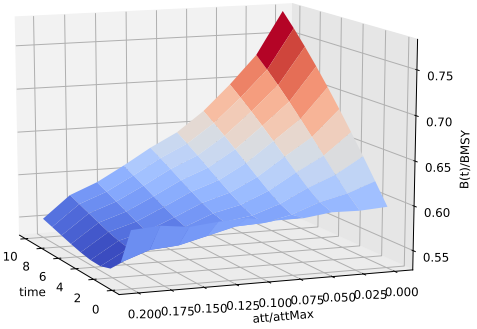
(a) $EE = 0.5EMSY$



(b) $EE = EMSY$



(c) $EE = 1.2EMSY$



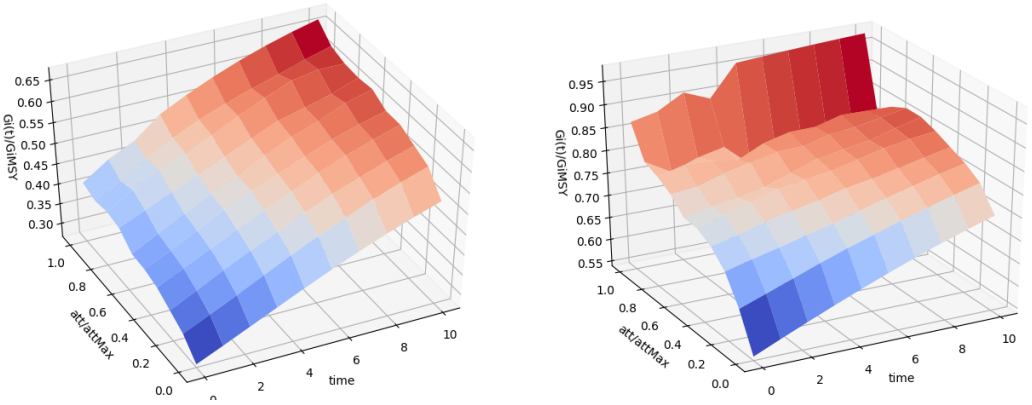
(d) $EE = 1.5MSY$

FIGURE 5.5 – Selective overexploitation, $B(t)/BMSY = f(t, greedyRate)$.

provoque des gains supérieurs tout au long de la simulation, mais ceux-ci restent quand même supérieur à $GMSY$ jusqu'à la fin.

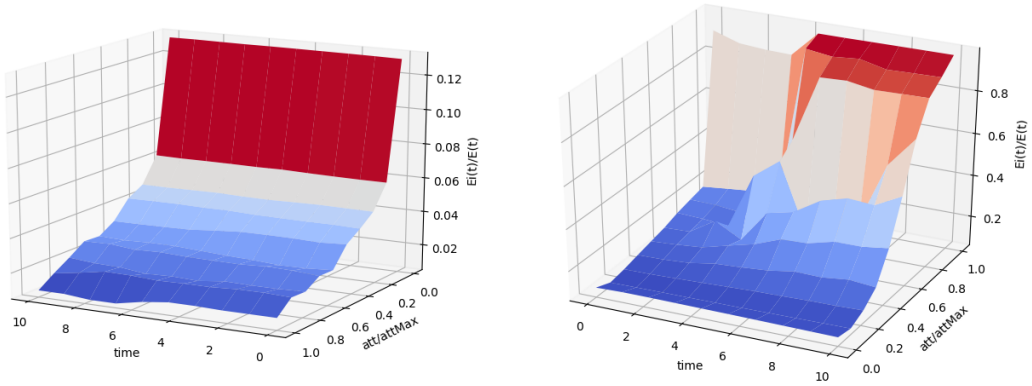
Cependant, même si les abus n'ont que peu de conséquences directement observables ici, la décroissance des gains au cours du temps pourrait provoquer des difficultés économiques en situation réelle et conduire à encore plus de comportements avides, alimentant le cercle vicieux de la tragédie des bien communs Hardin (1968).

Exploitation avec des niveaux de stock très variables Dans le cas d'une exploitation avec des niveaux de stock très variables, si nous regardons $B(t)/BMSY = f(t, GR)$ et $G(t)/GMSY = f(t, GR)$, les résultats sont très similaires à ceux d'une sous-exploitation avec une biomasse et des gains finaux légèrement inférieurs. Cependant, du fait de la variabilité des stocks, la représentation de l'évolution de la biomasse au cours du temps en fonction de l'attractivité économique, figure 5.9, nous mène à des conclusions bien différentes. Pour $GR = 0$, la biomasse tend vers $BMSY$ peu importe l'attractivité. En revanche, un faible $GR = 0.1$ suffit à provoquer de grandes diminutions de la biomasse avec l'augmentation



(a) $EE = 0.5EMSY$ (b) $EE = EMSY$

FIGURE 5.6 – surexploitation sélective, $G_i(t)/GiMSY = f(t, att/attMax)$ with $GR = 0$.



(a) $GR = 0$ (b) $GR = 0.2$

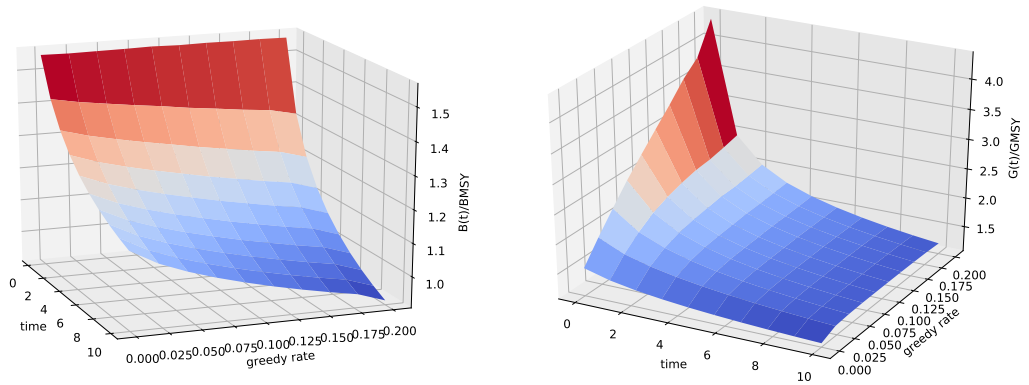
FIGURE 5.7 – surexploitation sélective, $E_i(t)/E(t) = f(t, att/attMax)$ with $EE = 1.2EMSY$.

de l'attractivité. Celles-ci devient alors rapidement inférieurs à $BMSY$.

Dans ce cas, même un faible effort applicable ne résout pas le problème. En effet, la trop grande variabilité de niveau de stock rend beaucoup trop sensible les stocks à forte attractivité économique et faible biomasse par rapport aux autres. Un durcissement des contrôles sur les espèces en question est alors nécessaire.

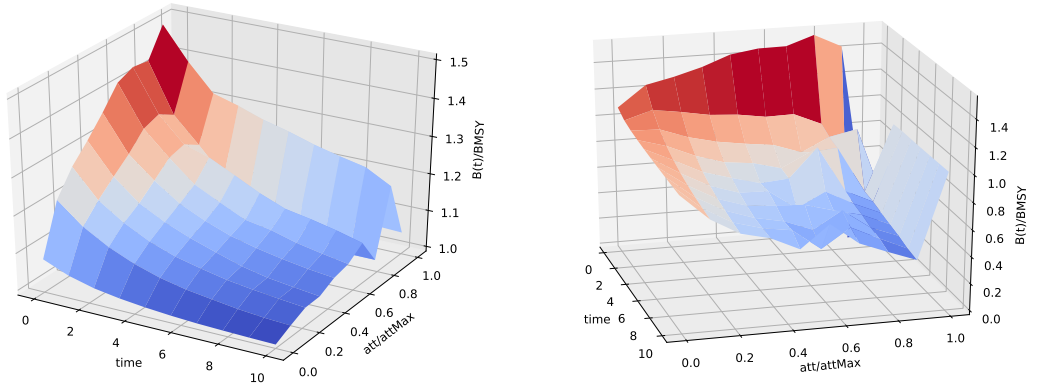
5.3.3 Exploitation non sélective

surexploitation En exploitation non sélective, la répartition d'un effort de $EMSY$ à travers les zones semble permettre une augmentation des stocks relativement rapide (figure 5.10a) pour des résultats économiques quasi optimaux. Sur la figure 5.10c, on constate que l'exploitation arrive à se stabiliser à un niveau économique viable même avec un GR élevé. L'exploitation de zones à fort potentiel économique



(a) $B(t)/BMSY = f(t, GR)$ (b) $G(t)/GMSY = f(t, GR)$

FIGURE 5.8 – Sous-exploitation sélective avec $EE = EMSY$.



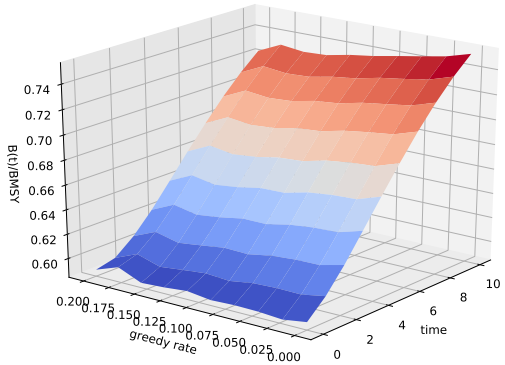
(a) $GR = 0$ (b) $GR = 0.1$

FIGURE 5.9 – Exploitation variable sélective $B(t)/BMSY = f(t, att/attMax)$.

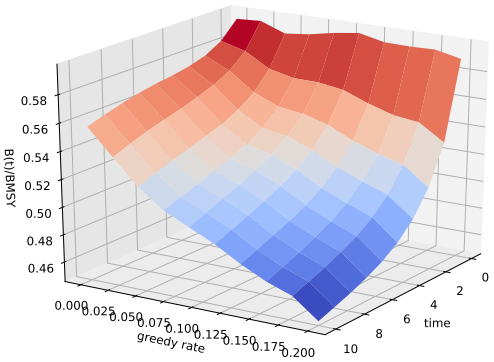
semble permettre des gains suffisants, même durant la période de restockage. Cela se traduit cependant par une irrégularité dans l'évolution des stocks. En effet, sur la figure 5.10d, nous pouvons constater qu'effectivement la majorité des stocks connaissent un accroissement de leur biomasse au cours du temps même avec un $GR = 0.2$. Cependant, les stocks à forte attractivité économique sont surexploités. Un compromis doit alors être décidé entre amélioration de la majorité des stocks et une légère diminution de ceux permettant la stabilité économique.

Enfin, la figure 5.10b montre que, contrairement à une exploitation sélective, un $EE = 1.2EMSY$ entraîne une surexploitation notable de l'ensemble des espèces, même en l'absence de comportements avides.

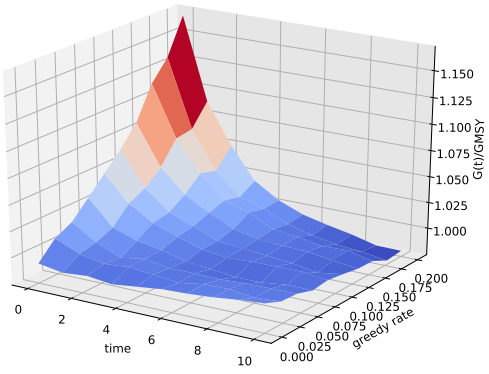
Sous-exploitation Contrairement au cas d'une exploitation sélective, un $EE = EMSY$ est déjà sensible à l'augmentation de GR provoquant une surexploitation à long terme et une diminution rapide



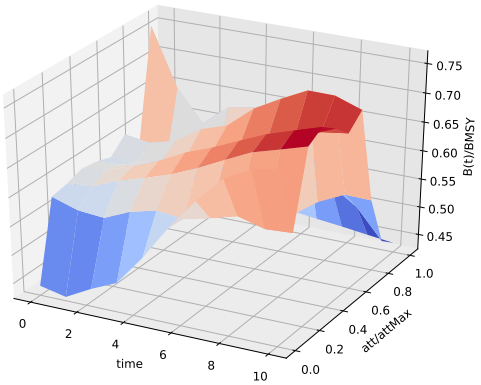
(a) $B(t)/BMSY = f(t, GR),\ EE = EMSY$



(b) $B(t)/BMSY = f(t, GR),\ EE = 1.2EMSY$



(c) $G(t)/GMSY = f(t, GR), EE = EMSY$

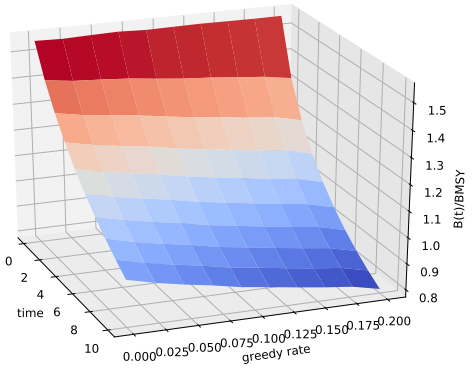


(d) $B(t)/BMSY = f(t, att/attMax),\ EE = EMSY,\ GR = 0.2.$

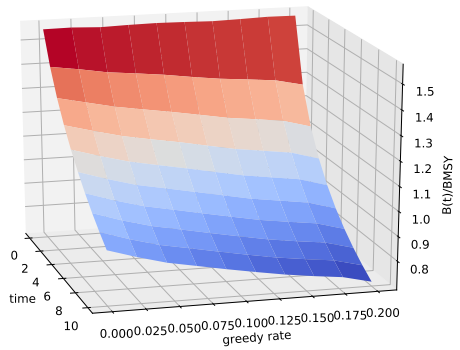
FIGURE 5.10 – surexploitation non sélective

des gains (figures 5.11a et 5.11c). Cela s'explique par le manque de sélection impliquant une répartition non-uniforme de l'effort par rapport à $EMSY$ de chaque espèce. Ses conséquences sont directement visibles sur la figure 5.11d, où la biomasse des espèces à faible attractivité économique n'évolue presque pas alors que toutes les autres sont en nette baisse. En revanche, la figure 5.11b montre que sur le plan écologique, ses résultats sont assez peu sensibles à une augmentation légère de l'effort. Ainsi, dans le cas d'une exploitation non sélective, il semble que même pour une situation de sous-exploitation, les contrôles sur les espèces à forte attractivité économique doivent être importants.

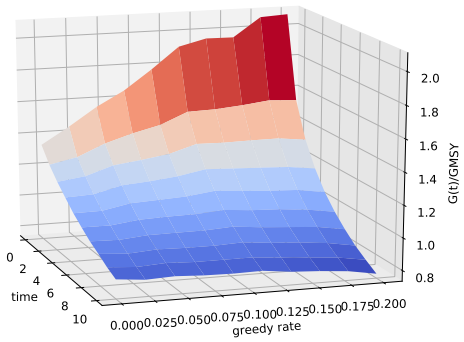
Exploitation avec des niveaux de stocks très variables Ce cas présente le même type de problème que lors d'une exploitation sélective avec des niveaux de stocks très variables. Du fait de la haute variabilité, la représentation de l'évolution des biomasses et des gains en fonction de GR au cours du temps ne suffit pas à conclure quant à la qualité de l'exploitation. Une diminution de l'effort jusqu'à $EE = 1.2EMSY$ est nécessaire afin d'obtenir des résultats acceptables. Sur les figures 5.12a et 5.12b,



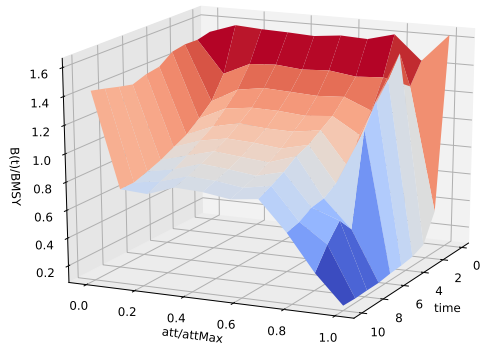
(a) $B(t)/BMSY = f(t, GR),\ EE = EMSY$



(b) $B(t)/BMSY = f(t, GR),\ EE = 1.2EMSY$



(c) $G(t)/GMSY = f(t, GR), EE = EMSY$

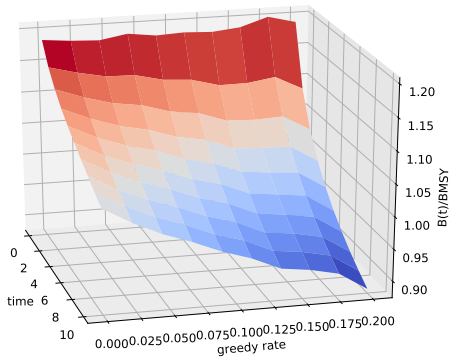


(d) $B(t)/BMSY = f(t, att/attMax),\ EE = 0,\ GR = 1.2EMSY$

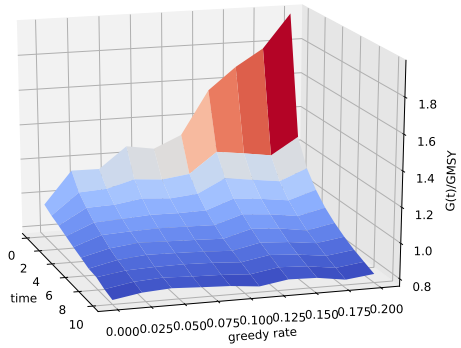
FIGURE 5.11 – Sous-exploitation non sélective

on remarque que les indicateurs écologiques et économiques sont très similaire à ceux du cas de sous-exploitation. En revanche, la figure 5.12c montre que même sans comportements avides, l'évolution des biomasses varie beaucoup en fonction de l'attractivité économique. Dans ce cas, les stocks de faible intérêt économique sont sous-exploités alors que les plus intéressants sont surexploités. Une majorité présente cependant des niveaux d'exploitation proches du MSY. Avec $GR = 0.2$, ce phénomène est accentué : les stocks à forte attractivité atteignent des niveaux dangereusement bas et, bien que proche du MSY, la majorité des autres stocks est en décroissance.

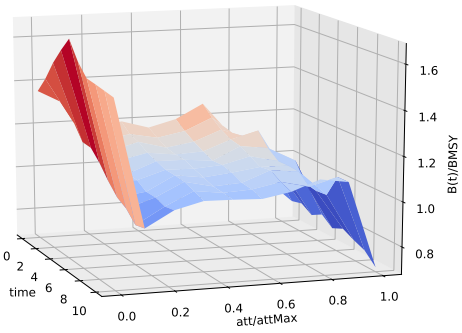
Sur ce type d'exploitation, il est donc très difficile de garantir le respect de l'ensemble des stocks. Des contrôles importants semblent donc nécessaires afin d'éviter une destruction des stocks les plus vulnérables et attractifs.



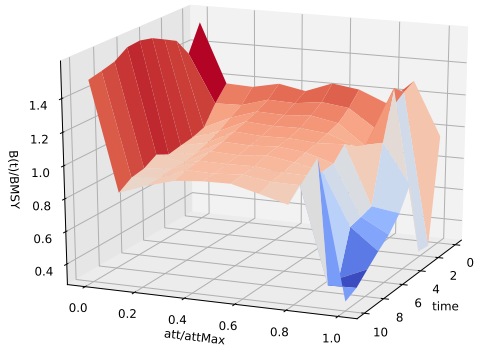
(a) $B(t)/BMSY = f(t, GR)$



(b) $G(t)/GMSY = f(t, GR)$



(c) $B(t)/BMSY = f(t, att/attMax), GR = 0$



(d) $B(t)/BMSY = f(t, att/attMax)$, $EE = EMSY$, $GR = 0.2$

FIGURE 5.12 – Exploitation variable non sélective $EE = -0.2$.

5.4 Conclusion du chapitre

Dans ce chapitre, nous avons montré différentes applications de l'optimisation et de l'apprentissage profond par renforcement.

En diminuant l'échelle temporelle de simulation pour arriver à la prise en compte de comportements individuels, à transitions stochastiques, dans un système complexe peu connu, l'apprentissage par renforcement trouve tout son intérêt. La formalisation en processus de décision markovien partiellement observable, ou de jeu de markov, permet de modéliser cela efficacement. Les comportements de petite échelle sont parfaitement gérés via apprentissage par renforcement. De plus, l'utilisation combinée d'optimisation, pour réaliser des calages imprécis à partir des connaissances, et d'apprentissage par renforcement, pour déterminer les stratégies à utiliser, permettent d'obtenir des méthodes de gestion adaptées, même en situation de connaissances faibles sur le système.

Finalement, nous avons étudié les résultats apportés par cette approche sur différents types d'envi-

ronnement, représentant différents cas de gestion de pêcheries. Les résultats sont concluants pour tous les cas, permettant d'aider à la bonne gestion et d'identifier les limites de l'approche. L'application présentée devra cependant être complexifiée afin de prendre en compte plus de phénomènes biologiques et économiques. En effet, ici les suppositions économiques et écologiques sont encore trop grossières pour mener à une utilisation réelle directe dans des cas complexes. Il est cependant possible de s'en servir comme aide à la décision, connaissant ses points forts et limites.

Cette approche permet également d'étudier les résultats de différents types d'environnement, représentant différents scénarios d'exploitation de la ressource halieutique. Dans toutes les situations, les résultats sont concluants. À court terme, ils permettront d'identifier les limites des méthodes et aideront à une meilleure gestion de la ressource.

Néanmoins, il semble nécessaire de complexifier l'approche présentée afin de prendre en compte des phénomènes biologiques et commerciaux qui correspondent davantage à la réalité du terrain. En effet, dans notre contexte, les hypothèses économiques et écologiques sont assez limitées et ne permettent pas de faire la transition à une application concrète pour des situations complexes.

D'un point de vue théorique, plus de tests seront nécessaires afin de généraliser l'approche et les conclusions. Les incertitudes de faisabilité et de sortie de modèle semblent par exemple difficilement gérables. Il sera alors nécessaire d'approfondir la question sur des cas d'application variés afin d'identifier au mieux les limites pour chaque cas d'incertitude.

CONCLUSION ET PERSPECTIVES

Comme nous avons pu le voir, l'étude de systèmes naturels, comme la gestion de ressources halieutiques, demande de nombreuses connaissances sur le système. En cas de données manquantes, incertaines voire parcellaires, les outils informatiques comme l'optimisation, l'optimisation robuste et l'apprentissage par renforcement permettent tout de même de proposer une aide à la décision.

Notre étude avait pour but d'identifier ce que les méthodes d'optimisation et d'apprentissage par renforcement, pouvaient apporter en termes d'aide à la décision dans une situation de faibles connaissances et de fortes incertitudes. Après avoir identifié et démontré la nature théorique des problèmes à résoudre, nous avons cherché à replacer les méthodes informatiques d'aide à la décision dans un contexte pluridisciplinaire, dans le but d'améliorer la gestion de la pêche en Corse.

Dans un premier temps, il a donc été question de la calibration d'un modèle de dynamique de population, d'abord à partir de données théoriques auto-générées puis à partir de données de la littérature. Nous avons vu, comme souvent quand il est question de calibrer un modèle sur des données réelles, qu'il est possible de modéliser cela par un problème des moindres carrés pour lequel un grand nombre de méthodes existent. Cependant, le manque de fiabilité des données et le problème d'équifinalité, souligné dans notre cas d'application, mènent indubitablement à l'incorporation de connaissances expertes au sein du processus de calibration. Il devient alors nécessaire d'utiliser des méthodes adaptées telles que les métaheuristiques pour l'optimisation difficile. Dans la plupart des cas, un espace continu non convexe est solution de la phase de calibration. Ces solutions, bien qu'évoluant de façon identique sur les données passées, peuvent réagir très différemment lors d'extrapolations pour simuler le futur.

En plus de cela, la difficulté d'acquisition de données due à la nature complexe et coûteuse du domaine halieutique, peut mener à des incohérences dans les entrées et donc rendre impossible le calage. Des méthodes de repérage et d'amélioration de ces données ont alors pu être proposées et utilisées, d'abord par différence aux résultats d'optimisation puis par une approche probabiliste Poiron-guidoni et al. (2020). Ces deux points rendent donc insuffisante l'exploitation des résultats de calibration par des méthodes

d'aide à la décision classiques.

Ainsi, nous avons orienté nos recherches vers les méthodes capables de gérer les incertitudes. L'optimisation robuste semblait particulièrement indiquée pour cela Poiron-Guidoni et al. (2020b). Cependant, l'espace des incertitudes étant l'espace des solutions de la calibration, il était difficile d'utiliser directement les solutions trouvées sans risque de biais dus aux méthodes de résolutions précédentes et à la possible non uniformité de la répartition des solutions trouvées au sein de l'espace des solutions réelles. Nous avons donc proposé une méthode basée sur une caractérisation plus approfondie de l'espace des incertitudes en utilisant une méthode de géométrie algorithmique. Elle nous a permis de garantir la robustesse de nos résultats et a ainsi permis l'identification de certaines solutions non pertinentes. La réduction automatique du nombre de solutions permet d'améliorer les résultats et de mieux prendre en compte des situations plus complexes.

De plus, la dimension d'évolution temporelle de l'étude nous permet d'utiliser des méthodes d'optimisation robuste ajustable, permettant de réduire l'espace des incertitudes au cours de l'optimisation via simulation. En effet, une année d'exploitation observée permet l'acquisition de nouvelles connaissances, donc une amélioration de la calibration et donc une diminution des incertitudes. Nous avons ainsi pu montrer que les résultats pouvaient être nettement améliorés.

Cependant, il nous est vite apparu que, plus l'espace des incertitudes augmentait, plus les solutions proposées par ces approches diminuaient en qualité. En l'absence d'intervention humaine ou de forte injection de connaissances expertes, cette approche trouve donc ses limites assez rapidement. Jusque-là, nous supposions un contrôle total des activités des pêcheurs, et donc que les quotas seraient parfaitement respectés. Pour prendre en compte ces incertitudes, il fallait modéliser les comportements des pêcheurs ce qui aurait été trop complexe à optimiser, car cela aurait nécessité une trop grande quantité de simulations Monte-Carlo et de paramètres à optimiser. Nous avons donc décidé d'orienter nos recherches vers les méthodes d'apprentissage par renforcement.

Elles permettent, au travers des processus de décision markoviens partiellement observables décentralisés, ou de jeux de markov, une modélisation optimale de la problématique de la pêche en Corse à toute échelle de modélisation temporelle et décisionnelle Poiron-Guidoni et al. (2020a). L'utilisation de méthodes d'optimisation, dans un premier temps pour générer des environnements d'apprentissage cohérents et variés, puis même au sein des simulations, montre l'importance et la complémentarité de ces deux approches. L'apprentissage peut s'avérer particulièrement efficace dans la prise de décision en fonction de connaissances partielles à l'état de croyances. L'optimisation, elle, s'avère particulièrement utile pour estimer l'état du système à partir des observations générées comme cela a pu être le cas en modélisant le problème sous forme de bandit manchot. Dans ce cas, elle a par exemple permis de déterminer les lois de probabilité des captures pour les pêcheurs. En utilisant des méthodes d'apprentissage profond par renforcement, l'optimisation a permis d'estimer les dynamiques de populations pour permettre une meilleure gestion de la part des décideurs.

Les résultats obtenus, bien que théoriques, nécessitent une complexification du modèle Leslie (1945, 1948). Ils permettent de montrer la puissance de l'approche et, nous espérons, d'initier de futurs travaux en biologie halieutique. Plus généralement, ces méthodes peuvent être applicables à tout processus d'aide

à la décision pour lequel toutes ou certaines étapes de sa construction et utilisation sont soumises à de fortes incertitudes.

D'un point de vue informatique, ces travaux nous ont permis de proposer une nouvelle méthode de gestion des incertitudes par optimisation robuste en caractérisant l'espace d'incertitudes par des diagrammes de Voronoï. Nous avons également pu proposer des méthodes d'identification et d'amélioration de données incohérentes en nous basant sur des estimations de la vraisemblance d'une situation réelle au regard d'un modèle de simulation. Nous avons également contribué à la mise en lumière des complémentarités d'utilisation des méthodes d'optimisation et d'apprentissage par renforcement. Bien que celles-ci soient limitées à notre contexte d'étude, la gestion d'un système aussi incertain que la petite pêche côtière Corse, nous semble d'une importance significative. Elles permettent donc de gérer plus efficacement des processus de décision markoviens partiellement observables décentralisés en couplant l'apport de connaissances engendrées par l'optimisation, et la prise de décision en situation de connaissances partielles de l'apprentissage par renforcement.

Nous pensons avoir également apporté une contribution au domaine de la biologie halieutique en proposant des méthodes de calibration, à la fois robustes et adaptable à un grand nombre de situations. Cela permet donc une amélioration de la phase d'acquisition des connaissances et donc une meilleure gestion future. Les méthodes d'optimisation robustes, et notamment d'optimisation robuste multiobjectif, proposées permettent également une meilleure visualisation des possibilités de gestion en fonction des connaissances. Les méthodes expérimentées en apprentissage par renforcement ne nous semblent pas encore assez matures pour être utilisées directement dans ce contexte, mais nous espérons que des améliorations seront proposées et que cela sera envisageable très bientôt.

Dans ce but, une complexification de l'environnement que nous avons proposé ici est nécessaire. Le plus important nous semble être l'incorporation d'un modèle économique réaliste. Lors de nos travaux, nous n'avons pas eu l'occasion de collaborer directement avec les sciences économiques pour identifier des modèles de marché cohérents avec les différentes solutions de gestion des pêches proposées par nos méthodes. Toutes nos applications étaient basées sur les quantités de captures ou supposaient des ventes optimales à prix constants, il nous est donc impossible d'évaluer leur impact sur une économie réelle. Bien que nous soyons assez confiants dans la généricité des méthodes proposées, une telle modification pourrait s'avérer significative et nécessiter certains ajustements pour répondre au mieux à la nouvelle problématique. Nous pourrions également prendre en compte certains paramètres moins significatifs, tels que la variation des coûts en carburant, d'entretien ou encore de personnel.

De plus, la réalité d'une exploitation marine est bien plus complexe qu'une simple équation d'évolution du stock. Les interactions entre espèces ou encore les migrations peuvent avoir un impact démesuré par exemple. L'utilisation de modèles biologiques plus complexes pourrait donc être nécessaire. Dans ce cas, la prise en compte de classes de tailles/masses pour chaque espèce pourrait nécessiter la prise en compte de différents engins au sens du processus d'apprentissage. Cela aurait pour conséquences d'agrandir considérablement l'espace d'action. Les observations s'en trouveraient également modifiées, induisant une adaptation des opérateurs de calculs de croyances ou d'optimisation par maximum de vraisemblance.

Nous avons également toujours cherché à proposer une gestion durable, et avons donc réalisé des

simulations d'au moins 10 ans. Dans un contexte de changement climatique, où les eaux se réchauffent d'année en année, impactant donc les populations marines, l'instabilité de cette situation est un paramètre à ne pas négliger. Une gestion robuste pourrait alors nécessiter la prise en compte des modèles climatiques pour extrapoler sur les changements à réaliser dès à présent dans les exploitations.

Néanmoins, nous pensons que l'approche par apprentissage et par optimisation s'avère suffisamment générique pour pouvoir s'adapter efficacement à tous ces problèmes, et bien d'autres, sur le long terme.

Chapitre 6

ANNEXES

Ces annexes sont constituées d'une section, ci-dessous, qui liste le lexique d'apprentissage par renforcement. La section suivante détaille les méthodes tabulaires d'apprentissage par renforcement. Dans une troisième section, nous donnons des informations complémentaires sur les méthodes d'optimisation sous contraintes et multimodales. La quatrième section présente la structure de la bibliothèque de méthodes d'optimisation que nous avons implémentée ainsi que le détail des méthodes que nous avons testées lors de nos travaux. Leur implémentation est validée dans la section suivante proposant des benchmarks d'optimisation globale, sous contraintes et multiobjectif. Enfin, dans la dernière section, nous détaillons des benchmarks supplémentaires réalisés sur les problèmes de calage de notre modèle afin de déterminer l'algorithme le plus efficace et sa paramétrisation optimale en fonction des cas.

6.1 Lexique d'apprentissage par renforcement

Actions : Les actions sont les méthodes de l'agent permettant d'interagir et de modifier l'environnement, et donc de passer d'un état à l'autre. La décision de l'action à choisir est prise grâce à la politique.

Actor-Critic : Lorsqu'on tente de résoudre un problème d'apprentissage par renforcement, on peut choisir entre deux méthodes principales : calculer les fonctions de valeur V des états ou les valeurs Q des couples états-actions et soit choisir les actions en fonction de celles-ci, soit calculer directement une politique qui définit les probabilités que chaque action soit entreprise en fonction de l'état actuel, et agir en fonction de celle-ci. Les algorithmes actor-critic combinent les deux méthodes afin de créer une méthode plus robuste en se basant sur la notion d'avantage.

Fonction d'avantage ou **Advantage function** : généralement désignée par A(s,a), la fonction d'avantage est une mesure de la qualité d'une action étant donné un certain état - ou plus simplement, quel est l'avantage de choisir une certaine action à partir d'un certain état. Elle est définie mathématiquement par :

$$A(s,a) = \mathbb{E}[r(s,a) - r(s)] \tag{6.1}$$

ou de manière plus évidente :

$$A(s,a) = Q(s,a) - V(s) \tag{6.2}$$

Agent : l'entité d'apprentissage et d'action d'un problème d'apprentissage par renforcement, qui essaie de maximiser les récompenses que lui donne l'environnement. C'est l'entité qui agit dans le système.

Bandits : généralisés en "bandits à k bras" d'après le surnom de "bandit manchot" donné aux machines à sous. Ils sont considérés comme le type le plus simple de tâches d'apprentissage par renforcement. Les bandits n'ont pas d'états différents, mais un seul - et la récompense prise en considération est uniquement la récompense immédiate. Par conséquent, les bandits peuvent être considérés comme ayant des épisodes à un seul état. Chacun des k bras est considéré comme une action, et l'objectif est d'apprendre la politique qui maximisera la récompense attendue après chaque action (ou tirage de bras). Les bandits contextuels sont une tâche légèrement plus complexe, où chaque état peut être différent et affecter le résultat des actions - donc à chaque fois le contexte est différent. Néanmoins, la tâche reste une tâche épisodique à un seul état et un contexte ne peut pas avoir d'influence sur les autres.

Problème de contrôle (*Control problem*) : problème consistant à trouver la politique optimale.

Équation de Bellman (*Bellman equation*) : formellement, l'équation de Bellman définit les relations entre un état donné (ou une paire état-action) et ses successeurs. Bien qu'il en existe de nombreuses formes, la plus courante rencontrée dans les tâches d'apprentissage par renforcement est l'équation de Bellman pour la valeur Q optimale, qui est donnée par :

$$Q^*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma max_{a'} Q^*(s',a')] \tag{6.3}$$

ou lorsqu'il n'y a pas d'incertitude (c'est-à-dire que les probabilités sont soit 1 soit 0) :

$$Q^*(s,a) = r(s,a) + \gamma max_{a'} Q^*(s',a') \tag{6.4}$$

où le signe astérisque indique la valeur optimale. Certains algorithmes, comme le Q-Learning, basent leur procédure d'apprentissage sur cette équation.

Tâches continues (*continuous tasks*) : tâches d'apprentissage par renforcement qui ne sont pas constituées d'épisodes, mais durent éternellement. Ces tâches n'ont pas d'état final. Pour simplifier, on suppose généralement qu'elles sont constituées d'un épisode sans fin.

Apprentissage par renforcement profond (*Deep reinforcement learning*) : l'utilisation d'un algorithme d'apprentissage par renforcement avec un réseau de neurones profond comme approximateur.Ceci est généralement utilisé pour faire face à des problèmes où le nombre d'états et d'actions possibles augmentent rapidement, et où une solution exacte n'est plus possible.

Facteur d'actualisation (γ) (*discount factor*) : le facteur d'actualisation, généralement désigné par γ, est un facteur multipliant la récompense future attendue, et varie dans l'intervalle $[0, 1]$. Il contrôle l'importance des récompenses futures par rapport aux récompenses immédiates. Plus le facteur d'atténuation est faible, moins les récompenses futures sont importantes, et l'agent aura tendance à se concentrer sur les actions qui lui rapporteront uniquement des récompenses immédiates.

Environnement : tout ce qui n'est pas l'agent, tout ce avec quoi l'agent peut interagir, directement ou indirectement. L'environnement change lorsque l'agent effectue des actions ; chaque changement de ce type est considéré comme une transition d'états. Chaque action effectuée par l'agent donne lieu à une récompense reçue par l'agent.

Épisode : tous les états qui se situent entre un état initial et un état final ; par exemple, la suite de positions et coups d'une partie d'échecs de son début jusqu'à sa fin. Le but de l'agent est de maximiser la récompense totale qu'il reçoit pendant un épisode. Dans les situations où il n'y a pas d'état terminal, nous considérons un épisode infini. Les épisodes sont complètement indépendants les uns des autres.

Tâches épisodiques (*Episod tasks*) : tâches d'apprentissage par renforcement qui sont composées de différents épisodes (ce qui signifie que chaque épisode a un état final).

Rendement attendu : parfois appelé "récompense globale" et parfois noté G, c'est la récompense attendue sur un épisode entier.

Reprise d'expérience (*experience replay*) : comme les tâches d'apprentissage par renforcement n'ont pas d'ensembles d'apprentissages pré-générés à partir desquels elles peuvent apprendre, l'agent doit conserver des enregistrements de toutes les transitions d'état qu'il a rencontrées (ou au moins les n dernières) afin de pouvoir en tirer des leçons plus tard. Le tampon mémoire utilisé pour stocker ces données est souvent appelé Experience Replay. Il existe plusieurs types et architectures de ces mémoires tampons, mais les plus courantes sont les mémoires tampons cycliques (qui garantissent que l'agent s'entraîne sur son nouveau comportement plutôt que sur des choses qui ne sont plus pertinentes) et les mémoires tampons basées sur l'échantillonnage de réservoirs (qui garantissent que chaque transition d'états enregistrée a une probabilité égale d'être insérée dans la mémoire tampon).

Exploitation et Exploration : Les tâches d'apprentissage par renforcement ne disposent pas d'ensembles d'apprentissage pré-générés à partir desquels elles peuvent apprendre - elles créent leur propre expérience et apprennent "à la volée". Pour ce faire, l'agent doit essayer de nombreuses actions différentes dans de nombreux états différents afin d'essayer d'apprendre toutes les possibilités disponibles et de trouver le chemin qui maximisera sa récompense globale ; c'est ce qu'on appelle l'exploration, car l'agent explore l'environnement. D'un autre côté, si l'agent ne fait qu'explorer, il ne maximisera jamais la récompense globale - il doit aussi utiliser les informations qu'il a apprises pour y parvenir. C'est ce qu'on appelle l'exploitation, car l'agent exploite ses connaissances pour maximiser les récompenses qu'il reçoit. Le compromis entre les deux est l'un des plus grands défis des problèmes d'apprentissage par renforcement, car les deux doivent être équilibrés afin de permettre à l'agent d'explorer suffisamment l'environnement, mais aussi d'exploiter ce qu'il a appris et de répéter le chemin le plus gratifiant qu'il a trouvé.

Politique gloutonne ou avide, politique ϵ-gloutonne (*greedy, ϵ-greedy policy*) : une politique avide signifie que l'agent effectue constamment l'action pour laquelle l'espérance de récompense est la plus élevée. De toute évidence, une telle politique ne permettra pas à l'agent d'explorer l'espace des possibilités. Afin de permettre une certaine exploration, une politique ϵ-greedy est souvent utilisée à la place : un nombre (nommé ϵ) dans la plage de [0,1] est sélectionné, et avant de sélectionner une action, un nombre aléatoire dans la plage de [0,1] est sélectionné. Si ce nombre est supérieur à ϵ, l'action avide

est sélectionnée - mais s'il est inférieur, une action aléatoire est sélectionnée. Donc, si $\epsilon = 0$, la politique devient la politique avide, et si $\epsilon = 1$, l'exploration sera toujours privilégiée.

Processus de décision Markovien (MDP) : la propriété de Markov signifie que chaque état dépend uniquement de son état précédent, de l'action sélectionnée prise à partir de cet état et de la récompense reçue immédiatement après l'exécution de cette action. Mathématiquement, cela signifie : $s' = s'(s, a, r)$, où s' est l'état futur, s est son état précédent et a et r sont l'action et la récompense. Aucune connaissance préalable de ce qui s'est passé avant s n'est nécessaire - la propriété de Markov suppose que s contient toutes les informations pertinentes. Un processus de décision de Markov est un processus de décision basé sur ces hypothèses.

Basé sur un modèle et sans modèle (*Model-based, model-free*) : model-based et model-free sont deux approches différentes qu'un agent peut choisir lorsqu'il tente d'optimiser sa politique. Prenons l'exemple de l'apprentissage du Blackjack. Il est possible de le faire de deux manières : la première consiste à calculer à l'avance, avant le début de la partie, les probabilités de gain de tous les états et toutes les probabilités de transition d'états pour toutes les actions possibles, puis à agir simplement en fonction de vos calculs. La deuxième option consiste à jouer simplement sans aucune connaissance préalable, et à obtenir des informations par "essai et erreur". En utilisant la première approche, on modélise essentiellement l'environnement, alors que la seconde approche ne nécessite aucune information sur l'environnement. C'est exactement la différence entre la méthode basée sur un modèle et la méthode sans modèle ; la première méthode est basée sur un modèle, tandis que la seconde est sans modèle.

Monte Carlo (MC) : les méthodes Monte Carlo sont des algorithmes qui utilisent un échantillonnage aléatoire répété afin d'obtenir un résultat. Elles sont souvent utilisées dans les algorithmes d'apprentissage par renforcement pour obtenir des valeurs attendues ; par exemple, le calcul d'une fonction de valeur d'état en retournant au même état à plusieurs reprises et en calculant la moyenne de la récompense cumulative réelle reçue à chaque fois.

Sur Politique et hors politique (*On-policy/Off-policy*) : chaque algorithme d'apprentissage par renforcement doit suivre une certaine politique afin de décider des actions à effectuer à chaque état. Cependant, la procédure d'apprentissage de l'algorithme ne doit pas tenir compte de cette politique pendant l'apprentissage. Les algorithmes qui se préoccupent de la politique qui a donné lieu à des décisions d'action dans les états précédents sont appelés algorithmes on-policy, tandis que ceux qui l'ignorent sont appelés off-policy. Un algorithme hors politique bien connu est le Q-Learning, car sa règle de mise à jour utilise l'action qui donnera la valeur Q la plus élevée, alors que la politique réelle utilisée pourrait restreindre cette action ou en choisir une autre. La variante de Q-Learning sur politique est connue sous le nom de Sarsa, où la règle de mise à jour utilise l'action choisie par la politique suivie.

Politique (π) (*policy*) : la politique, désignée par π (ou parfois $\pi(a|s)$), est une correspondance entre un certain état s et les probabilités de sélection de chaque action possible étant donné cet état. Par exemple, une politique avide produit pour chaque état l'action avec la plus grande valeur Q attendue.

Probème de prédiction (*prediction problem*) : problème consistant à estimer la fonction de valeur de π.

Q-Learning : Q-Learning est un algorithme d'apprentissage par renforcement hors politique, considéré

comme l'un des plus basiques. Dans sa forme la plus simplifiée, il utilise une table pour stocker toutes les valeurs Q de toutes les paires état-action possibles. Il met à jour cette table en utilisant l'équation de Bellman, tandis que la sélection des actions se fait généralement avec une politique ϵ-greedy. Dans sa forme la plus simple (aucune incertitude dans les transitions d'état et les récompenses attendues), la règle de mise à jour du Q-Learning est :

$$Q(s,a) = r(s,a) + \gamma max_a Q(s',a) \tag{6.5}$$

Une version plus complexe, et beaucoup plus populaire aujourd'hui, est la variante Deep Q-Network (qui est parfois même appelée simplement Deep Q-Learning ou simplement Q-Learning par abus de langage). Cette variante remplace la table d'état-action par un réseau de neurones afin de faire face aux tâches à grande échelle, où le nombre de paires état-action possibles peut être énorme.

Valeur Q (fonction Q) (*Q-value (Q-function)*) : habituellement désignée par $Q(s,a)$ (parfois avec un indice π, et parfois comme $Q(s,a;\Theta)$ en Deep RL). La valeur Q est une mesure de la récompense globale attendue en supposant que l'agent est dans l'état s et effectue l'action a, puis continue à jouer jusqu'à la fin de l'épisode en suivant une certaine politique π. Son nom est une abréviation du mot "Qualité", et elle est définie mathématiquement comme :

$$Q(s,a) = \mathbb{E}\left[\sum_{n=0}^{N} \gamma^n r_n\right] \tag{6.6}$$

où N est le nombre d'états depuis l'état s jusqu'à l'état terminal, γ est le facteur d'actualisation et r est la récompense immédiate reçue après avoir effectué l'action a dans l'état s.

Algorithmes REINFORCE : les algorithmes REINFORCE sont une famille d'algorithmes d'apprentissage par renforcement qui mettent à jour les paramètres de leur politique en fonction du gradient de la politique par rapport aux paramètres de la politique. Le nom est généralement écrit en utilisant uniquement des majuscules, car il s'agit à l'origine d'un acronyme pour la conception originale du groupe d'algorithmes : "REward Increment = Nonnegative Factor x Offset Reinforcement x Characteristic Eligibility"

Apprentissage par renforcement (RL) : l'apprentissage par renforcement est, comme l'apprentissage supervisé et l'apprentissage non supervisé, l'un des principaux domaines de l'apprentissage automatique et de l'intelligence artificielle. Il s'intéresse au processus d'apprentissage d'un être arbitraire, formellement appelé agent, dans le monde qui l'entoure, appelé environnement. L'agent cherche à maximiser les récompenses qu'il reçoit de l'environnement et effectue différentes actions afin d'apprendre comment l'environnement réagit, et d'obtenir davantage de récompenses. L'un des plus grands défis des tâches de RL est d'associer des actions à des récompenses différées - qui sont des récompenses reçues par l'agent longtemps après que l'action génératrice de récompense ait été effectuée. Il est donc largement utilisé pour résoudre différents types de jeux, du Tic-Tac-Toe aux échecs, en passant par l'Atari 2600 et jusqu'au Go et StarCraft.

Récompense (*reward*) : Valeur numérique que l'agent reçoit de l'environnement en réponse directe

à ses actions. L'objectif de l'agent est de maximiser la récompense globale qu'il reçoit au cours d'un épisode, et les récompenses sont donc la motivation dont l'agent a besoin pour adopter un comportement souhaité. Toutes les actions produisent des récompenses, qui peuvent être divisées en trois types : les récompenses positives qui soulignent une action souhaitée, les récompenses négatives qui soulignent une action dont l'agent devrait s'écarter, et zéro, qui signifie que l'agent n'a rien fait de spécial ou d'unique.

Sarsa : l'algorithme Sarsa est en grande partie l'algorithme Q-Learning avec une légère modification afin d'en faire un algorithme on-policy. La règle de mise à jour du Q-Learning est basée sur l'équation de Bellman pour la Q-Valeur optimale, et donc dans le cas où il n'y a pas d'incertitudes dans les transitions d'états et les récompenses attendues, la règle de mise à jour du Q-Learning est la suivante :

$$Q(s,a) = r(s,a) + \gamma max_a Q(s',a) \tag{6.7}$$

Afin de transformer ceci en un algorithme on-policy, le dernier terme est modifié :

$$Q(s,a) = r(s,a) + \gamma Q(s',a') \tag{6.8}$$

où ici, les deux actions a et a' sont choisies par la même politique. Le nom de l'algorithme est dérivé de sa règle de mise à jour, qui est basée sur (s,a,r,s',a'), tous provenant de la même politique.

État (*state*) : chaque scénario que l'agent rencontre dans l'environnement est formellement appelé un état. L'agent passe d'un état à l'autre en effectuant des actions. Il convient également de mentionner les états terminaux, qui marquent la fin d'un épisode. Il n'y a plus d'état possible après qu'un état terminal a été atteint, et un nouvel épisode commence. Très souvent, un état terminal est représenté comme un état spécial où toutes les actions transitent vers le même état terminal avec la récompense 0.

Différence temporelle (TD) (*Temporal Difference*) : la différence temporelle est une méthode d'apprentissage qui combine à la fois la programmation dynamique et les principes de Monte Carlo ; elle apprend "à la volée" comme Monte Carlo, mais met à jour ses estimations comme la programmation dynamique. L'un des algorithmes de différence temporelle les plus simples est connu sous le nom de TD à une étape (*one-step TD*) ou TD(0). Il met à jour la fonction de valeur selon la règle de mise à jour suivante :

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1} - V(s_t)] \tag{6.9}$$

où V est la fonction de valeur, s est l'état, r est la récompense, γ est le facteur d'actualisation, α est le taux d'apprentissage, t est le pas de temps. Le terme qui se trouve entre les crochets est connu comme l'erreur de différence temporelle.

Upper Confident Bound (UCB) : L'UCB est une méthode d'exploration qui tente de garantir que chaque action est bien explorée. Considérons une politique d'exploration totalement aléatoire, c'est-à-dire que chaque action possible a la même chance d'être sélectionnée. Il y a une chance que certaines actions soient beaucoup plus explorées que d'autres. Moins une action est sélectionnée, moins l'agent peut être confiant quant à sa récompense attendue, et sa phase d'exploitation peut en être affectée. L'exploration par UCB prend en compte le nombre de fois où chaque action a été sélectionnée, et donne

un poids supplémentaire à celles qui sont moins explorées. En formalisant cela mathématiquement, l'action sélectionnée est choisie par :

$$action = argmax_a \left[R(a) + c\sqrt{\frac{lnt}{N(a)}} \right] \tag{6.10}$$

où $R(a)$ est la récompense globale attendue de l'action a, t est le nombre de pas effectués (combien d'actions ont été sélectionnées au total), $N(a)$ est le nombre de fois où l'action a été sélectionnée et c est un hyperparamètre configurable. Cette méthode est aussi parfois appelée "exploration par l'optimisme", car elle donne aux actions moins explorées une valeur plus élevée, ce qui encourage le modèle à les sélectionner.

Fonction de valeur (*Value function*) : généralement désignée par $V(s)$, la fonction de valeur est une mesure de la récompense globale attendue en supposant que l'agent est dans l'état s et qu'il continue à agir jusqu'à la fin de l'épisode en suivant une certaine politique π. Elle est définie mathématiquement par :

$$V(s) = \mathbb{E}\left[\sum_{n=0}^{N} \gamma^n r_n \right] \tag{6.11}$$

Bien qu'elle semble similaire à la définition de Q Value, il existe une différence implicite - mais importante : pour n=0, la récompense r_0 de $V(s)$ est la récompense attendue du simple fait d'être dans l'état s, avant toute action jouée, alors que dans Q Value, r_0 est la récompense attendue après qu'une certaine action ait été jouée. Cette différence permet également d'obtenir la fonction Avantage

6.2 Détails sur les méthodes tabulaires d'apprentissage par renforcement

Cette annexe est en grande partie basée sur le livre de Sutton and Barto (1998) détaillant une grande partie des méthodes d'apprentissage par renforcement classiques. Nous en faisons ici un condensé centralisé sur les méthodes tabulaires historiques à la base des méthodes les plus récentes que nous utilisons dans ce manuscrit. Celles-ci sont détaillées en section 5.1.2.

Ces méthodes sont utilisables dans le cas où les espaces d'actions et d'états sont suffisamment petits pour que la fonction de valeur soit représentable sous forme de tableau. Dans ce cas, les méthodes peuvent aller jusqu'à trouver l'exacte politique optimale, contrastant ainsi avec les méthodes par approximation qui sont décrites en section 5.1.2.

La première partie présente le cas particulier où un seul état est possible : le problème du bandit manchot.

Les trois parties suivantes décriront les classes fondamentales permettant de résoudre les problèmes de décisions markoviens finis :

— la programmation dynamique : bien développée mathématiquement mais requérant un modèle com-

plet et précis de l'environnement.

— les méthodes de Monte Carlo, conceptuellement plus simples mais peu adaptées pour des calculs pas à pas incrémentaux.

— les méthodes de différences temporelles (*Temporal-difference methods* qui ne nécessitent pas forcément de connaissances sur le modèles et sont très incrémentales mais complexes à analyser.

Ces méthodes diffèrent dans leur efficacité et vitesse de convergence en fonction des cas.

Enfin, les deux dernières parties présenteront comment combiner ces trois méthodes afin d'allier les meilleures fonctionnalités de chacune.

6.2.1 Bandit manchot à plusieurs bras

Ici, nous étudierons l'aspect évaluatif de l'apprentissage par renforcement sur des configurations simples qui n'incluent pas vraiment d'apprentissage pour agir sur d'autres situations que la présente. Il évite donc une grande partie de la complexité des problèmes d'apprentissage par renforcement complet, mais étudier ces problèmes et méthodes permet de mieux visualiser l'importance du processus évaluatif.

Ainsi, pour commencer, nous étudions une version simple du problème du bandit manchot à k bras.

Problème du bandit manchot à k bras Ce problème, présenté pour la première fois en 1933 Thompson (1933) et beaucoup étudié depuis Robbins (1985) Bellman (1956) peut être défini de la façon suivante. À chaque pas de temps il faut choisir entre k actions, chacune générant une récompense suivant une loi de probabilité stationnaire. L'objectif étant de maximiser la récompense totale après une période de temps, par exemple 1000 itérations.

Dans notre cas, chaque action a une récompense moyenne, appelée valeur de l'action. On a donc :

$$q_*(a) = \mathbb{E}[R_t | A_t = a] \tag{6.12}$$

avec :

— $q_*(a)$: la récompense attendue

— R_t : la récompense à l'instant t

— A_t : l'action choisie à l'instant t

En sachant la valeur de chaque action, le problème serait trivial. Ainsi, ces valeurs ne sont pas connues et le but est de les estimer. La valeur estimée à l'instant t est notée $Q_t(a)$ et nous cherchons a ce qu'il soit proche de $q_*(a)$. La ou les action(s) qui ont la valeur estimée la plus élevée peuvent être appelées actions avides ($greedy - actions$) et choisir l'une d'elles c'est exploiter les connaissances actuelles. Le terme *greedy* est également utilisé dans les domaines des heuristiques Pearl (1984) et représente le même concept, à savoir, choisir la solution qui semble la meilleure immédiatement. À l'inverse, choisir une *nongreedy* action c'est **explorer**, afin d'améliorer nos estimations de ces solutions. Ainsi, l'exploitation est la chose à faire pour maximiser la récompense sur le court terme, mais l'exploration peut permettre de meilleurs résultats sur le long terme.

Ainsi, par la suite, nous nous intéresserons à la façon de gérer le compromis exploration/exploitation.

Méthodes basées sur la valeur d'action Une façon simple d'estimer la valeur d'une action est la suivante :

$$Q_t(a) = \frac{somme\ des\ récompenses\ quand\ a\ est\ choisie}{nombre\ de\ fois\ où\ a\ est\ choisie} \tag{6.13}$$

Si le dénominateur vaut 0, on définit $Q_t(a)$ à une valeur par défaut, en général 0. À l'inverse, quand le dénominateur tend vers l'infini, $Q_t(a)$ converge vers $q_*(a)$. Ce type de méthode est appelé "sample-average" methods.

Une règle de sélection simple est de sélectionner une des actions avec la plus grande valeur estimée Thathachar and Sastry (1985), étant ainsi une greedy action. Ainsi, la méthode de greedy sélection est notée :

$$A_t \doteq argmax_a Q_t(a) \tag{6.14}$$

Cette méthode permet une exploitation maximum des connaissances mais donc, n'explore que très peu l'espaces des possibilités. Ainsi, elle est souvent couplée à une petite probabilité ϵ de sélection aléatoire. On appelle cette méthode ϵ-greedy Watkins (1989).

Exemple sur un bandit à 10 bras La figure 6.1 montre la répartition des récompenses en fonction des actions sur un test de bandit manchot à 10 bras. Chacune suit une loi normale centrée sur $q_*(a)$ et de variance 1. Pour résoudre ce problème d'apprentissage par renforcement nous utilisons les solutions proposées précédemment. Ainsi, la figure 6.2 montre les résultats en prenant une méthode de sélection ϵ-greedy. Chaque courbe représente les résultats pour une valeur différente d'ϵ.

L'avantage de la méthode ϵ-greedy n'est valable que pour une certaine variance dans la fonction d'évaluation. En effet, si la variance était égale à 0, une simple méthode avide aurait été plus efficace. Dans le cas où la variance aurait été plus élevée par contre, une méthode non avide aurait été plus efficace mais aurait pris beaucoup plus de temps à converger.

Implémentation incrémentale Quand un grand nombre d'itérations est effectué, le calcul de la moyenne peut poser un problème de temps de calcul et d'espace mémoire. En effet, la méthode triviale nécessite le stockage de toutes les valeurs de récompense précédentes. Pour pallier cela, une formule plus efficace peut être utilisée :

$$Q_{n+1} = \frac{1}{n}\sum_{i=1}^{n} R_i \tag{6.15}$$

$$Q_{n+1} = \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \tag{6.16}$$

$$Q_{n+1} = \frac{1}{n}(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i) \tag{6.17}$$

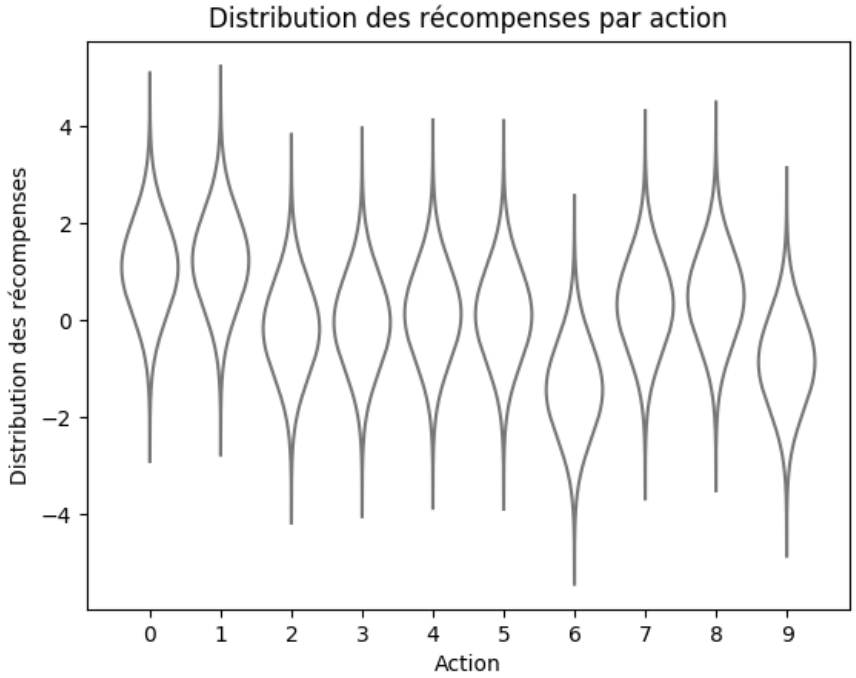


FIGURE 6.1 – Exemple de problème de bandit à 10 bras. La valeur $q_*(a)$ de chaque action suit une loi normale de variance 1

$$Q_{n+1} = \frac{1}{n}(n-1)Q_n \tag{6.18}$$

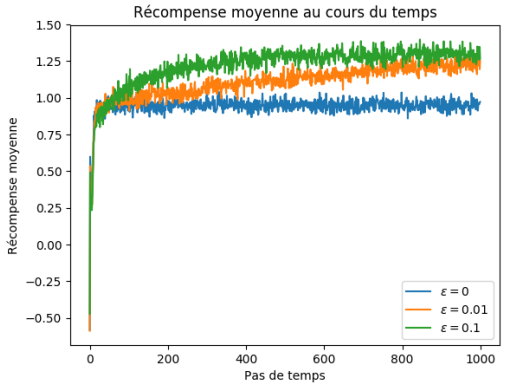
$$Q_{n+1} = \frac{1}{n}(R_n + nQ_n - Q_n) \tag{6.19}$$

$$Q_{n+1} = Q_n + \frac{1}{n}[R_n - Q_n] \tag{6.20}$$

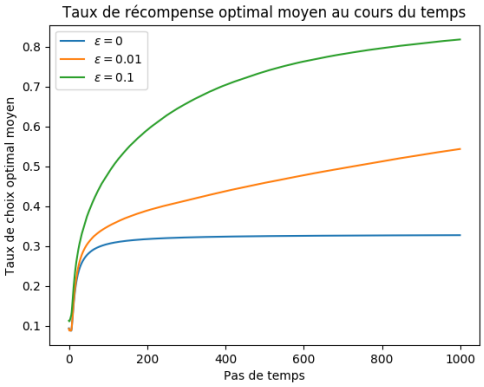
Ainsi, Q_{n+1} ne dépend plus que de Q_n et de R_n.

Problème non stationnaire La méthode proposée jusque-là peut se montrer efficace en environnement stationnaire. En revanche, lorsque les probabilités de récompenses peuvent changer au cours du temps (ce que l'on appelle donc, un environnement non stationnaire) il peut être intéressant de pondérer les récompenses pour donner plus de poids aux plus récentes.

$$Q_{n+1} = Q_n + \alpha[R_n - Q_n] \tag{6.21}$$



(a) Récompense moyenne au cours du temps



(b) Taux de choix de l'action optimale au cours du temps

FIGURE 6.2 – Résultats en faisant varier ϵ

$$Q_{n+1} = (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-1} R_i \tag{6.22}$$

α constant $\in]0;1]$

Attention : Q_1 représente la valeur initiale de Q, donc Q_0 souvent.

Valeur initiale optimiste Les méthodes présentées précédemment sont biaisées (au sens statistique du terme) par la valeur initiale estimée. Pour la méthode de la moyenne, le biais disparaît quand toutes les actions ont été sélectionnées au moins une fois. Par contre, pour la méthode de α constant, il est permanent, même s'il s'atténue, au cours du temps. En pratique, ce biais est rarement gênant, il peut même être utile dans certains cas.

On peut également s'en servir pour encourager l'exploration. Par exemple, dans le cas présenté en figure 6.3, une initialisation très optimiste Sutton and Singh (1994) de +5 va permettre d'explorer très rapidement l'espace.

Upper-Confidence-Bound Action Selection (UCB) Plutôt que de toujours prendre une action avide indépendemment du potentiel des autres solutions, il peut être intéressant de considérer chaque solution en fonction de leur potentiel Lai and Robbins (1985), c'est à dire, prendre en compte à quel point l'action-value est proche de son estimation optimale. On peut ainsi modifier la politique pour obtenir Auer et al. (2002) :

$$A_t = argmax_a \left[Q_t(a) + c\sqrt{\frac{lnt}{N_t(a)}} \right] \tag{6.23}$$

avec $c > 0$ contrôle le degré d'exploration. Ici, la racine carrée est une mesure de l'incertitude, ou variance, de l'estimation de la valeur de a. À chaque fois que a est choisie, l'incertitude diminue due à

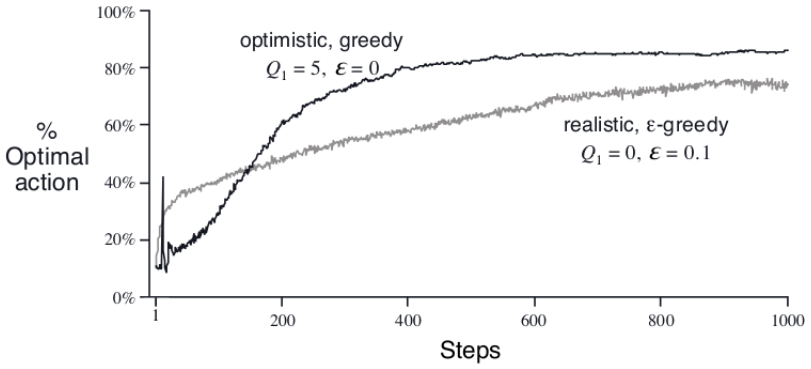


FIGURE 6.3 – Bandit avec initialisation optimiste tiré de Sutton (1988)

$N_t(a)$. En revanche, la présence de t au numérateur entraîne une augmentation de l'incertitude à chaque fois qu'une autre action est choisie, mais l'utilisation du logarithme permet de réduire cet impact au cours du temps.

Gradient Bandit Algorithms Jusque-là, nous avons utilisé des méthodes estimant une action-value et avons déterminé les actions à partir de celle-ci. La méthode présentée Greensmith et al. (2004) Dick (2015) ici va considérer apprendre une préférence numérique pour chaque action, notée $H_t(a)$. Seule la préférence relative d'une action sur l'autre est importante. Ainsi, ajouter une constante à toutes les préférences n'aura aucun effet sur les probabilités d'actions, déterminées par une distribution soft-max (i.e Gibbs ou Boltzmann).

$$PRA_t = a \doteq \frac{e^{h_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a) \tag{6.24}$$

Avec $\pi_t(a)$ la probabilité de sélectionner l'action a à l'instant t.

Initialement, toutes les préférences sont égales, en général $H_1(a) = 0 \forall a$. L'apprentissage se fait de la façon suivante :

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha(R_t - \overline{R_t})(1 - \pi_t(A_t)) \tag{6.25}$$

$$H_{t+1}(a) \doteq H_t(a) - \alpha(R_t - \overline{R_t})\pi_t(a)\ \forall a \neq A_t \tag{6.26}$$

avec $\alpha > 0$ et $\overline{R_t}$ la moyenne de toutes les récompenses obtenues jusque-là.

6.2.2 Programmation dynamique

Jusque-là, nous avons considéré des tâches non associatives, c'est-à-dire que la sélection d'actions ne nécessite pas la prise en compte de l'état courant. Plus clairement, il n'y a pas besoin d'associer différentes

actions avec des situations différentes. Ceci n'est en général pas le cas en apprentissage par renforcement.

La programmation dynamique (DP) Bellman and Dreyfus (2015) Minsky (1967) fait référence à un ensemble d'algorithmes permettant de calculer une politique optimale dans le cas de modèle parfait de l'environnement représenté sous forme de MDP, fini ou non. En général, la DP est difficilement utilisable en pratique du fait du modèle parfait nécessaire et de la grande demande en temps de calcul. Ces algorithmes sont cependant importants d'un point de vue théorique.

L'idée principale, est d'utiliser la value-function pour organiser la recherche de bonnes politiques. Ainsi, nous pouvons facilement déterminer la politique optimale une fois que les value-functions v_* et q_* optimales, satisfont les équations d'optimalité de Bellman ;

$$v_*(s) = max_a \mathbb{E}[R_{t+1} + \gamma v_x(S_{t+1})|S_t = s, A_t = a \tag{6.27}$$

$$v_*(s) = max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{6.28}$$

ou

$$q_*(s, a) = \mathbb{E}[R_{t+1} + \gamma max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \tag{6.29}$$

$$q_*(s, a) = \sum_{s',r} p(s', r|s, a)[R + \gamma max_{a'} q_*(s', a')] \tag{6.30}$$

Évaluation de politique (Prediction) Dans le cas où la dynamique de l'environnement est parfaitement connue on a un système de $|S|$ équations à $|S|$ inconnues. En principe la solution est trouvable sans problème, mais fastidieuse en temps de calcul. Ainsi, des solutions itératives sont préférables. En partant de v_0 une value-function mapant $\mathcal{S}^+$ dans $\mathbb{R}$ et définie arbitrairement, on définit les approximations suivantes en utilisant l'équation de Bellman pour v_π :

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1})|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s') \tag{6.31}$$

avec $\mathbb{E}_\pi$ représentant l'espérance suivant la politique π

La séquence v_k converge généralement vers v_π pour $k \to \infty$. On appelle cet algorithme *iterative policy evaluation.*

Pour réaliser l'approximation de v_{k+1} à partir de v_k, on applique la même opération sur chaque état s : on remplace l'ancienne valeur de s par une nouvelle, obtenue à partir des anciennes valeurs des successeurs de s et des récompenses immédiates obtenues en un pas de temps. Cette opération est appelée mise à jour attendue (*expected update*). Ainsi, à chaque itération, on met à jour l'ensemble des valeurs de chaque état pour produire la nouvelle approximation de la value-function v_{k+1}.

L'algorithme 11 montre le déroulement d'un algorithme d'évaluation itérative Howard (1964) de politique in-place.

Algorithme 11 Évaluation itérative de politique

Entree π : la politique à évaluer
Initialise $V(s) = 0\ \forall s \in S^+$
repeat
  $\Delta = 0$
  pour tout s $\in S$ **faire**
    $v = V(s)$
    $V(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta = max(\Delta, |v - V(s)|$
until $\Delta < \theta$ (un petit entier positive)
Sortie : $V \approx v_\pi$

Amélioration de politique On calcule la fonction de valeur d'une politique afin de trouver la meilleure. On peut donc comparer les politiques grâce à la fonction état-valeur, ainsi $v_{\pi'}(s) \geq v_\pi(s)\ \forall s \in \mathcal{S}$ implique que la politique π' est meilleur que la politique π.

Une fois la valeur d'une politique connue, comment savoir s'il serait mieux de changer pour une nouvelle ? Une façon de répondre à cette question est de ne pas suivre la politique pendant une itération puis de se remettre à la suivre. Ainsi, si $q_\pi(s, \pi'(s)) \geq v_\pi(s)$ alors la politique π' est au moins aussi bonne que π. Ainsi, on passe de politique π à π', chacune meilleure que la précédente par itérations successives en copiant π excepté pour un $\pi'(s) = a \neq \pi(s)$ tel que $q_\pi(s, a) > v_\pi(s)$ $v_{\pi'} = v_\pi$ implique que l'on a atteint la politique optimale.

Programmation dynamique asynchrone Jusqu'à présent, nous n'avons parlé que de méthodes de DP opérant sur l'ensemble des états à chaque itération. Cela présente l'inconvénient majeur de nécessiter une puissance de calcul beaucoup trop importante dans le cas où nous avons un grand ensemble d'états. Pour pallier ce problème, des algorithmes de DP asynchrones Bertsekas (1982) Bertsekas (1983) ont été développés. Ceux-ci mettent à jour les valeurs d'états dans n'importe quel ordre en utilisant les valeurs des autres états disponibles. Les valeurs de certains états peuvent être mis à jour plusieurs fois avant que celles d'autres ne le soient une fois. Pour converger correctement, un algorithme asynchrone doit continuer de mettre à jour les valeurs de tous les états : il ne peut pas en ignorer.

Efficacité de la programmation dynamique Littman et al. (1995) Les méthodes de DP ne sont pas pratiques sur de très gros problèmes, mais sont quand même efficaces comparées à d'autres méthodes de résolution des MDP. Elles garantissent de trouver une politique optimale en un temps polynomial même si le nombre total de politiques (déterministes) est k^n avec k le nombre d'actions possibles et n le nombre d'états possibles. Ainsi, les méthodes de DP sont exponentiellement plus rapides que toute recherche directe dans l'espace des politiques, car celles-ci vont chercher de façon exhaustive dans l'espace des politiques plutôt que de ne considérer que les intéressantes.

DP est cependant limitée quand beaucoup de variables d'états sont présentes. Avec la puissance de calcul actuelle, la DP peut être utilisée jusqu'à plusieurs millions d'états, mais devient rapidement inefficace pour plus.

6.2.3 Méthodes de Monte Carlo

Pour la première fois, nous allons considérer des méthodes permettant d'estimer des fonctions de valeur et découvrir des politiques optimales sans une connaissance complète de l'environnement. Les méthodes de monte carlo ne nécessitent que des expériences, c'est-à-dire des séquences d'états, actions et récompenses. Apprendre à partir d'expériences est remarquable, car cela ne requiert pas de connaissance a priori sur les dynamiques de l'environnement. Un modèle est cependant requis, mais il n'y a pas besoin de complètement connaître la distribution de probabilités de toutes les transitions possibles comme en DP.

Ces méthodes seront utilisées sur des tâches épisodiques. Les politiques et fonctions de valeur seront mises à jour seulement après un épisode (ou plusieurs). Ainsi, ces méthodes sont incrémentales au sens d'épisode par épisode et non pas par pas.

Comme pour les méthodes de bandits, les méthodes de Monte Carlo sont souvent basées sur la moyenne des retours de chaque paire état-action. La différence principale est qu'ici on prend en compte de multiples états. Cette spécificité n'était possiblement prise en compte que dans le bandit contextuel. La valeur de retour, après avoir choisie une action dans un état, dépend également des actions effectuées dans les états suivants. Ainsi, le problème devient non-stationnaire du point de vue de l'état passé. Pour prendre cela en compte, nous adapterons l'idée d'itération de la politique générale (*General Policy Iteration* (GPI)) de la DP mais en apprenant la fonction de valeur à partir des exemples retournés par le MDP. Les fonctions de valeur et politiques correspondantes intéragissent ensemble pour atteindre l'optimum.

Monte carlo Prediction Nous allons commencer par considérer une méthode de Monte Carlo permettant d'apprendre la fonction état-valeur pour une politique donnée Barto and Duff (1994). Pour rappel, la valeur d'un état est la récompense cumulée espérée actualisée (*expected cumulative future discounted reward*) en partant de l'état en question. La façon la plus évidente est de moyenner les valeurs de retour de l'ensemble des états observés après l'apparition de cet état. Plus il y a de retours observés, plus la moyenne convergera vers la valeur espérée. C'est la base de toutes les méthodes de Monte Carlo.

En considérant $v_\pi(s)$, la valeur de l'état s sous la politique π en prenant un ensemble d'épisodes π passant par s. Chaque occurrence de l'état s dans un épisode est appelé *visite*. Un état peut cependant être visité plusieurs fois pendant un épisode. On distingue alors deux algorithmes, first-visit MC prediction et every-visit MC prediction Singh and Sutton (1996).

Monte Carlo Estimation of Action Values Si aucun modèle n'est disponible, il est particulièrement utile d'estimer l'action-state value plutot que la state-value. Ainsi ici, nous chercherons à déterminer $q_\pi(s, a)$, la valeur de retour espérée en partant de l'état s, en prenant l'action a puis en suivant la politique π. Les méthodes énoncées précédemment peuvent s'appliquer en utilisant q plutôt que v à la différence qu'ici, certaines paires état-action ne seront jamais visitées. Si π est déterministe, alors en suivant cette politique nous observerons toujours la même action en partant du même état. Or, pour comparer les alternatives, il faut pouvoir estimer toutes les actions à partir de chaque état, pas seulement celle qui semble la meilleure à l'instant t. On revient donc au problème de maintien de l'exploration

présenté en annexe 6.2.1.

Un algorithme intéressant dans ce cas est l'*exploringstarts*. Celui-ci propose, pour chaque épisode, de commencer dans un état donné avec toute paire état-action ayant une probabilité non nulle d'être explorée. Bien qu'intéressante théoriquement, cette proposition est difficilement utilisable en pratique. En effet, en apprenant directement via interaction avec l'environnement, la condition de départ est difficilement respectable. Ainsi, on préfère souvent considérer des politiques stochastiques.

Contrôle par Monte-Carlo Comme en programmation dynamique (DP), le contrôle MC se base sur les notions d'évaluation et d'amélioration. Il propose de partir d'une politique arbitraire π_0 et de l'améliorer au cours des itérations pour finir sur π_* et q_*. On a ainsi le diagramme suivant :

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} q_{\pi_*} \tag{6.32}$$

avec $\xrightarrow{E}$ une évaluation de la politique et $\xrightarrow{I}$ une amélioration. Celui-ci est réalisé de la même façon que pour la DP, c'est à dire par utilisation de comportement glouton : pour toute $q(s,a)$ et $\forall\ s \in \mathcal{S}$, on choisit l'action avec l'action-value maximale :

$$\pi(s) = argmax_a q(s,a) \tag{6.33}$$

Il y a cependant un problème. En effet, pour obtenir la garantie de convergence, nous supposons que les épisodes ont une possibilité d'exploration initiale Michie and Chambers (1968) et que l'évaluation peut se faire en un nombre infini d'épisodes. En pratique, il faudra au moins supprimer la deuxième.

Une façon simple d'y parvenir est de ne pas estimer avec précision l'évaluation de la politique actuelle à chaque itération, mais plutôt de rapidement aller vers la phase d'amélioration. La précision des évaluations s'améliorera ainsi petit à petit. Ainsi, après chaque épisode, les observations sont utilisées pour l'évaluation de la politique et celle-ci est améliorée (*improved*) pour chaque état rencontré.

Sans départ exploratoire, il faut pouvoir garantir que chaque action pourra être sélectionnée par l'agent. Pour cela, il y a deux types de méthodes :

— sur politique (*on-policy*) : vise à évaluer ou améliorer la politique utilisée pour prendre des décisions. L'exploring start présenté avant est un exemple de méthode sur politique.

— hors politique (*off-policy*) : évalue ou améliore une politique différente de celle utilisée pour générer les données.

Un exemple classique de méthode sur politique sans exploration initiale est simplement d'utiliser une politique ϵ-greedy.

Prédiction hors politique via Importance Sampling Les méthodes hors politique consistent à utiliser deux politiques. Une qui apprendra et deviendra la politique optimale et une autre plus exploratrice utilisée pour générer des comportements différents. La première étant appelée politique cible (*target policy*) et la seconde, politique comportementale (*behavior policy*). On dit que l'apprentissage

se fait sur des données hors politique cible et le processus complet est donc appelé apprentissage hors politique (*off-policy learning*).

Les méthodes hors politique sont souvent plus difficiles à utiliser et présentent une variance et un temps de convergence supérieurs. Elles sont en revanche plus puissantes et plus génériques Sutton and Barto (1998). Dans le cas particulier où la politique comportementale et la politique cible sont identiques, on retombe sur une méthode sur politique.

Presque toutes les méthodes hors politique utilisent la notion d'*importance sampling* Hesterberg (1988) Shelton (2001), une technique générale d'estimation des valeurs attendues sous une distribution donnée à partir d'échantillons d'une autre distribution. Le ratio importance-sampling (grossièrement assimilable au ratio exploration-approfondissement) est donné par l'équation suivante :

$$p_{t:T-1} = \frac{\Pi_{k=t}^{T-1}\pi(A_k|S_k)p(S_{k+1}|S_k,A_k)}{\Pi_{k=t}^{T-1}b(A_k|S_k)p(S_{k+1}|S_k,A_k)} = \Pi_{k=t}^{T-1}\frac{\pi(A_k|S_k)}{b(A_k|S_k)} \tag{6.34}$$

Méthode hors politique de contrôle de Monte Carlo Les méthodes de contrôle de Monte Carlo hors politique utilisent l'une des techniques présentées dans les sections précédentes. Elles suivent la politique de comportement tout en apprenant et en améliorant la politique cible. Ces techniques exigent que la politique comportementale ait une probabilité non nulle de sélectionner toutes les actions qui pourraient être sélectionnées par la politique cible. Pour explorer toutes les possibilités, nous exigeons que la politique comportementale soit souple (c'est-à-dire que la probabilité de sélection de toutes les actions dans tous les états soit non nulle).

6.2.4 Apprentissage par différence temporelle (Temporal difference learning (TD))

TD est une combinaison de DP et MC. Les méthodes de TD peuvent apprendre directement à partir d'expériences sans connaissance sur le modèle. Comme pour la DP, elles peuvent mettre à jour certaines estimations à partir de celles apprises précédemment.

Prédiction TD Comme les méthodes de Monte-Carlo, TD utilise des expériences pour résoudre le problème de prédiction. En prenant quelques expériences qui suivent la politique π, elles mettent à jour V de v_π pour les états non terminaux S_t de cette expérience. MC attend que la valeur de retour soit connue et l'utilise comme cible pour $V(S_t)$. Une méthode every-visit simple en environnement non stationnaire est :

$$V(S_t) = V(S_t) + \alpha[G_t - V(S_t)] \tag{6.35}$$

avec G_t le retour actuel au temps t et α un le paramètre constant $step-size$. Cette méthode est appelée $constant-\alpha$ MC.

À l'inverse des méthodes de MC, TD n'a besoin d'attendre que le prochain pas de temps pour déterminer la variation de $V(S_t)$. Ainsi, au temps $t+1$, elles peuvent déjà effectuer une mise à jour en utilisant la récompense R_{t+1} observée et l'estimation de $V(S_{t+1})$. La méthode de TD la plus simple

réalise la mise à jour dès la transition vers S_{t+1} générant R_{t+1} :

$$V(S_t) = V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{6.36}$$

La cible pour la mise à jour de Monte Carlo est G_t alors que celle de TD est $R_{t+1} + \gamma V(S_{t+1})$. Cette méthode est appelée $TD(0)$ Sutton (1988) ou *one − step TD*. On a alors l'algorithme 12. Elle a été prouvée convergente par ses auteurs et avec une probabilité 1 par Dayan (1992).

Algorithme 12 Différence temporelle à une étape (TD(0))

Entree π : la politique à évaluer
Initialise $V(s) = 0\ \forall s \in S^+$
repeat
  Initialise s
  repeat
    a=$\pi(s)$
    r, s' =pas_de_simulation(a)
    $V(s) = V(s) + \alpha[r + \gamma V(s') - V(s)]$
    $s = s'$
  until s est terminal
until $nEpisode$

Avantages des méthodes de prédiction par TD Un des avantages évident par rapport aux méthodes de DP est que celles-ci ne nécessitent pas forcément de connaissances sur le modèle de l'environnement, les récompenses et les probabilités de transitions. Par rapport aux méthodes de Monte Carlo, celles-ci sont naturellement implémentées de façon on-line et complètement incrémentales. De plus, certaines applications nécessitent de très longs épisodes (voir infini dans le cas de tâches continues) ce qui peut être problématique et long avec les méthodes de MC. De plus, les méthodes de MC doivent souvent ignorer les épisodes où certaines actions expérimentales sont prises, ce qui peut considérablement ralentir l'apprentissage. Les méthodes TD sont beaucoup moins sensibles à ce problème puisqu'elles apprennent par rapport à chaque transition.

Sarsa : contrôle sur politique par TD Cette méthode se base sur la GPI mais en utilisant cette fois une méthode de TD pour l'estimation de la partie prédictive.

La première étape est donc d'apprendre une fonction de valeur d'action plutôt qu'une fonction de valeur d'état. En particulier pour une on-policy method, il faut estimer $q_\pi(s, a)$ pour la politique de comportement (*behavior policy*) π pour chaque état s et action a. Cela peut être fait avec la méthode de TD présentée précédemment pour v_π en considérant plutôt les transitions de paire état-action à paire état-action et en apprenant ces valeurs. On a donc :

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{6.37}$$

Cette mise à jour étant faite après chaque transition non terminale. Si S_{t+1} est terminal, alors

$Q(S_{t+1}, A_{t+1})$ est définie à 0. On suit alors l'algorithme Rummery and Niranjan (1994) Sutton (1996) défini en algorithme 13.

Algorithme 13 Sarsa : contrôle sur politique par TD

Initialise $Q(s,a)\ \forall s \in S, a \in A(s)$, arbitrairement et $Q(etats\ terminaux) = 0$
repeat
  Initialise s
  choisir a à partir de s en utilisant la politique dérivée de Q (ϵ-greedy par exemple)
  repeat
    $r, s' =$pas_de_simulation(a)
    choisir a' à partir de s' en utilisant la politique dérivée de Q (ϵ-greedy par exemple)
    $Q(s,a) = Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
    $s = s'$
    $a = a'$
  until s est terminal
until $nEpisode$

La convergence de cet algorithme dépend de la nature de la politique. Avec des politiques ϵ-greedy la probabilité de convergence tend vers 1 quand t tend vers l'infini. On peut également utiliser des politiques ϵ-greedy à ϵ variable comme $\epsilon = 1/t$.

Q-learning : contrôle par TD hors politique Le Q-learning a été introduit pour la première fois dans Watkins (1989) et se base sur la relation maîtresse suivante :

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{6.38}$$

Dans ce cas, l'apprentissage de la fonction Q approxime directement q_*, la fonction de valeur d'action optimale, indépendamment de la politique actuellement suivie. Cette méthode simplifie drastiquement les analyses de l'algorithme et permet une convergence Watkins and Dayan (1992) rapide. La politique a toujours un effet en déterminant quelle paire état-action sera visitée et mise à jour.

Algorithme 14 Q-learning : contrôle par TD hors politique

Initialise $Q(s,a)\ \forall s \in S, a \in A(s)$, arbitrairement et $Q(etats\ terminaux) = 0$
repeat
  Initialise s
  repeat
    choisir a à partir de s en utilisant la politique dérivée de Q (ϵ-greedy par exemple)
    $r, s' =$pas_de_simulation(a)
    $Q(s,a) = Q(s,a) + \alpha[r + \gamma max_a Q(s',a) - Q(s,a)]$
    $s = s'$
  until s est terminal
until $nEpisode$

Sarsa espéré (*Expected Sarsa*) Cet algorithme Sutton and Barto (1998) Van Seijen et al. (2009) est très proche du Q-learning à la différence qu'il utilise la valeur espérée à la place du maximum de la prochaine paire état-action. La règle de mise à jour est alors la suivante :

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}Q(S_{t+1}, a - Q(S_t, A_t)] \tag{6.39}$$

Ainsi, cet algorithme évolue de façon déterministe dans la même direction que le Sarsa *espère* bouger, c'est pour cela qu'il est appelé *Expected Sarsa*.

On peut utiliser cette méthode sur politique mais en général il vaut mieux utiliser une politique de comportement différente, devenant ainsi un algorithme hors politique.

Biais de maximisation et double apprentissage Tous les algorithmes de contrôle présentés précédemment utilisent la notion de maximisation dans la construction de leur politique. Dans des cas relativement particuliers, cela peut mener à ce qu'on appelle un biais de maximisation dû à une estimation positive trop significative d'un biais Hasselt (2010). Par exemple, quand la valeur espérée est 0 mais qu'une estimation devient positive. Cette notion est illustrée par l'exemple présenté en figure 6.4.

Dans ce cas, on a un MDP à 2 états non terminaux, A et B. Un épisode débute toujours dans l'état A et a toujours deux choix, droite ou gauche. S'il va à doite dans l'état A, l'épisode se termine avec la récompense 0. S'il va à gauche il passe dans l'état B toujours avec une récompense nulle. Dans l'état B, un grand nombre d'actions sont possibles, chacune entraînant la fin de l'épisode et une récompense suivant une loi normale de moyenne -0.1 et de variance 0.1. Ainsi, choisir l'action de gauche est toujours une erreur. or, comme les actions de B peuvent engendrer un biais positif, on remarque sur le graphique de la figure 6.4 que la méthode de Q-learning estime que l'action gauche est un bon choix, du moins au début.

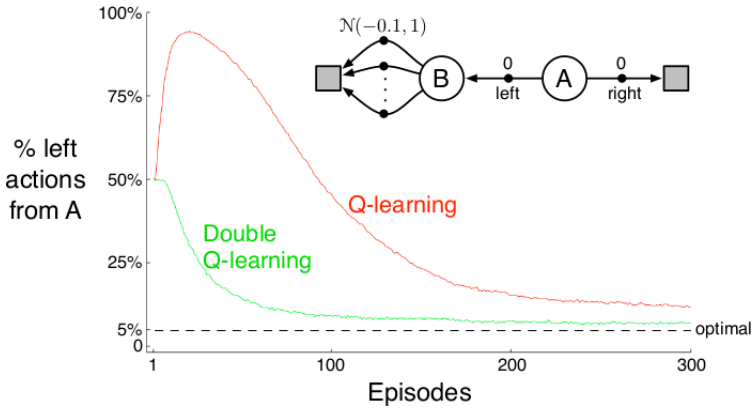


FIGURE 6.4 – Exemple biais de maximisation tiré de Sutton (1988)

Notion d'*afterstates* Les méthodes présentées ici sont applicables à un grand nombre de cas. Cependant, des cas particuliers existent où d'autres méthodes peuvent être préférable. Par exemple, tra-

ditionnellement on apprend sur une fonction de valeur ou une valeur d'action. Or, dans les cas où on connaît le résultat immédiat d'une action, mais pas forcément ses conséquences, une approche *afterstates-value* Van Roy et al. (1997) peut être intéressante. Cette approche consiste à estimer la valeur de l'état suivant, résultant de l'action choisie. Cela peut par exemple être utile aux échecs. En effet, pour un état donné (l'état de l'échiquier) et une action (un coup), on est sûr de la nouvelle position engendrée, l'$afterstate$, il est en revanche difficile de prévoir ce que notre adversaire répondra et les conséquences, $afterstate - value$.

De plus, il est possible, en partant de deux états et actions différentes, d'arriver au même $afterstate$. On peut donc ainsi avoir de meilleures estimations rapidement.

6.2.5 Bootstrapping à n étapes

Les méthodes de MC ou TD à une étape présentant certaines limites, certains auteurs ont proposé de les combiner afin d'en tirer profit à travers des méthodes TD à n étapes plus générales. Leur principal avantage est qu'elles nous libèrent de la *tyrannie* du pas de temps. Dans beaucoup d'application, nous cherchons à mettre à jour le plus rapidement possible afin de prendre en compte la moindre variation d'état. Mais il est parfois préférable d'attendre afin de vraiment observer des variations significatives.

Prédiction par TD à n étapes Les méthodes de MC considèrent une estimation de v_π sur des épisodes entiers générés par π et mettent à jour chaque état en se basant sur la séquence toute entière. Les méthodes TD à une étape à l'inverse, mettent à jour les fonctions de valeur à chaque pas de temps, en se basant sur la récompense instantanée. Ainsi, une méthode intermédiaire peut simplement être de mettre à jour à intervalle de temps n régulier : n-step TD.

Sarsa à n étapes Pour réaliser un contrôle plûtot qu'une prédiction, on peut réutiliser la méthode Sarsa présentée précédemment en prenant en compte n étapes. Cette méthode va principalement consister à utiliser les paires état-action plutôt que simplement les valeurs d'états et utiliser une politique ϵ-greedy.

Modèles et planification L'expression "modèle de l'environnement" fait référence à tout ce que l'agent peut utiliser pour prédire ce que l'environnement va répondre à ses actions. Ainsi, en prenant un état et une action, le modèle va produire une prédiction de l'état et de la récompense engendrée. Dans le cas où le modèle est stochastique, il y a plusieurs nouveaux états et récompenses possibles chacun ayant une certaine probabilité d'apparaître.

Le mot planification (*planning*) désigne, la plupart du temps, tout processus prenant un modèle en entrée et créant ou améliorant une politique. Deux approches existent. La première, *State-space planning* cherche la meilleure politique possible liant l'espace des états à un but modélisé par des récompenses produites par un ensemble d'actions et de transitions d'états. La deuxième, *plan-space planning*, est une recherche dans l'espace des plans. Elle consiste en la transformation d'un plan en un autre et les fonctions de valeur sont définies dans l'espace des plans. Ces méthodes sont cependant difficilement applicables efficacement dans des processus de décision stochastiques.

Toutes les méthodes *state-space planning* présentent deux idées de base communes :

— elles calculent toutes des fonctions de valeur afin d'améliorer la politique

— elles calculent les fonctions de valeur par mise à jour et backup appliquées à des expériences simulées.

Les notions de *planning* et *learning* sont à distinguer. Le terme *planning* se réfère à des méthodes utilisant des expériences simulées, générées par un modèle alors que les méthodes de *learning* utilisent de vraies expériences générées par l'environnement réel. Ces méthodes présentent cependant une structure commune permettant facilement d'adapter des algorithmes d'une méthode en l'autre. Par exemple, les méthodes de *learning* ne requièrent que des données d'expériences en entrées, elles peuvent donc facilement être appliquées à des données d'expériences simulées.

Ainsi, il est souvent possible de considérer une vue unifiée de *planning* et *learning* et leur non distinction est rarement problématique.

Quand le modèle est faux Les modèles peuvent être incorrects parce que l'environnement est stochastique et que seul un nombre limité d'échantillons a été observé, ou parce que le modèle a été appris en utilisant des approximations qui se sont généralisées imparfaitement, ou simplement parce que l'environnement a changé et son nouveau comportement n'a pas encore été observé. Lorsque le modèle est incorrect, le processus de planification est susceptible de calculer une politique sous-optimale.

Dans certains cas, la politique sous-optimale calculée par la planification conduit rapidement à la découverte et à la correction de l'erreur de modélisation. Cela a tendance à se produire lorsque le modèle est optimiste en ce sens qu'il prédit une plus grande récompense ou de meilleures transitions d'états qu'il n'est réellement possible. La politique prévue tente d'exploiter ces possibilités et, ce faisant, découvre qu'elles n'existent pas.

Balayage prioritaire (*Prioritized sweeping*) En général, les transitions simulées sont sélectionnées uniformément. Mais ce type de sélection est rarement le meilleur. En effet, la planification peut être beaucoup plus efficace en ne se concentrant seulement sur des paires état-action particulières. Ainsi, il peut être intéressant de ne considérer que les états dont la valeur a changé. Mais cela impliquera également la variation de plusieurs autres états, liés à celui-ci. Or, il n'est utile de mettre à jour que les états menant, ou découlant de celui-ci. En effet, les mises à jour des états voisins provoqueront, à l'itération suivante, la propagation des modifications aux autres états. Cette idée générale est appelée *backward focusing*.

De même, quand beaucoup de changement ont lieux, il peut être intéressant de ne prioriser la mise à jour que des états voisins de ceux qui ont subi une grande modification. En effet, ceux-ci sont plus susceptibles d'également subir une grande modification. On priorise donc par urgence, c'est l'idée derrière le *prioritized sweeping*.

Mise à jour espérée et mise à jour par échantillonage Les mises à jour peuvent être espérées (*expected update*), c'est à dire qu'elles considèrent tous les événements possibles qui peuvent arriver, ou (*sample update*), un seul exemple. En l'absence d'un modèle de distribution, les méthodes de mise à jour

espérée ne sont pas utilisables. Ces méthodes, bien que semblant plus efficaces, demandent cependant une puissance de calcul bien plus importante. Elles ne sont donc pas toujours à préférer.

La différence entre ces deux types de mise à jour est particulièrement significative dans le cas d'un environnement stochastique, particulièrement quand, pour un état et une action donnés, un grand nombre d'états suivants sont possibles. Si un seul état est possible, ces méthodes sont équivalentes. Sinon, les différences peuvent être très importantes. Ainsi, la mise à jour espérée a l'avantage de proposer des valeurs exactes. En revanche, la mise à jour par échantillonnage est moins coûteuse en temps de calcul, car ne requiert qu'une transition d'états. En pratique, les calculs requis par les opérations de mise à jour sont généralement dominées par le nombre de paires état-action pour lesquelles la fonction Q est évaluée. Ainsi, pour une paire s, a de départ et b le facteur d'embranchement, le nombre d'état s' pour lequel $p(s'|s, a) > 0$, la mise à jour espérée requiert b fois plus de calcul qu'une mise à jour par échantillonage.

Recherche par heuristique En recherche par heuristique, pour chaque état, un arbre avec un grand nombre de possibles continuations est considéré et la fonction de valeur est appliquée sur les feuilles et ramenée à l'état initial. Une fois la fonction de *back-up* calculée pour chaque noeud jusqu'à l'état initial, le meilleur est choisi comme action courante et les autres valeurs sont supprimées.

L'équation utilisée pour le back-up est la suivante :

$$v_*(s) = max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \tag{6.40}$$

Algorithmes de déploiement (*Rollout algorithms*) Les algorithmes de déploiement sont des algorithmes de *deicsion-time planning* basés sur le contrôle de Monte-Carlo appliqué à des trajectoires simulées, partant toutes de l'état courant. Ils estiment la valeur d'action pour une politique donnée en prenant le retour d'un grand nombre de trajectoires simulées, partant de toutes les actions possibles puis en suivant la politique. Quand l'estimation des valeurs d'actions est considérée comme suffisamment précise, l'action avec la meilleure valeur est exécutée et le processus est réitéré à partir du nouvel état.

Recherche par arbre de Monte-Carlo (*Monte Carlo tree search, MCTS*) MCTS est un algorithme de déploiement, mais ajoute la prise en compte des valeurs calculées précédemment afin de progressivement orienter les recherches vers les trajectoires les plus prometteuses.

Pour chaque état rencontré, le MCTS est exécuté afin de sélectionner l'action à choisir pour cet état, puis le prochain et ainsi de suite. Comme pour les algorithmes de déploiement, c'est un processus itératif qui simule un grand nombre de trajectoires en partant de l'état courant jusqu'à un état terminal (ou une condition d'arrêt). L'idée principale des MCTS est de se concentrer petit à petit sur les trajectoires qui ont reçu les meilleures valeurs précédemment et de les étendre. En général, on ne conserve pas les valeurs d'une exécution à l'autre, mais cela n'est pas impossible.

Généralement, les trajectoires simulées suivent une politique simple, appelée politique de déploiement (*rollout-policy*). Généralement, comme pour les méthodes de Monte-Carlo classiques, la valeur d'une paire état-action est estimée par la moyenne des retours simulés à partir de cette paire, mais il est possible

de considérer le pire ou le meilleur cas par exemple. Les valeurs estimées ne sont conservées que pour les sous-ensembles des paires action-states les plus susceptibles d'être atteintes en quelques étapes (voir figure 6.5). Ces algorithmes étendent ensuite progressivement l'arbre en ajoutant des nœuds représentant les états qui semblent prometteurs d'après les résultats des trajectoires simulées.

La politique de déploiement est utilisée pour sélectionner les actions mais dans l'arbre on utilise une autre politique, la politique de l'arbre (*tree policy*), afin de gérer exploration et exploitation. Elle peut par exemple choisir des actions en utilisant ϵ-greedy ou UCB.

Ainsi, chaque itération d'un MCTS basique suit les quatre étapes présentées en figure 6.5.

— Sélection : En partant du noeud racine, la politique de l'arbre est utilisée pour explorer les possibilités et choisir un noeud feuille.

— Expansion : Sur quelques itérations, l'arbre est étendu à partir du noeud feuille en ajoutant un ou plusieurs noeuds fils vers des actions inexplorées.

— Simulation : À partir d'un des nouveaux noeuds, une simulation d'un épisode complet est exécuté en choisissant les actions à partir de la politique de déploiement.

— *Backup* : le retour généré par l'épisode simulé est propagé pour mettre à jour (ou initialiser) les valeurs d'actions des noeuds traversés par la politique de l'arbre de cette itération du MCTS. Aucune valeur n'est gardée pour les noeuds traversés par la politique de déploiement. En général, le retour entier de la simulation est envoyé au premier noeud.

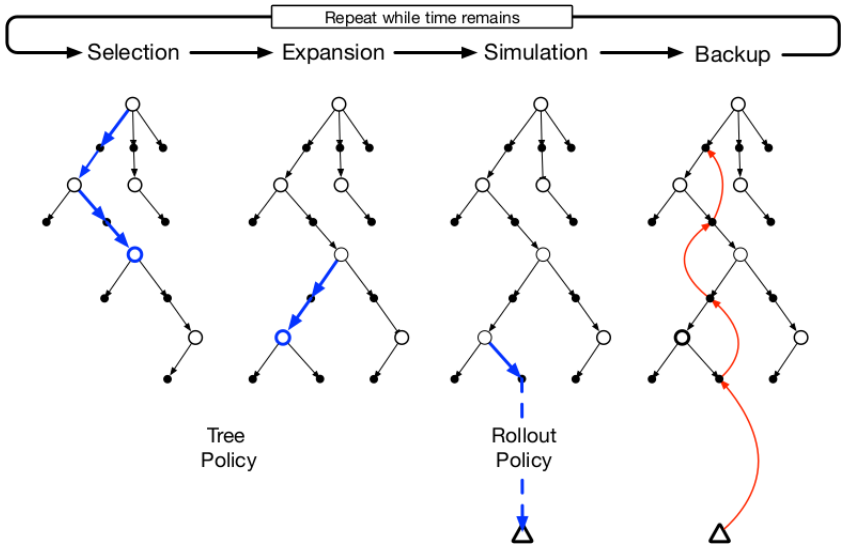


FIGURE 6.5 – Déroulement MCTS (tiré de Sutton (1988)

Ces étapes sont itérées jusqu'à une condition d'arrêt (temps de calcul, nombre d'itérations etc.) puis une action est choisie à partir du noeud racine, l'état actuel de l'environnement. Cette sélection peut se faire de différentes façons. Elle peut simplement se faire par choix de l'action avec la meilleure valeur, ou celle qui a été visitée le plus de fois par exemple, mais des mécanismes plus complexes peuvent être

imaginés. Une fois la transition effectuée, MCTS est réitéré, souvent en repartant de l'arbre calculé précédemment à partir de ce point. Les autres noeuds sont éliminés.

6.3 Détails sur les types d'optimisation

6.3.1 Précision sur les types d'optimisation sous contraintes

Méthodes de pénalisation

Méthode générale Les méthodes de pénalisation transforment le problème initial en ajoutant une fonction de pénalité, dépendante des contraintes. Le problème devient :

$$minimiser\ f(x) + p(x) \tag{6.41}$$

$$p(x) = 0\ si\ x\ \in \mathcal{F} \tag{6.42}$$

$$p(x) > 0\ sinon \tag{6.43}$$

$p(x)$ étant la fonction de pénalité et $\mathcal{F}$ l'ensemble des solutions réalisables.

Souvent, la fonction de pénalité est définie de la façon suivante :

$$p(x) = F(\sum_{j=0}^{m} \alpha_j v_j^{\beta}(x)) \tag{6.44}$$

avec :

— F : une fonction croissante pouvant dépendre du temps ou du nombre d'itérations

— m : nombre de contraintes

— α_j : des coefficients de pénalisation, souvent égaux pour chaque contrainte

— β : paramètre, généralement égal à 2

— v_j : mesure de la violation de la $j$ème contrainte

Pour les contraintes d'inégalité :

$$v_j = max(0, g_j(x)) \tag{6.45}$$

Pour les contraintes d'égalité :

$$v_j = |h_j(x)| \tag{6.46}$$

Cette méthode peut poser un problème. En effet, si la pénalisation est mal gérée, ce qui est fréquent quand on ne connaît pas bien l'espace de recherche, il est possible d'avoir des solutions ne respectant

pas les contraintes qui présentent une meilleure évaluation que l'optimum global, c'est le problème de sous-pénalisation.

Méthode de la peine de mort Cette méthode permet de pallier le problème de sous-pénalisation en utilisant une pénalité infinie :

$$p(x) = \pm\infty \tag{6.47}$$

Cela permet, en fonction des algorithmes, de garder des solutions non réalisables plus ou moins longtemps, assurant ainsi une certaine exploration de l'espace non réalisable. Son efficacité est cependant très dépendante du problème, de la population initiale et surtout de la difficulté de satisfaction des contraintes.

Ainsi, ses résultats sont souvent faibles et elle est en pratique rarement privilégiée.

Pénalités statiques/dynamiques L'approche générale présentée en section 6.3.1 suppose des coefficients de pénalités α_j statiques. Or, comme vu précédemment, si ces coefficients sont mal gérés, des phénomènes de sous-pénalisation peuvent apparaître. Il est également possible de voir apparaître une sur-pénalisation empêchant l'exploration de se dérouler correctement. Ce cas est typiquement présent lors de la méthode de la peine de mort (section 6.3.1).

Afin d'éviter cela, Homaifar et al. (1994) proposent une méthode pour définir des coefficients de pénalité appropriés à chaque famille d'intervalle de violation de contraintes. Cette méthode demande cependant un grand nombre de paramètres, ce qui la rend souvent difficile à utiliser avec efficacité.

Des méthodes de pénalités dynamiques ont également été envisagées Joines and Houck (1994). Elles proposent de faiblement pénaliser les solutions au début afin de favoriser l'exploration de l'espace non réalisable. La pénalité sera augmentée au fur et à mesure des itérations pour finalement favoriser le retour de solutions réalisables.

Le faible nombre de paramètres à caler rend son utilisation plus facile et les résultats obtenus sont souvent meilleurs.

Pénalités adaptatives, auto-adaptatives Le principe de ces méthodes est d'introduire une variable dépendante de l'état du processus d'optimisation. Celle-ci sera en général modifiée à chaque itération selon certains critères spécifiques à la méthode. Il nous est donc impossible de toutes les détailler ici. Ainsi, le lecteur intéressé pourra se référer à :

— Ben Hadj-Alouane and Bean (1997) : qui utilisent une pénalité dépendante de la qualité de la meilleure solution connue

— Smith (1993) : incorpore la notion de degré de violation

— Hamida and Schoenauer (2000) : proposent d'ajuster la pénalité en fonction de la proportion des solutions qui respectent les contraintes

— Coello (1999) et Tessema and Yen (2006) : amènent la notion de pénalité auto-adaptative

Méthode de supériorité des individus réalisables

Ces méthodes définissent toute solution réalisable comme meilleure que toute solution non réalisable.

La première méthode proposée est celle de Powell and Skolnick (1993), qui définit une heuristique pour faire en sorte que les individus réalisables soient toujours mieux évalués que les autres. Ainsi, une mauvaise solution réalisable aura une meilleure performance qu'une solution non réalisable, mais très proche de l'optimum.

Cette approche présente l'avantage de pouvoir s'appliquer à n'importe quel algorithme à population de solutions. En effet, la plupart des autres méthodes de ce type Deb (2000) , Runarsson and Yao (2000) sont souvent exclusivement applicables aux algorithmes évolutionnaires, car elles utilisent leur fonction de sélection (section 6.4.2). Ces approches présentent cependant de très bons résultats dans la plupart des cas. Nous avons donc implémenté la méthode de Stochastic Ranking proposée par Runarsson and Yao (2000), méthode que nous détaillerons en section 6.4.14.

Méthodes hybrides

D'autres approches basées sur la préservation de la faisabilité des solutions ont été proposées. Elles se basent généralement sur des algorithmes d'optimisation globale classiques mais en y incluant des heuristiques de satisfaction de contraintes. Cette approche est dite hybride. Elle est cependant très souvent spécifique au problème.

Des méthodes génériques ont cependant été proposées. On peut notamment penser au système GENOCOP Michalewicz and Janikow (1991) , Michalewicz and Nazhiyath (1995). Cette méthode modifie les contraintes au préalable afin de définir un espace de faisabilité convexe et de pouvoir y définir des opérateurs fermés, permettant ainsi de maintenir la faisabilité des solutions. Cette méthode, bien que très intéressante, est difficilement automatisable.

6.3.2 Précision sur les méthodes d'optimisation multimodale

Mesure de distance

Dans le cas des algorithmes génétiques à codage binaire, une distance génotypique par la méthode de distance de Hamming a historiquement été utilisée Sareni (1999). Elle n'est cependant appropriée que dans le cas de cette forme de codage et pour cet algorithme. Des mesures de distance basées sur les valeurs réelles des paramètres, parfois appelées distances phénotypiques, ont donc été proposées. Dans ces cas, la mesure de distance historique Deb and Goldberg (1989) est donnée par :

$$d^{\beta}(a_1, a_2) = \frac{1}{n^{1/\beta}} \left(\sum_{i=1}^{n} |\Delta x_i|^{\beta} \right)^{1/\beta} \tag{6.48}$$

$$\Delta x_i = \frac{x_{1i} - x_{2i}}{x_{imax} - x_{imin}} \tag{6.49}$$

avec :

— n le nombre de paramètres

— x_{1i} et x_{2i} la valeur du ième paramètre de l'individu a_1 et a_2 respectivement

— x_{imin} et x_{imax} les valeurs minimale et maximale du paramètre i respectivement

— Δx_i l'écart normalisé

— $\beta > 0$ un facteur caractérisant la forme de la niche

Le paramètre β permet également d'obtenir différents types de distance en fonction de sa valeur. Pour $\beta = 1$, d^β donne la valeur moyenne des écarts normalisés alors que $\beta = 2$ donne la distance euclidienne moyenne. Pour une valeur théorique de $\beta = \infty$, on définit le maximum des écarts normalisés :

$$d^\infty(a_1, a_2) = max_{i=1..n} \left| \frac{x_{1i} - x_{2i}}{x_{imax} - x_{imin}} \right| \tag{6.50}$$

Des méthodes de nichage par surpeuplement ont été historiquement proposées pour les algorithmes génétiques. Elles proposent de restreindre la compétition entre individus. Le lecteur intéressé pourra se référer à De Jong (1975); Cavicchio (1970); Mahfoud (1995) pour plus de détails. Ces méthodes ont semblé peu efficaces sur des problèmes complexes jusqu'à Thomsen (2004) proposant, CrowdingDE plus tard améliorée par Wong et al. (2012), une variante d'évolution différentielle basée sur le surpeuplement pour l'optimisation multimodale et a démontré de très bonne performance.

Méthode de partage

Les méthodes de nichage se basent donc généralement sur la notion de partage. La première méthode proposée Goldberg et al. (1987) consiste à ajuster l'évaluation (l'adaptation dans les algorithmes génétiques) des individus en fonction des ressources disponibles et du nombre d'individus dans son voisinage. Cela réduit l'évaluation des individus dans les régions à forte densité pour encourager les recherches dans d'autres zones. On utilise alors :

$$f_i^s = \frac{f_i}{\sum_{j=1}^{N} sh(d_{ij})} \tag{6.51}$$

La fonction de partage, sh, dépendante de la distance d_{ij} entre deux individus i et j, mesure la similarité entre les individus et vaut 0 si $d_{ij} > \sigma_s$, sinon :

$$sh(d_{ij}) = 1 - (d_{ij}/\sigma_s)^\alpha \tag{6.52}$$

où α est un paramètre fixant la forme de la décroissance de la fonction de partage en fonction de la distance. σ_s est un paramètre de la méthode appelé rayon de nichage. Sa valeur optimale est difficile à estimer, car nécessite de connaître la distance a priori entre les optimums. Le paramétrage du nombre d'individus optimal dans la population peut également s'avérer difficile, car le nombre d'optimums trouvable théorique dépendant directement de ce paramètre. Il est donc crucial.

Une méthode séquentielle, basée sur de multiples optimisations unimodales a également été proposée Beasley et al. (1993), méthode rappelant les méthodes de scalarisation en optimisation multiobjectif (section 1.7).

Méthode d'éclaircissement

Cette méthode Petrowski et al. (1997) de spéciation se base sur le principe de ressource limitée. Plutôt que de partager les ressources entre les individus, elles sont attribuées uniquement aux meilleurs, ne préservant alors que les k individus les mieux évalués d'une niche. Une niche est, cette fois encore, définie par la notion de distance entre les individus par rapport à un paramètre, σ_s, cette fois appelé rayon d'éclaircissement, et peut être adapté dynamiquement Pétrowski et al. (1997).

Pour s'avérer efficace, cette méthode doit souvent être couplée avec des stratégies basées sur l'élitisme, conservant à coup sûr les meilleurs individus des sous-populations.

Méthodes récentes basées sur le clustering

Le clustering, est une méthode populaire de machine learning utilisée pour séparer un ensemble d'objets en différentes classes. Les méthodes présentées jusque-là ont incorporé une première forme de clustering. Elles semblent donc intuitivement adaptées à l'optimisation multimodale.

Ainsi récemment, il a été proposé de coupler l'utilisation d'essaim particulaire et de méthodes de clustering Mehmood et al. (2018). Ces approches se basent principalement sur le *k-means clustering* Rana et al. (2013); Niu et al. (2015) mais d'autres variantes peuvent exister avec par exemple le *k-horminic means clustering* Abshouri and Bakhtiary (2012); Bouyer and Hatamlou (2018). D'autres méthodes basées sur l'évolution différentielle ont pu être proposées avec efficacité telles que Wang et al. (2017) ou encore Wang et al. (2019b).

6.4 Implémentation des méthodes d'optimisation

6.4.1 Bibliothèque de méthodes d'optimisation

Pour réaliser nos travaux d'optimisation, la première étape a été de réaliser une bibliothèque de méthodes d'optimisation continue, se voulant les plus génériques possibles afin de pouvoir les appliquer facilement à tous les problèmes que nous allions rencontrer.

Le but de cette bibliothèque est de proposer des méthodes d'optimisation pouvant chacune s'appliquer à un maximum de problèmes. Cela permet de tester des méthodes différentes en quelques lignes pour pallier le problème mis en lumière par le "no free lunch theorem" [Wolpert and Macready (1997)] à savoir qu'il est impossible de déterminer à l'avance quelle métaheuristique sera la plus efficace sur un problème donné. La grande généricité des méthodes mises en oeuvre peut cependant avoir un coût en efficacité. Celui-ci est très souvent négligeable en pratique, mais si une performance extrême est nécessaire, il peut être à envisager de ré-implémenter la méthode sélectionnée afin de la rendre spécifique au problème en question.

Afin d'assurer une vitesse d'exécution optimale, nous avons réalisé cette bibliothèque entièrement en C++. Une première version en C avait été proposée, mais sa difficulté d'utilisation nous a contraints à la porter sur du C++ afin de proposer des constructions plus courantes et faciles d'utilisation. De même, nous n'avons utilisé aucune bibliothèque non standard afin d'assurer une portabilité optimale. Une seule

exception a été faite plus tard pour l'utilisation de l'algorithme de Mersenne Twister afin de proposer une génération de nombres pseudo-aléatoire optimale.

La figure 6.6 présente un diagramme de séquence simplifié de la mise en place d'une optimisation. On distingue deux phases :

— l'initialisation du problème, détaillée en section 6.4.1, valable pour n'importe quelle optimisation.

— l'optimisation, dépendante de chaque algorithme.

Le déroulement de l'optimisation peut également varier en fonction des problèmes et des critères d'arrêt définis par l'utilisateur. Cet aspect sera détaillé en section 6.4.1.

Dans un premier temps, l'utilisateur initialise le problème à résoudre. Il définit ensuite le ou les algorithmes d'optimisation à utiliser. La première génération sera créée à travers des *Solutions* contenant chacune des jeux de *Paramètres* correspondant aux critères définis dans le *Probleme*. À chaque itération, l'algorithme d'optimisation modifiera les *Solutions*, voire en créera de nouvelles, ou en supprimera. Ces solutions seront finalement évaluées et, si un des critères d'arrêt est validé, la meilleure solution sera retournée à l'utilisateur. À noter que, dans le cas d'un algorithme à population de solutions, l'ensemble de la dernière génération reste stocké dans le *Probleme*.

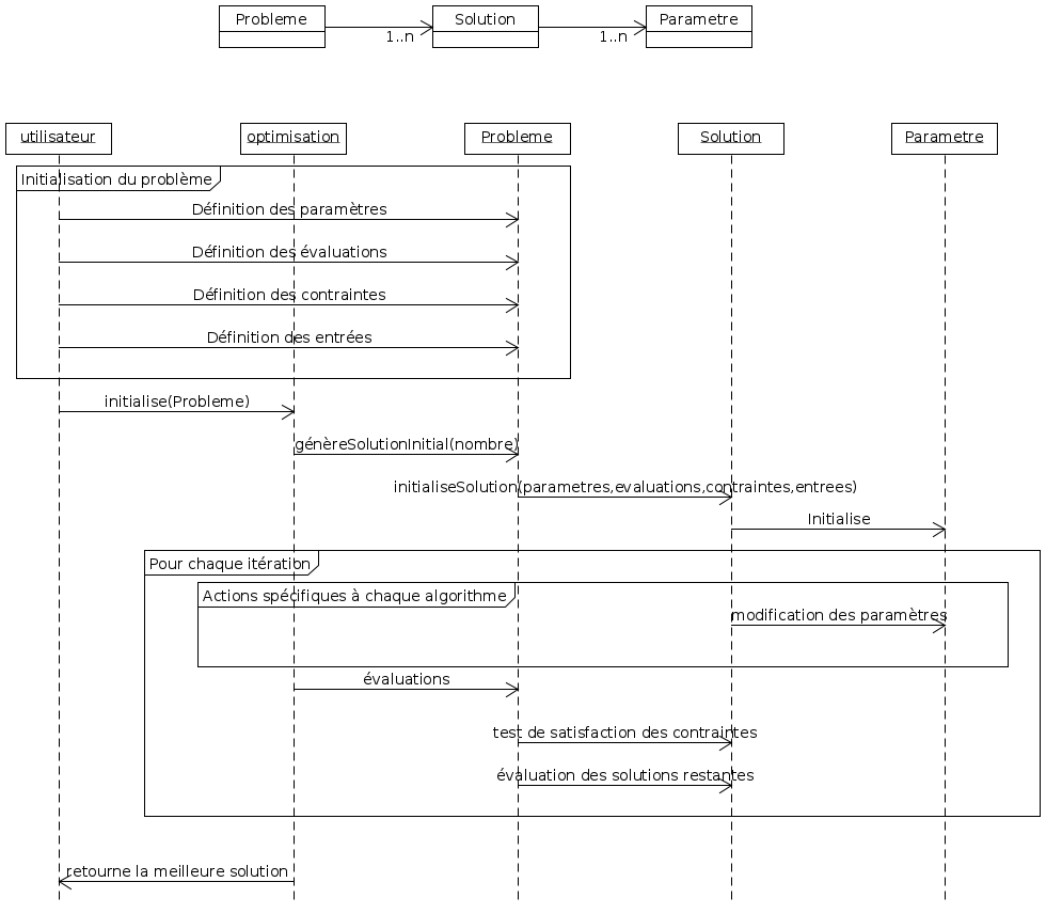


FIGURE 6.6 – Présentation générale

Formalisme de définition d'un problème

Un problème d'optimisation vise à trouver un jeu de paramètres permettant de répondre à certaines contraintes tout en atteignant un ou des objectifs spécifiques. Ainsi, notre classe *Probleme* est principalement composée de trois parties :

— les paramètres : ensemble de ce qui va varier au cours de l'optimisation afin de chercher le meilleur résultat

— les objectifs : ensemble des fonctions permettant d'évaluer une solution

— les contraintes : ensemble de fonctions à satisfaire afin que la solution proposée soit valide

À cela nous pouvons ajouter un ensemble d'entrées qui pourront être utilisées dans les fonctions d'évaluation et de contraintes, mais qui ne sont pas destinées à varier. Les critères d'arrêt de l'optimisation seront également stockés dans le problème en lui-même, mais utilisés uniquement dans l'optimisation, nous les détaillerons donc en section 6.4.1.

Paramètres Pour les applications auxquelles nous allions devoir faire face, les paramètres prennent toujours la forme de nombres réels définis sur un intervalle continu (ou sur R). C'est le paramétrage par défaut dans notre bibliothèque, mais il est également possible de gérer des nombres entiers ou des booléens.

Évaluations L'évaluation est l'étape la plus importante en optimisation. C'est elle qui va déterminer ce que l'algorithme va chercher à accomplir. Pour gérer les méthodes de scalarisation (section 1.7) en optimisation multiobjectif, il est possible d'ajouter des poids sur chaque objectif.

Un vecteur de pointeurs de fonctions gère l'exécution de l'ensemble des fonctions d'évaluation que l'utilisateur peut définir. Dans le cas d'optimisation via simulation, il est possible de se servir d'une fonction d'évaluation pour réaliser la simulation et évaluer la solution directement. Cette méthode est souvent suffisante en optimisation mono-objectif sur un modèle simple. L'utilisateur peut également définir une fonction de *simulation* qui sera exécutée automatiquement avant chaque évaluation. Un vecteur de $void*$ permet de stocker les sorties de simulations afin de pouvoir s'en servir pour les évaluations.

Contraintes Les contraintes permettent de définir la validité ou non d'une solution selon des critères. Nous en avons implémenté plusieurs types pour gérer différents besoins et types d'algorithmes. Celles-ci sont gérées par des classes contenant un pointeur de fonction pour que l'utilisateur puisse définir n'importe quelle fonction de contrainte voulue. Elles sont exécutées automatiquement lors de la phase d'évaluation. Elles peuvent servir soit à pénaliser l'évaluation par ajout de scalaire ou en multipliant sa valeur par un certain facteur, soit à éliminer la solution (méthode de la peine de mort).

La classe *Contrainte* contient également un pointeur de fonction pouvant être utilisé pour proposer une heuristique de satisfaction de contrainte.

Entrées Les entrées sont des variables plus complexes que de simples paramètres. Elles sont principalement destinées à des problèmes d'optimisation via simulation où, une entrée sera un objet servant à réaliser la simulation. Elles sont stockées dans un vecteur de $void*$.

En figure 6.7, nous avons représenté un diagramme de séquence simplifié correspondant à l'optimisation de la répartition de l'effort de pêche au sein des différentes zones d'une pêcherie. Les entrées sont utilisées pour stocker les différentes zones de pêche. Chaque paramètre représente l'effort de pêche à appliquer sur une zone à un instant. La simulation permet de déterminer le nombre de captures qui seront réalisées, élément principal de l'évaluation de ce problème. Celle-ci ne peut donc être réalisée qu'après simulation des effets du jeu de paramètres sur la pêcherie.

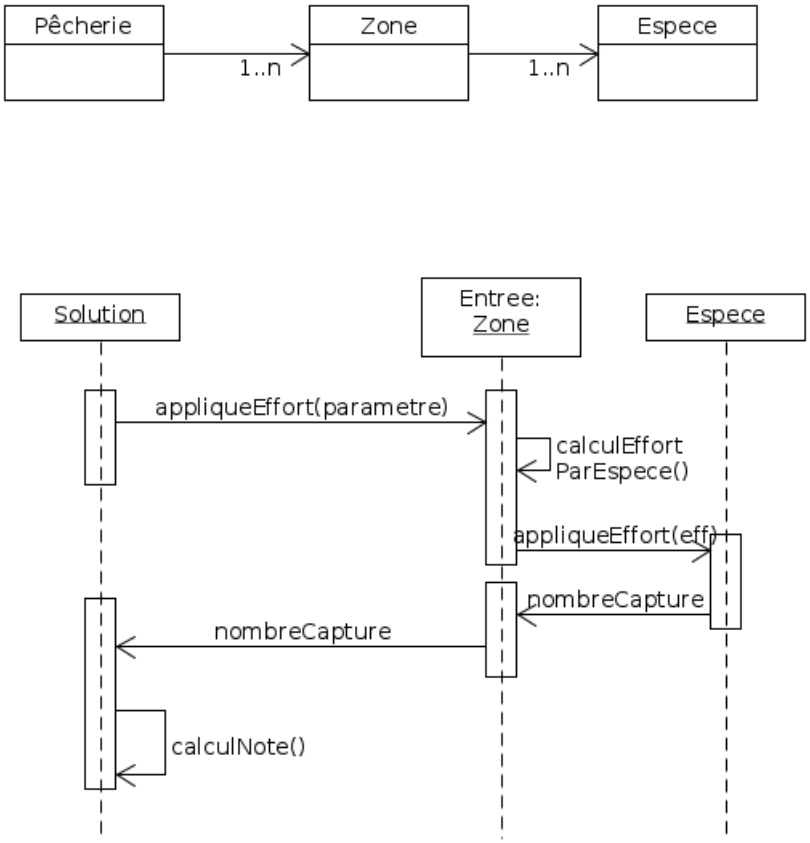


FIGURE 6.7 – Diagramme de séquence simplifié de l'évaluation d'un problème de répartition d'effort de pêche dans des zones

Déroulement d'une optimisation

En figure 6.8, on peut voir que chaque méthode d'optimisation est une classe héritant de la classe abstraite *Optimisation*. Sur ce diagramme seulement deux méthodes sont partiellement représentées par souci de lisibilité. Elles seront toutes détaillées dans la section 6.4.

La classe *Optimisation* contient uniquement la méthode à redéfinir, une variable *nThread*, permettant de gérer la parallélisation quand cela est possible, et le nom de la méthode uniquement utilisé lors de sauvegardes dans des fichiers.

Chaque classe fille contient un certain nombre de pointeurs de fonction permettant d'utiliser n'importe quelle variante de l'algorithme dans la même classe. Chacun prend un pointeur vers un objet du type de la classe en paramètre. Cette construction permet à n'importe quel utilisateur qui le souhaite de définir une nouvelle variante d'une des fonctions de l'algorithme sans avoir à modifier la classe. Lors de l'optimisation, ces fonctions sont appelées en passant l'objet actuel en argument (*this*). Ainsi, il est possible d'utiliser

des variables et méthodes de la classe depuis une fonction définie à l'extérieur.

Bien sûr, nous sommes conscients que ce type de construction peut engendrer des problèmes de sécurité. Le but de cette bibliothèque n'étant que d'effectuer des tests de méthodes et non de servir lors de procédés critiques, nous avons choisi de négliger ce point au profit de la généricité.

Chaque classe fille possède également des variables qui lui sont spécifiques afin de gérer les différentes variantes possibles de la méthode.

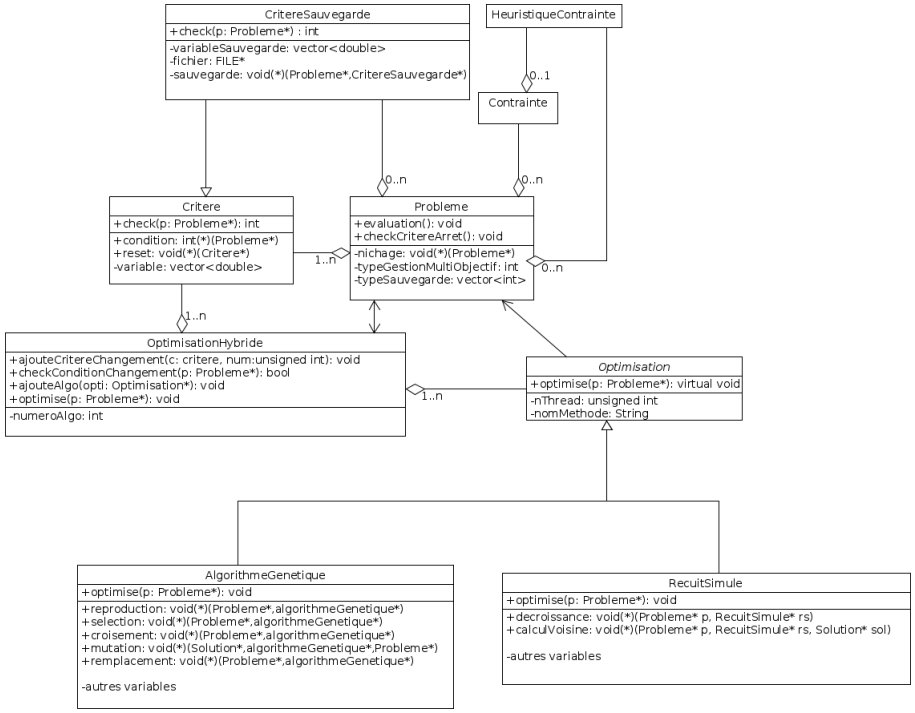


FIGURE 6.8 – Diagramme de classe simplifié de la gestion des méthodes d'optimisation

Critères La classe *Critere* sert à définir les critères d'arrêt d'une optimisation classique et les critères de changement d'algorithme lors d'une optimisation hybride. Nous reviendrons sur ce point en section 6.4.1.

Les critères d'arrêt de l'optimisation sont stockés au sein de la classe *Probleme*. Pour chaque itération de l'optimisation, la fonction *check* de chacun de ces critères est appelée. Elle permet de vérifier s'il est validé. Cette fonction se servira du vecteur de variables comme de paramètres. Pour faciliter l'utilisation, nous avons créé un certain nombre de critères classiques directement initialisables en une fonction, comme les critères de nombre d'itérations, de temps de calcul ou de convergence.

Chaque critère possède également un pointeur de fonction *reset* servant à réinitialiser les variables lors d'optimisation hybride ou pour créer des critères de sauvegarde par exemple.

Sauvegarde La classe *Probleme* peut également contenir des *CritereSauvegarde*. Cette classe hérite de *Critere* et fonctionne de la même manière à la différence que si le critère est validé, le pointeur de fonction sauvegarde sera exécuté. Un pointeur vers un fichier est également stocké afin de pouvoir y écrire. De plus un autre vecteur de variables permet, dans le cas de sauvegardes complexes, dont les variables ne sont pas directement stockées dans *Probleme*, d'en définir autant que désiré.

Comme pour les critères d'arrêt, un certain nombre de critères de sauvegarde ont été définis et peuvent être initialisés en une ligne afin de faciliter l'utilisation et la compréhension de l'utilisateur. Mais comme pour les critères d'arrêt, n'importe quel critère de sauvegarde peut être imaginé.

Optimisation hybride Pour rappel, l'optimisation hybride consiste à tirer profit de différentes méthodes d'optimisation en les enchaînant selon des critères particuliers. Souvent, les métaheuristiques sont combinées à des heuristiques spécifiques à un problème, mais des enchaînements de métaheuristiques peuvent également être possibles.

Nous proposons la classe *OptimisationHybride* pour réaliser cela. La variable la plus importante est certainement le vecteur d'*Optimisation* appelé *fonction*. Les méthodes d'optimisation à enchaîner doivent être stockées ici. Ainsi, la méthode *optimise* va commencer par lancer la première méthode. À chaque itération, en plus des critères d'arrêt et de sauvegarde habituels, seront vérifiés les critères de changement d'algorithmes.

Critères de changement d'algorithmes Les critères de changement d'algorithmes fonctionnent de la même manière que les critères définis précédemment, à ceci près qu'en cas de validation, ils lanceront un nouvel algorithme.

Lorsqu'un critère de changement est validé, la fonction *reset* de chaque critère de changement d'algorithme est appelée. Cette fonction sert à remettre le critère dans son état d'origine pour ne pas fausser les cas cycliques.

6.4.2 Algorithme génétique

Présentation générale Les algorithmes évolutionnaires, dont l'algorithme génétique fait partie, s'inspirent directement de la théorie de l'évolution darwinienne. Les solutions potentielles du problème sont des individus soumis à l'évolution et leur évaluation détermine leurs capacités de reproduction et de survie. L'ensemble des individus constitue la population. Celle-ci évolue à chaque itération appelée génération.

L'algorithme génétique est certainement la métaheuristique la plus connue. Il a fait ses preuves depuis longtemps et reste encore très fréquemment utilisé encore aujourd'hui dans des domaines variés.

Dans le domaine énergétique on peut par exemple citer Ahmadi et al. (2015) travaillant sur l'efficacité de pompes à chaleur, ou encore Yu et al. (2015) où les auteurs cherchent à optimiser le design des immeubles pour une meilleure conservation de l'énergie thermique. Leur problème est multiobjectif et basé sur des simulations de la consommation d'énergie ainsi que le confort des habitations.

Dans un tout autre registre on peut également citer Gil et al. (2018) pour l'optimisation de l'alignement des ontologies du web sémantique ou encore Shima et al. (2006) dans le domaine militaire pour de la coopération entre véhicules aériens inhabités.

Algorithme 15 Algorithme génétique

```
Initialisation, évaluation de la population initiale
tant que testCritèresArrêt() faire
    Sélection
    Croisement
    Mutation
    Évaluation
    Remplacement
fin tant que
```

L'algorithme 15 présente le déroulement global. Dans un premier temps, la population initiale est générée aléatoirement et évaluée. À chaque itération, un certain nombre d'individus seront sélectionnés pour reproduction. Ainsi, leurs caractéristiques seront croisées pour obtenir une génération fille. Au cours de la reproduction, des mutations peuvent s'opérer. La nouvelle génération est ensuite évaluée et une phase de remplacement a lieux.

Au cours du temps, un très grand nombre de variantes ont été proposées. Ainsi, dans notre bibliothèque, chaque fonction utilisée dans la boucle générationnelle est passée en paramètre ce qui rend l'algorithme complètement générique et valide pour n'importe quelle implémentation désirée. Bien sûr, nous n'avons pas pu toutes les écrire. Nous nous sommes pour l'instant concentré sur les plus fréquentes.

Représentation Les caractéristiques des individus doivent généralement être codées sous forme de gènes. En pratique, la plupart du temps, elles sont codées sous forme binaire, entière ou réelle. Ainsi, l'implémentation que nous avons faite des paramètres du problème (section 6.4.1) correspond directement à ce codage.

La version classique de l'algorithme génétique présente une phase de codage / décodage des caractéristiques des individus. Nous avons fait le choix de ne pas l'inclure directement au sein de la boucle générationnelle. En effet, si une représentation différente de celle admise par nos paramètres s'avère nécessaire, elle serait certainement spécifique au problème en question et donc non générique. Si toutefois une phase de codage spécifique était nécessaire à l'utilisateur, une version de l'algorithme a été prévue à cet effet. L'incorporation de mécanismes de transcodage pour chaque représentation classique (matrice, vecteur etc.) et chaque algorithme peut être une future amélioration possible de la bibliothèque afin de faciliter son utilisation.

Les différentes phases de l'optimisation, notamment le croisement et la mutation, dépendent souvent du codage des informations. Ainsi, dans les sections suivantes, nous ne détaillerons que les cas les plus courants, à savoir les codages binaires, entiers et réels. Pour des problèmes mettant en oeuvre, par exemple,

des graphes orientés, une forme de codage par chemins ou séquences peut exister. Nous n'avons cependant pas encore eu le temps de l'implémenter. De plus, ce type de problème présente souvent des contraintes particulières nécessitant des applications spécifiques plutôt que génériques. Cette amélioration possible n'est donc pas une priorité pour nous.

Sélection Les opérateurs de sélection regroupent souvent ce que nous avons appelé *sélection* et *remplacement*. Le premier correspond à la sélection pour reproduction alors que le second représente la sélection environnementale, pour la survie. Dans cette partie, nous ne détaillerons que la sélection pour la reproduction.

Au cours des itérations, le nombre d'individus reste fixe la plupart du temps, il peut varier mais devra toujours être contrôlé. Ainsi, le rôle principal des opérateurs de sélection est de déterminer le nombre de fois qu'un individu se reproduira. Les individus dont l'évaluation est la plus importante seront prioritaires.

Sélection proportionnelle Le premier type de sélection à avoir été conçu est la **sélection proportionnelle**. Elle est définie Holland (1992b) pour tout problème de maximisation positive ce qui peut nécessiter des transformations de la fonction d'évaluation pour pouvoir l'utiliser. Ainsi, pour chaque solution on a :

$$nEnfants_i = \frac{nEnfantsTotal}{\sum_{j=1}^{n} f_j} f_i \tag{6.53}$$

avec :

— $nEnfants_i$: le nombre d'enfants que pourra espérer l'individu i

— $nEnfantsTotal$: le nombre total d'enfants voulu pour cette génération (souvent fixe)

— n : le nombre total d'individus cherchant à se reproduire

— f_i : l'évaluation de l'individu i

Or, le nombre d'individus est forcément un entier, c'est pourquoi ce qui vient d'être calculé n'est qu'une espérance. En effet, les individus vont devoir subir une méthode d'échantillonnage stochastique à savoir la **méthode de la roulette**Holland (1992b), aussi appelée loterie biaisée, ou la **méthode d'échantillonage stochastique universel** Baker (1987).

Loterie biaisée La loterie biaisée consiste à tirer au hasard les parents de chaque enfant. Un individu aura d'autant plus de chances d'être parent que son évaluation sera élevée. Cette méthode présente le gros désavantage de permettre des cas extrêmes comme de ne sélectionner que de mauvais parents pour une génération et donc régresser dans les solutions proposées, engendrant ainsi une dérive génétique.

Méthode d'échantillonage stochastique (SuS) La méthode d'échantillonage stochastique universel quant à elle consiste à répartir aléatoirement les individus sur un segment. La place que chacun occupera sera proportionnelle à son évaluation. On répartit ensuite un ensemble de points de façon équidistante

sur ce segment, la position de chaque point correspondant à un futur parent. Une représentation de cette méthode est visible en figure 6.9.

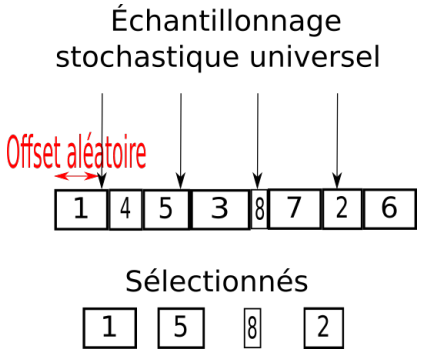


FIGURE 6.9 – Méthode d'échantillonnage stochastique universel

Pression de sélection Ces types de sélection peuvent s'avérer peu efficaces si la pression de sélection est mal gérée. En effet, si les évaluations sont trop proches, toutes les solutions auront autant de chances ou presque d'être sélectionnées. Ainsi, un ajustement, linéaire ou exponentiel, de la fonction évaluation peut être nécessaire.

Sélection déterministe La sélection peut également se faire en fonction du rang de l'individu. La **sélection déterministe**, par exemple, consiste à sélectionner simplement les n meilleurs individus d'une population. Cette sélection engendre cependant bien souvent une convergence prématurée de l'ensemble de solution.

Sélection par tournois La **sélection par tournois** est souvent préférable. Les **tournois déterministes**, avec ou sans remise, consistent à tirer de façon équiprobable n individus et les faire "s'affronter". Le vainqueur est celui qui a la note la plus élevée. Les **tournois stochastiques** en revanche laissent une chance de victoire à n'importe quel individu. Le vainqueur est tiré au hasard mais plus la note d'un individu est élevée, meilleures sont ses chances. On peut également citer les **tournois de Boltzmann** Maza and Tidor (1993) et Goldberg et al. (1990) que nous n'avons pas encore implémentés.

Croisement Un grand nombre de types de croisement existent dans la littérature, nous ne détaillerons ici que ceux que nous avons directement implémentés. Le lecteur intéressé pourra se référer à Picek et al. (2013) pour trouver d'autres exemples ainsi que des comparaisons entre eux.

Par la suite, nous supposerons que les croisements s'effectuent sur deux parents. Bien que biologiquement ce soit impossible, dans ce type d'algorithme, rien n'interdit le croisement multi-parents Eiben et al. (1995). Il est cependant rarement utilisé en pratique. Nous ne l'avons donc pas implémenté pour le moment.

Croisements binaires Les croisements qui vont être détaillés ici s'appliquent généralement sur des solutions à représentations binaires, mais ils peuvent également s'appliquer à des représentations entières. Il est souvent quand même préférable que ces entiers soient bornés.

Croisement un point Le croisement un point Holland (1992a) est certainement le plus simple qui existe. Il consiste à simplement définir un point de coupure dans chacun des parents, tiré aléatoirement dans l'ensemble de reproduction. Comme vous pouvez voir en figure 6.10, la partie gauche du premier parent sera copiée dans un des enfants alors que la partie droite ira dans le deuxième et inversement pour le deuxième parent. On obtient ainsi deux nouveaux individus.

À noter que la position du point de coupure est généralement choisie au hasard à chaque itération, voire pour chaque couple de parents, afin d'éviter une convergence trop rapide. En effet, avec un point fixe, deux parents qui se reproduiraient plusieurs fois donneraient à chaque fois les mêmes enfants.

Croisement multi-point, croisement uniforme Le principe du croisement un point peut se généraliser. En pratique, on utilise souvent le croisement deux points ou le croisement uniforme Ackley (1987). Il peut être considéré comme un croisement multi-points dont le nombre de points de coupure est indéterminé. À chaque itération, ou pour chaque couple de parents, un masque de croisement est généré. Ce masque est un mot binaire avec autant de bits qu'il y a de dimensions au problème. La présence d'un 1 à la $n$ième position de ce mot indique l'échange de ce bit avec l'autre parent. La figure 6.10 illustre ce principe. En général, la proportion de 0 ou de 1 des masques de croisement uniforme est tirée de façon équiprobable.

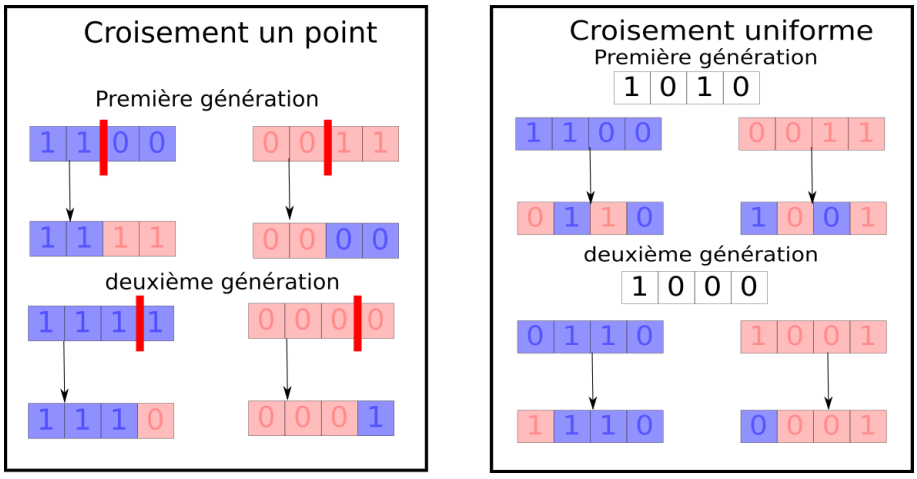


FIGURE 6.10 – Croisement binaire

Croisements réels Les croisements par points détaillés précédemment peuvent s'appliquer aux problèmes à représentation réelle. Ils sont alors appelés croisement par échange de composantes.

Croisement BLX-α volumique Le croisement BLX-α volumique Nomura and Shimohara (2001) et Picek et al. (2013) consiste à générer deux nouveaux individus à l'intérieur d'un hyper-rectangle tels que les deux parents et *alpha* définissent une de ses plus grandes diagonales. En pratique, la formule suivante est simplement utilisée pour chaque composant de z :

$$z_i = x_i - \alpha(y_i - x_i) + (1 + 2\alpha)(y_i - x_i)\mathcal{U}(0, 1) \tag{6.54}$$

avec :

— z_i : $i$ème composant de l'enfant z

— x_i : $i$ème composant du parent x

— y_i : $i$ème composant du parent y

— α : paramètre de cette fonction de croisement

— $\mathcal{U}(0, 1)$: nombre choisi au hasard, uniformément sur $[0, 1]$.

Une valeur typique de α utilisée est 0.5.

Une représentation en dimension 2 de cette méthode peut être trouvée en figure 6.11.

Croisement BLX-α linéaire Ce croisement peut être trouvé sous plusieurs dénominations comme "croisement arithmétique" ou encore "recombinaison intermédiaire".

Comme pour le croisement BLX-α volumique, il permet de générer deux enfants à la différence que ceux-ci se situeront sur une droite passant par les deux parents (figure 6.11.

En pratique il est réalisé via la formule suivante :

$$z = x - \alpha(y - x) + (1 + 2\alpha)(y - x)\mathcal{U}(0, 1) \tag{6.55}$$

Les termes de cette formule sont les mêmes que pour le croisement BLX-*alpha* volumique.

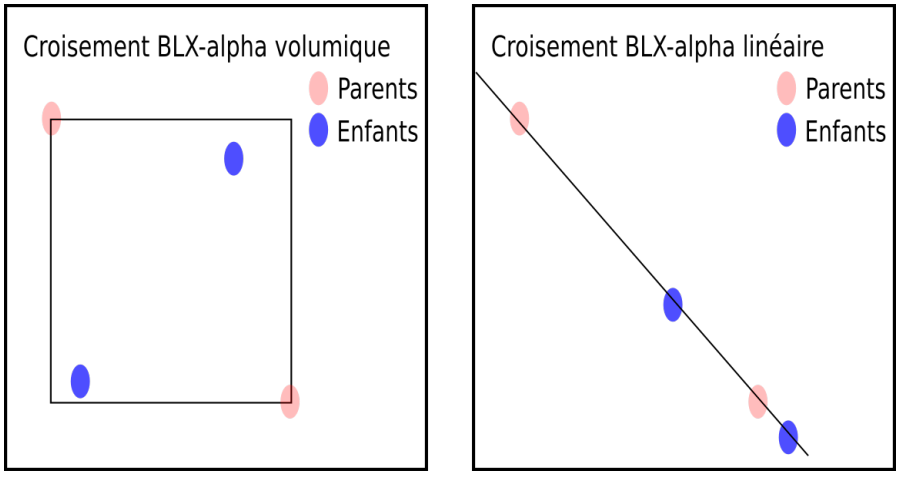


FIGURE 6.11 – Croisement BLX-α

Mutation La mutation est l'opérateur de variation qui va servir principalement à augmenter l'exploration de l'espace des solutions.

Comme vous pouvez le voir dans notre exemple en figure 6.12, par simple croisement, le bit de poids faible sera toujours à 1. Ainsi, la moitié de l'espace des solutions ne sera jamais explorée. L'opérateur de mutation permet de pallier ce problème en introduisant, la plupart du temps, des variations aléatoires des caractéristiques des individus.

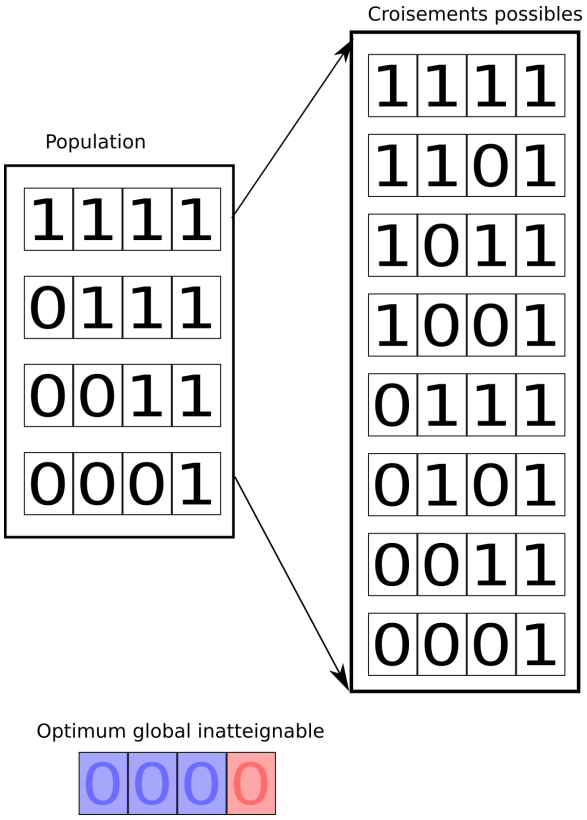


FIGURE 6.12 – Limite de l'opérateur de croisement

Mutation binaire En représentation binaire, l'opérateur de mutation le plus fréquemment utilisé consiste en une inversion de bit ($0 \rightarrow 1$, $1 \rightarrow 0$).

On peut par exemple citer :

— La **mutation déterministe** : fixe un nombre de bits à modifier par itération par individu. Les bits en questions sont choisis au hasard.

— La **mutation bit-flip** : chaque bit de chaque individu à une chance de muter selon le taux de mutation choisi. En pratique, ce taux est fixé et faible afin d'éviter une trop grande variation des individus, rendant inutile la phase de croisement.

Mutations réelles

Mutation uniforme La mutation la plus classique en représentation réelle consiste en l'ajout d'une variable aléatoire appartenant à un hypercube de coté a, a étant un paramètre de la mutation. Ce type de mutation ne permet cependant pas de s'échapper d'un optimum local situé sur un pic plus large que l'hypercube. Ainsi, l'utilisation de distributions à support non borné est souvent préconisé.

Mutation gaussienne Comme pour la mutation uniforme, la mutation gaussienne ajoute une variable aléatoire à la solution proposée. En revanche, cette variable suit une fonction gaussienne centrée sur 0 et dont l'écart-type σ est à déterminer. Ce type de mutation permet, en théorie, de s'échapper de n'importe quel optimum local. En pratique, les valeurs extrêmes ont des probabilités tellement faibles d'être tirées qu'il peut être préférable d'utiliser des distributions dites "à queues plus épaisses" telles que les distributions de Cauchy ou Laplace Yao and Liu (1996) et Montana and Davis (1989). Il est cependant souvent préféré de modifier σ au cours de l'optimisation par des méthodes auto-adaptatives.

Règle des 1/5 Après une étude approfondie, Rechenberg (1973) a été en mesure de déterminer empiriquement que si σ est optimal, environ 1/5 des mutations seront bénéfiques (provoquera une amélioration de la solution). Ainsi, il propose la méthode auto-adaptative suivante : si le taux de mutation bénéfique est supérieur à 1/5 : augmenter σ. Sinon, le réduire. Schwefel (1981) propose la valeur 0.85 comme facteur de variation de σ pour $10n$ mutations avec n le nombre de dimensions de l'espace de recherche.

Mutation polynomiale La mutation polynomiale Deb and Goyal (1996) est plus complexe algorithmiquement que la plupart des autres méthodes de mutation. Elle semble cependant présenter de très bons résultat en optimisation multiobjectif Hamdan (2012). Nous ne présenterons ici que la version initiale (algorithme 16) mais d'autres versions ont été proposée Hamdan (2012). Dans cet algorithme, on a :

- X_i : valeur du paramètre i pour la solution X
- X_i^{Upper}, X_i^{Lower} : respectivement les valeurs maximales et minimales du paramètre i
- η_m : paramètre de l'algorithme. Plus celui-ci est élevé plus les solutions filles pourront être éloignées des parents
- P_m : probabilité de mutation pour chaque paramètre

Remplacement Cette phase, aussi appelée sélection environnementale, permet de déterminer quels individus survivront jusqu'à la prochaine génération.

Remplacement générationnel Ce remplacement consiste simplement à tuer tous les parents et ne garder que les enfants d'une génération à l'autre. Ce remplacement suppose qu'à chaque génération, autant d'enfants sont créés qu'il y avait de parents.

Algorithme 16 Mutation polynomiale

$i \leftarrow 0$
pour $i < n;\ i++$ **faire**
  $r \leftarrow \mathcal{U}(0,1)$
  si $r <= P_m$ **alors**
    $\delta \leftarrow \frac{min(X_i^{Upper} - X_i, X_i - X_i^{Lower}}{X_i^{Upper} - X_i^{Lower}}$
    $r \leftarrow \mathcal{U}(0,1)$
    si $r <= 0.5$ **alors**
      $\delta_q \leftarrow [2r + (1-2r)(1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}} - 1$
    sinon
      $\delta_q \leftarrow 1 - [2(1-2r) + 2(r-0.5)(1-\delta)^{\eta_m+1}]^{\frac{1}{\eta_m+1}}$
    fin si
    $X_i \leftarrow X_i + \delta_q(X_i^{Upper} - X_i^{Lower})$
  fin si
fin pour

Remplacement des stratégies d'évolution C'est un remplacement déterministe qui consiste simplement à garder les n meilleurs enfants. n étant le nombre d'individus que l'ont veut par génération. En général, ce type de remplacement est utilisé avec une phase de croisement générant un nombre d'enfants supérieur à n afin de ne sélectionner que les meilleurs.

Remplacement stationnaire Ce type de remplacement est très particulier puisqu'il suppose qu'à chaque génération seulement quelques enfants sont engendrés et remplaceront des parents de la population initiale de façon stochastique ou selon des critères de performances.

Il peut être utile lorsque la représentation d'une solution se répartit sur plusieurs individus, mais ce cas est assez rare. En pratique, il favorise souvent la dérive génétique De Jong and Sarma (1993), il est donc peu utilisé.

Stratégies élitistes Ces stratégies de remplacement consistent à garder au moins l'individu ayant la meilleure performance. Ainsi, une stratégie élitiste peut s'appliquer à n'importe quelle stratégie que nous avons présentée précédemment.

6.4.3 Recherche par harmonie

Présentation Contrairement à un grand nombre de métaheuristiques, la recherche par harmonie s'inspire de la création de nouveaux accords musicaux. Elle se base sur le fait que, lors d'une improvisation, un musicien part de se qu'il connaît déjà et y applique de petites variations. De même, lors de la recherche d'un accord parfait pour un nouveau morceau, il partira d'un accord harmonique auquel il appliquera de faibles modifications en espérant l'améliorer progressivement.

La première version de l'algorithme a été proposée en 2001 Geem et al. (2001) et est présentée en algorithme 17. Il consiste, dans un premier temps, à générer une population initiale et l'évaluer. Pour

chaque itération tant qu'un critère d'arrêt n'est pas satisfait, l'algorithme va chercher à créer de nouveaux individus de plus en plus harmonieux. Quand un individu est créé, une phase de remplacement a lieu afin d'évaluer s'il est plus utile que les accords déjà présents dans la mémoire harmonique.

Algorithme 17 Recherche par harmonie

```
Initialisation, évaluation de la population initiale
tant que testCritèresArrêt() faire
  Improvisation
  Remplacement
fin tant que
```

Improvisation Pour rester dans la métaphore musicale, chaque dimension du problème est considérée comme un instrument. L'improvisation consiste en la création d'un nouvel accord en s'inspirant des différents instruments déjà présents dans la mémoire harmonique. Ainsi, pour chaque instrument, la note peut être soit choisie au hasard, soit prise dans les accords déjà évalués. Cette probabilité est un paramètre de l'algorithme appelé HMCR (Harmony Memory Considering Rate). Ce paramètre est souvent assez élevé pour permettre l'exploitation de la mémoire : entre 0.7 et 0.98. De même, une fois l'accord sélectionné, la tonalité peut être ajustée ou non selon une probabilité appelée PAR (Pitch Adjusting Rate) valant environ 0.3 historiquement. Cette variation se fait en pourcentage de l'amplitude des valeurs, en général entre 1 et 10%.

Vous trouverez en algorithme 18 l'algorithme correspondant.

— $\mathcal{U}([x : y])$: distribution uniforme entre x et y

— v_i : valeur sur la dimension i

— $m_{j,i}$: valeur de la dimension i du $j$ème élément de la mémoire harmonique

— β : variation maximale

Remplacement Le nouvel accord généré est évalué afin d'être comparé à ceux déjà présents dans la mémoire harmonique. La variante la plus simple est de remplacer l'accord le moins bon à condition que le nouveau soit meilleur. Il peut être cependant intéressant de vérifier que notre accord ne corresponde pas à un autre déjà présent dans la mémoire afin d'éviter les doublons et maintenir une certaine diversité des solutions. Cela empêche une convergence prématurée.

Variantes

IHS Contrairement à beaucoup de méthodes récentes, la recherche harmonique a connu un assez grand succès. Ainsi, des propositions d'améliorations ont été faites. La plus connue est sûrement la recherche harmonique améliorée (Improved Harmony Search : IHS) Mahdavi et al. (2007) proposée en 2007. Cette variante propose de modifier les valeurs des paramètres au cours du temps suivant les formules suivantes :

Algorithme 18 Recherche par harmonie : improvisation

pour chaque paramètre du problème : i **faire**
  si $\mathcal{U}([0:1]) <= HMCR$ **alors**
    $j = \mathcal{U}([1 : tailleMemoire])$
    si $\mathcal{U}([0:1] <= PAR)$ **alors**
      $v_i = m_{j,i} + \mathcal{U}([-1:1]) \times \beta$
    sinon
      $v_i = m_{j,i}$
    fin si
  sinon
    $v_i = \mathcal{U}([min_i \ : max_i])$
  fin si
fin pour

$$PAR(gn) = PAR_{min} + \frac{PAR_{max} - PAR_{min}}{NI} \tag{6.56}$$

avec :

— NI : nombre d'itérations maximum avant arrêt. Bien sûr, ce paramètre peut être défini à part et l'algorithme sera arrêté suivant d'autres critères

— gn : génération actuelle

$$\beta(gn) = \beta_{max} exp^{c} \tag{6.57}$$

$$c = \frac{ln(\frac{\beta_{min}}{\beta_{max}})}{NI} \tag{6.58}$$

GHS Aux propositions précédentes, Omran and Mahdavi (2008) ajoute une modification de la phase d'improvisation. Il remplace la formule d'adaptation initiale, ainsi : $v_i = m_{j,i} + \mathcal{U}([-1:1]) \times \beta$ devient $v_i = m_{best,\mathcal{U}([1:nombreDimension])}$, avec *best* étant l'indice de la meilleure solution connue à cet instant. Cette variante est connue sous le nom de Global-best Harmony Search.

IAHS L'Improved Adaptative Harmony Search Zhang (2015) propose de repartir de l'IHS en modifiant les valeurs des paramètres au cours du temps. Les nouvelles formules de HMRC, PAR et β deviennent :

$$HMRC(gn) = HMRC_{max} - \frac{(HMCR_{max} - HMRC_{min})}{NI} \tag{6.59}$$

$$PAR(gn) = \frac{PAR_{max} - PAR_{min}}{\pi/2}(gn) + PAR_{min} \tag{6.60}$$

$$\beta(gn) = (\beta_{max} - \beta_{min}) \times exp^{-gn} + \beta_{min} \tag{6.61}$$

Les fonctions de croissance et décroissance proposées pour PAR et β semblent cependant converger beaucoup trop vite. Ainsi, la version proposée par Li et al. (2015) semble plus correcte. Elle garde les mêmes valeurs que IHS pour PAR et β mais ajoute la variation de HMRC proposée précédemment.

Nous proposons une autre variante :

$$PAR(gn) = \frac{PAR_{max} - PAR_{min}}{\pi/2}(gn/\tau) + PAR_{min} \tag{6.62}$$

$$\beta(gn) = (\beta_{max} - \beta_{min}) \times exp^{-gn/\tau} + \beta_{min} \tag{6.63}$$

avec $\tau = \frac{NI}{5}$.

6.4.4 Algorithme de colonie de fourmis

Présentation générale Cet algorithme est directement inspiré de l'intelligence collective des fourmis. En effet, une colonie de fourmis est capable de réaliser des tâches complexes alors qu'individuellement une fourmi est limitée. L'exemple typique est la tâche de recherche de nourriture. Chaque fourmi part chercher de la nourriture en déposant des phéromones sur son chemin. Les fourmis suivantes s'orienteront en fonction de la quantité de phéromones présente sur chaque chemin. Ainsi, le chemin le plus court aura une quantité de phéromones de plus en plus élevée, ayant pour effet d'attirer de plus en plus de fourmis sur le bon chemin.

Cet algorithme est particulièrement utilisé dans des problèmes présentables sous forme de graphe. Ainsi, nous ne l'utiliserons pas par la suite dans la section Benchmark (6.5). De plus, il nécessite de pouvoir estimer l'attractivité de chaque valeur possible de chaque paramètre ce qui est bien souvent problématique.

La première version de l'algorithme Dorigo et al. (1996), présenté en algorithme 19, commence par initialiser un taux de phéromones à chaque chemin possible du graphe. Pour commencer, les phéromones ne doivent pas impacter les choix des fourmis. Elles sont donc toutes initialisées à la même valeur. Une valeur d'intérêt semble être $\tau_0 = n/C$ avec n : le nombre de fourmis et C : le coût d'une solution obtenue par un algorithme glouton.

À chaque itération, une nouvelle génération de fourmis sera initialisée en prenant en compte l'attractivité de chaque chemin et le taux de phéromones présent sur celui-ci. Les solutions sont évaluées et la meilleure est stockée. Une phase de mise à jour des phéromones a ensuite lieu, composée d'une évaporation puis d'un dépôt.

Ant System (AS)

Construction d'une solution Historiquement, ce problème a été utilisé pour la recherche de plus court chemin dans un graphe complet. Cet exemple étant particulièrement bien adapté pour la compréhension,

Algorithme 19 Colonie de fourmis de base : AS

```
Initialisation des phéromones
tant que testCritèresArrêt() faire
  Initialisation de la nouvelle génération
  Évaluation
  Remplacement
  Mise à jour des phéromones
fin tant que
```

nous l'utiliserons ici.

Tant que la fourmi n'a pas fait le tour du graphe jusqu'à revenir à sa position initiale, elle choisit le prochain arc à emprunter en suivant la relation suivante :

$$p_{ij}^k(t) = \frac{\tau_{ij}(t)^\alpha \times \eta_{ij}^\beta}{\sum_{l \in N_{i]^k}} \tau_{il}(t)^\alpha \times \eta_{il}^\beta} \tag{6.64}$$

avec :

— τ_{ij} : la quantité de phéromones présente entre i et j

— η_{ij} : l'attractivité, aussi appelée visibilité, de l'arc i, j. Dans notre exemple elle est définie comme l'inverse de la distance entre i et j.

— α : paramètre permettant de régler l'influence des phéromones, typiquement 1

— β : paramètre permettant de régler l'influence de l'attractivité, typiquement entre 2 et 5.

— N_i^k : ensemble des chemins possibles à cet instant.

Mise à jour des phéromones Après chaque itération, les phéromones présentes sur chaque chemin s'évaporent selon la relation :

$$\tau_{ij}(t+1) = \tau_{ij}(t) \times (1 - \rho) \tag{6.65}$$

ρ étant un paramètre appelé taux d'évaporation. Typiquement, dans cette implémentation, il vaut 0,5.

Après évaluation, notée L^k : la longueur du chemin parcouru dans notre exemple, chaque fourmi dépose une quantité de phéromones, notée Δ_{ij}^k inversement proportionnelle à L^k. Ainsi, la formule de mise à jour de phéromones devient :

$$\tau_{ij}(t+1) = \tau_{ij}(t) \times (1 - \rho) + \sum_{k=1}^{n} \Delta_{ij}^k \tag{6.66}$$

Variantes de AS

Max-Min Ant System (MMAS) Cette variante Stutzle and Hoos (1997) propose des modifications sur la gestion des phéromones. Elle instaure des bornes au taux de phéromones afin d'éviter que la quantité de phéromones de certains arcs tende vers 0. Les bornes sont généralement fixées, mais certaines variantes proposent de diminuer τ_{max} au cours du temps. La quantité initiale de phéromones est fixée à τ_{max}. En cas de stagnation, une réinitialisation est possible.

Lors de la mise à jour des phéromones, uniquement la meilleure fourmi dépose des phéromones sur les arcs qu'elle a parcourus. La fourmi en question peut être soit la meilleure de l'itération, soit la meilleure rencontrée jusque-là.

AS_{rank} L'algorithme AS_{rank} Bullnheimer et al. (1999) propose une variante de la stratégie élitiste précédente en impliquant un nombre σ d'individus dans le dépôt des phéromones.

Ant Colony System (ACS) ACS Dorigo and Gambardella (1997) est une des variantes les plus efficaces sur de nombreux problèmes, elle est donc particulièrement utilisée.

Elle supprime le paramètre α pour ajouter un paramètre noté q_0 permettant de régler le ratio exploitation/exploration. La création de nouvelles solutions est modifiée. Un chemin j est choisi tel que :

$$Si(\mathcal{U}([0:1]) < q_0) : j = argmax_{j \in N_i^k}(\tau_{il} \times \eta_{il}^{\beta}) \tag{6.67}$$

$$Sinon : p_{ij}^k(t) = \frac{\tau_{ij}(t) \times \eta_{ij}^{\beta}}{\sum_{l \in N_{i]^k}} \tau_{il}(t) \times \eta_{il}^{\beta}} \tag{6.68}$$

La mise à jour des phéromones suit la même stratégie élitiste proposée dans MMAS mais la formule n'est appliquée que sur les arcs parcourus par la meilleure fourmi. Elle devient :

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \frac{\rho}{L^+} \tag{6.69}$$

Cela est compensé par une évaporation locale à chaque fois qu'une fourmi passe sur un arc :

$$\tau_{ij} = (1 - \xi)\tau_{ij} + \xi\tau_0 \tag{6.70}$$

ξ étant un nouveau paramètre du système appelé évaporation locale.

Enfin, ACS propose d'utiliser des heuristiques de recherche locale spécifiques au problème défini.

Des valeurs typiques de paramètres sont données en table 6.1, m définissant le nombre de fourmis, n le nombre de sommets et C une estimation du coût d'une solution.

Paramètre	Valeur
β	2 à 5
ρ	0.1
τ_0	1/nC
m	10
ξ	0.1
q_0	0.9

TABLE 6.1 – Valeurs de paramètres pour ACS

6.4.5 Essaim particulaire

Présentation générale L'optimisation par essaim particulaire Eberhart and Kennedy (1995) est la première méthode d'optimisation basée sur un mécanisme de coopération sans sélection. C'est une méthode à population de solutions, mais ici, une population est composée de particules qui peuvent avoir plusieurs rôles tels que l'exploration ou la mémorisation. Une particule est composée d'au moins deux informations, à savoir sa position et sa vitesse. Sa position est ce que nous avons appelé "solution" pour les algorithmes précédents, c'est-à-dire, l'ensemble des paramètres à optimiser dans le problème. La vitesse est un vecteur contenant la variation de position de la particule sur chaque dimension du problème. Elle est, bien sûr, vouée à changer au cours du temps.

Cet algorithme est très apprécié de par sa facilité de codage, mais également grâce à sa capacité d'adaptation à un grand nombre de problèmes. En témoignent les études bibliographiques trouvables très régulièrement sur le sujet Zhang et al. (2015), Kothari et al. (2012), Mavrovouniotis et al. (2017).

La version de base est donnée en algorithme 20. Comme pour les algorithmes présentés précédemment, les différentes fonctions présentes au sein de l'algorithme basique sont en réalité passées en paramètre de l'algorithme afin de pouvoir gérer un grand nombre de variantes de celui-ci.

Il commence par l'initialisation et l'évaluation des particules, initialisation qui consiste en la génération aléatoire de solutions possibles au problème. À chaque itération, les particules subiront un déplacement, c'est à dire, une variation de chacune de ses composantes en fonction de mécanismes que nous détaillerons plus tard. Une phase de confinement a ensuite lieu afin d'éviter que les particules puissent se déplacer à l'extérieur de l'espace de définition. À noter cependant que certaines variantes permettent des dépassements. Après l'évaluation des solutions vient la phase de mémorisation. Elle permet de mettre à jour les mémoriseurs et les liens entre les différentes particules. Ces opérations seront répétées jusqu'à validation d'un critère d'arrêt.

Pour commencer, nous détaillerons les différents points de l'algorithme en nous basant sur la version standard de 1998. Puis nous expliquerons les quelques variantes avec deux autres versions que nous avons implémentées à savoir SPSO (Standard Particle Swarm Optimization) 2007 et SPSO 2011.

SPSO 1998

Algorithme 20 Essaim particulaire

```
Initialisation
Evaluation
tant que testCritèresArrêt() faire
  Déplacement
  Confinement
  Evaluation
  Mémorisation
fin tant que
```

Initialisation Dans cette version, la taille de l'essaim est fixe au cours du temps. L'initialisation des positions se fait aléatoirement selon une distribution généralement uniforme. L'initialisation de la vitesse suit la formule suivante pour chaque dimension d de chaque particule i :

$$v_{i,d}(0) = (\mathcal{U}(x_{min_d}, x_{max_d}) - x_{i,d}(0))/2 \tag{6.71}$$

avec :

- $v_{i,d}$: la vitesse de la particule i sur la dimension d
- $\mathcal{U}(x_{min_d}, x_{max_d})$: une distribution uniforme entre les valeurs minimales et maximales possibles sur la dimension d du problème.
- $x_{i,d}$: la position de la particule i sur la dimension d

La topologie est globale : chaque particule informe et est informée par chaque autre.

Déplacement Lors de la phase de déplacement, chaque particule suit les règles suivantes :

$$v_{i,d}(t+1) = wv_{i,d}(t) + c_1(p_{i,d}(t) - x_{i,d}(t)) + c_2(p_{g,d}(t) - x_{i,d}(t)) \tag{6.72}$$

avec :

- w : inertie, aussi appelée coefficient de confiance en soi. Compris entre 0 et 1. Classiquement, on n'utilise la valeur 0.72.
- c_1 : coefficient de confiance cognitive supérieur à 1.
- c_2 : coefficient de confiance sociale supérieur à 1. Typiquement, $c1 = c2 = 1.2$.
- $p_{i,d}$: mémoriseur de la particule i
- $p_{g,d}$: meilleure position mémorisée par l'ensemble des particules.

Si la vitesse obtenue par la particule est supérieure à v_{max} elle est ramenée à cette valeur. Il en est de même si elle est inférieure à v_{min}.

Ainsi, la position de la particule est mise à jour :

$$x_i(t+1) = x_i(t) + v_i(t) \tag{6.73}$$

Confinement Dans cette version de l'algorithme, la phase de confinement est très simple : elle consiste, pour chaque dimension de chaque particule, à vérifier si elle est dans l'espace de définition. Si oui, aucun changement ne s'applique. Si non, la valeur sur cette dimension est ramenée à la valeur maximale ou minimale en fonction du côté de dépassement. La vitesse de la particule sur cette dimension est alors mise à 0.

Mémorisation Après évaluation, si la position de la particule est meilleure que celle de son mémoriseur, le mémoriseur prend pour valeur cette nouvelle position. Il existe également un mémoriseur global. Celui-ci prend la valeur de la meilleure solution rencontrée jusque-là. Ainsi, à chaque itération, si une particule à une meilleure évaluation que lui, il prend sa valeur, sinon il ne change pas.

Variantes implémentées Au cours du temps, un grand nombre de méthodes ont été proposées. Certaines sont problème spécifiques. La plupart ne consiste qu'en la variation des coefficients de confiance au cours du temps, mais n'en restent pas moins intéressante. Le lecteur intéressé pourra se référer à Eslami et al. (2012) pour plus d'informations. Nous ne présenterons ici que les variantes, dites standard, que nous avons implémentées dans notre bibliothèque.

Problèmes de la version originelle Des variantes standard ont été développées en 2007 et 2011 afin de pallier certains problèmes de la version de base. En effet, cette version entraîne souvent une convergence prématurée. Ce problème semble être la cause de la topologie globale. De plus, il nécessite la fixation de vitesse maximale, qui peut impacter les performances de l'algorithme. Enfin, le comportement de l'algorithme semble dépendre du système de coordonnées choisi.

SPSO 2007 La topologie utilisée n'est plus globale mais locale Clerc (2012b). À chaque itération n'ayant pas apporté d'amélioration globale, chaque particule génère un certain nombre de liens, fixes ou aléatoires bornés voire via une probabilité suivant une courbe en cloche. Ces liens permettent le transfert d'information entre les particules. Ainsi, une particule peut avoir de une informatrice à l'ensemble de l'essaim, comme c'était le cas dans la version de 1998.

Les coefficients de confiance suivent maintenant des règles.

$$0.7 < w < 0.9 \tag{6.74}$$

$$c1 = c2 = \frac{(w+1)^2}{2} \tag{6.75}$$

Dans le cas de plusieurs informatrices Mendes et al. (2004), la formule se généralise de la façon suivante :

$$\sum_{k=1}^{n} c_k <= (w+1)^2 \tag{6.76}$$

Ainsi, la formule de déplacement devient :

$$v_{i,d}(t+1) = wv_{i,d}(t) + \sum_{k=1}^{n} c_k(p_{\alpha(k),d}(t) - x_{i,d}(t)) \tag{6.77}$$

SPSO 2011 Cette version reprend les modifications apportées par SPSO 2007, auxquelles sont ajoutées des modifications sur les phases d'initialisation et de déplacement.

L'espace de définition doit être normalisé en hypercube. On a ainsi l'initialisation de la vitesse suivante :

$$U_{i,d}(0) = U(x_{min} - x_{i,d}(0), x_{max} - x_{i,d}(0)) \tag{6.78}$$

La phase de déplacement est totalement revue. Elle s'effectue en plusieurs temps. Pour chaque particule, le centre de l'hypersphère formée par la particule et ces informatrices est calculé.

$$G = \frac{x_i(t) \sum_{j=1}^{n} p_i(t)}{n+1} \tag{6.79}$$

avec :

— G : centre de l'hypersphère

— n : nombre d'informatrices. Si la meilleure solution rencontrée par cette particule est également la meilleure solution globale, celle-ci n'est comptée qu'une seule fois.

Le rayon de l'hypersphère est donné par la distance entre la particule et G.

Un point x' est alors pris au hasard dans l'hypersphère. Ce choix peut se faire selon une distribution uniforme, mais il semble qu'une distribution non uniforme, de densité décroissante avec la distance au centre soit la plus efficace. La nouvelle position de la particule devient :

$$x_i(t+1) = x' + wv(t) \tag{6.80}$$

Et la nouvelle vitesse :

$$v_i(t+1) = x_i(t+1) - x_i(t) \tag{6.81}$$

Pour plus de clarté, un exemple très simple est proposé en figure 6.13

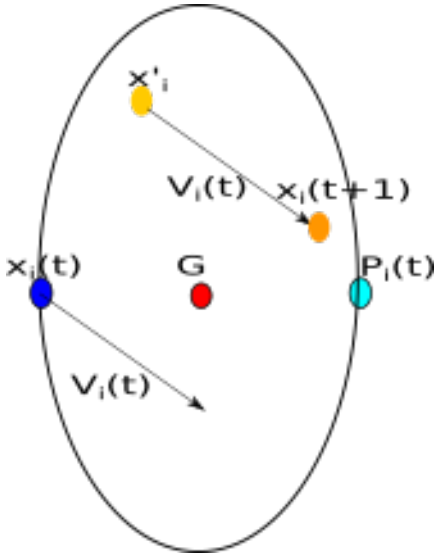


FIGURE 6.13 – Déplacement SPSO 2011 dans un espace 2D avec une seule informatrice

6.4.6 Recuit simulé

Origines Contrairement aux algorithmes précédents, le recuit simulé est une méthode inspirée de phénomènes physiques tels que la solidification des liquides. En ferronnerie, les métaux sont chauffés à une température élevée puis refroidis progressivement en marquant des paliers afin d'éviter l'apparition de défauts. Si une erreur est survenue, les défauts peuvent être éliminés par réchauffement local. C'est ce qu'on appelle le recuit. C'est l'observation de cette technique qui a donné naissance à cet algorithme.

Il a été inventé en 1983 par 3 chercheurs d'IBM Kirkpatrick et al. (1983) et a depuis fait ses preuves dans des domaines tels que l'optimisation de la disposition des composants de circuits électriques.

Présentation L'algorithme 21 prend 3 paramètres :

— T : la température initiale du système

— $temperatureArret$: la température finale

— $maxTconst$: le nombre maximum d'itérations avant changement de solution.

Il consiste, dans un premier temps, à partir d'une solution générée aléatoirement. À chaque itération, la solution sera légèrement perturbée (Calcul voisinage) afin d'explorer l'espace des solutions. L'énergie de la solution est alors calculée, ce qui correspond, pour garder nos termes habituels, à son évaluation. Si la solution trouvée est meilleure que la précédente, celle-ci est remplacée. Sinon, elle peut quand même l'être mais avec une probabilité à calculer. Cette probabilité suit généralement la règle de Metropolis Metropolis et al. (1953) :

$$P = e^{-}\frac{\Delta E}{T} \tag{6.82}$$

avec ΔE : la variation d'énergie, c'est à dire, la différence d'évaluation entre la solution précédente et la solution évaluée.

Algorithme 21 Recuit simulé

```
Initialisation aléatoire d'une solution
tant que testCritèresArrêt() faire
  Calcul voisinage
  Calcul énergie
  Remplacement
  Mise à jour des paramètres
fin tant que
```

Après cette phase de remplacement, les paramètres sont mis à jour grâce à une fonction de décroissance de la température. Si un remplacement a eu lieu, elle permet le refroidissement de la solution. Dans le cas contraire, on considère qu'on a atteint un palier de température pendant *maxStop* itérations. La fonction la plus simple, fréquemment utilisée et fonction par défaut dans notre bibliothèque est $T = \lambda T$ avec $\lambda = 0.99$. Des fonctions bien plus complexes peuvent être imaginées. Ainsi, l'utilisateur est libre d'en définir une et de la passer en paramètre de notre algorithme.

Bibliographie Des études théoriques ont été menées Aarts and Van Laarhoven (1985) et ont permis de montrer que la convergence vers un optimum, au moins local, est garantie sous certaines conditions qui restent cependant discutables. On peut par exemple citer Hajek (1988) et Hajek and Sasaki (1989) qui énoncent que l'algorithme converge vers un optimum global si,

$$\forall t \to \infty, decroissance(T) < \frac{C}{log(t)} \tag{6.83}$$

avec C : constante définie en fonction de la profondeur des puits d'énergie. Ses preuves théoriques, sont presque uniques dans le domaine de l'optimisation combinatoire ce qui en fait un avantage notable pour cette méthode. En pratique, celle-ci est très souple, mais les paramètres sont difficiles à caler pour une utilisation optimale en matière de résultats et temps de calcul.

En pratique, le recuit simulé a beaucoup été utilisé en traitement des images, tel que la reconstitution de données incomplètes, en limitant le temps de calcul de chaque opération. Son efficacité a notamment été prouvée théoriquement par Geman (1985), où une approche bayésienne a été utilisée pour la restauration d'images brouillées. Il est également souvent appliqué à des problèmes d'ordonnancement Van Laarhoven et al. (1992), mais certaines caractéristiques particulières peuvent le mettre en difficulté Fleury (1995).

6.4.7 Recherche à voisinage variable générale

Présentation Afin de pallier au mieux le problème du compromis entre exploration et exploitation, cette métaheuristique couple deux méthodes :

— la recherche locale : permettant d'exploiter au mieux une solution vers un minimum local

— les perturbations : permettant l'exploration

Cette méthode est utilisée dans de très nombreux problèmes, tels que la classification non supervisée avec l'algorithme k-means MacQueen et al. (1967), ou encore lors d'optimisation hybride Knowles (2002), Maringer and Kellerer (2003) et Kelner et al. (2008).

L'algorithme 22 propose de partir d'une solution initiale générée aléatoirement et de lui appliquer une perturbation à chaque itération. Cette solution perturbée servira d'entrée à un algorithme de recherche local, ici la descente à voisinage variable. La solution ainsi obtenue sera comparée à la meilleure solution connue et la remplacera si elle est meilleure.

Les méthodes présentées ici ne sont que des exemples de fonction servant à exploiter ou explorer. Elles peuvent être remplacées par n'importe quelle autre. Ainsi, par la suite, nous expliquerons quelques une des méthodes que nous avons implémentées pour ce cas.

Algorithme 22 Recherche à voisinage variable globale

```
Initialisation aléatoire d'une solution
tant que testCritèresArrêt() faire
  Perturbation
  Phase 1 : Descente à voisinage variable :
  pour Chaque transformation utilisée faire
    1. Phase 2 : Recherche locale sur chaque fonction de transformation :
    1.1. Calcul voisinage
    1.2. Évaluation
    1.3. Remplacement
  fin pour
  2. Remplacement
fin tant que
```

Les algorithmes d'exploitation

La recherche locale classique Cet algorithme consiste simplement à effectuer des variations élémentaires de la solution courante afin d'explorer son voisinage à la recherche d'une meilleure solution. Ces transformations élémentaires s'effectuent en général dimension par dimension à la recherche de la meilleure valeur possible pour chacune. En théorie, cette méthode permet de trouver un optimum local à coup sûr. En pratique, le nombre de combinaisons possibles ne permet pas de complètement explorer un voisinage, il est donc impossible de garantir l'optimalité de la solution en temps acceptable.

Descente à voisinage variable (DVV) Plutôt que de ne considérer qu'une fonction de calcul de voisinage, la descente à voisinage variable permet d'en utiliser autant que l'on veut. Ainsi, pour chaque transformation à utiliser, l'algorithme calcule un grand nombre de voisinages afin de sélectionner le meilleur. Cet algorithme est représenté par la phase 1 dans l'algorithme 22.

Fonctions de calcul de voisinage Les fonctions de calcul de voisinage dont nous avons parlées dans les paragraphes précédents peuvent être n'importe quelle fonction partant d'une solution pour en générer une autre. Bien sûr, la notion de voisinage implique que cette transformation ne soit pas trop importante. On peut par exemple citer les mutations, présentées précédemment dans les algorithmes génétiques, à condition d'utiliser un coefficient de variation relativement faible.

Les algorithmes d'exploration

Algorithme multi-start Afin d'augmenter l'exploration, l'algorithme multi-start a été beaucoup utilisé. Celui-ci consiste simplement à partir d'un ensemble de solutions générées aléatoirement et à effectuer une recherche locale sur chacune. Cette méthode est souvent très efficace si un faible nombre d'optimums locaux sont présents.

Perturbations Une solution souvent plus adéquate est de revenir sur notre descente à voisinage variable et d'y incorporer une phase de perturbation (voir algorithme 22).

Cette méthode consiste à partir de la meilleure solution connue et d'y appliquer une perturbation avant de s'en servir d'entrée pour un algorithme de recherche locale tel que la DVV. Comme les fonctions de calcul de voisinage, un très grand nombre de fonctions de perturbation peuvent être imaginées. Pour reprendre l'exemple précédent, on peut également se servir d'une des fonctions de mutation de l'algorithme génétique en prenant une amplitude de variation plus élevée.

Ainsi, en couplant perturbation et recherche locale, on obtient une recherche à voisinage variable de base. En remplaçant la recherche locale par la DVV on obtient une recherche à voisinage variable générale. Il est également possible d'enlever la phase de recherche locale et d'adapter les fonctions de perturbations. On obtient ainsi une recherche à voisinage variable réduite.

6.4.8 Colonie d'abeilles artificielles

Présentation Cet algorithme est directement inspiré du fourragement des abeilles mellifères. Il s'appuie principalement sur les sources de nourriture identifiées, $N_{exploitante}$, des abeilles éclaireuses, $N_{eclaireuss}$, des spectatrices, $N_{spectatrice}$ et des exploitantes.

Nous n'en avons implémenté qu'une variante. Ainsi, nous ne développerons que l'algorithme principal défini par Karaboga (2005) et Karaboga and Basturk (2007).

Au début de l'algorithme (23), $N_{exploitante}$ sources de nourriture sont initialisées aléatoirement de façon uniforme. Chaque source correspond à un point de l'espace des solutions. À chaque itération, les abeilles exploitantes sortiront pour exploiter les sources déjà connues. Les spectatrices se répartissent ensuite en fonction des sources intéressantes. Enfin, une phase de remplacement des sources de nourriture peu fructueuses à lieu afin de maintenir une certaine exploration.

Algorithme 23 Colonie d'abeilles artificielles

```
Initialisation des sources de nourriture
tant que testCritèresArrêt() faire
  Sortie des abeilles exploitantes
  Sortie des abeilles spectatrices
  Remplacement des sources de nourriture
fin tant que
```

Sortie des abeilles exploitantes Chaque abeille exploitante va explorer l'espace autour de sa source de nourriture en opérant une variation sur une dimension, k choisie aléatoirement, selon la formule suivante :

$$v_{i,k} = s_{i,k} + \mathcal{U}([-1,1]) \times (s_{i,k} - s_{n,k}) \tag{6.84}$$

avec :

— $v_{i,k}$: dimension k de la nouvelle solution v mutée de la source i

— $s_{i,k}$: dimension k de la source i

— $s_{n,k}$: dimension k d'une autre source d'indice n choisi aléatoirement

Si la solution trouvée est moins bonne que la source d'origine, une variable e correspondant au nombre d'exploitations est augmentée. Dans le cas contraire, cette variable est remise à 0 et cette solution remplace la source.

Sortie des abeilles spectatrices Le rôle des spectatrices est similaire à celui des exploitantes à la différence qu'au lieu de chacune exploiter une source définie, elles vont choisir laquelle exploiter selon la loi de probabilité suivante :

$$p_i = \frac{q(f(s_i))}{\sum_{q \in S} q(f(s))} \tag{6.85}$$

$f(s)$ définissant la fonction d'évaluation d'une solution. $q(f(s))$ est défini comme :

$$Si(f(s) >= 0) \ : q(f(s)) = \frac{1}{1 + f(s)} \tag{6.86}$$

$$Sinon \ : q(f(s)) = 1 + |f(s)| \tag{6.87}$$

Remplacement des sources de nourriture Pour chaque source, si son nombre d'exploitations e est supérieur au paramètre e_{max}, celle-ci est potentiellement remplaçable. À chaque itération, seul $N_{eclaireuse}$ sources peuvent être remplacées. Ainsi, les sources qui ont été le plus expoitées sont remplacées en priorité. Les autres sont gardées jusqu'à l'itération suivante.

Paramétrage par défaut Typiquement, il y a autant de sources que d'abeilles spectatrices. De plus, une étude Karaboga and Akay (2009) semble montrer que la valeur optimale de e_{max} était $\frac{D \times N}{2}$ avec D le nombre de dimensions du problème et $N = N_{exploitante} + N_{spectatrice}$. Nous rappelons qu'une abeille exploitante exploite une source définie, ainsi $N_{exploitante} = N_{source}$

6.4.9 Évolution différentielle

Comme pour la colonie d'abeilles artificielle, nous n'avons pour l'instant implémenté que la version de base Storn and Price (1997) de cet algorithme pour ce qui est de l'optimisation globale. Il est cependant possible d'utiliser une de ses variantes en tant que stratégie d'évolution lors de l'utilisation d'un algorithme génétique multiobjectif (section 6.4.13 et benchmark 6.5.3).

Presentation L'algorithme à évolution différentielle est un algorithme évolutionnaire. Comme l'algorithme génétique, il utilise de phases de croisement, mutation et sélection (voir algorithme 24).

Algorithme 24 Differential Evolution

```
Initialisation de la population initiale
tant que testCritèresArrêt() faire
  pour Chaque solution dans la population faire
    Mutation basée sur la différence de vecteurs
    Croisement
  fin pour
  Selection (remplacement)
fin tant que
```

Nous ne développerons ici que la version originale de l'algorithme. Le lecteur intéressé pourra se référer à Qin et al. (2009) pour un exemple d'algorithme à évolution différentielle avec stratégie d'adaptation, ou encore à Das and Suganthan (2011) pour un état de l'art complet sur le sujet.

Mutation À la différence des algorithmes génétiques, la phase de mutation nécessite le tirage de trois autres solutions distinctes, $s1$, $s2$, $s3$, dans la population. Ces solutions seront utilisées pour créer une solution mutante selon la formule suivante :

$$\forall i \in D, x_{m,i} = x_{s1,i} + F.(x_{s2,i} - x_{s3,i}) \tag{6.88}$$

avec :

— D : nombre de paramètres

— $x_{m,i}$: valeur du ième paramètre de la solution mutante

— F : paramètre de l'algorithme, $F \in [0, 2]$

Croisement Au début de la phase de croisement, un paramètre aléatoire, N, est tiré. Le croisement suit l'algorithme 25.

Algorithme 25 Differential Evolution : Crossover

pour Chaque paramètre i **faire**
  si $\mathcal{U}(0, 1) < CR$ **or** $i == N$ **alors**
    $x_{G+1,i} = x_{m,i}$
  sinon
    $x_{G+1,i} = x_{G,i}$
  fin si
fin pour

Ainsi, la valeur du paramètre N sera toujours remplacée par celle du vecteur mutant. Selon un paramètre, $CR \in [0, 1]$, de l'algorithme, un certain nombre d'autres paramètres seront remplacés par ceux du vecteur mutant.

Sélection La seule phase de sélection de cet algorithme est équivalente à une sélection environnementale pour un algorithme génétique.

Dans la version de base de l'algorithme à évolution différentielle, cette phase consiste simplement en le remplacement des parents dont l'évaluation est moins bonne que celle de leurs enfants.

Paramètres Les valeurs des paramètres F et CR sont par défaut initialisées à 0.5 et 0.1 respectivement, ce qui semblent être des valeurs intéressantes dans la plupart des cas d'optimisation globale Storn and Price (1997). Les auteurs proposent cependant d'autres paramètres en fonction des situations. Par exemple, une valeur de CR de 0.9 voire 1 permet une convergence rapide. Cela peut être utile pour vérifier si une solution rapide est suffisante. Le couple $(F = 1, CR = 0.5)$ semble également souvent être utilisé en optimisation multiobjectif Li and Zhang (2009).

Les auteurs précisent également qu'une valeur de taille de population adéquate semble être entre 5 et 10 fois plus qu'il y a de paramètres à optimiser.

6.4.10 Grey Wolf Optimizer

Présentation Cet algorithme Mirjalili et al. (2014) d'intelligence en essaim est directement inspiré par le comportement des loups gris. Il imite la hiérarchie de dominance et le mécanisme de chasse des loups gris dans la nature. Les trois principales étapes de la chasse, à savoir la recherche de proies, leur encerclement et l'attaque des proies, sont mises en œuvre. Elles peuvent être résumées par l'algorithme 26.

Initialisation et mises à jour des paramètres Cet algorithme comprend un certain nombre de paramètres mais les auteurs en définissent des valeurs afin d'observer le comportement souhaité. Le

Algorithme 26 Grey wolf optimizer

```
Initialisation de la population initiale et des paramètres
Initialisation des loups α, β, δ
tant que testCritèresArrêt() faire
    Mise à jour des positions des loups
    Mise à jour des paramètres
    Mise à jour des loups α, β, δ
fin tant que
```

premier paramètre a a pour valeur 2 au début de l'algorithme et sa valeur décroît jusqu'à 0 de manière linéaire en fonction soit du nombre d'itérations, soit du temps maximum alloué. r_1 et r_2 sont des vecteurs de valeur aléatoire entre 0 et 1 : $r_1, r_2 \in [0,1]^{nWolfs^2}$

Initialisation et mise à jour des loups α, β, δ La solution avec la meilleure évaluation est considérée comme étant le loup α. De même, la deuxième et troisième meilleure solution sont les loups β et γ. Tous les autres sont considérés comme des loups ω.

Mise à jour des positions des loups Les équations suivantes sont utilisées :

$$D = |CX_p(t) - X(t)| \tag{6.89}$$

$$X(t+1) = X_p(t) - AD \tag{6.90}$$

avec t l'itération actuelle, X_p la position de la proie, X la position d'un loup et A et C sont des vecteurs de coefficients qui suivent les équations suivantes :

$$A = 2ar_1 - a \tag{6.91}$$

$$C = 2r_2 \tag{6.92}$$

La position de la proie n'est cependant pas connue. L'algorithme suppose donc que les loups α, β et δ ont une meilleure connaissance de la position de la proie et les positions de chacun seront mises à jour en fonction des leurs :

$$D_\alpha = |C_1 X_\alpha - X|, D_\beta = |C_2 X\beta - X|, D_\delta = |C_3 X_\delta - X| \tag{6.93}$$

$$X_1 = X_\alpha A_1 D_\alpha, X_2 = X_\beta - A_2 D_\beta, X_3 = X_\delta - A_3 D_\delta \tag{6.94}$$

$$X(t+1) = \frac{X_1 + X_2 + X_3}{3} \tag{6.95}$$

La valeur décroissante de a, et les valeurs aléatoires de r_1 et r_2 permettent de mimer les processus d'encerclement, de chasse, d'attaque (exploitation) et de recherche de proies (exploration) durant l'évolution de l'algorithme.

6.4.11 Improved Grey Wolf Optimizer

Présentation Cet algorithme Wang and Li (2019) est une variante du *Grey wolf optimizer* auquel des mécanismes d'évolution différentielle (annexe 6.4.9) et d'élimination sont ajoutés. L'algorithme 27 présente son déroulement.

Algorithme 27 Improved Grey wolf optimizer

```
Initialisation de la population initiale et des paramètres
Initialisation des loups α, β, δ
tant que testCritèresArrêt() faire
    Mise à jour des positions des loups
    Opérations d'évolution différentielle
    Remplacement des plus mauvais
    Mise à jour des paramètres
    Mise à jour des loups α, β, δ
fin tant que
```

Les phases d'initialisation et de mise à jour des paramètres, des positions, et des loups α, β et γ sont identiques à celles présentées en annexe 6.4.10.

Opérations d'évolution différentielle Cette phase présente trois opérations : mutation, croisement et sélection.

La mutation consiste, comme pour les algorithmes génétiques (annexe 6.4.2) et l'évolution différentielle (annexe 6.4.9), en la modification d'un ou plusieurs paramètres en fonction des valeurs des paramètres d'autres individus de la population. Pour cette étape, l'algorithme suit l'équation 6.88. Les loups α, β et δ sont utilisés comme parents. La valeur du paramètre F de l'algorithme d'évolution différentielle est également adaptée pour suivre une décroissance linéaire entre f_{min} et f_{max} au cours du temps ou à l'itération par rapport au nombre maximum d'itérations.

Le croisement obéit également à la méthode habituelle de croisement d'évolution différentielle : l'algorithme 25.

La sélection consiste simplement à choisir l'individu ayant la meilleure évaluation entre l'individu nouvellement généré par croisement et mutation et l'individu ayant permis cette génération.

Remplacement des plus mauvais Pour garantir une exploration efficace, à chaque itération, les R plus mauvais loups sont éliminés et remplacés par R nouveaux générés de manière totalement aléatoire. Les auteurs préconisent l'utilisation d'une valeur aléatoire entière de R telle que $R \in [n/\epsilon, n/0.75 \times \epsilon]$ avec n le nombre d'individus dans la population et ϵ le facteur d'échelle de mise à jour des loups.

Valeurs préconisées pour les paramètres Après avoir comparé leur algorithme sur un grand nombre de problèmes d'optimisation globale, les auteurs préconisent l'utilisation des valeurs suivantes pour les

paramètres $N = 30$, $f_{min} = 0.25$, $f_{max} = 1.5$, $CR = 0.7$, $\epsilon = 5$

6.4.12 Whale Optimization Algorithm

Présentation Cet algorithme Mirjalili and Lewis (2016) d'intelligence en essaim s'inspire de la stratégie de chasse aux filets à bulles des baleines. Cette stratégie permet d'encercler et de bloquer les proies dans des spirales de bulles qu'elles ne peuvent pas traverser. Comme pour l'algorithme des loups gris (annexe 6.4.10), cette stratégie consiste en une phase de recherche de proies (exploration) et un encerclement puis une attaque de la proie (exploitation).

Ainsi, il peut même être considéré comme une variante de celui-ci, il suit un algorithme très similaire (l'algorithme 28) mais avec des paramètres et méthodes de mise à jours des positions différents.

Algorithme 28 Whale optimizer

```
Initialisation de la population initiale et des paramètres
Initialisation de la meilleure solution
tant que testCritèresArrêt() faire
    Mise à jour des positions des agents de recherche
    Mise à jour des paramètres
    Mise de la meilleure solution
fin tant que
```

Initialisation et mise à jour des paramètres Comme pour l'algorithme des loups gris, cet algorithme utilise un paramètre a suivant une décroissance linéaire de 2 à 0 en fonction du temps ou de l'itération par rapport au nombre d'itérations maximum. Il possède également un vecteur de paramètres $r \in [0, 1]^{nbSolutions}$ permettant l'exploration du voisinage de la meilleure solution connue actuellement.

Mise à jour de la meilleure solution Plutôt que de considérer les trois meilleures solutions comme dans l'algorithme des loups gris, ici seule la meilleure solution est considérée. Cela implique que dans les autres étapes de l'algorithme, la position de la proie ne sera pas estimée en fonction des meilleures solutions. Les solutions s'orienteront en fonction de la position de la meilleure connue.

Mise à jour des positions des agents de recherche Trois comportements sont possibles en fonction des cas détaillés en algorithme 29.

Le premier est un comportement d'encerclement très proche de celui utilisé dans l'algorithme des loups gris :

$$D = |CX^*(t) - X(t)| \tag{6.96}$$

$$X(t+1) = X^*(t) - AD \tag{6.97}$$

Avec t l'itération actuelle, $X^*(t)$ la meilleure position connue, $X(t)$ la position actuelle de l'agent (solution)

considéré et :

$$A = 2ar - a \tag{6.98}$$

$$C = 2r \tag{6.99}$$

Un deuxième comportement consiste en la mise à jour de la position en suivant une spirale :

$$X(t+1) = D' e^{bl} cos(2\pi l) + X^*(t) \tag{6.100}$$

avec b une constante définissant la forme de la spirale logarithmique, l une variable aléatoire telle que $l = \mathcal{U}(-1, 1)$ et :

$$D' = |X^*(t) - X(t)| \tag{6.101}$$

Le dernier comportement est la recherche des proies suivant cette fois les équations suivantes :

$$D = |CX_{rand} - X| \tag{6.102}$$

$$X(t+1) = X_{rand} - AD \tag{6.103}$$

Avec X_{rand} un nouvel individu généré totalement aléatoirement dans l'espace des paramètres.

Algorithme 29 Whale optimizer : mise à jour des positions de chaque agent de recherche

```
pour Chaque agent de recherche faire
  si U(0,1) < 0.5 alors
    si |A| < 1 alors
      mise à jour de la position par encerclement (eq 6.97)
    sinon
      mise à jour de la position par recherche de proie (eq 6.103
    fin si
  sinon
    mise à jour de la position par spirale (eq 6.100
  fin si
fin pour
```

6.4.13 NSGA-II : Non-dominated Sorting Geneteic Algorithm 2

Cet algorithme est une version multiobjectif de l'algorithme génétique. Dans notre bibliothèque, il est utilisable directement à partir de la classe *AlgorithmeGenetique*, en utilisant les fonctions adéquates que nous détaillerons ici.

Cette méthode est basée sur la notion de dominance et donc un classement de Pareto. Ainsi, l'algorithme de base est le même que l'algorithme génétique 15. La principale différence vient de l'utilisation

d'une autre méthode de classement des individus.

Méthode de classement de Pareto La méthode de classement est donnée par l'algorithme 30. Ce classement se déroule en deux phases. Pendant la phase d'initialisation, chaque individu i de la population P se voit associer deux variables :

— α_i : un compteur donnant le nombre d'individus qui dominent i

— S_i : l'ensemble des individus dominés par i

L'ensemble $\mathcal{F}_x$ représente l'ensemble des individus non dominés de rang x. En fin de boucle, une valeur $alpha_i$ nulle signifie que l'individu i n'est pas dominé. Il appartient ainsi à l'ensemble $\mathcal{F}_0$.

Une fois ce premier ensemble construit, on peut facilement déterminer les ensembles d'individus non dominés de rangs supérieurs. Pour chaque individu, la variable α, correspondant à chaque solution qu'il domine, sera décrémentée. Quand un α_i tombe à 0, alors l'individu i appartient à l'ensemble non dominé de rang supérieur $r + 1$.

Reproduction La phase de reproduction se fait via une sélection par tournoi appelée tournoi de surpeuplement. C'est une méthode de sélection par tournoi comme présentée plus tôt (section 6.4.2) à la différence que les individus sont comparés via un opérateur $\prec_n$ appelé opérateur de surpeuplement, défini de la façon suivante pour deux individus i et j :

$$i \prec_n j \iff r_i < r_j \; ou (r_i == r_j \; et \; d_i > d_j) \tag{6.104}$$

d_i est appelé distance de surpeuplement et est défini de la façon suivante :

$$d_i = \sum_{m=0}^{c} \frac{f_m^+(i) - f_m^-(i)}{f_m^{max} - f_m^{min}} \tag{6.105}$$

avec :

— f_m^{max} : valeur maximale de l'objectif m

— f_m^{min} : valeur minimale de l'objectif m

— $f_m^+(i)$: plus proche valeur supérieure de $f_m(i)$ dans $\mathcal{F}_r$.

— $f_m^-(i)$: plus proche valeur inférieure de $f_m(i)$ dans $\mathcal{F}_r$.

— m : nombre d'objectifs

Pour les individus extrêmes on fixe les valeurs de f_m^+ et f_m^- à l'infini pour un et 0 pour tous les autres.

Ce calcul est illustré en figure 6.14.

Algorithme 30 NSGA-II : classement

1 : Initialisation
pour chaque individu $i \in P$ **faire**
  $S_i \leftarrow \emptyset$
  $\alpha_i \leftarrow 0$
  pour chaque individu $j \in P$ **faire**
    si $i \prec j$ **alors**
      $S_i \bigcup j$
    sinon si $j \prec i$ **alors**
      $\alpha_i + +$
    fin si
  fin pour
  si $\alpha_i == 0$ **alors**
    $\mathcal{F}_0 \leftarrow \mathcal{F}_0 \bigcup i$
  fin si
fin pour
2 : affectation des rangs
$r \leftarrow 0$
tant que $\mathcal{F}_r \neq \emptyset$ **faire**
  $\mathcal{F}_{r+1} \leftarrow \emptyset$
  pour chaque individu $i \in \mathcal{F}_r$ **faire**
    pour chaque individu $j \in S_i$ **faire**
      $\alpha_j - -$
      si $\alpha_j == 0$ **alors**
        $\mathcal{F}_{r+1} \bigcup j$
      fin si
    fin pour
  fin pour
  $r + +$
fin tant que

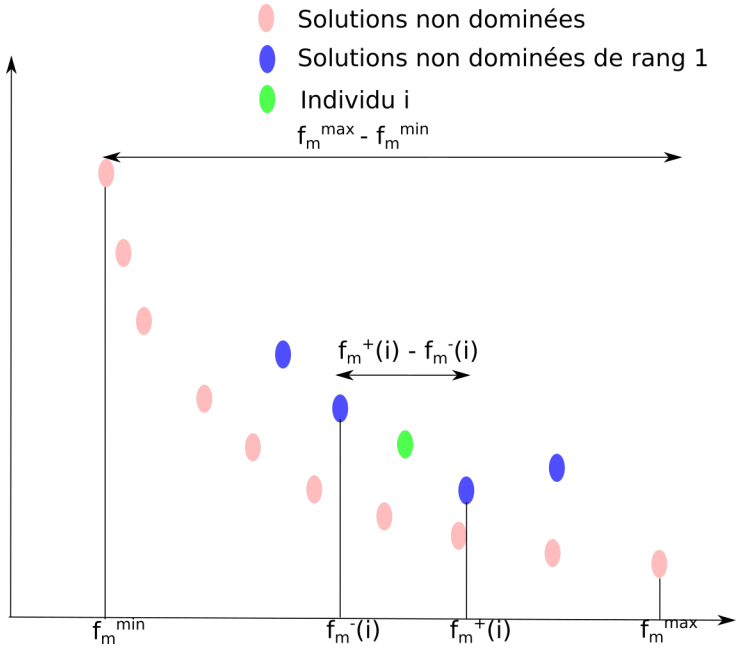


FIGURE 6.14 – Calcul de la distance de surpeuplement

L'utilisation de cette méthode constitue une méthode de nichage sans paramètre. Elle est un des principaux avantage de NSGA-II par rapport à son prédécesseur NSGA.

À cette phase viennent s'ajouter les phases de croisements de mutation classique des algorithmes génétiques. N'importe quelle variante peut faire l'affaire. On utilise souvent des méthodes de type évolution différentielle et mutation polynomiale Zhang and Li (2007).

Remplacement (sélection environnementale) Le remplacement se fait de façon élitiste en fonction des rangs de domination. Une taille de population μ est fixée au début de l'algorithme. À chaque itération, tant que l'on ne dépasse pas μ tous les individus appartenant à $\mathcal{F}_i$, i croissant, seront ajoutés à la population P. Si l'utilisation complète d'un niveau $\mathcal{F}_i$ provoque le dépassement de μ, les individus sont rangés selon l'opérateur de surpeuplement $\prec_n$ et seulement les meilleurs sont sélectionnés pour la génération suivante.

6.4.14 Stochastic Ranking

La méthode de *stochastic ranking* Runarsson and Yao (2000) est une variante d'algorithme génétique adaptée à l'optimisation sous contrainte. Elle utilise une méthode de pénalisation statique (section 1.5) potentiellement modifiable par l'utilisateur. Cela permet de créer un équilibre entre les fonctions d'objectif et de pénalisation.

Comme pour NSGA-II, l'algorithme est en tout point similaire à l'algorithme génétique à l'utilisation de la fonction de classement stochastique près. La méthode de classement proposée suit l'algorithme 31.

N représente le nombre maximum de balayages de la population. Ainsi, pour chaque individu I_j sauf le dernier de la population contenant λ individus, si I_j et I_{j+1} respectent les contraintes ($\Phi == 0$), ou si un tirage aléatoire uniforme entre 0 et 1, $\mathcal{U}(0,1)$ est inférieur à $\mathcal{P}_f$, un paramètre de l'algorithme : les fonctions objectifs sont comparées. Dans le cas d'une minimisation de la fonction objectif, on inverse les deux individus si $f(I_j) > f(I_j + 1)$.

Sinon, si les solutions ne respectent pas les contraintes et que le tirage aléatoire est défavorable, l'écart au respect des contraintes Φ est utilisé. La solution avec le plus petit Φ est placé avant l'autre.

Algorithme 31 Stochastic Ranking

```
pour i = 0 a N faire
  pour j = 0 a λ − 1 faire
    si Φ(I_j) = Φ(I_{j+1}) == 0) ou U(0,1) < P_f alors
      si f(I_j) > f(I_{j+1}) alors
        inverser I_j et I_{j+1}
      fin si
    sinon si Φ(I_j) > Φ(I_{j+1}) alors
      inverser I_j et I_{j+1}
    fin si
  fin pour
  break s'il n'y a eu aucune inversion
fin pour
```

Pour être utiles, les méthodes de sélection et remplacement doivent être dépendantes des rangs des individus dans la population plutôt que de leur évaluation globale. On peut par exemple penser à des sélections élitistes et remplacement générationnel.

La valeur du paramètre $\mathcal{P}_f$ permet de plus ou moins orienter les recherches vers la satisfaction des contraintes. Une étude approfondie des auteurs Runarsson and Yao (2000) semble montrer que la valeur $\mathcal{P}_f = 0.45$ est optimale pour un grand nombre de problèmes. Le choix est bien sûr donné à l'utilisateur de respecter ou pas cette valeur par défaut.

6.5 Benchmarks de validation des implémentations

Dans cette section nous allons présenter des benchmarks réalisés pour les méthodes implémentées. Nous ne détaillerons pas ici de choix de paramètres, car le but était de valider les implémentations. Pour chaque méthode, nous avons donc utilisé les mêmes paramètres que dans la publication de référence développant la méthode.

6.5.1 Benchmarks pour l'optimisation globale

Très souvent, les tests de performances s'évaluent sur le nombre d'itérations d'un algorithme pour se rapprocher de l'optimum. Or nous avons plusieurs fonctions à comparer avec des caractéristiques très différentes. En effet, une itération d'un algorithme génétique n'est pas équivalente à une itération de recherche harmonique par exemple. Ainsi, nous avons décidé d'effectuer nos comparaisons en limitant les algorithmes sur une certaine durée.

Pour cette première série de tests, nous avons limité le temps d'exécution de chaque méthode à une seconde sans parallélisation. Toute machine n'étant pas équivalente, pour permettre des comparaisons ultérieures avec d'autres machines, nous avons effectué le test de définition d'une unité standard de temps (stu) tirée de Shcherbina et al. (2002) et obtenons $1stu \approx 20s$.

Les tests ont été effectués sur un ensemble de fonctions tirées de Yang (2010) sur plusieurs dimensions (tableau **??**). À noter que dans cette publication, une erreur s'est glissée sur la première fonction de Perm Rahnamayan et al. (2007). Elle vaut en réalité :

$$f(x) = \sum_{j=1}^{n}(\sum_{i=1}^{n}(i^j + \beta)[(\frac{x_i}{i})^j - 1])^2 \tag{6.106}$$

Vous trouverez l'ensemble des formules correspondantes en annexe 6.5.4.

Pour chaque algorithme, nous réalisons 30 optimisations indépendantes sur chaque fonction avec pour critère de réussite une différence d'évaluation de 0.0001 par rapport à l'optimum. Pendant chaque test, nous calculons et enregistrons parmi les solutions finales :

- la meilleure solution trouvée
- la moins bonne
- la solution médiane
- l'évaluation moyenne
- l'écart type
- le nombre de réussites
- le temps moyen pour atteindre une différence à l'optimum de 0.01
- le temps moyen pour atteindre une différence à l'optimum de 0.0001
- l'évaluation moyenne de la meilleure solution de chaque itération par pas de temps.

Les temps moyens sont calculés uniquement sur les solutions qui ont effectivement atteint l'évaluation requise. De plus, pour ne pas risquer d'influencer les performances d'une quelconque façon, tous ces paramètres sont stockés en ram puis enregistrés à la fin. Cela garantit que les entrées/sorties n'influencent en aucune manière les performances.

Fonction	Nombre de dimensions	Optimum	Intervalle de définition
Ackley	2 / 5 / 10 / 20	0.000000	[-32.768,32.768]
DeJong1	2 / 5 / 10 / 20	0.000000	[-100,100]
DeJong2	2/ 5 / 10 / 20	0.000000	[-100,100]
DeJong3	2/ 5 / 10 / 20	0.000000	[-1 , 1]
Easom	2 / 10 / 20	-1.000000	[-100,100]
Michaelwicz	2 5 10 20	-1.801303 -4.687658 -9.660152 -19.637014	[0, π]
Perm1	2 / 5	0.000000	[-nbDimension , nbDimension]
Perm2	2/ 5 / 10 / 20	0.000000	[-1 , 1]
Rastrigin	2/ 5 / 10 / 20	0.000000	[-5.12,5.12]
Rosenbrock	2/ 5 / 10 / 20	0.000000	[-5,5]
Schwefel	2/ 5 / 10 / 20	0.000000	[-500,500]
Six_Hump	2	-1.031600	$x \in$ [-3,3] $y \in$ [-2,2]
Shubert	2	-186.730900	[-10,10]
Zakharov	2/ 5 / 10 / 20	0.000000	[-5,10]

TABLE 6.2 – Définition des fonctions de tests de l'optimisation globale

Essaims particulaires : illustration du "no free lunch" théorème Les benchmarks réalisés nous montrent un phénomène intéressant. La figure 6.15 montre que l'essaim particulaire de 1998 semble être le plus efficace. En dimension 2, il a un taux de réussite de 100% sur chacun des problèmes et converge vers l'optimum plus vite que les autres variantes sur presque tous les problèmes.

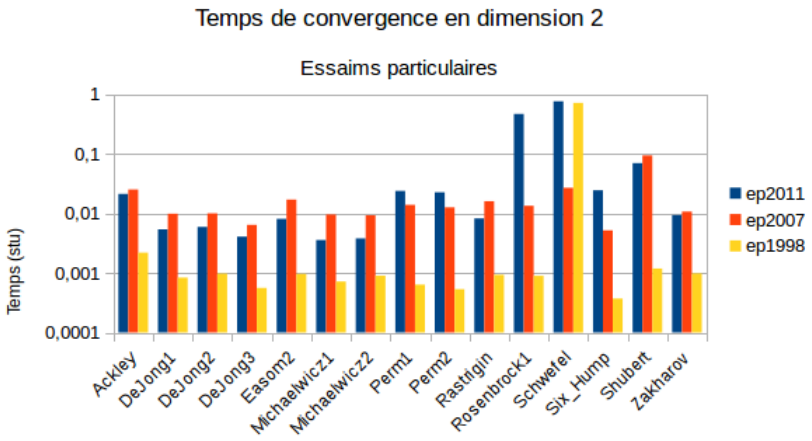


FIGURE 6.15 – Comparaison essaim particulaire en dimension 2

Il est en revanche très mauvais sur ces mêmes problèmes en dimension 5 où sa convergence rapide l'entraîne vers des solutions non optimales. En figure 6.16 nous avons représenté les résultats des tests

en dimension 5 en enlevant ep1998. L'absence de barre signifie que ce problème n'a pas été résolu pour l'algorithme en question, ou trop peu pour que les résultats soient significatifs. Sur ce graphique, la version de 2007 semble être la plus efficace. On remarque cependant que certains problèmes ne sont résolus que par une des deux versions.

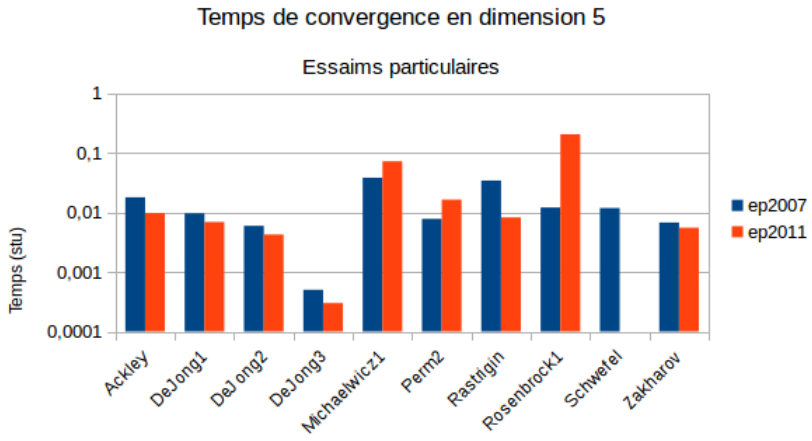


FIGURE 6.16 – Comparaison essaim particulaire en dimension 5

Ainsi, en figure 6.17, nous avons représenté l'évolution de la moyenne des évaluations des différentes variantes testées sur le problème de Zakharov en dimension 10. On remarque qu'ici ep1998 converge rapidement sans atteindre l'optimum. Bien qu'il semble plus lent au début, ep2007 finit par dépasser ep2011 et converger à chaque fois vers une solution satisfaisant le critère d'arrêt d'évaluation.

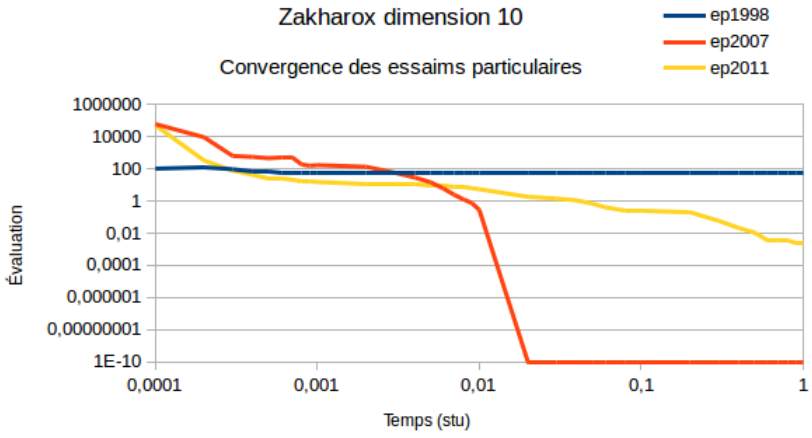


FIGURE 6.17 – Comparaison essaim particulaire sur le problème de Zakharov en dimension 10

Finalement, le tableau 6.3 montre les résultats de ces algorithmes sur des problèmes à 20 dimensions.

	ep1998		ep2007		ep2011	
Problème	Moyenne	Écart-type	Moyenne	Écart-type	Moyenne	Écart-type
Ackley	1,98E+01	2,50E-01	2,43E+00	5,70E-01	2,00E-02	4,00E-02
DeJong1	2,84E+04	3,38E+03	1,96E+01	2,83E+01	0,00E+00	1,00E-02
DeJong2	7,57E+02	1,07E+02	4,20E-01	4,40E-01	1,00E-02	2,00E-02
DeJong3	6,00E-02	4,00E-02	0	0	0	0
Easom2	0	0	-3,60E-01	3,80E-01	-8,60E-01	3,40E-01
Michaelwicz1	-7,11E+00	5,70E-01	-1,67E+01	9,90E-01	-1,11E+01	2,48E+00
Perm2	1,14E+03	4,09E+02	8,03E+00	1,27E+01	7,19E+01	5,04E+00
Rastrigin	2,13E+02	1,27E+01	1,96E+01	9,25E+00	1,38E+00	3,60E+00
Rosenbrock1	4,87E+04	1,31E+04	2,42E+01	7,83E+00	1,86E+01	2,70E-01
Schwefel	5,81E+03	3,45E+02	0	0	7,71E+03	2,40E+02
Zakharov	9,04E+07	4,87E+08	6,81E+00	5,96E+00	1,68E+00	1,97E+00

TABLE 6.3 – Essaims particulaires : résultats sur des problèmes en dimension 20

	RH		IAHS		GHS	
	Moyenne	Écart-type	Moyenne	Écart-type	Moyenne	Écart-type
Ackley	5,32E+00	3,22E-01	5,18E+00	4,45E-01	3,43E-04	2,84E-04
DeJong1	4,74E-01	9,81E-02	4,62E-01	1,06E-01	0,00E+00	0,00E+00
DeJong2	3,94E+00	7,93E-01	3,59E+00	7,64E-01	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,99E-01	7,66E-02	-1,90E-01	7,51E-02	-1,00E+00	0,00E+00
Michaelwicz1	-1,84E+01	2,48E-01	-1,84E+01	2,54E-01	-1,95E+01	3,71E-02
Perm2	2,73E+00	1,91E+00	3,18E+00	2,02E+00	4,90E-01	6,15E-01
Rastrigin	1,47E+01	2,31E+00	1,56E+01	2,08E+00	0,00E+00	0,00E+00
Rosenbrock1	1,11E+02	3,02E+01	1,14E+02	3,47E+01	1,49E+01	1,73E+01
Schwefel	5,30E+01	1,36E+01	6,05E+01	2,34E+01	0,00E+00	0,00E+00
Zakharov	9,12E+00	1,63E+00	8,82E+00	1,74E+00	0,00E+00	0,00E+00

TABLE 6.4 – Recherches harmoniques : résultats sur des problèmes en dimension 20

Ce phénomène illustre bien le "no free lunch theorem" Wolpert and Macready (1997). En effet, on remarque bien qu'il n'y a pas d'algorithme en surpassant un autre, ils ont tous leurs points forts.

Comparaison des recherches harmoniques La version d'IAHS proposée dans la littérature (voir annexe 6.4.3) semble engendrer une convergence très rapide des différentes variables, entraînant le retour vers une forme de recherche harmonique plus classique. En effet, on remarque dans le tableau 6.4 que les différences entre RH classique et IAHS sont bien souvent minimes en dimension 20. Ce phénomène est également constatable pour les autres dimensions. En revanche, on remarque que GHS présente d'excellents résultats, très souvent atteignant le critère d'arrêt d'évaluation, exception faite pour le problème de Rosenbrock.

Nous avons représenté, en figure 6.18, l'évolution des différentes versions de la recherche harmonique que nous avons implémentées, sur le problème de Rastrigin en dimension 20. Comme énoncé précédemment, RH et IAHS sont extrêmement proches, voire confondus. Ainsi, notre proposition d'IAHS

semble plus efficace et celle proposée en 2015 (annexe 6.4.3) commence à présenter de bons résultats, similaires à ceux de IHS. GHS semble en revanche bien meilleur que les autres sur ce type de problème.

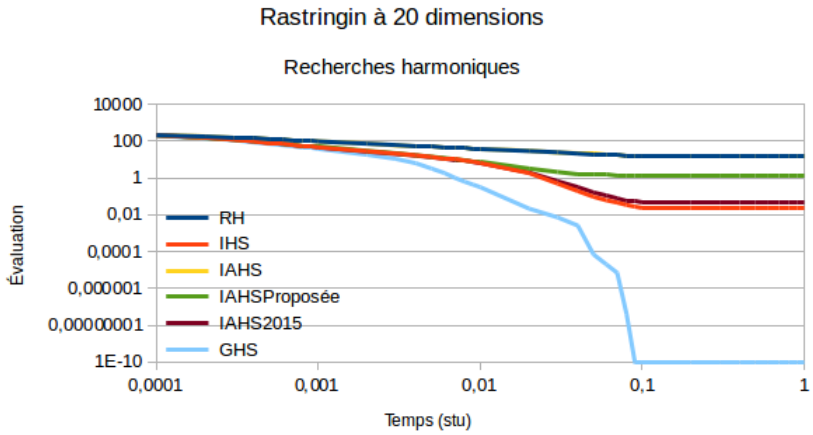


FIGURE 6.18 – Comparaison des recherches harmoniques sur Rastrigin en dimension 20

Bien sûr, comme pour les essaims particulaires, ces fonctions sont soumises au "no free lunch theorem". Ainsi, sur d'autres problèmes, les résultats pourraient être tout autre. En témoigne la figure 6.19 où GHS est légèrement plus lente que les autres pour trouver l'optimum. À noter toutefois que certains problèmes en dimension 2 ne sont pas résolus par toutes les variantes. En effet, Ackley n'est pas représenté ici, car aucune des variantes ne la résout de manière optimale, bien que présentant des résultats bien souvent proches (0,0002 de moyenne pour certains alors que l'on demande 0,0001 pour être considéré comme résolu). Il en est de même pour la fonction de Rosenbrock où seulement 10 essais sur 30 de la recherche harmonique classique ont abouti contre 30 pour IHS et aucune pour GHS.

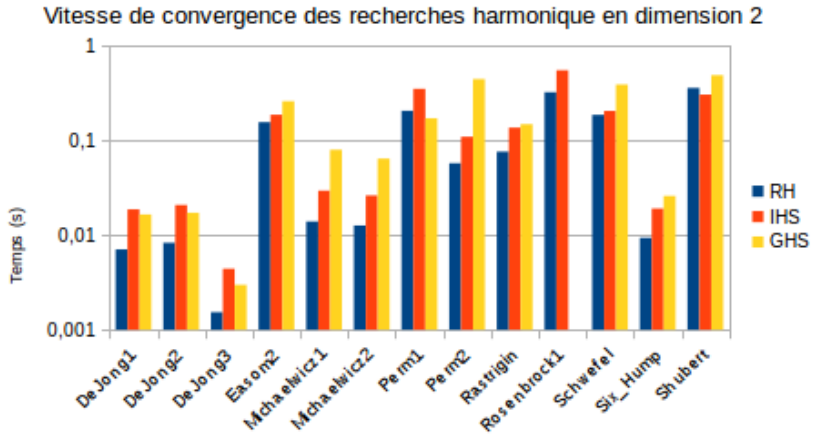


FIGURE 6.19 – Comparaison des recherches harmoniques sur les problèmes en dimension 2

Il semble donc que GHS soit très efficace sur des problèmes complexes alors que IHS le soit plus sur des problèmes de moindre difficulté.

Comparaison des algorithmes génétiques Nous avons implémenté beaucoup de variantes de l'algorithme génétique (voir annexe 6.4.2). Nous en avons testé 32 sur nos benchmarks. Il nous est donc difficile de toutes les comparer.

Bien que présentant souvent des résultats acceptables, les variantes utilisant une sélection élitiste et/ou une mutation complètement aléatoire semblent moins intéressantes que les autres. En effet, elles ne sont jamais meilleures que toutes les autres sur un problème donné, du moins sur nos tests.

Les figures 6.20 et 6.21 montrent les évaluations moyennes des solutions finales d'une douzaine de variantes d'algorithmes génétiques. La composition de ces variantes est visible dans le tableau 6.5.

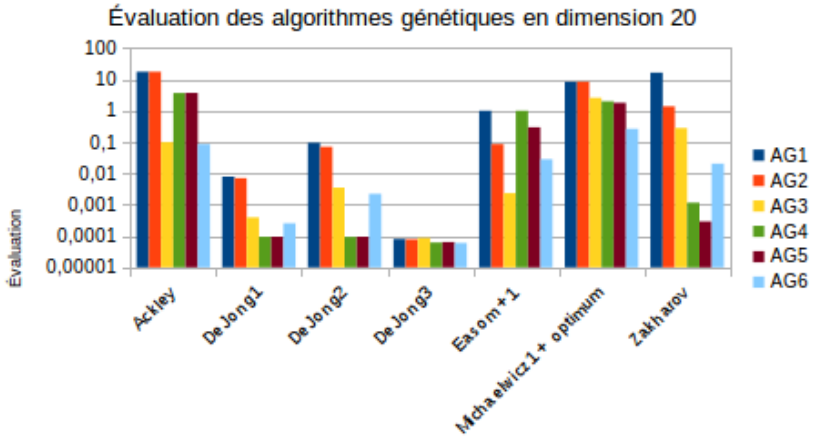


FIGURE 6.20 – Comparaison entre les algorithmes génétiques en dimension 20

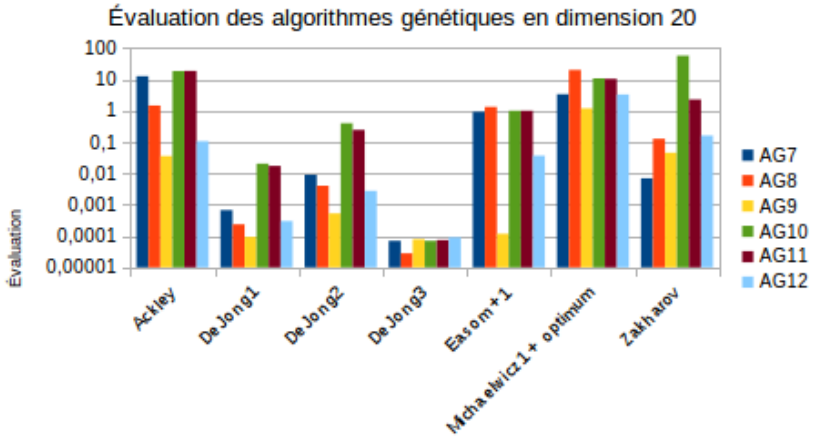


FIGURE 6.21 – Comparaison entre les algorithmes génétiques en dimension 20 (suite)

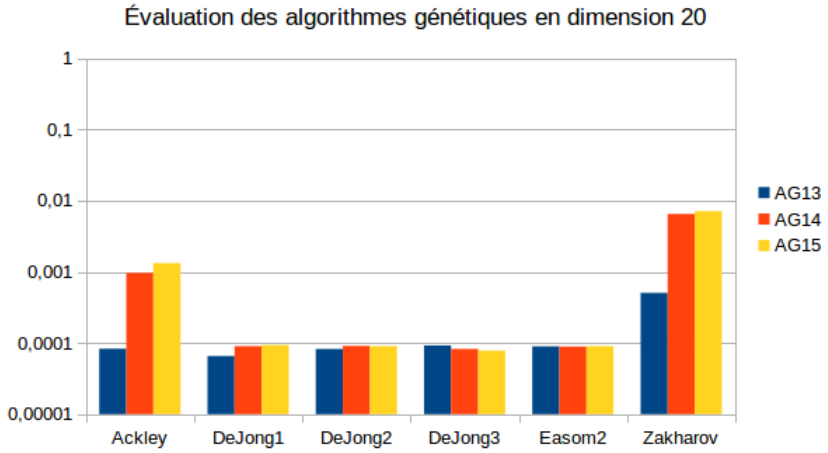


FIGURE 6.22 – Comparaison entre les algorithmes génétiques en dimension 20 (suite)

On remarque qu'aucune des variantes ne semble se distinguer des autres radicalement. Les résultats obtenus peuvent cependant grandement varier d'un problème à l'autre, mais il semble qu'il y ait toujours une variante qui donne une solution acceptable. Cela n'est cependant pas le cas sur le problème de Schwefel que nous n'avons pas représenté sur ces graphiques pour une question de visibilité. En effet, les résultats obtenus sont souvent très mauvais.

	Sélection	Croisement	Mutation
AG1	Tournois stochastique avec remise	Uniforme	Gausienne
AG2	Tournois stochastique avec remise	Uniforme	Gausienne règle 1/5
AG3	Tournois stochastique avec remise	Uniforme	Réelle classique
AG4	Tournois stochastique avec remise	BLX linéaire	Gausienne
AG5	Tournois stochastique avec remise	BLX linéaire	Gausienne règle 1/5
AG6	Tournois stochastique avec remise	BLX linéaire	Réelle classique
AG7	Tournois stochastique avec remise	Un point aléatoire	Gausienne
AG8	Tournois stochastique avec remise	Un point aléatoire	Gausienne règle 1/5
AG9	Tournois stochastique avec remise	Un point aléatoire	Réelle classique
AG10	Tournois stochastique avec remise	BLX volumique	Gausienne
AG11	Tournois stochastique avec remise	BLX volumique	Gausienne règle 1/5
AG12	Tournois stochastique avec remise	BLX volumique	Réelle classique
AG13	Tournois stochastique avec remise	BLX Linéaire	Polynomiale
AG14	Tournois stochastique avec remise	BLX volumique	Polynomiale
AG15	Tournois stochastique avec remise	Uniforme	Polynomiale

TABLE 6.5 – Correspondance noms/fonctions des algorithmes génétiques

Autres méthodes D'autres méthodes présentent également de très bons résultats. On peut notamment citer la recherche à voisinage variable générale. Dans ces tests, nous avons utilisé une descente à voisinage variable dont la fonction de recherche locale est une fonction de variation réelle à hauteur de 5% maximum de la plage de valeur. La fonction de perturbation utilisée permet une grande variation d'un paramètre à la fois. Les résultats obtenus sont visibles en tableau 6.6. On remarque de très bons résultats même sur des fonctions complexes, que d'autres algorithmes ont du mal à résoudre, tel que Perm2 ou Rosenbrock. Les résultats sont cependant assez variables pour ces deux fonctions.

Problème	Optimum	Meilleur	Médiane	Pire	Moyenne	Écart-type
Ackley	0,00E+00	9,00E-04	1,14E-03	1,89E-03	1,18E-03	2,28E-04
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	0,00E+00
Michaelwicz1	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	0,00E+00
Perm2	0,00E+00	6,16E-02	2,25E-01	1,87E+01	1,73E+00	3,54E+00
Rastrigin	0,00E+00	2,00E-06	7,00E-06	1,30E-05	8,00E-06	3,00E-06
Rosenbrock1	0,00E+00	1,78E-01	9,35E+00	1,45E+01	7,38E+00	5,80E+00
Schwefel	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Zakharov	0,00E+00	7,42E-02	1,58E-01	2,15E-01	1,51E-01	3,64E-02

TABLE 6.6 – Benchmark en dimension 20 : recherche à voisinage variable limitée à 0.1stu

De même, la colonie d'abeilles artificielles présente d'excellents résultats (tableau 6.7) sur l'ensemble des problèmes en dimension 20 à l'exception de Zakharov. La grande différence entre les solutions proposées, représentée par l'écart-type élevé, nous laisse cependant penser que des tests sur une plus longue

durée pourraient être bénéfiques.

Problème	Optimum	Meilleur	Médiane	Pire	Moyenne	Écart-type
Ackley	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	0,00E+00
Michaelwicz1	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	3,36E-03
Perm2	0,00E+00	1,35E-02	1,48E+00	1,91E+01	3,28E+00	5,22E+00
Rastrigin	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Rosenbrock1	0,00E+00	0,00E+00	3,83E-03	6,09E-01	3,43E-02	1,14E-01
Schwefel	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Zakharov	0,00E+00	8,09E+00	5,33E+01	1,64E+02	5,99E+01	3,34E+01

TABLE 6.7 – Benchmark en dimension 20 : colonie d'abeilles artificielles, limité à 0.1stu

Ainsi, nous avons utilisé les mêmes benchmarks, mais en autorisant jusqu'à 1 stu de temps de calcul. Bien sûr, ces tests ont été réalisés sur la même machine qu'auparavant et avec autant de lancements indépendants. Les résultats sont disponibles en tableau 6.8 pour la RVVG et 6.9 pour la colonie d'abeilles. Nous avons également effectué ces mêmes tests sur l'évolution différentielle. Seuls les résultats des tests avec 10 secondes sont présentés en tableau 6.10.

La RVVG présente alors des solutions quasi optimales à tous les problèmes et avec une très petite variabilité, à l'exception de Perm2 où de mauvaises solutions peuvent être trouvées sur certains essais.

Les résultats de la colonie d'abeilles sont également très bons sur l'ensemble des solutions et sont très comparables à ceux de la RVVG. On peut cependant noter une légère supériorité pour la RVVG sur le problème de Zakharov.

Les résultats de l'algorithme à évolution différentielle sont excellents sur la plupart des problèmes. Il est par exemple le meilleur sur Zakharov.

Problème	Optimum	Meilleur	Médiane	Pire	Moyenne	Écart-type
Ackley	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-1,00E+00	-0,00E+00	-9,67E-01	1,80E-01
Michaelwicz1	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	0,00E+00
Perm2	0,00E+00	3,15E-03	1,44E+00	3,29E+01	3,23E+00	6,62E+00
Rastrigin	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Rosenbrock1	0,00E+00	9,04E-04	1,58E-01	1,03E+00	1,71E-01	1,91E-01
Schwefel	0,00E+00	2,55E-04	2,55E-04	2,57E-04	2,55E-04	0,00E+00
Zakharov	0,00E+00	1,26E-03	2,17E-03	3,16E-03	2,23E-03	3,81E-04

TABLE 6.8 – Benchmark en dimension 20 : recherche à voisinage variable limité à 1 stu

	Optimum	Meilleur	Pire	Moyenne	Écart-type
Ackley	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-1,65E-07	-4,34E-01	4,95E-01
Michaelwicz1	-1,96E+01	-1,63E+01	-8,45E+00	-1,37E+01	2,16E+00
Perm2	0,00E+00	2,95E-03	4,53E+01	1,01E+01	1,35E+01
Rastrigin	0,00E+00	0,00E+00	4,09E+00	1,36E-01	7,34E-01
Rosenbrock1	0,00E+00	1,42E+01	1,80E+01	1,62E+01	8,31E-01
Schwefel	0,00E+00	2,97E+03	4,27E+03	3,60E+03	3,54E+02
Zakharov	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00

TABLE 6.11 – Benchmark en dimension 20 : Grey wolfs limité à 1 stu

Problème	Optimum	Meilleur	Médiane	Pire	Moyenne	Écart-type
Ackley	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	0,00E+00
Michaelwicz1	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	-1,96E+01	0,00E+00
Perm2	0,00E+00	8,37E-04	8,37E-03	3,29E+01	2,13E+00	6,66E+00
Rastrigin	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Rosenbrock1	0,00E+00	0,00E+00	5,21E-04	4,77E-03	1,13E-03	1,34E-03
Schwefel	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Zakharov	0,00E+00	5,83E-02	8,94E-01	3,29E+00	1,09E+00	8,54E-01

TABLE 6.9 – Benchmark en dimension 20 : colonie d'abeilles artificielles limité à 1 stu

Problème	Optimum	Meilleur	Médiane	Pire	Moyenne	Écart-type
Ackley	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	0,00E+00
Michaelwicz1	-1,96E+01	-1,96E+01	-1,96E+01	-1,95E+01	-1,96E+01	2,28E-02
Perm2	0,00E+00	0,00E+00	1,33E-03	3,00E-03	1,48E-03	8,19E-04
Rastrigin	0,00E+00	0,00E+00	1,68E-08	1,88E-06	1,04E-07	2,42E-07
Rosenbrock1	0,00E+00	4,35E+00	5,89E+00	9,74E+00	6,41E+00	2,01E+00
Schwefel	0,00E+00	1,24E-04	3,54E-04	2,87E-03	9,62E-04	1,02E-03
Zakharov	0,00E+00	7,90E-05	9,50E-05	1,00E-04	9,30E-05	7,00E-06

TABLE 6.10 – Benchmark en dimension 20 : Évolution différentielle limité à 1 stu

	Optimum	Meilleur	Pire	Moyenne	Écart-type
Ackley	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-5,00E-10	-5,34E-01	4,98E-01
Michaelwicz1	-1,96E+01	-1,69E+01	-8,69E+00	-1,38E+01	2,12E+00
Perm2	0,00E+00	3,06E-03	1,97E+01	2,91E+00	4,72E+00
Rastrigin	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Rosenbrock1	0,00E+00	1,43E+01	1,79E+01	1,57E+01	8,05E-01
Schwefel	0,00E+00	9,87E+02	4,29E+03	2,81E+03	8,26E+02
Zakharov	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00

TABLE 6.12 – Benchmark en dimension 20 : Improved Grey wolfs limité à 1 stu

	Optimum	Meilleur	Pire	Moyenne	Écart-type
Ackley	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong2	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
DeJong3	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Easom2	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00	-1,00E+00
Michaelwicz1	-1,96E+01	-1,53E+01	-1,00E+01	-1,26E+01	1,53E+00
Perm2	0,00E+00	1,21E-01	6,26E+01	1,88E+01	1,95E+01
Rastrigin	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
Rosenbrock1	0,00E+00	8,80E+00	9,96E+00	9,38E+00	3,34E-01
Schwefel	0,00E+00	0,00E+00	1,75E+00	1,84E-01	3,57E-01
Zakharov	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00

TABLE 6.13 – Benchmark en dimension 20 : Whales limité à 1 stu

Conclusion de ces tests Comme le "no free lunch theorem" l'énonçait, il est extrêmement difficile de prévoir quel algorithme est meilleur qu'un autre sur un problème donné. En effet, même après avoir réalisé des tests sur d'autres problèmes, ou en les simplifiant, les résultats peuvent être surprenants. On a pu le voir plusieurs fois, notamment avec les différences de résultats entre les dimensions 2 et 5 des essaims particulaires.

Certains algorithmes semblent en revanche être des valeurs sûres :

— la recherche à voisinage variable générale, qui a toujours présenté de très bons résultats, même si ceux-ci peuvent bien sûr être surpassés par d'autres fonctions dans certains cas

— la colonie d'abeilles artificielle

— la recherche harmonique globale qui, même si elle semble souvent plus lente que ses homologues, présente généralement de meilleurs résultats sur les problèmes complexes

— l'algorithme génétique : la grande diversité des variantes qu'il propose permet bien souvent de résoudre n'importe quel problème

Bien sûr, les fonctions non citées ici ne sont pas non plus à inintéressantes. Il a en effet été montré que la plupart d'entre elles peuvent être bonnes sur certains problèmes. De plus, pour un algorithme donné, nous avons utilisé les mêmes paramètres sur chacune des fonctions de tests. Il est possible qu'un changement de paramètres permette d'obtenir de meilleurs résultats dans certains cas.

6.5.2 Optimisation sous contraintes

Protocole Pour cette série de tests, nous avons repris le protocole proposé par Runarsson and Yao (2000). Celui-ci propose 13 problèmes d'optimisation sous contraintes et une méthode de résolution à partir d'un algorithme génétique qui base sa phase de sélection sur une méthode de tri stochastique (voir section 6.4.14 pour plus de précisions). Ainsi, nous avons implémenté cette méthode afin de comparer les résultats avec nos algorithmes.

Ce protocole limite le nombre d'itérations des algorithmes génétiques à 1750. Nous ne pouvons pas utiliser ce critère d'arrêt pour les autres algorithmes, ainsi nous l'avons implémenté en premier afin de mesurer le temps de calcul que cela représentait et l'avons reporté sur nos autres algorithmes. Le nombre de réplications est également fixé à 30.

Pour ces tests, nous avons comparé plusieurs algorithmes présentés précédemment et plusieurs méthodes de prise en compte des contraintes :

— degré de violation avec coefficient de pénalisation statique :
  — minimisation de la fonction objectif
  — considérant qu'un individu réalisable est forcément supérieur à un qui ne l'est pas Powell and Skolnick (1993)

— degré de violation avec pénalité adaptative

— stochastic ranking, utilisable uniquement dans les algorithmes génétiques

Pour cette section, nous ne comparerons que très peu les algorithmes entre eux mais plutôt les façons de prendre en compte les contraintes. En effet, nous avons déjà suffisamment comparé les algorithmes précédemment, confirmant ainsi une fois encore le "no free lunch theorem". Il n'est donc pas nécessaire de réitérer l'exercice.

Il est à noter que les fonctions d'égalité stricte, souvent notées $h(x)$, ont été simplifiées. Elles sont considérées comme valides si l'écart à l'égalité est suffisamment faible, typiquement 0.001. Cette simplification des contraintes est normale lors de benchmark, car les fonctions utilisées ont un sens mathématique plutôt que réaliste. En réalité, des contraintes de stricte égalité sont extrêmement rares lors d'utilisation de variables réelles.

Lors de ces tests, nous nous sommes rendu compte que la méthode de pénalité adaptative peut être très efficace, mais est bien souvent dépendante du problème et de l'algorithme et présente des résultats similaires à une pénalité statique adaptée. Ainsi, nous ne détaillerons pas ces résultats par la suite. Il est cependant à noter que sur des problèmes plus complexes, cette méthode présente souvent d'excellents résultats.

Attention : une erreur s'est glissée dans la publication dont nous avons extrait le protocole, en effet la fonction g08 originale doit être maximisée et non minimisée Koziel and Michalewicz (1999). Cependant, lors de ces tests, nous transformons les fonctions de maximisation en fonction de minimisation (passage au négatif). L'annexe 6.5.4 présente l'ensemble des fonctions utilisées pour ces tests.

Premières comparaisons

RVVG Pour commencer, nous avons voulu tester si un algorithme à solution unique pouvait être utilisable sur ce type de problèmes. En effet, l'espace de définition étant rarement continu du fait des contraintes à respecter, l'exploration peut s'avérer très difficile avec une solution unique.

Notre recherche à voisinage variable générale présente quelques résultats intéressants (voir figure 6.23) notamment sur les problèmes g01 et g12 mais ils sont souvent très faibles comparés aux algorithmes que nous verrons par la suite. Il semble que sur ce type d'algorithme, l'utilisation de méthodes priorisant le respect des contraintes à la qualité des solutions soit inefficace.

Les meilleurs résultats trouvés par cette méthode sont cependant souvent très bons. Ils sont malheureusement irréguliers. Vous trouverez ces valeurs dans les tableaux récapitulatifs 6.14 et 6.15.

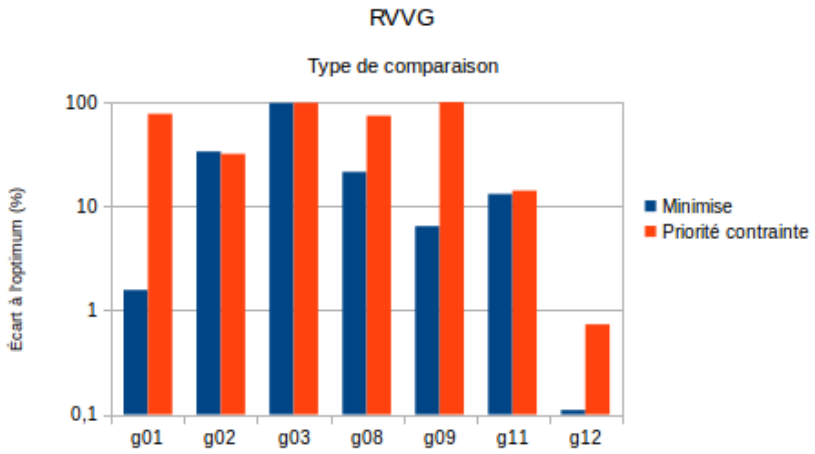


FIGURE 6.23 – RVVG : moyenne des résultats avec coefficient de pénalisation statique

EP2007 La version de 2007 de l'essaim particulaire présente des résultats mitigés. Elle semble très dépendante de la valeur du coefficient de pénalisation, ainsi que du type de comparaison utilisé. En effet, on remarque (figure 6.24) que sur g06 aucune solution respectant les contraintes n'est trouvée en utilisant la minimisation de la fonction objectf. De très bons résultats sont cependant observés en priorisant le respect des contraintes sauf pour $k = 10000$. Des résultats opposés sont observés sur g13 et g07. L'absence de barres signifie qu'aucune, ou trop peu de solutions réalisables n'ont été trouvées pour que les résultats soient significatifs.

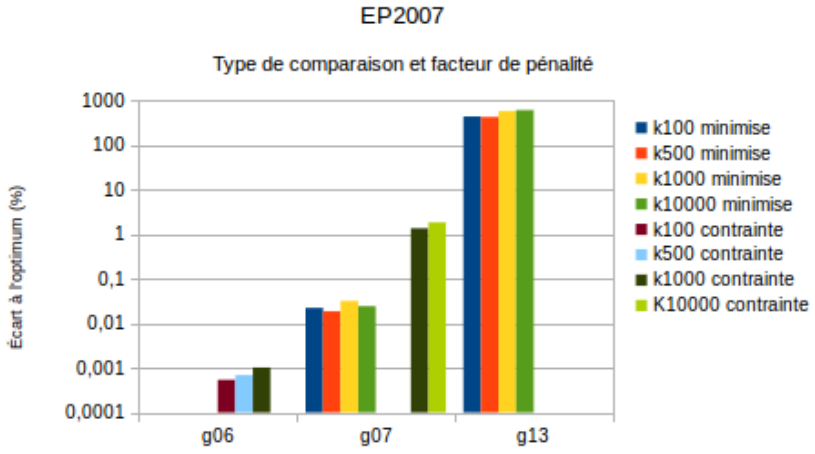


FIGURE 6.24 – ep2007 comparaison facteur pénalité

À noter que les résultats moyens sur g13 sont loin de l'optimum mais ce problème est particulièrement difficile à résoudre, ces résultats sont parmi les meilleurs que nous ayons obtenus sur ce problème. Ils sont

cependant variables, allant de l'optimum global à plusieurs fois la valeur de celui-ci. En effet, l'optimum global de ce problème semble très difficile à localiser alors qu'un des optimums locaux (présentant une évaluation 8 fois supérieure) semble facilement atteignable. Cette dernière affirmation est cependant à nuancer, elle n'est le résultat que de suppositions quant aux résultats que nous avons obtenus. Une étude plus approfondie de cette fonction serait nécessaire à sa confirmation.

Comparaison des résultats sur les algorithmes génétiques La plupart des méthodes de gestion des contraintes proposées dans la littérature sont basées sur les algorithmes évolutionnaires, génétiques. Nous n'avons pour l'instant implémenté que la méthode de *stochastic ranking* ainsi que la méthode consistant à classer les individus de sorte que toutes les solutions respectant les contraintes soient mieux évaluées que les autres.

Degré de violation avec facteur de pénalité statique Un premier test a été de comparer les résultats de la gestion des contraintes par prise en compte des degrés de violation avec facteur de pénalité statique. On remarque (en figure 6.25) que pour une variante fixée d'un algorithme génétique, le changement de valeur du facteur k peut entraîner des résultats variables. En effet, on constate que sur g01 l'utilisation d'un facteur $k = 10000$ lors d'utilisation de la méthode de priorisation des contraintes, entraîne de bien meilleurs résultats que pour les autres valeurs. Pour g03, même si les résultats sont toujours proches de l'optimum, des variations sont également observables.

Pour les prochains résultats utilisant un facteur de pénalité statique, nous ne montrerons que le meilleur résultat obtenu. Il est également à noter que les résultats dépendent également de la variante d'algorithme génétique utilisée. De même, nous ne présenterons que les résultats de la meilleure.

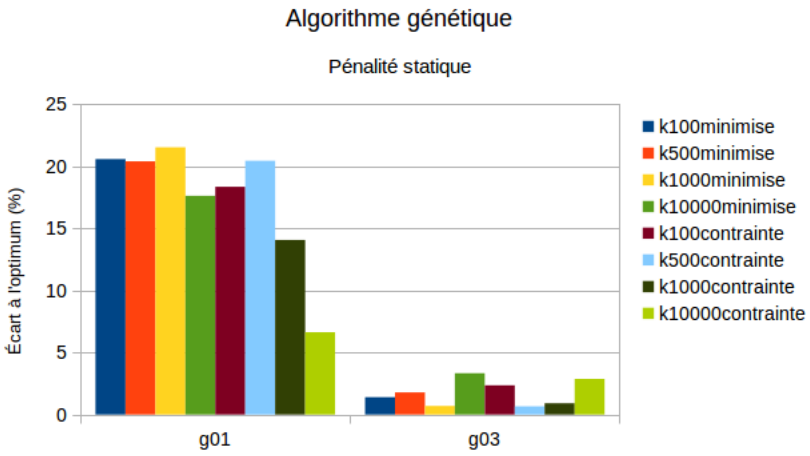


FIGURE 6.25 – Algorithme génétique comparaison facteur pénalité

Stochastic ranking Cette méthode possède l'énorme avantage de ne pas dépendre d'un facteur de pénalité, fixe ou dynamique, souvent fortement problème dépendant du problème. Des variations du paramètre Pf peuvent cependant être à tester mais l'étude de l'auteur Runarsson and Yao (2000) semble montrer qu'une valeur fixe de $Pf = 0.45$ est bien souvent optimale. La plupart du temps, la distance au respect des contraintes est souvent élevée au carré lors de l'utilisation de cette méthode. Sur nos prochains graphiques, cette variante correspond à *stochastic ranking* 2. En effet, nous avons également testé sans cette mise au carré, *stochastic ranking*, afin de comparer les résultats.

Pour un algorithme génétique basé sur une sélection élitiste, un remplacement générationnel, une mutation réelle, un croisement $BLX-\alpha$ linéaire et une stratégie de reproduction classique, nous obtenons les résultats présentés en figure 6.26. On remarque que le *stochastic ranking* présente d'excellents résultats, souvent équivalents voire meilleurs que ceux proposés par les autres méthodes. Il est le seul à proposer des solutions réalisables pour g10 mais n'en trouve cependant pas pour g05.

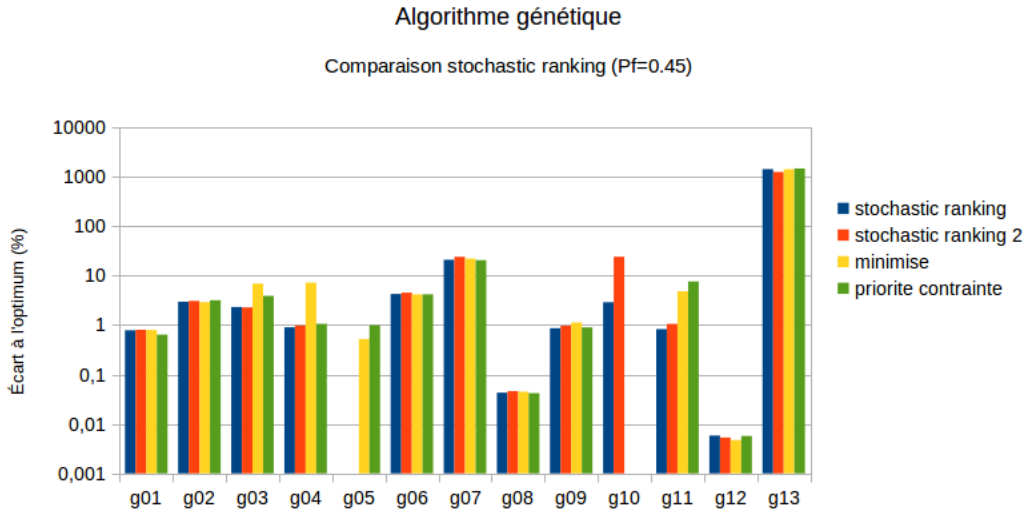


FIGURE 6.26 – Algorithme génétique comparaison résultats globaux

La fonction de classement utilisée par cet algorithme est cependant bien plus lente qu'une simple comparaison en fonction de l'évaluation. Les résultats présentés ci-dessus ont été observés après 1750 itérations comme le préconisait notre protocole. Ces itérations ont cependant été réalisées bien plus rapidement lors d'utilisation de degrés de violation des contraintes. Ainsi, nous présentons en figure 6.27, les différences de convergence de ces méthodes en fonction du temps. Les courbes bleu et rouge montrent la convergence lors de l'utilisation du protocole initial. Ainsi, au bout d'une seconde, le nombre maximum d'itérations est atteint et l'algorithme s'arrête.

Nous avons donc cherché à comparer les résultats du *stochastic ranking* avec ceux du degré de pénalisation pour un même effort de calcul en terme de durée. On obtient ainsi de meilleurs résultats pour la plupart des problèmes. Ils sont représentés par les courbes marron et bleu ciel sur la figure 6.27.

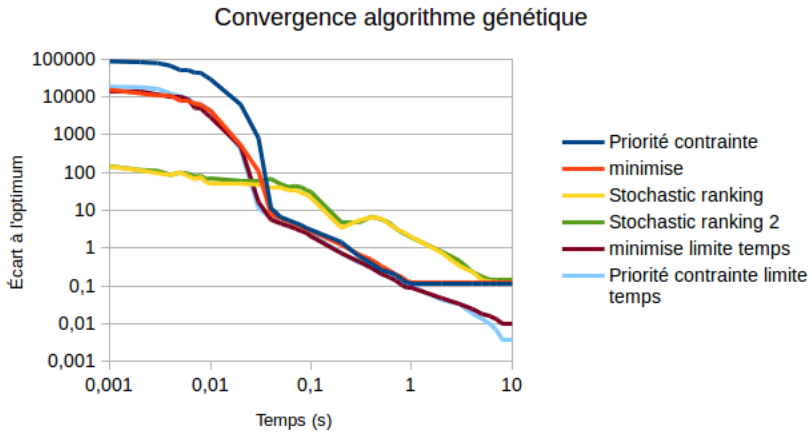


FIGURE 6.27 – Algorithme génétique comparaison convergence sur g01

Bien que ces résultats doivent être gardés à l'esprit, ils peuvent cependant être totalement différents avec d'autres variantes d'algorithme génétique et dépendent également de la proportion d'effort de calcul que représente le classement des solutions. En effet, en optimisation via simulation par exemple, l'effort de calcul est souvent bien plus réparti sur les simulations, rendant la différence de performances de la méthode de classement négligeable.

Conclusion et tableau récapitulatif Les différents tests effectués tendent à montrer que des méthodes simples, telles que la pénalisation à coefficients statiques peuvent être utilisées avec la plupart des algorithmes et s'avèrent efficaces sur des problèmes tels que ceux présentés ici. Ces méthode présentent cependant le désavantage de demander une phase de test pour trouver les paramètres optimaux.

Les algorithmes à solution unique semblent peu efficaces sur des problèmes où des solutions respectant les contraintes sont difficilement trouvables. Sur un grand nombre de tests, des solutions très correctes sont cependant trouvées. Il pourrait donc être intéressant de les utiliser en les couplant à d'autres algorithmes à population de solutions lors d'optimisation hybride.

Les tableaux suivants présentent les résultats optimaux (tableau 6.14) et moyens (tableau 6.15) de quelques un des algorithmes présentés précédemment. Bien que présentant d'excellents résultats lors de l'optimisation globale, la colonie d'abeilles artificielle (ABC dans les tableaux) semble plutôt moyenne ici. On peut également noter que EP2007 présente de meilleurs résultats dans les problèmes où il trouve des solutions mais il n'est pas régulier.

Problème	*stochastic ranking*	AG 1750 itérations	EP2007	ABC	RVVG
g01	0,7123	0,5380			0,0553
g02	1,3635	2,3402	35,4413	20,5427	17,6737
g03	0,1971	0,1577		81,9713	89,9246
g04	0,6213	3,5530		0,8831	0,0052
g05		0,3693	0,0003		1,6475
g06	1,5125	2,0132		5,8679	2,8468
g07	17,0172	15,5336	0,0042	15,2006	8,9169
g08	0,0042	0,0449		0,0052	0,0000
g09	0,4830	0,5662	0,0000	0,5863	0,3256
g10	0,1776			3,6758	
g11	0,0000	0,0000	0,0000	0,0000	0,0000
g12	0,0036	0,0036	0,0015	0,0011	0,0000
g13	824,0019	798,3633	0,9138	545,5348	65,6423

TABLE 6.14 – Meilleur écart à l'optimum (en pourcentage)

Problème	*stochastic ranking*	AG 1750 itérations	EP2007	ABC	RVVG
g01	0,8511	0,8435			1,5923
g02	3,1204	3,4489	48,8806	27,4809	35,7036
g03	1,4934	3,8792		97,7541	98,5456
g04	0,9810	3,5520		1,2823	0,0642
g05		1,5335	13,5965		154,1729
g06	4,0397	3,5177		15,6086	11,7833
g07	26,8822	30,7905	0,0184	94,3982	411,8351
g08	0,0511	0,0605		0,0522	31,4010
g09	0,7396	0,9165	0,0000	1,9929	16,6057
g10	4,5269			29,6992	
g11	0,0000	0,0000	0,0000	0,0000	0,0000
g12	0,0064	0,0056	0,0061	0,0053	0,0000
g13	1304,0778	1221,6775	418,3540	2246,5802	1823,7238

TABLE 6.15 – Écart moyen à l'optimum (en pourcentage)

D'autres tests ont également été effectués sans être présentés ici car les résultats sont peu concluants. Par exemple, l'utilisation de contraintes éliminatoires provoque souvent un temps de recherche de solutions acceptables très important sans utiliser d'heuristiques de satisfaction de contraintes qui sont problème dépendantes. Ceci dit, une fois une population de solutions acceptables initialisée, les résultats sont souvent très bons. Les temps de calcul nécessaires sont cependant souvent plus importants.

De plus, en éliminant les solutions par l'évaluation, le fait de de ne pas inclure de méthode afin de quantifier l'écart qui sépare la solution étudiée d'une solution potentiellement acceptable peut potentiellement provoquer la suppression de très bonnes solutions au profit de mauvaises.

6.5.3 Optimisation multiobjectif

Protocole N'ayant pas implémenté un grand nombre d'algorithmes consacrés spécifiquement à l'optimisation multiobjectif, nous avons simplifié le protocole de test proposé par Li and Zhang (2009). Ainsi, nous n'utilisons que les 3 premières fonctions qu'il propose ainsi que la classique fonction de Binh and Korn pour tester la prise en compte des contraintes dans le multiobjectif. Les caractéristiques de chaque problème testé sont données en table 6.16. Les tests sont effectués sur la même machine que précédemment.

Problème	Nombre de dimensions	Temps de calcul autorisé (s)
F1	10	60
F2	30	120
F3	30	120
Binh and Korn	2	10

TABLE 6.16 – Problèmes benchmark multiobjectif

Les algorithmes et méthodes testés sont les suivants :

— NSGA-II : avec et sans mutation polynomiale

— algorithme génétique classique avec nichage par la méthode du partage et procédure d'éclaircissement :

  — algorithme génétique avec procédure d'éclaircissement

  — essaim particulaire de 2007

L'efficacité des algorithmes a été mesurée par l'IGD. La méthode naïve, consistant à réaliser de multiples optimisations en modifiant les poids des objectifs, présente souvent de très bons résultats, mais un temps de calcul très élevé. Nous ne l'utiliserons donc pas ici.

NSGA-II L'algorithme NSGA-II présente de bons résultats sur l'ensemble des problèmes testés (voir tableau 6.17) avec une légère supériorité lors de l'utilisation de la mutation polynomiale. Les fronts de pareto que nous avons obtenus pour chacune des fonctions sont visibles en figure 6.28.

Bien que les IGD soient très proches, les solutions obtenues diffèrent beaucoup. En effet, sur F2, la mutation réelle couvre une grande partie du front de pareto mais avec des solutions non optimales. Avec une mutation polynomiale en revanche, on obtient très souvent des groupements de solutions très proches du front de Pareto optimal, mais le recouvrement n'est pas total. Les résultats présentés ici ne sont qu'un cas parmi tous les tests que nous avons réalisés, les formes prises peuvent varier, mais on constate presque toujours ce phénomène.

Pour NSGA-II-DE, c'est-à-dire en utilisant une stratégie d'évolution différentielle, nos résultats sont équivalents à ceux présentés par Li and Zhang (2009) (voir tableau 6.17 et figure 6.29. Ils comparent également leurs résultats avec un autre algorithme : MOEA/D-DE. Les résultats qu'ils présentent sur ces tests sont assez impressionnants. Ainsi, nous ferons de son implémentation une priorité pour une

prochaine version de notre bibliothèque.

Que ce soit avec une mutation polynomiale ou non, avec DE ou pas, les résultats du test de la fonction de Binh and Korn mènent à des solutions quasi optimales. Nous ne l'avons donc pas comparée ici. Ce résultat est toutefois visible en figure 6.31.

NSGA-II mutation polynomiale					
nomProbleme	meilleur	median	pire	moyenne	ecart-type
F1	0,004285	0,00456	0,004879	0,004543	0,000213
F2	0,034464	0,06269	0,072815	0,059338	0,013977
F3	0,027563	0,030879	0,061392	0,048203	0,012912
NSGA-II mutation réelle					
F1	0,006254	0,006743	0,007127	0,006713	0,000236
F2	0,047121	0,055959	0,063073	0,055384	0,006525
F3	0,045868	0,046279	0,055308	0,049152	0,004356
NSGA2-DE mutation polynomiale					
F1	0.002580	0.002898	0.003827	0.002983	0.000451
F2	0.022513	0.053883	0.085495	0.048697	0.023116
F3	0.023418	0.030474	0.038009	0.030602	0.005148

TABLE 6.17 – Résultats NSGA-II

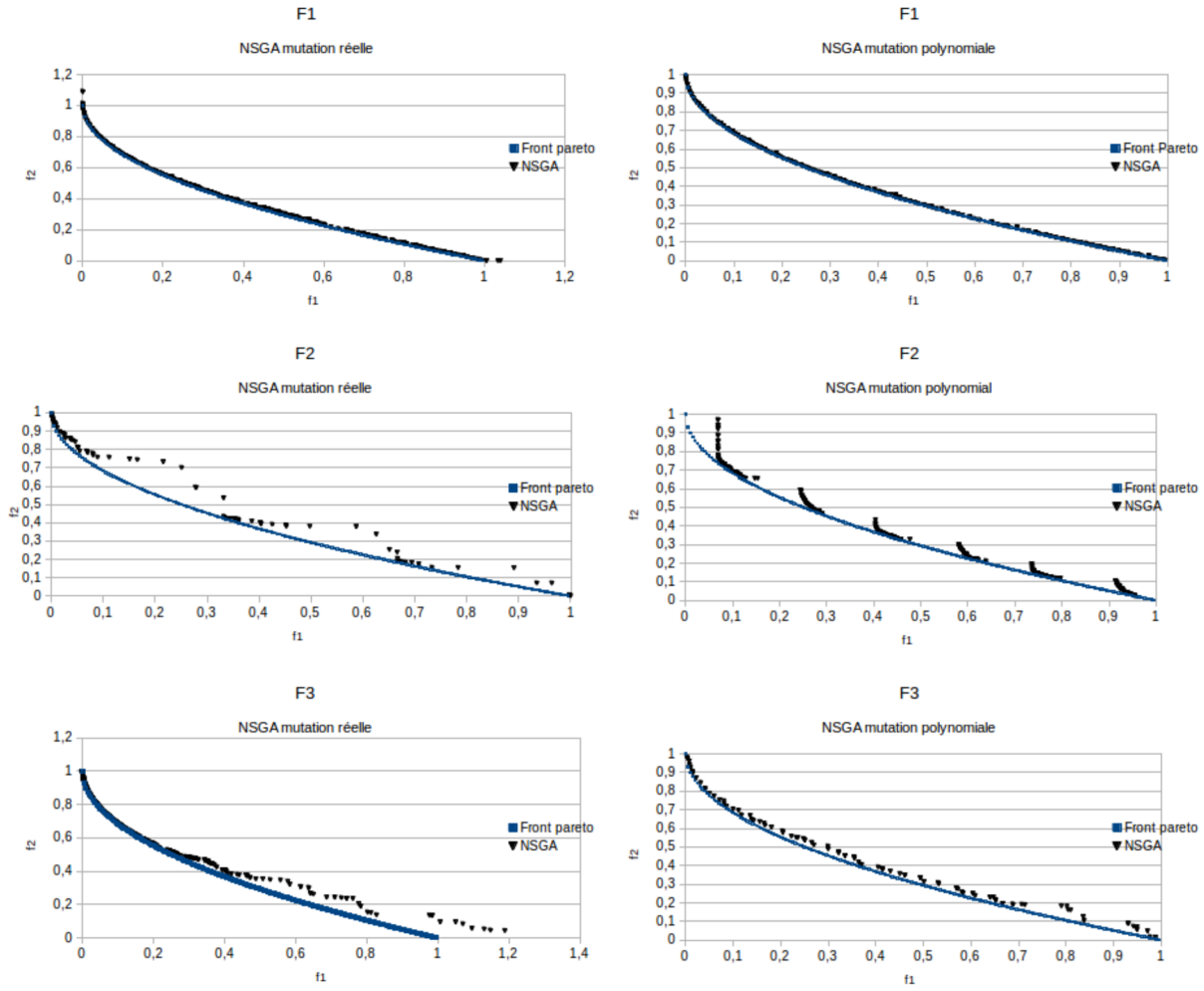


FIGURE 6.28 – Front pareto NSGA-II

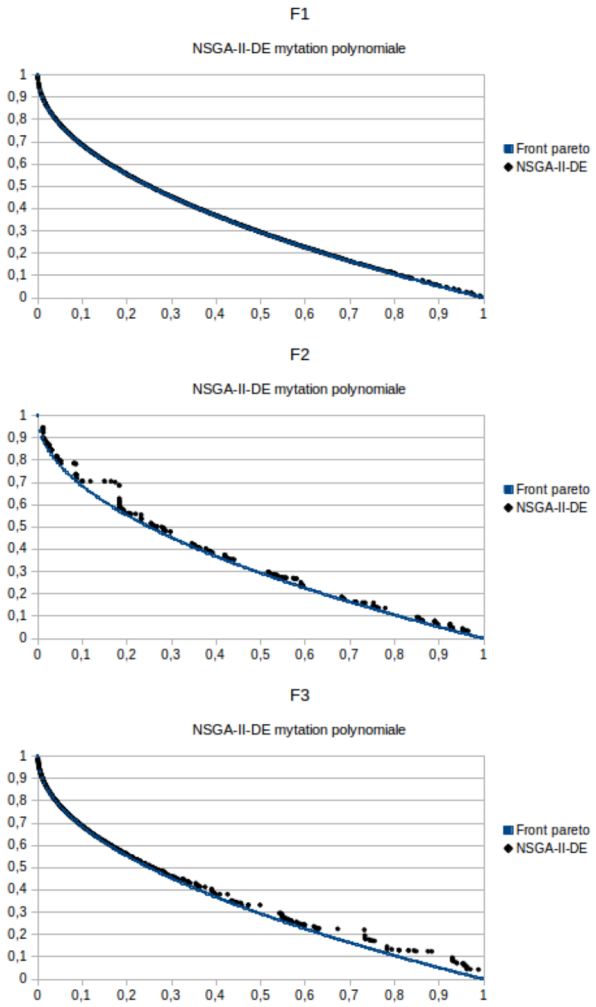


FIGURE 6.29 – Front pareto NSGA-II avec évolution différentielle

Algorithmes classiques avec méthodes de nichage Cette méthode présente l'avantage de pouvoir s'appliquer à n'importe quel algorithme à population de solutions. Son efficacité semble cependant assez faible sur des problèmes complexes. En effet, le tableau 6.18 et la figure 6.30, montrent des résultats bien moins intéressants que ceux proposés par NSGA-II. L'ensemble des solutions a été représenté pour F1 afin de montrer la dispersion des solutions. On constate sur F2 et F3 que lorsqu'on se limite aux solutions non dominées, le front est très peu et mal couvert par la méthode du partage. La procédure d'éclaircissement semble présenter de meilleurs résultats, mais n'assure pas le recouvrement du front.

AG éclaircissement					
nomProbleme	meilleur	median	pire	moyenne	ecart-type
F1	0,124614	0,164328	0,182431	0,152432	0,024138
F2	0,082036	0,189592	0,959896	0,316927	0,325297
F3	0,137192	0,166822	0,183706	0,161754	0,016582
AG partage					
F1	0,184518	0,204792	0,355187	0,234888	0,063696
F2	0,162553	0,223352	0,682555	0,300177	0,193318
F3	0,17855	0,222276	0,261015	0,219808	0,026363
EP 2007 éclaircissement					
F1	0,151732	0,207545	0,233335	0,200948	0,029462
F2	0,284143	0,385967	0,518762	0,398143	0,164732
F3	0,283581	0,346324	0,429744	0,351672	0,074354

TABLE 6.18 – Résultats algorithmes classiques avec nichage

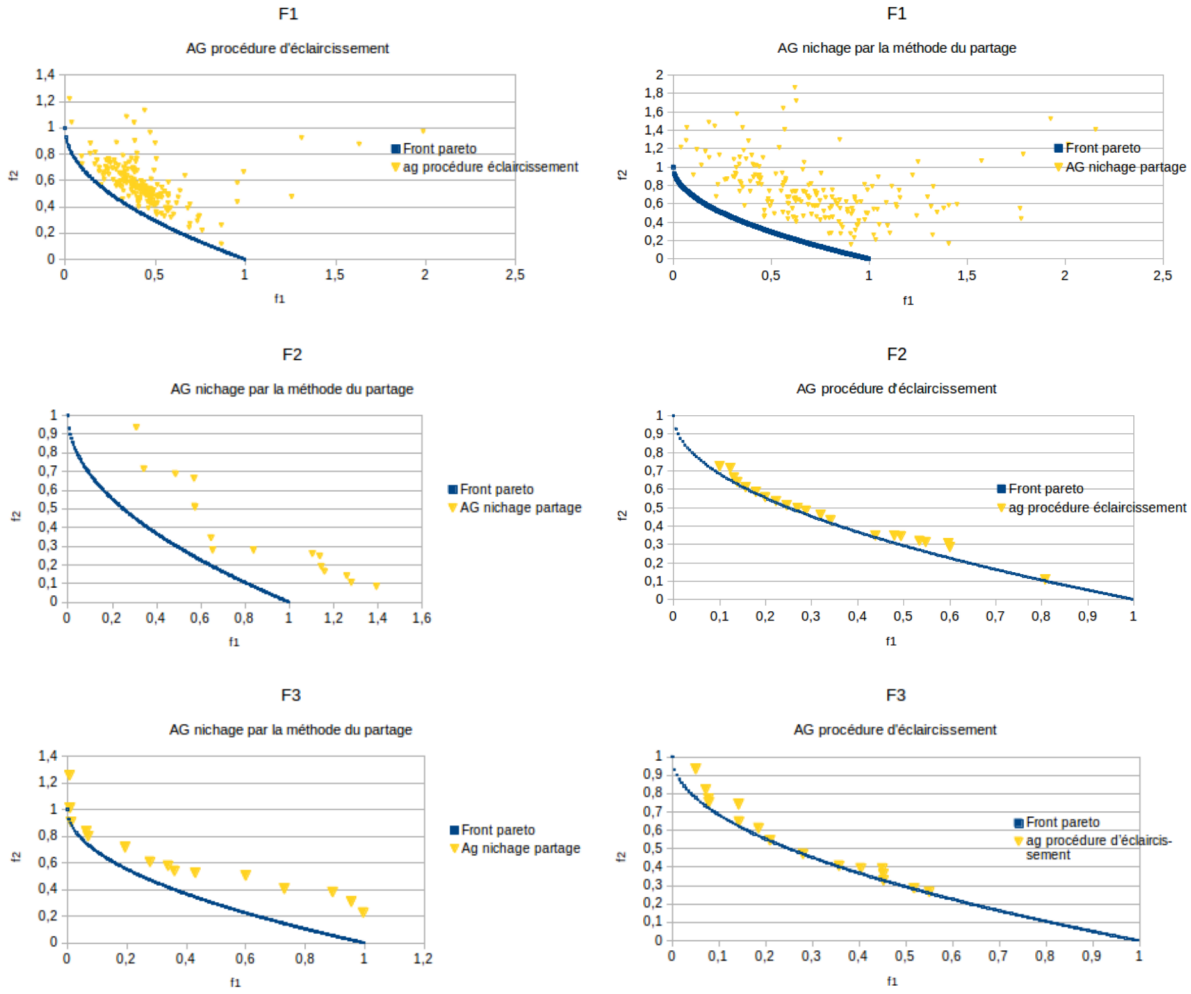


FIGURE 6.30 – Front pareto : algorithme génétique avec nichage

Finalement, sur des problèmes simples tel que Binh and Korn (voir figure 6.31), la simple utilisation du nichage semble suffisante. Les résultats sont toutefois très éloignés de ceux proposés par NSGA qui couvrent entièrement le front de Pareto.

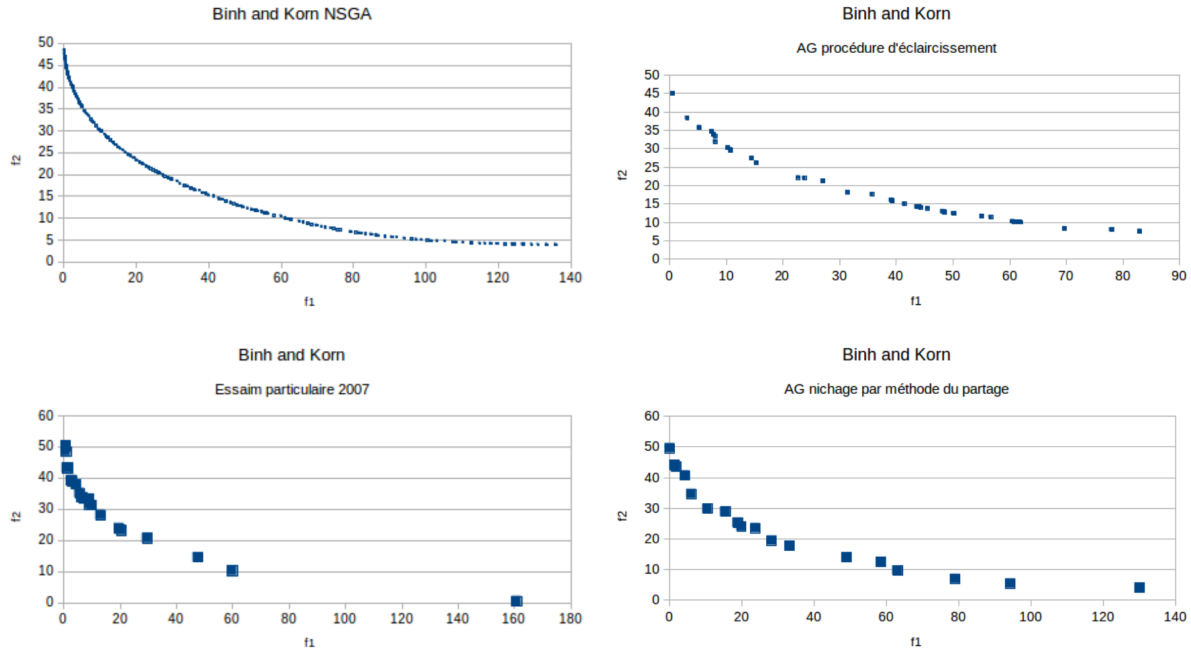


FIGURE 6.31 – Front pareto : Binh and Korn

Conclusion Les simples méthodes d'optimisation multimodale ne permettent pas à elles seules d'obtenir de bons résultats lors d'optimisation multiobjectif complexe. Ainsi, l'utilisation d'algorithmes tels que NSGA-II ou MOEA est à privilégier. Ainsi, à l'avenir, nous porterons un intérêt particulier à toutes les méthodes que nous avons pu citer en section 1.7 afin de les inclure à notre bibliothèque afin de fournir aux utilisateurs un choix de méthodes aussi grand que pour de l'optimisation globale.

Pour revenir sur les sections précédentes, l'ensemble de ces benchmarks nous ont surtout montré l'importance de pouvoir tester un grand nombre de méthodes afin de comparer les résultats. En effet, il est très simple de juger de la qualité d'une solution sur des benchmarks. Il est en revanche beaucoup plus difficile de déterminer si un algorithme est efficace sur un problème réel si on ne l'a pas comparé à d'autres.

6.5.4 Fonctions utilisées lors des benchmarks

Optimisation globale

Fonction d'Ackley

$$f(x) = -20exp\left(-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n}cos\left(2\pi x_i\right)\right) + 20 + e \tag{6.107}$$

avec $n = 1, 2...$, et $-32.768 < x_i < 32.768\ \forall i$. L'optimum global se situe en $x_* = (0, 0, ..., 0)$ et vaut $f_* = 0$

Fonctions de De Jong

$$f(x) = \sum_{i=1}^{n}x_i^2 \tag{6.108}$$

avec $n = 1, 2...$, et $-5.12 < x_i < 5.12\ \forall i$. L'optimum global se situe en $x_* = (0, 0, ..., 0)$ et vaut $f_* = 0$

$$f(x) = \sum_{i=1}^{n}ix_i^2 \tag{6.109}$$

avec $n = 1, 2...$, et $-5.12 < x_i < 5.12\ \forall i$. L'optimum global se situe en $x_* = (0, 0, ..., 0)$ et vaut $f_* = 0$

$$f(x) = \sum_{i=1}^{n}|x_i|^{i+1} \tag{6.110}$$

avec $n = 1, 2...$, et $-1 < x_i < 1\ \forall i$. L'optimum global se situe en $x_* = (0, 0, ..., 0)$ et vaut $f_* = 0$

Fonction d'Easom

$$f(x) = -(-1)^n\left(\prod_{i=1}^{n}cos^2\left(x_i\right)\right)exp\left(-\sum_{i=1}^{n}\left(x_i - \pi\right)^2\right) \tag{6.111}$$

avec $n = 1, 2, ...,$ et $-100 < x_i < 100\ \forall i$. L'optimum global se situe en $x_* = (\pi, \pi, ..., \pi)$ et vaut $f_* = -1$

Fonction de Michaelwicz

$$f(x) = -\sum_{i=1}^{n}sin\left(x_i\right)\left(sin\left(\frac{ix_i^2}{\pi}\right)\right)^2 0 \tag{6.112}$$

avec $n = 1, 2, ...,$ et$0 < x_i < \pi\ \forall i$.

Fonctions de Perm

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{n}\left(i^j + \beta\right)\left[\left(\frac{x_i}{i}\right)^j - 1\right]\right)^2\ \forall\beta >\ 0 \tag{6.113}$$

avec $n = 1, 2...$, et $-n < x_i < n \ \forall i$. L'optimum global se situe en $x_* = (1, 2, ..., n)$ et vaut $f_* = 0$

$$\sum_{j=1}^{n} \left(\sum_{i=1}^{n} \left(i^j + \beta \right) \left[x_i^j \left(\frac{1}{i} \right)^j \right] \right)^2 \quad \forall \beta > \ 0 \tag{6.114}$$

avec $n = 1, 2...$, et $-1 < x_i < 1 \ \forall i$. L'optimum global se situe en $x_* = (1, 1/2, ..., 1/n)$ et vaut $f_* = 0$

Fonction de Rastrigin

$$f(x) = 10n + \sum_{i=1}^{n} \left[x_i^2 - 10cos\left(2\pi x_i\right) \right] \tag{6.115}$$

Avec $-5.12 \leq x_i \leq 5.12 \ \forall i \in [0, n]$. L'optimum global se situe en $x_* = (0, 0, ..., 0)$ et $f_* = 0$

Fonction de Rosenbrock

$$f(x) = \sum_{i=1}^{n-1} \left(x_i - 1 \right)^2 + 100 \left(x_{i+1} - x_i^2 \right)^2 \tag{6.116}$$

Avec $-5 \leq x_i \leq 5 \ \forall i \in [0, n]$. L'optimum global se situe en $x_* = (1, 1, ..., 1)$ et $f_* = 0$

Fonction de Schwefel

$$f(x) = -\sum_{i=0}^{n} x_i sin\left(\sqrt{|x_i|} \right) \tag{6.117}$$

Avec $-500 \leq x_i \leq 500 \ \forall i \in [0, n]$. L'optimum global se situe en $x_* = (420.9687, ..., 420, 9687)$ et $f_* = -418.9829n$

Fonction six hump

$$f(x, y) = \left(4 - 2.1x^2 + \frac{1}{3}x^4 \right) x^2 + xy + 4\left(y^2 - 1 \right) y^2 \tag{6.118}$$

Avec $-3 \leq x \leq 3 \, et \, -2 \leq y \leq 2$. Cette fonction à deux optimum globaux : $(x_*, y_*) = (0.0898, -0.7126) \ et \, (-0.0898, 0.71$ pour $f_* = -1.0316$

Fonction de Shubert

$$f(x) = \left[\sum_{i=1}^{n} icos\left(i + (i+1)\, x \right) \right] . \left[\sum_{i=1}^{n} icos\left(i + (i+1)\, y \right) \right] \tag{6.119}$$

Cette fonction à plusieurs optimum globaux qui changent en fonction de n. Pour $n = 5$, $f_* = -186.7309$ avec $-10 \leq x, y \leq 10$

Fonction de Zakharov

$$f(x) = \sum_{i=1}^{n} x_i^2 + \left(\frac{1}{2}\sum_{i=0}^{n} i x_i\right)^2 + \left(\frac{1}{2}\sum_{i=0}^{n} i x_i\right)^4 \tag{6.120}$$

Avec $-5 \leq x_i \leq 10\ \forall i \in [0, n]$. L'optimum global se situe en $x_* = (0, 0, ..., 0)$ et $f_* = 0$

Optimisation sous contraintes

g01 minimiser

$$f(x) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i \tag{6.121}$$

sous les contraintes :

$$g_1(x) = 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \tag{6.122}$$

$$g_2(x) = 2x_1 + 2x_2 + x_{10} + x_{12} - 10 \leq 0 \tag{6.123}$$

$$g_3(x) = 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \tag{6.124}$$

$$g_4(x) = -8x_1 + x_{10} \leq 0 \tag{6.125}$$

$$g_5(x) = -8x_2 + x_{11} \leq 0 \tag{6.126}$$

$$g_6(x) = -8x_3 + x_{12} \leq 0 \tag{6.127}$$

$$g_7(x) = -2x_4 - x_5 + x_{10} \leq 0 \tag{6.128}$$

$$g_8(x) = -2x_6 + x_7 + x_{11} \leq 0 \tag{6.129}$$

$$g_9(x) = -2x_8 - x_9 + x_{12} \leq 0 \tag{6.130}$$

$x_i \in [0, 1]\, \forall i \in [1, 9] + x_{13}\ x_i \in [0, 1]\, \forall i \in [10, 12]$
$x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ et $f(x^*) = -15$

g02 minimiser

$$f(x) = \frac{\sum_{i=1}^{n} cos^4(x_i) - 2\Pi_{i=1}^{n} cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \tag{6.131}$$

sous les contraintes :

$$g_1(x) = 0.75 - \Pi_{i=1}^{n} x_i \leq 0 \tag{6.132}$$

$$g_2(x) = \sum_{i=1}^{n} x_i - 7.5n \leq 0 \tag{6.133}$$

avec $n = 20$
$x_i \in [0, 10] \, \forall i$
L'optimum est inconnu, le meilleur qui a été trouvé est : $f(x^*) = -0.803619$

g03 minimiser

$$f(x) = \sqrt{n}^n \Pi_{i=1}^{n} x_i \tag{6.134}$$

sous la contrainte :

$$h_1(x) = \sum_{i=1}^{n} x_i^2 - 1 = 0 \tag{6.135}$$

avec $n = 10$
$x_i \in [0, 1] \, \forall i$
$x_i^* = 1/\sqrt{n} \forall i$ et $f(x^*) = 1$

g04 minimiser

$$f(x) = 5.3578547 x_3^2 + 0.8356891 x_1 x_5 + 37.293239 x_1 - 40792.141 \tag{6.136}$$

sous les contraintes :

$$g_1(x) = 85.334407 + 0.0056858 x_2 x_5 \leq 0 \tag{6.137}$$

$$g_2(x) = -85.334407 - 0.0056858 x_2 x_5 - 0.0006262 x_1 x_4 + 0.0022053 x_3 x_5 \leq 0 \tag{6.138}$$

$$g_3(x) = 80.51249 + 0.0071317 x_2 x_5 + 0.0029955 x_1 x_2 + 0.0021813 x_3^2 - 110 \leq 0 \tag{6.139}$$

$$g_4(x) = -80.51249 - 0.0071317 x_2 x_5 - 0.0029955 x_1 x_2 - 0.0021813 x_3^2 + 90 \leq 0 \tag{6.140}$$

$$g_5(x) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \leq 0 \tag{6.141}$$

$$g_6(x) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0 \tag{6.142}$$

$x_1 \in [78, 102]$ $x_2 \in [33, 45]$ $x_i \in [27, 45] \forall i \in [3, 5]$
$x^* = (78, 33, 29.995256025682, 45, 36.775812905788)$ et $f(x^*) = -30665.539$

g05 minimiser

$$f(x) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3 \tag{6.143}$$

sous les contraintes :

$$g_1(x) = -x_4 + x_3 - 0.55 \leq 0 \tag{6.144}$$

$$g_2(x) = -x_3 + x_4 - 0.55 \leq 0 \tag{6.145}$$

$$h_1(x) = 1000sin(-x_3 - 0.25) + 1000sin(-x_4 - 0.25) + 894.8 - x_1 = 0 \tag{6.146}$$

$$h_2(x) = 1000sin(x_3 - 0.25) + 1000sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0 \tag{6.147}$$

$$h_3(x) = 1000sin(-x_4 - 0.25) + 1000sin(x_4 - x_3 - 0.25) + 1294.8 = 0 \tag{6.148}$$

$x_1, x_2 \in [0, 1200]$ $x_3, x_4 \in [-0.55, 0.55]$
$x^* = (679.9453, 1026.067, 0.1188764, -0.3962336)$ et $f(x^*) = -5126.4981$

g06 minimiser

$$f(x) = (x_1 - 10)^3 + (x_2 - 20)^3 \tag{6.149}$$

sous les contraintes :

$$g_1(x) = -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \tag{6.150}$$

$$g_2(x) = (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0 \tag{6.151}$$

$x_1 \in [13, 100]$ $x_2 \in [0, 100]$
$x^* = (14.095, 0.84296)$ et $f(x^*) = -6961.81388$

g07 minimiser

$f(x) = x_1^2 + x_2^2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_10 - 7)^2 + 45$

sous les contraintes :

$$g_1(x) = -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0 \tag{6.152}$$

$$g_2(x) = 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \tag{6.153}$$

$$g_3(x) = -8x_1 + 2x_2 + 5x_9 - 2x_10 - 12 \leq 0 \tag{6.154}$$

$$g_4(x) = 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \tag{6.155}$$

$$g_5(x) = 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0 \tag{6.156}$$

$$g_6(x) = x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0 \tag{6.157}$$

$$g_7(x) = 0.5(x_1 - 8)^2 + 2(x_2 - 4) + 3x_5^2 - x_6 - 30 \leq 0 \tag{6.158}$$

$$g_8(x) = -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_10 \leq 0 \tag{6.159}$$

$x_i \in [-10, 10] \, \forall i$

$x^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$ et $f(x^*) = 24.3062091$

g08 minimiser

$$f(x) = \frac{sin^3(2\pi x_1) \, sin(2\pi x_2)}{x_1^3 (x_1 + x_2)} \tag{6.160}$$

sous les contraintes :

$$g_1(x) = x_1^2 - x_2 + 1 \leq 0 \tag{6.161}$$

$$g_2(x) = 1 - x_1 + (x_2 - 4)^2 \leq 0 \tag{6.162}$$

$x_1, x_2 \in [0, 10]$

$x^* = (1.2279713, 4.2453733)$ et $f(x^*) = 0.095825$

g09 minimiser

$$f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \quad (6.163)$$

sous les contraintes :

$$g_1(x) = -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \quad (6.164)$$

$$g_2(x) = -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \quad (6.165)$$

$$g_3(x) = -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \quad (6.166)$$

$$g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_3 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \quad (6.167)$$

$x_i \in [-10, 10] \, \forall i$
$x^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$ et $f(x^*) = 680.6300573$

g10 minimiser

$$f(x) = x_1 + x_2 + x_3 \quad (6.168)$$

sous les contraintes :

$$g_1(x) = -1 + 0.0025(x_4 + x_6) \leq 0 \quad (6.169)$$

$$g_2(x) = -1 + 0.0025(x_5 + x_7 - x_4) \leq 0 \quad (6.170)$$

$$g_3(x) = -1 + 0.01(x_8 - x_5) \leq 0 \quad (6.171)$$

$$g_4(x) = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.333 \leq 0 \quad (6.172)$$

$$g_5(x) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 \leq 0 \quad (6.173)$$

$$g_6(x) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 \leq 0 \quad (6.174)$$

$x_1 \in [0, 10000] \; x_2, x_3 \in [1000, 10000] \; x_i \in [10, 1000] \, \forall i \in [4, 8]$

$x^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 219.9799, 286.4162, 395.5979)$ et $f(x^*) = 7049.3307$

g11 minimiser

$$f(x) = x_1^2 + (x_2 - 1)^2 \tag{6.175}$$

sous la contrainte :

$$h_1(x) = x_2 - x_1^2 = 0 \tag{6.176}$$

$x_1, x_2 \in [-1, 1]$
$x^* = \left(1/\sqrt{2}, 1/2\right) U \left(-1/\sqrt{(2), 1/2}\right)$ et $f(x^*) = 0.75$

g12 minimiser

$$f(x) = \frac{100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2}{100} \tag{6.177}$$

sous la contrainte :

$$g_1(x) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - 5)^2 - 0.0625 \leq 0 \tag{6.178}$$

$x_i \in [0, 10] \, \forall i \; p, q, r = 1, 2, ..., 9$
$x^* = (5, 5, 5)$ et $f(x^*) = 1$

g13 minimiser

$$f(x) = e^{x_1 x_2 x_3 x_4 x_5} \tag{6.179}$$

sous les contraintes :

$$h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \tag{6.180}$$

$$h_2(x) = x_2 x_3 - 5 x_4 x_5 = 0 \tag{6.181}$$

$$h_3(x) = x_1^3 + x_2^3 + 1 = 0 \tag{6.182}$$

$x_1, x_2 \in [-2.3, 2.3]$ $x_3, x_4, x_5 \in [-3.2, 3.2]$
$x^* = (-1.717143, 1.595709, 1.827247, -0.7636413, -0.763645)$ et $f(x^*) = 0.0539498$

Optimisation multiobjectif

F1 Minimise :

$$f_1 = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left(x_j - x_1^{0.5\left(1.0 + \frac{3(j-2)}{n-2}\right)} \right)^2 \tag{6.183}$$

$$f_2 = 1 - \sqrt{x_1} + \frac{2}{|J_1|} \sum_{j \in J_2} \left(x_j - x_1^{0.5\left(1.0 + \frac{3(j-2)}{n-2}\right)} \right)^2 \tag{6.184}$$

avec $J_1 = 2k, k \in N/|2k \leq n$ et $J_2 = 2k+1, k \in N/|2k+1 \leq n$

Le front de pareto est défini tel que $x_j = x_1^{0.5\left(1.0 + \frac{3(j-2)}{n-2}\right)} \forall j = 2, 3, ..., n$

F2 Minimise :

$$f_1 = x_1 + \frac{2}{|J_1|} \sum_{J \in J_1} \left(x_j - sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \right)^2 \tag{6.185}$$

$$f_2 = 1 - \sqrt{x_1} x_1 + \frac{2}{|J_2|} \sum_{J \in J_2} \left(x_j - sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \right)^2 \tag{6.186}$$

avec $J_1 = 2k, k \in N/|2k \leq n$ et $J_2 = 2k+1, k \in N/|2k+1 \leq n$

Le front de pareto est défini tel que $x_j = sin\left(6\pi x_1 + \frac{j\pi}{n}\right) \forall j = 2, 3, ..., n$

F3 Minimise :

$$f_1 = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left(x_j - 0.8 x_1 cos\left(6\pi x_i + \frac{j\pi}{n}\right) \right)^2 \tag{6.187}$$

$$f_2 = 1 - \sqrt{x_1} + \frac{2}{|J_1|} \sum_{j \in J_2} \left(x_j - 0.8 x_1 sin\left(6\pi x_i + \frac{j\pi}{n}\right) \right)^2 \tag{6.188}$$

avec $J_1 = 2k, k \in N \ |2k \leq n$ et $J_2 = 2k+1, k \in N \ |2k+1 \leq n$

Le front de pareto est défini tel que $x_j = 0.8 x_1 cos\left(6\pi x_i + \frac{j\pi}{n}\right) \forall j \in J_1$ et $x_j = 0.8 x_1 sin\left(6\pi x_i + \frac{j\pi}{n}\right) \forall j \in J_1$

6.6 Benchmarks des problèmes de calage

6.6.1 Protocole

Pour tester la fiabilité des différents algorithmes sur nos problèmes de calibration nous avons testé trois paramètres :

1. le temps de simulation : 10 ans, correspondant à la durée de nouvelles données que l'on pourrait espérer avoir ; 58 ans, correspondant à la durée d'estimation disponible dans Le Manacha et al. (2011).

Paramètre	Borne inférieure	Borne supérieure
q	0.01	0.5
r	0.01	0.5
Biomasse initiale	100	2000
k	100	2000

TABLE 6.19 – Bornes des paramètres du modèle

2. la variation maximale entre les efforts sur les différentes années : 0, 5, 10, 20 ou 30%. Plus cette valeur est importante, plus l'espace des solutions diminue, augmentant la difficulté de résolution.
3. le taux d'incertitude : 0 ; 10 ou 20%. Il va définir la largeur de l'intervalle de recherche des efforts pêche à caler.

La génération des efforts de pêche au cours du temps est donnée en algorithme 32 et les bornes des paramètres de dynamique de population sont donnés en tableau **??**. À cela s'ajoute une phase de vérification de la validité des données et de la dynamique générée afin de s'assurer que cet ensemble donne bien une dynamique cohérente et donc un calage possible. Pour cela, une simple simulation est effectuée à partir des paramètres de la dynamique et des efforts générés afin de valider que la population ne s'éteint pas.

Algorithme 32 Benchmark calage : génération des efforts et des bornes de recherche

Entrée: $variationMax \geq 0$ $baseEffort > 0$ $nbAnnee > 0$ $incertitudes \geq 0$
  pour $i = 0;\ i < nbAnnee;\ i++$ **faire**
    si $randBooleen()$ **alors**
      $effort[i] = baseEffort/(1 + random(0, variationMax))$
    sinon
      $effort[i] = baseEffort \times (1 + random(0, variationMax))$
    fin si
    $borneMinEffort[i] = effort[i]/(1 + incertitudes)$
    $borneMaxEffort[i] = effort[i] \times (1 + incertitudes)$
  fin pour

Chaque problème est réalisé 100 fois pour un total de 3000 problèmes testés. Afin de pouvoir comparer les algorithmes entre eux sur ces problèmes, nous utiliserons un critère d'arrêt de temps maximum fixé à 1 stu. Un critère d'arrêt de qualité de la solution est fixé à 10^{-10} de sorte à avoir suffisamment de décimales pour comparer les résultats des algorithmes les plus performants sans non plus rencontrer des problématiques d'arithmétique flottante nous obligeant à modifier les types de variables et donc fausser les temps de calcul.

La fonction d'évaluation est :

$$f(x) = \sum_{t=0]^{nbAnnees}} (C(t) - Cs(t))^2 \tag{6.189}$$

Avec $C(t)$ les captures générées et $Cs(t)$ les captures simulées pour la solution testée, à l'instant t.

Une calibration sera considérée réussie si, pour la meilleure solution proposée, les différences d'aucune année ne dépasse 1% des captures de cette année. Ce critère est complètement arbitraire, il vise à garder une certaine cohérence avec ce qui est recherché en situation réelle, c'est-à-dire des courbes quasiment superposées, en gardant toutefois un certain niveau d'exigence.

Ainsi dans un premier temps, nous évaluerons les algorithmes indépendamment les uns des autres en testant différents jeux de paramètres pour déterminer lequel est le plus efficace sur chaque problème. Ils seront jugés uniquement sur le pourcentage de réussite à chaque problème pour le moment.

Par la suite, les meilleures configurations de chaque algorithme seront comparées en fonction également de la qualité des solutions proposées et du temps de convergence.

6.6.2 Résultats par type d'algorithme

Essaim particulaire Le tableau 6.20 montre les résultats pour 40 jeux de paramètres de l'essaim particulaire de Clerc (2012a), SPSO2007.

Comme préconisé par les auteurs, l'intervalle $w \in [0.7, 0.9]$ est testé et $c_1 = c_2 = \frac{(w+1)^2}{2}$.

On peut constater de très bons résultats notamment quand il y a peu de paramètres à caler, seulement 10 ans de simulation ou une incertitude nulle, avec des taux de réussite avoisinant les 100%. Cette méthode semble cependant trouver ses limites sur ce type de problèmes quand trop de paramètres sont présents. Ses taux de réussite sur 58 ans de simulations sont en effet nettement inférieurs aux précédents.

Évolution différentielle Cet algorithme présente de très bons résultats notamment sur les problèmes avec un taux d'incertitude faible. Il semble également bien gérer une variabilité importante quand il y a peu de paramètres à caller.

Algorithme des loups gris Selon l'implémentation standard de cet algorithme, le seul paramètre est la taille de la population. Sur les problèmes de calage, celle-ci semble assez peu significative bien qu'on remarque que les résultats globaux se dégradent légèrement avec augmentation de la population. Ceux-ci ne semblent cependant pas significatifs car le test avec la population la plus grande présente les meilleurs résultats sur les problèmes de 10 ans sans incertitudes.

Les résultats, présentés dans le tableau 6.22, sont cependant assez faibles comparés à d'autres algorithmes.

Algorithme des loups gris amélioré Les résultats de cet algorithme sont présentés en tableau 6.23. Bien que cet algorithme semble être l'un des meilleurs sur les benchmarks habituels d'optimisation globale, ses performances sont relativements faibles sur ces problèmes. Il reste cependant légèrement meilleur que la version standard de l'algorithme des loups gris.

Algorithme des baleines Les résultats de cet algorithme, présentés en tableau 6.24 semblent être les moins bons de ceux que nous avons vu jusque là alors que celui-ci présente de très bonnes performances sur les problèmes de benchmark d'optimisation globale habituels.

				10 ans					58 ans				
pop	w	c	total	u0	u10	u20	var20	var30	u0	u10	u20	var20	var30
200	0.72	1.193	**85**	99,6	**99,8**	**99,4**	**99,3**	**99**	**100**	62,6	48,6	58,7	**60,3**
100	0.84	1.6928	83,2	97,4	99,4	98,2	97,3	97	97,6	63,8	43	58,3	55
100	0.72	1.193	83	99,8	99,6	99,4	99,3	99	99,8	58	41,2	52,3	56,3
100	0.78	1.5842	82,7	97,8	99,4	98,8	97,7	97,7	96,8	59,6	44	50,7	53
200	0.78	1.5842	82,5	91,8	99,6	98,8	96	94	90,2	66,2	48,4	54,3	55,3
200	0.75	1.5312	82,5	88,6	99,8	99,2	94	94	89,6	64,8	53,2	56,3	56
100	0.75	1.5312	82,3	96,6	99,4	99,2	97	96,7	96,2	59,2	43,2	55,3	54,3
200	0.84	1.6928	82,2	83,8	99,2	99,4	92	93,3	88,2	67,2	**55,6**	58,7	58,7
100	0.81	1.6381	82	94,4	99,6	98	96,3	93,7	95,6	59,4	45	52,3	53,7
200	0.81	1.6381	81,8	82,2	99,8	99,2	92,7	92	91,8	67,2	50,4	**59,3**	58,7
100	0.87	1.7485	81,7	93	99,2	98,2	96,3	94,3	95,8	62	42,2	57	53,3
50	0.81	1.6381	81,5	99	99,2	99,2	98,3	98,7	98,2	54,6	38,6	51,3	47,7
200	0.87	1.7485	81,4	**100**	99,8	98	98,3	98	99,8	52,4	38,4	49,7	52,7
50	0.72	1.193	81,4	83,2	99,6	99	93,3	88,7	87,8	**68,2**	50,8	58	57,7
100	0.9	1.805	81,3	90,4	98,2	98	92,3	*92,7*	93	59,6	48,4	52	55,7
50	0.87	1.7485	81,1	98,4	98,4	97	96,3	95,3	97,8	54	41,2	50	52
50	0.84	1.6928	81	99,6	98	97,6	96	97	98,8	56,8	35	51,3	56,3
50	0.75	1.5312	80,9	99,2	99,4	98	98,7	97	98,8	53,8	36,2	53	50,3
50	0.95	1.9012	80,6	99,6	99,8	98,4	98	99	98,8	52,4	34,6	50	46,3
50	0.78	1.5842	80,6	98,4	97	94,2	95	93,7	97	59,8	37,4	51,7	52,7
100	0.95	1.9012	80	88,4	97,8	97,6	91,7	90,3	89	59,8	47,2	53,3	50
50	0.9	1.805	79,8	97	99,4	96	95,3	94,3	96,6	55,8	34	50,7	49,7
30	0.84	1.6928	78,4	99,6	97,6	92,8	94,3	95,3	98,6	52,2	29,8	49	51
30	0.81	1.6381	78,2	99,8	96,8	91,6	94,3	93,3	99,2	49,4	32,6	51,7	48
30	0.87	1.7485	77,2	99,4	95,6	91,4	93,7	94,3	98,6	48,6	29,4	45,7	47,3
30	0.78	1.5842	76,8	100	94,8	86,6	92	92	98,2	50,4	30,8	48	52
30	0.95	1.9012	76,6	99	92,8	89,6	91,7	91,7	98,2	50	29,8	51,7	44,7
30	0.9	1.805	76,5	99,6	93,6	89,8	93	93	98	48,4	29,4	50	47,3
30	0.75	1.5312	75,6	99,6	91,8	84,8	90,3	90	98,4	49,2	29,6	47,3	46,7
200	0.9	1.805	74,7	65,4	98,8	98,8	84	82,7	72,8	61	51,2	51,3	51,3
30	0.72	1.193	73,6	99,6	87,6	75,8	88,3	87,7	99,6	48	31,2	51,7	44,7
20	0.84	1.6928	68,4	99,8	78,6	60,8	79,7	75,7	98,6	45,8	26,6	46,7	45,7
20	0.87	1.7485	67,7	99	80	58,2	75	77,7	98,8	44,2	25,8	44,3	44
20	0.95	1.9012	67	98	78,8	63,8	81,7	79	98,6	40,8	22,2	43	42,7
200	0.95	1.9012	66,7	64,6	73,8	99,4	77	77,7	59,8	52	50,8	40	42
20	0.81	1.6381	66,2	98,8	77	56,4	80,3	75,3	97,4	43,2	24,4	45,7	42,3
20	0.9	1.805	65,5	99	76,4	55,2	75	76,3	97,6	41,8	23	45,7	44
20	0.78	1.5842	64,1	99,2	73	44,8	72	70,7	98,6	43,4	25,4	50,7	43
20	0.75	1.5312	64,1	98,8	71,4	48,8	74,3	70,7	98,8	42,2	24,6	45,7	43,3
20	0.72	1.193	59,4	100	62,6	32,6	66	65,7	99,4	38,4	23,2	47,7	44

TABLE 6.20 – Benchmark problèmes de calage : comparaison des essaims particulaires (pourcentage de réussite)

				10 ans					58 ans				
pop	F	CR	total	u0	u10	u20	var20	var30	u0	u10	u20	var20	var30
30	0,5	0,9	**88,9**	99	99,8	92,8	**94,7**	**92,3**	**100**	**80**	61,6	**61**	**61**
50	0,5	0,9	85	99,6	99,4	89	91,7	89	99,4	70,8	51,8	60,3	58,7
20	0,5	0,1	84,4	91,8	99,4	97,4	91,7	91,3	84,8	68	**65**	59	55,7
100	0,5	1	82,4	99,8	**100**	88,8	91	90,3	98,8	66,4	40,6	54	54,7
50	0,5	1	81,7	99,6	99,2	87,6	90	89,3	99	64,4	40,6	57,7	60
20	0,5	0,5	81,2	95	100	84,6	84	86,7	92	63	52,4	61	56,7
30	0,5	0,1	80,1	91,4	99,4	97,6	93	91,3	79,2	58,8	54,2	51,7	48
20	0,5	0,9	79,8	88,6	99,8	90,8	86	85,7	84,4	68,6	46,6	54	51,3
100	0,5	0,9	78,5	99,8	99,6	85,6	87,3	88	99,2	55,4	31,6	52,7	55,7
20	1	0,1	78,5	90,8	100	**98,8**	92,3	93	69,8	61,2	50,4	49	49,7
30	0,5	0,5	76,5	99,8	97,6	78	85,7	84,3	98,6	51,8	33,2	52	52,3
50	0,5	0,1	76	89,6	99,8	92	88	85,3	73,2	54,4	47,2	47,7	39,3
200	0,5	1	75,5	**100**	100	82,6	86,7	84,3	98	43,8	28,8	48,7	45,7
30	1	0,1	75,1	91	99,4	95,6	90,7	90,3	68,6	54,6	41,4	44	37,7
30	1	0,9	71,6	97,2	99	73	80,7	79	96,8	42,6	20,8	47	47,3
50	1	0,1	71,2	89	99,4	89,6	88	85,7	64	46,4	39	38,3	35,3
50	0,5	0,5	71,1	99,8	89,6	63	78	75,7	98	45,2	30,8	50	51,3
200	0,5	0,9	70,9	99,8	96	68,2	82,7	76	98,2	40,6	22,4	46,7	46,7
100	0,5	0,1	70,4	90	98,4	81,8	81,3	86	65,2	48,4	38,8	40	39
20	1	0,9	68,6	76,6	96	77,2	73,3	79	70,2	51	40,4	47,7	48
30	0,5	1	67,3	89,6	80,6	58,4	74,7	74,3	87,2	51,2	36,8	49,3	55,7
50	1	1	66,7	99,6	84,4	58,2	75	72	99,4	38	20,8	46,3	49,7
20	1	0,5	64,9	99,6	74,8	49,8	72,7	69	98,4	44,4	22,6	50	48,3
100	1	0,1	64,8	88,6	94,8	76,2	79,3	82	60,6	40	28,6	30,7	27,3
30	1	1	64,5	91,4	73,4	51,2	73,3	72	91	49,4	30,8	51	49,7
100	0,5	0,5	63,2	97,6	73	46,4	67,7	72,3	96,6	39	26,8	48	48,3
50	1	0,9	60,8	99,6	70,6	39,8	65	67,3	99	36,8	19,2	47,7	44
30	1	0,5	60,4	100	64,4	38,2	64,7	63	98,8	38	23,2	52	46,7
200	0,5	0,1	60,2	88,8	85	63,2	70,3	72	59,2	35,6	29,2	26,7	29,3
100	1	1	57,3	99,8	58	35	64,3	58,7	99,8	32,2	19	46,7	46
50	1	0,5	57,2	98,4	53,6	34	59	59	96	34	27	46,3	49,7
200	0,5	0,5	56,7	96,4	58,4	33,2	58,7	65,3	93,6	33,6	25,2	45,7	45
100	1	0,9	54,2	100	47,4	30,4	57,3	56	99	31	17,6	43,3	43,7
200	1	0,1	52,8	83,6	77	52,4	68	66,7	48,2	31,4	24,4	25,3	24,3
100	1	0,5	51,7	94,8	53,4	29,6	58,7	59,3	87,4	29	16,2	40,3	43
200	1	1	51,1	96,8	48	26,4	52,3	56,7	95	24,6	15,6	40	44,7
200	1	0,9	50,2	95,4	47	21,2	55,7	56,3	93,6	27,8	16,4	42,7	41,7
20	0,5	1	44,3	72,8	46,4	27,2	46,7	48,3	59,8	37,4	22,2	34,3	29,3
200	1	0,5	43,5	83,6	44,2	24	52	45,3	65,2	28	16,2	31	33,3
20	1	1	43,1	56	58,8	33	45	45,7	41,6	44,2	25,2	26,7	37,3

TABLE 6.21 – Benchmark problèmes de calage : comparaison des algorithmes d'évolution différentielle (pourcentage de réussite)

		10 ans					58 ans				
pop	total	u0	u10	u20	var20	var30	u0	u10	u20	var20	var30
20	**65,7**	74	**92,6**	69,4	61,7	59,7	75,4	**51,8**	**31,2**	**42,3**	43,7
30	65,5	81	92,4	**73**	**67**	**65,7**	74,4	46,6	25,4	40,3	43
50	65,4	82,8	89	68,8	66	61	**79,8**	45,6	26,6	47	44,3
100	62,2	80,2	85,6	65,4	61,3	63	77,2	40,2	24,8	41	**45,3**
200	56,4	**83,8**	70	60	58,3	63,7	74,2	28,4	22,2	38,7	39,7

TABLE 6.22 – Benchmark problèmes de calage : comparaison des algorithmes des loups gris (pourcentage de réussite)

Colonie d'abeille artificielle Nous testons deux types de colonies d'abeilles artificielles, l'originale publiée dans Karaboga and Basturk (2007) (type A) et notre version modifiée (type B). Les résultats sont présentés dans le tableau 6.25 Dans les deux cas nous testons le nombre d'individus dans la population, donc le nombre de source, et le nombre d'abeilles spectatrices.

On constate que cet algorithme semble particulièrement sensible au taux de variation notamment sur les simulations de 58 ans.

6.6.3 Comparaison des algorithmes entre eux

Le tableau 6.26 montre les résultats de différents algorithmes sur des problèmes de calage avec différents taux de variation dans les efforts au cours du temps ainsi que différentes durées de simulation et différents niveaux de connaissances. Ils sont comparés en utilisant une méthode de Friedman par rang Hodges and Lehmann (2012). Il nous permet de constater que quand le nombre de paramètres est faible, c'est-à-dire ici les cas où les efforts sont parfaitement connus, l'essaim particulaire de 2007 et l'évolution différentielle sont très proches dans la plupart des cas. Nous nous sommes donc intéressé à leur temps de convergence. Le tableau 6.28 montre que dans la plupart des cas il est très difficile de trancher entre les deux algorithmes.

En revanche, dès l'introduction d'incertitudes dans les valeurs d'efforts de pêche, ils trouvent leurs limites au profit de la colonie d'abeilles artificielles ou de la recherche à voisinage variable générale.

Les problèmes avec 58 ans de simulations et de trop fortes variabilités ne présente cependant pas d'algorithmes dont les performances sont significativement meilleures que les autres.

Le tableau 6.27 montre les moyennes et écart-types de l'ensemble de ces tests.

					10 ans					58 ans				
pop	CR	fmin	fmax	total	u0	u10	u20	var20	var30	u0	u10	u20	var20	var30
20	0,9	0,25	1,5	**68**	**84,8**	93,8	**78,6**	**73,3**	**75,3**	78,6	45,4	27	39,7	43,7
20	0,7	0	2	67,5	84,2	92,6	73,8	68,7	66,3	75,6	47	**32**	45,7	45,3
20	0,7	0,25	1,5	67,2	81,4	92,6	74	69,7	66,7	76,8	49,2	29,4	45	42,7
30	0,5	0	2	67,1	83,2	90,8	75,2	68,3	72	77,8	43,2	32,4	**47,7**	41,7
30	0,5	0,25	1,5	67	83,4	92,4	76,4	70,3	68	76,4	45,4	28,2	47	42,3
30	0,9	0,25	1,5	66,7	84,2	89,4	73,8	68,3	66	78,6	44,6	29,6	43,7	44,3
20	0,5	0,25	1,5	66,7	80,8	**94,2**	74,4	71,7	66,7	76,2	46,4	28,4	42	41,3
30	0,7	0,25	1,5	66,4	82,4	88,2	73	66,3	64,7	77,8	48	29	43,7	**47,7**
20	0,1	0,25	1,5	66,3	80	87,6	71,4	61,7	61,3	78,4	**50,4**	29,8	42	44
30	0,1	0,25	1,5	66,3	83,6	90	72,8	69	63,7	75,6	46,8	29	43,3	44,3
20	0,5	0	2	66,1	80,8	90,4	73,2	69	61,7	74,6	50,2	27,6	44,7	41,7
30	0,7	0	2	66	83,4	87	70,8	68	64	77,4	45,6	32	47	45,3
50	0,9	0,25	1,5	65,9	86,6	86	74,6	71,3	71,7	**80,6**	42	25,6	46	41,7
50	0,5	0	2	65,5	83	89,4	75	72,7	71,7	75	42,8	28	40,7	44
20	0,9	0	2	65,5	83,2	89,4	75,6	72,3	65,7	76,8	41,6	26,2	36,3	46,7
50	0,9	0	2	65,2	81,4	87,2	71	69,3	67,3	79,6	45,6	26,4	42,7	44,7
30	0,9	0	2	64,9	81,6	88	72,4	65	68,3	77,4	43,8	26	44,3	44
50	0,5	0,25	1,5	64,8	82	88	72,2	63,3	69,3	79	39,6	28	43,3	40
50	0,7	0,25	1,5	64,2	82,8	89,4	69,8	66,7	66,3	76	38,2	29	41,7	46,3
50	0,7	0	2	63,3	81,2	83,8	73,4	64,3	64	75,6	40,2	25,8	41,3	43,3
50	0,1	0,25	1,5	62,8	79,2	87,2	67,8	66,3	63,7	73,4	39,8	29,4	40	42,3
100	0,7	0	2	61,3	83,8	80,6	64,4	64,7	64	75,6	37	26,2	42,7	42,7
100	0,9	0,25	1,5	61,1	84,6	83,2	64,2	67	67	77,2	34,4	23,2	39,7	40,3
100	0,7	0,25	1,5	60,7	85	79,4	66,6	68,3	62,3	77,4	33	23	34,3	39,7
100	0,1	0,25	1,5	60,5	81,4	80,6	65,2	56,3	66,7	76,6	38	21,2	39,3	36
100	0,5	0,25	1,5	60,5	83,8	81	62,8	63,3	63	79,6	33,4	22,6	42,3	45,7
100	0,5	0	2	60,1	84	82,6	63,4	66,3	66	75,2	34,8	20,4	38	40,3
100	0,9	0	2	59,7	84,2	80,6	65	69	65,7	73,4	34,6	20,4	40,7	35
200	0,7	0,25	1,5	54,4	86,8	63,2	51,8	63,7	57,3	77,2	29	18,2	36	40,7
200	0,5	0,25	1,5	53,9	82,2	63,2	54,4	53,7	58,7	76,2	27,8	19,6	39,3	37
200	0,9	0	2	53,3	84,2	65,2	53,4	57,7	63	74,2	25,4	17,2	40	36
200	0,1	0,25	1,5	53	83,4	64,2	47,8	55,7	55	75,8	28,4	18,6	38,7	36,7
200	0,5	0	2	52,9	84,8	61,4	54,2	61	55,7	76	23	18	35,3	35,3
200	0,7	0	2	52,9	82,8	64	50,8	61	53,7	76,6	26,4	16,6	37	40,3
200	0,9	0,25	1,5	52,4	81,6	61,6	52,6	56,7	55	76,4	25,8	16,6	39,7	40,7

TABLE 6.23 – Benchmark problèmes de calage : comparaison des algorithmes des loups gris améliorés (pourcentage de réussite)

			10 ans					58 ans				
pop	b	total	u0	u10	u20	var20	var30	u0	u10	u20	var20	var30
200	5	48	40,2	**56**	**50,4**	32	**25,3**	**32,2**	**59,8**	49,6	21	20,3
200	2	43,7	35,8	57,4	48,8	29,3	27,3	20	52,4	47,6	15	**16,3**
100	5	42,9	37,4	52,6	44	27,3	26	22,4	53,2	47,6	19	15,3
200	1	41,1	31,4	56,2	43,6	31,7	25	23,8	48,2	43,2	17,3	14
200	0,3	38,8	32,6	48,8	38,2	26,3	26	18,6	51,2	43,6	18	18,3
50	5	38,7	31,2	47,6	41,6	20,7	24,7	24,8	45,6	41,4	19,7	13
200	0,2	38,4	34,2	48,2	38,6	**26,3**	26,7	16,4	50,6	42,6	16,3	13,7
100	2	38,2	26,8	47,6	44,6	26,3	19	18,8	53,2	38,2	17,7	15,3
200	0,5	38,1	33,6	49,4	37	31,7	22,3	20,6	46	42	**16**	14,7
30	5	37,9	33,6	45	39,4	22,7	25,7	20,8	48	40,8	19	14,3
200	0,1	37,6	30,6	51,2	39	25,7	25	17,4	47	40,2	15,3	16
50	2	36,9	31,8	44,8	39,6	29,3	23,7	20,2	45	40,2	17	11,7
200	0,15	36,4	27,4	48	38,8	28,7	22	18,2	46,2	**39,6**	13,3	10
20	5	35,9	29,8	45,2	38,2	25,7	22	21,6	43,2	37,6	14,7	14
100	1	35,7	30	45,4	40,6	24	23	18,2	42,4	37,8	12,7	14,3
100	0,2	35,5	28,4	44,2	36,4	20,3	20,7	18,4	46,8	38,8	14,3	16
100	0,3	35,3	27,8	46	38,8	24,3	25	16,4	43,8	38,8	17	10,7
100	0,1	34,9	31,4	45,8	36	24,3	25	15,2	44,4	36,4	13	13,3
100	0,15	34,7	32,8	43,8	36	23,3	28	16,6	43,4	35,8	15	10,3
100	0,5	34	28,4	42	35,4	25	21	20,4	41,4	36,6	13,7	14,3
30	2	33,8	30,8	39,4	36,8	24	21,7	17,6	43	35,4	12,7	10,3
50	0,5	33,2	32,8	42,8	32,4	27,7	22,7	16,2	39,6	35,4	12	11
20	2	33	28,4	41,4	34	23,3	21,3	21,4	41,8	31	13	13,3
50	0,15	32,8	30,8	41,2	31,4	23,7	18,7	18	40	35,4	12	12
50	1	32,8	28,6	37,4	36,4	18,7	20,3	19,8	39	35,8	12,3	12,3
50	0,3	32	29,2	39,6	36,6	25	18	16,2	37,8	32,6	14	9,7
30	1	31,8	28,4	38,6	35	20,3	22	17,2	39,2	32,6	11,3	11,7
50	0,1	31,1	27,8	37,8	33	17	18,3	16	40,6	31,6	11,3	10,7
30	0,2	30,5	25,4	39,8	35	17	23,3	16,6	35,4	31	10,3	8,3
50	0,2	30,4	27,6	39	32,2	22,3	16,3	15	36,2	32,2	8,7	12,3
30	0,3	30,3	28,4	36,6	32,4	18,3	16,3	17,2	35,6	31,4	8	10
30	0,1	30,1	29	37	34,8	23,7	18,7	16,4	31,6	32	13,7	13,7
30	0,5	30,1	26,8	39,8	32,6	21,3	19	13,8	37,2	30,4	8	13
30	0,15	29	24	35,8	31,4	17,3	17,3	15,6	35,8	31,2	10,7	10,3
20	0,2	29	28,8	36,2	31,2	22,3	18,3	14,8	33,6	29,6	9,3	9,7
20	0,1	28,9	31	33,8	24,8	20,7	19	16,2	35,4	32,2	13,7	9,7
20	0,5	28,7	23,8	35,6	32,2	21,7	18,3	14	35	31,6	9,3	10,7
20	1	28,5	26	32,4	30,2	19,7	17,7	16,4	36	30	6,7	10
20	0,15	28,3	27,6	35	31,6	21	18,7	12,4	32,2	30,8	13,3	8,7
20	0,3	27,8	21,6	35,2	29	16,3	18,7	15,6	34,4	30,8	9,3	9,3

TABLE 6.24 – Benchmark problèmes de calage : comparaison des algorithmes des baleines (pourcentage de réussite)

				10ans					58ans				
type	pop	nSpec	total	u0	u10	u20	var20	var30	u0	u10	u20	var20	var30
B	20	20	**87,7**	98,8	**100**	94	94	**94**	**78,8**	86,2	68,6	47,3	49,7
B	50	50	87,4	97,6	99,8	94	93	93	73,6	**86,8**	**72,8**	**50**	**50,7**
B	200	200	87	**99,2**	100	**94,4**	**95,7**	93,7	78,4	80,6	69,2	46,3	45,3
B	30	30	85,7	96,2	100	93,4	92,3	92	69	84,8	70,8	46	45,3
B	100	100	85	95,4	100	94,2	91,2	91,2	72,8	80	68,2	45,3	45,3
A	50	10	76,9	79,4	99,2	89,2	78,7	78	46,2	78,8	68,6	32,7	32,3
A	50	5	76,8	78,8	99,4	83,8	75	72,3	47,2	81,4	70	35,3	37
A	50	20	76,2	78,8	99,8	85,6	79,3	72	48	78,4	66,4	33,3	26,3
A	100	5	76,1	84,6	99	87	80,7	79,3	54,8	73,4	57,8	30,7	30
A	100	10	75,7	88	99,8	88	84	81,7	52	70,2	56,4	31	24,7
A	100	20	75,1	88,6	99,8	86,2	82,3	79,7	55,8	67,8	52,6	30	25
A	30	20	73,9	67,4	99,8	85,2	73	70,7	42	80	69	33,7	27,7
A	50	35	73,8	78,6	99,8	85,8	75,7	76,7	42,2	75,2	61	27,7	24,7
A	100	35	73,7	89,8	100	85,8	83,7	78,7	56,8	62,8	47	29,7	23
A	30	5	73,7	73,4	98,6	84,4	76	68,3	36,6	82,2	67	34	26
A	30	10	73,6	69,8	99,6	84,6	76,3	68,3	39,6	79,4	68,8	31	26,7
A	50	50	73,4	78,2	99,2	86,4	76,3	74,7	46,8	69,6	60	28,7	24,7
A	100	50	72,8	87,6	99,6	86,6	81,3	79,3	57,8	60,2	45,2	26,7	30
A	20	5	72	67,8	99,2	82,4	69	71	33	79,8	70	32	28,7
A	30	35	71,6	65,8	99,2	82,4	68,3	71,7	39,2	79	63,8	26,3	25
A	20	10	71,4	65,6	99	84,2	70,3	70,3	32,4	78,8	68,2	31,3	23,3
A	200	10	71,2	92,2	99,8	85,6	83,7	80,7	66,8	50,6	32	28,7	29,3
A	200	20	71,2	90,4	99,4	88,4	84	82	68	45,6	35,4	27	25,7
A	30	50	70,7	63,2	99,6	86,4	73	69	35,4	76,6	63,2	27,3	21
A	20	20	70,3	58,6	99	83,2	72	65	35,6	77,6	68	27,3	26,3
A	20	35	69,8	62	98,4	81,8	72	63	31,8	79	65,8	28,7	22,7
A	200	35	69,1	93	99,6	84,8	83,3	79,7	62	43,2	31,8	23,3	24
A	200	50	69	93,8	99,4	83,8	84,3	79,7	63,6	43,6	30	27,7	23
A	200	5	68,9	91,8	100	84,4	84,7	78,7	57,2	47,2	33	22	21,7
A	100	100	67,8	88,6	99,2	83,8	82,3	73,3	55,8	48,2	31,4	21,7	22,7
A	20	50	67,1	55,4	98,2	81,2	68	63,7	29,6	74,2	64,2	21,3	24,3
A	50	100	66,3	77	99	81,6	75,7	66,3	45,4	55,6	39,4	22,7	18,3
A	200	100	65,6	92,2	99	82,2	82	75,7	66,4	34,2	19,4	24,7	25,3
A	30	100	62,7	60,6	98,4	82	70,3	60,3	34,4	56,2	44,4	14,7	14,7
A	200	200	62,4	92,4	98,6	78,2	77,7	75	61,2	30	13,8	22	21,7
A	100	200	61,7	83,4	99	80,4	73,7	71	53	33	21,6	16	17
A	20	100	60,7	55,2	97,4	78,2	61,7	59,7	25,2	59,4	49	23	16
A	50	200	58,5	71,2	98,8	79,4	68,7	63	44	36	21,6	16	12,3
A	200	500	54,4	87,2	87,6	64,2	65	58	57,4	20,6	9,2	15	19
A	30	200	53,7	61,4	97,4	76,8	62,3	57,3	30	35,2	21,2	11,7	9,7
A	100	500	51,4	83	89,6	67,6	64,3	55,7	45,4	15,4	7,4	10	13
A	20	200	48	43,8	96,4	77,8	61,7	51,3	21,2	28,6	20,2	8	10,3
A	50	500	47,5	69,8	91,2	66,2	64,3	51,3	38,8	13	5,8	9,7	8,3
A	30	500	38,8	47	84,8	62,2	48,7	41	24	10	4,6	4,7	7,3
A	20	500	33,9	42	78,6	56,8	45	36	17,4	5,4	3	6	3

TABLE 6.25 – Benchmark problèmes de calage : comparaison des colonies d'abeilles artificielles (pourcentage de réussite)

t	var	u	ABC	PSO	GW	IGW	Baleines	RVVG	DE	P-value
10	0	0	5,65	**2,2**	3,4	3,85	6,45	3,7	2,75	3,11E-15
10	5	0	4,8	1,8	4,2	4,4	5,5	5,55	**1,75**	1,11E-16
10	10	0	4,85	**1,6**	3,8	4,2	6,2	5,5	1,85	1,11E-16
10	20	0	4,9	1,85	3,7	3,55	6,45	6,05	**1,5**	1,11E-16
10	30	0	5,3	1,65	3,65	3,95	6,7	5,4	**1,35**	1,11E-16
58	0	0	6,05	**1,85**	4,15	3,95	5,85	3,9	2,25	1,11E-16
58	5	0	4,4	**1,475**	3,85	5,1	5,9	5,75	1,525	1,11E-16
58	10	0	5	1,55	3,75	4,25	6,25	5,75	**1,45**	1,11E-16
58	20	0	5,65	1,85	3,5	3,5	6,2	5,85	**1,45**	1,11E-16
58	30	0	5,8	**1,5**	3,3	4,25	6	5,65	**1,5**	1,11E-16
10	0	10	4,45	2,15	4,35	4,4	6,9	**1**	4,75	1,11E-16
10	5	10	4,7	2,1	4,25	4,15	6,8	**1**	5	1,11E-16
10	10	10	4,3	2,05	3,9	4,1	6,85	**1,3**	5,5	1,11E-16
10	20	10	3,3	**1,5**	4,75	3,8	6,3	3,95	4,4	2,86E-13
10	30	10	3,15	**1,25**	4,95	3,8	6,75	4	4,1	1,11E-16
58	0	10	2	5,8	4,1	5,05	5,5	**1,25**	4,3	1,11E-16
58	5	10	2,25	5,65	4,2	4,65	5,45	**1**	4,8	1,11E-16
58	10	10	2,1	5,6	3,85	4,35	6,1	**1,1**	4,9	1,11E-16
58	20	10	3,85	4,75	4,5	**2,85**	4,35	3,85	3,85	1,22E-01
58	30	10	3,4	5,9	**2,9**	3,45	4,75	3,4	4,2	3,40E-05
10	0	20	5	2	3,55	4,35	6,8	**1,3**	5	1,11E-16
10	5	20	4,65	2,05	3,75	4,1	6,85	**1,05**	5,55	1,11E-16
10	10	20	4,9	1,95	3,35	4,4	6,95	**1,2**	5,25	1,11E-16
10	20	20	3,45	**1**	4,5	3,55	5,85	5,3	4,35	1,11E-16
10	30	20	3,7	**1,15**	4,5	4,05	5,7	4,65	4,25	7,93E-12
58	0	20	2,4	6	4,35	4,35	5,25	**1,1**	4,55	1,11E-16
58	5	20	2	5,7	4,25	4,1	5,55	**1,05**	5,35	1,11E-16
58	10	20	2,15	6,05	3,45	4	5,95	**1,1**	5,3	1,11E-16
58	20	20	4,35	4,7	4,05	3,5	**3,15**	3,75	4,5	2,27E-01
58	30	20	3,5	4,15	4,2	3,7	**3,3**	4,65	4,5	3,52E-01

TABLE 6.26 – Résultats de différents algorithmes sur différents problèmes de calage, comparaison par test de friedman

t	var	u		ABC	SPSO	GW	IGW	Baleines	RVVG	DE
10	0	0	moy	2,53E-03	6,68E-08	3,11E+00	3,14E-06	8,44E+01	5,70E-03	4,14E-03
			std	5,11E-03	2,71E-08	1,36E+01	6,43E-06	3,67E+02	1,57E-02	1,80E-02
	5	0	moy	2,53E-02	9,29E-04	1,82E-02	3,47E-02	1,52E-01	1,19E+00	1,19E-03
			std	3,27E-02	3,81E-03	3,64E-02	5,62E-02	2,24E-01	4,11E+00	4,71E-03
	10	0	moy	3,42E-02	2,20E-04	5,43E+00	6,59E-02	3,46E+00	1,48E+00	5,65E-04
			std	5,15E-02	7,35E-04	2,33E+01	2,34E-01	6,26E+00	3,33E+00	2,26E-03
	20	0	moy	4,41E-02	3,62E-02	4,94E-03	4,03E-03	7,03E+00	2,03E+00	7,40E-08
			std	6,91E-02	1,09E-01	1,06E-02	9,47E-03	1,17E+01	3,19E+00	1,94E-08
	30	0	moy	8,93E-02	1,17E-07	6,25E-04	5,03E-03	1,22E+01	5,82E-01	6,95E-08
			std	1,05E-01	2,05E-07	8,97E-04	1,56E-02	2,87E+01	9,47E-01	2,30E-08
58	0	0	moy	8,54E-03	6,11E-08	2,16E-04	8,54E-06	5,18E-01	4,60E-02	8,50E-07
			std	1,86E-02	2,85E-08	5,89E-04	1,09E-05	1,25E+00	1,15E-01	1,63E-06
	5	0	moy	1,62E-01	4,37E-06	1,56E-01	1,11E+00	3,49E+00	3,14E+00	5,81E-06
			std	1,57E-01	1,88E-05	3,74E-01	2,54E+00	5,51E+00	4,48E+00	2,50E-05
	10	0	moy	4,22E-01	5,97E-08	1,51E-01	7,39E-02	1,79E+01	4,88E+00	6,05E-08
			std	7,72E-01	2,93E-08	3,79E-01	1,06E-01	4,54E+01	7,11E+00	2,46E-08
	20	0	moy	9,63E-01	6,45E+00	6,46E+00	1,82E-03	2,91E+01	2,27E+00	6,86E-08
			std	2,54E+00	2,81E+01	2,82E+01	5,30E-03	9,42E+01	3,16E+00	2,54E-08
	30	0	moy	1,10E+00	6,24E-08	1,83E+00	1,32E-01	1,50E+01	4,60E+00	7,09E-08
			std	2,38E+00	2,48E-08	7,98E+00	3,88E-01	4,10E+01	1,59E+01	1,98E-08
10	0	10	moy	4,34E-04	8,43E-08	7,63E-03	3,86E-04	2,23E+00	3,65E-10	5,20E-02
			std	4,88E-04	1,69E-08	2,19E-02	9,63E-04	3,47E+00	5,50E-10	1,22E-01
	5	10	moy	4,25E-04	7,85E-08	4,33E+01	8,82E-04	4,00E+00	4,22E-09	1,21E-01
			std	4,56E-04	1,48E-08	1,88E+02	2,51E-03	5,44E+00	1,57E-08	1,91E-01
	10	10	moy	3,75E-04	7,87E-08	5,02E-03	3,93E-03	4,65E+00	1,49E+00	1,55E-01
			std	5,43E-04	2,02E-08	1,91E-02	1,07E-02	9,49E+00	6,50E+00	2,25E-01
	20	10	moy	7,13E-02	1,18E+01	1,30E+00	2,61E+01	6,43E+01	5,99E+00	6,14E-01
			std	1,42E-01	5,15E+01	3,10E+00	1,14E+02	2,19E+02	1,32E+01	1,23E+00
	30	10	moy	3,46E-02	8,41E-08	2,05E+00	9,61E-02	1,28E+02	1,58E+01	2,68E-01
			std	6,95E-02	1,57E-08	4,55E+00	1,48E-01	3,84E+02	4,99E+01	3,16E-01
58	0	10	moy	1,12E-03	9,51E+01	1,38E+00	2,71E+00	2,09E+01	5,76E-03	4,14E+00
			std	1,26E-03	1,95E+02	3,61E+00	6,58E+00	4,64E+01	1,89E-02	8,66E+00
	5	10	moy	3,96E-03	6,79E+01	6,33E-01	1,19E+00	3,68E+01	4,14E-05	2,29E+01
			std	6,25E-03	1,43E+02	1,04E+00	1,56E+00	6,42E+01	1,80E-04	6,20E+01
	10	10	moy	3,07E-03	1,08E+02	6,84E-01	4,79E-01	4,50E+01	4,77E-04	3,33E+01
			std	8,21E-03	3,39E+02	1,57E+00	6,16E-01	8,36E+01	2,05E-03	9,54E+01
	20	10	moy	1,35E+02	3,77E+02	1,18E+02	2,08E+01	3,62E+02	2,01E+02	9,38E+01
			std	3,02E+02	7,69E+02	1,66E+02	3,66E+01	6,65E+02	5,25E+02	2,07E+02
	30	10	moy	4,55E+01	9,45E+02	1,11E+02	6,53E+01	8,78E+02	1,64E+02	1,22E+02
			std	7,02E+01	1,46E+03	2,42E+02	1,24E+02	2,42E+03	4,58E+02	2,00E+02
10	0	20	moy	1,07E-03	8,01E-08	2,69E-03	4,41E-02	3,51E+01	7,03E-03	4,10E-01
			std	1,28E-03	1,23E-08	1,07E-02	1,61E-01	1,40E+02	3,06E-02	1,35E+00
	5	20	moy	7,35E-04	7,56E-08	7,04E-02	1,08E-04	3,71E+00	7,94E-09	2,34E-01
			std	9,27E-04	1,28E-08	3,06E-01	1,71E-04	9,43E+00	3,10E-08	2,91E-01
	10	20	moy	8,39E-04	7,98E-08	8,09E-04	9,64E-03	4,92E+00	3,84E-05	1,72E-01
			std	9,09E-04	1,33E-08	3,38E-03	4,07E-02	1,55E+01	1,67E-04	3,11E-01
	20	20	moy	4,01E+00	8,05E-08	2,07E+01	1,95E+00	7,13E+01	1,63E+02	4,66E+00
			std	1,45E+01	1,49E-08	4,38E+01	2,87E+00	1,12E+02	3,38E+02	7,96E+00
	30	20	moy	4,61E+00	8,32E-08	8,20E+01	1,82E+00	1,06E+02	9,92E+01	3,57E+00
			std	9,92E+00	9,97E-09	2,96E+02	2,21E+00	2,23E+02	3,33E+02	8,64E+00
58	0	20	moy	3,00E-03	1,87E+02	5,98E+00	5,31E+00	8,87E+00	3,94E-05	2,56E+01
			std	3,16E-03	2,31E+02	2,15E+01	1,49E+01	2,05E+01	9,75E-05	5,83E+01
	5	20	moy	1,83E-03	1,26E+02	2,83E+00	2,16E+00	2,26E+01	1,69E-04	5,34E+01
			std	4,05E-03	2,39E+02	7,66E+00	4,72E+00	4,39E+01	6,46E-04	1,30E+02
	10	20	moy	8,66E-03	1,54E+02	6,05E-01	2,92E-01	1,78E+01	2,89E-03	2,35E+01
			std	1,70E-02	2,81E+02	1,26E+00	2,90E-01	2,27E+01	1,17E-02	5,15E+01
	20	20	moy	1,68E+03	1,30E+03	7,22E+02	7,17E+02	2,36E+04	6,00E+02	1,42E+03
			std	2,52E+03	1,64E+03	1,47E+03	1,67E+03	1,01E+05	9,22E+02	2,57E+03
	30	20	moy	4,64E+02	1,77E+03	7,92E+02	3,01E+02	5,20E+02	1,23E+03	2,50E+03
			std	7,87E+02	3,89E+03	1,31E+03	3,55E+02	1,28E+03	1,82E+03	5,00E+03

TABLE 6.27 – Résultats de différents algorithmes sur différents problèmes de calage, moyennes et écart types

			PSO			DE		
t	var	u	moyenne	ecart-type	Temps	moyenne	ecart-type	Temps
10	0	0	**6,68E-08**	**2,71E-08**	1,36E-02	4,14E-03	1,80E-02	**7,96E-03**
10	5	0	**9,29E-04**	**3,81E-03**	2,45E-02	1,19E-03	4,71E-03	**1,55E-02**
10	10	0	**2,20E-04**	**7,35E-04**	6,72E-02	5,65E-04	2,26E-03	**1,03E-02**
10	20	0	3,62E-02	1,09E-01	4,92E-02	**7,40E-08**	**1,94E-08**	**1,07E-02**
10	30	0	1,17E-07	2,05E-07	3,03E-02	**6,94E-08**	**2,30E-08**	**9,43E-03**
58	0	0	**6,10E-08**	**2,85E-08**	3,23E-02	8,50E-07	1,63E-06	**1,77E-02**
58	5	0	4,37E-06	1,88E-05	4,42E-02	**5,81E-06**	**2,50E-05**	**3,61E-02**
58	10	0	5,97E-08	2,93E-08	3,74E-02	**6,05E-08**	**2,46E-08**	**1,93E-02**
58	20	0	6,45E+00	2,81E+01	2,72E-02	**6,86E-08**	**2,54E-08**	**1,32E-02**
58	30	0	**6,24E-08**	2,48E-08	2,86E-02	7,09E-08	**1,98E-08**	**1,52E-02**
10	20	10	**1,18E+01**	5,15E+01	**4,49E-02**	6,14E-01	**1,23E+00**	5,17E-01
10	30	10	**8,41E-08**	**1,57E-08**	**4,09E-02**	2,68E-01	3,16E-01	4,57E-01
10	20	20	**8,05E-08**	**1,49E-08**	**3,61E-02**	4,66E+00	7,96E+00	2,33E-01
10	30	20	**8,32E-08**	**1,00E-08**	**3,48E-02**	3,57E+00	8,64E+00	4,76E-01

TABLE 6.28 – Comparaison des temps de convergence de DE et PSO sur les problèmes sur lesquels ils sont en compétitions

BIBLIOGRAPHIE

Aarts, E. H. and Van Laarhoven, P. J. (1985). Statistical cooling : A general approach to combinatorial optimization problems. *Philips J. Res.*, 40(4) :193–226.

Abshouri, A. A. and Bakhtiary, A. (2012). A new clustering method based on firefly and khm. *Journal of Communication and Computer*, 9(4) :387–391.

Ackley, D. H. (1987). The model. In *A Connectionist Machine for Genetic Hillclimbing*, pages 29–70. Springer.

Aguilera, S. (2018). Measuring squid fishery governance efficacy : A social-ecological system analysis. *International Journal of the Commons*, 12(2).

Ahmadi, M. H., Ahmadi, M. A., Bayat, R., Ashouri, M., and Feidt, M. (2015). Thermo-economic optimization of stirling heat pump by using non-dominated sorting genetic algorithm. *Energy Conversion and Management*, 91 :315–322.

Akinbulire, T., Falcon, R., Abielmona, R., and Schwartz, H. (2018). Responding to illegal, unreported and unregulated fishing with evolutionary multi-objective optimization. In *2018 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*, pages 1–6. IEEE.

Akinbulire, T., Schwartz, H., Falcon, R., and Abielmona, R. (2017). A reinforcement learning approach to tackle illegal, unreported and unregulated fishing. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE.

Aldrich, J. (1997). Ra fisher and the making of maximum likelihood 1912-1922. *Statistical science*, 12(3) :162–176.

Althoen, S. C. and Mclaughlin, R. (1987). Gauss-jordan reduction : A brief history. *The American mathematical monthly*, 94(2) :130–142.

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3) :175–185.

Anderson, E. A. (2002). Calibration of conceptual hydrologic models for use in river forecasting. *Office of Hydrologic Development, US National Weather Service, Silver Spring, MD*.

Angel, R., Caudle, W., Noonan, R., and Whinston, A. (1972). Computer-assisted school bus scheduling. *Management Science*, 18(6) :B–279.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3) :235–256.

August, M. and Hernández-Lobato, J. M. (2018). Taking gradients through experiments : Lstms and memory proximal policy optimization for black-box quantum control. In *International Conference on High Performance Computing*, pages 591–613. Springer.

Azadivar, F., Truong, T., Stokesbury, K. D., and Rothschild, B. J. (2002). Simulation based optimization in fishery management. In *Proceedings of the Winter Simulation Conference*, volume 1, pages 525–531. IEEE.

Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., and Kautz, J. (2016). Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv :1611.06256*.

Baker, J. E. (1987). Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, pages 14–21.

Balbi, J. H., Morandini, F., Silvani, X., Filippi, J. B., and Rinieri, F. (2009). A physical model for wildland fires. *Combustion and Flame*, 156(12) :2217–2230.

Barto, A. and Duff, M. (1994). Monte carlo matrix inversion and reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 687–694.

Barzilai, J. and Borwein, J. M. (1988). Two-point step size gradient methods. *IMA journal of numerical analysis*, 8(1) :141–148.

Beasley, D., Bull, D. R., and Martin, R. R. (1993). A sequential niche technique for multimodal function optimization. *Evolutionary computation*, 1(2) :101–125.

Begley, J. and Howell, D. (2004). An overview of gadget, the globally applicable area-disaggregated general ecosystem toolbox. ICES.

Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pages 449–458. PMLR.

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6) :503–515.

Bellman, R. (1956). A problem in the sequential design of experiments. *Sankhyā : The Indian Journal of Statistics (1933-1960)*, 16(3/4) :221–229.

Bellman, R. E. and Dreyfus, S. E. (2015). *Applied dynamic programming*, volume 2050. Princeton university press.

Ben Hadj-Alouane, A. and Bean, J. C. (1997). A genetic algorithm for the multiple-choice integer program. *Operations research*, 45(1) :92–101.

Ben-Tal, A. and Nemirovski, A. (2002). Robust optimization–methodology and applications. *Mathematical programming*, 92(3) :453–480.

Ben-Tal, A., Nemirovski, A., and Roos, C. (2002). Robust solutions of uncertain quadratic and conic-quadratic problems. *SIAM Journal on Optimization*, 13(2) :535–560.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv :1912.06680*.

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. (2002). The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4) :819–840.

Bertsekas, D. (1982). Distributed dynamic programming. *IEEE transactions on Automatic Control*, 27(3) :610–616.

Bertsekas, D. P. (1983). Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1) :107–120.

Beven, K. and Binley, A. (1992). The future of distributed models : model calibration and uncertainty prediction. *Hydrological processes*, 6(3) :279–298.

Beyer, H.-G. and Sendhoff, B. (2007). Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering*, 196(33-34) :3190–3218.

Bhowmik, B. (2010). Dynamic programming. its principles, applications, strengths, and limitations. *Criterion*, 4(7).

Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming*. Springer Science & Business Media.

Bisgambiglia, P.-A., Rossi, J.-L., Franceschini, R., Chatelon, F.-J., Rossi, L., Bisgambiglia, P., and Marcelli, T. (2017). Dimzal : A software tool to compute acceptable safety distance. *Open Journal of Forestry*, 7 :11–33.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152.

Bottou, L. and Bousquet, O. (2011). 13 the tradeoffs of large-scale learning. *Optimization for machine learning*, page 351.

Bousquet, O. and Bottou, L. (2008). The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pages 161–168.

Bouyer, A. and Hatamlou, A. (2018). An efficient hybrid clustering method based on improved cuckoo optimization and modified particle swarm optimization algorithms. *Applied Soft Computing*, 67 :172–182.

Bowyer, A. (1981). Computing dirichlet tessellations. *The computer journal*, 24(2) :162–166.

Brandt, G., Merico, A., Vollan, B., and Schlüter, A. (2012). Human adaptive behavior in common pool resource systems. *PloS one*, 7(12) :e52763.

Bullnheimer, B., Hartl, R. F., and Strauss, C. (1999). An improved ant system algorithm for thevehicle routing problem. *Annals of operations research*, 89 :319–328.

Burdakov, O., Dai, Y.-H., and Huang, N. (2019). Stabilized barzilai-borwein method. *arXiv preprint arXiv :1907.06409*.

Calvez, B. (2007). Le calibrage de modèles à base d'agents pour la simulation de systèmes complexes. *These de doctorat, Université d'Evry Val d'Essonne*.

Cantrell, C. D. (2000). *Modern mathematical methods for physicists and engineers*. Cambridge University Press.

Cauchy, A. et al. (1847). Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847) :536–538.

Cavicchio, D. J. (1970). Adaptive search using simulated evolution. Technical report.

Chaloupka, M. and Balazs, G. (2007). Using bayesian state-space modelling to assess the recovery and harvest potential of the hawaiian green sea turtle stock. *Ecological modelling*, 205(1-2) :93–109.

Chang, P. B., Williams, B. J., Santner, T. J., Notz, W. I., and Bartel, D. L. (1999). Robust optimization of total joint replacements incorporating environmental variables.

Chen, G., Peng, Y., and Zhang, M. (2018). An adaptive clipping approach for proximal policy optimization. *arXiv preprint arXiv :1804.06461*.

Chen, L., Chen, W., and Bletzinger, K.-U. (2019). A gradient descent akin method for inequality constrained optimization. *arXiv preprint arXiv :1902.04040*.

Cholesky, A.-L. (2005). Sur la résolution numérique des systèmes d'équations linéaires. *Bulletin de la Sabix. Société des amis de la Bibliothèque et de l'Histoire de l'École polytechnique*, (39) :81–95.

Chrysafi, A. and Kuparinen, A. (2016). Assessing abundance of populations with limited data : Lessons learned from data-poor fisheries stock assessment. *Environmental Reviews*, 24(1) :25–38.

Chu, P. C. and Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1) :63–86.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv :1412.3555*.

Clerc, M. (2012a). Beyond standard particle swarm optimisation. In *Innovations and Developments of Swarm Intelligence Applications*, pages 1–19. IGI Global.

Clerc, M. (2012b). Standard particle swarm optimisation.

Coello, C. A. C. (1999). Self-adaptive penalties for ga-based optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, pages 573–580. IEEE.

Coget, J.-F., Haag, C., and Bonnefous, A.-M. (2009). Le rôle de l'émotion dans la prise de décision intuitive : zoom sur les réalisateurs-décideurs en période de tournage. *M@ n@ gement*, 12(2) :118–141.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6) :791–812.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2018). Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Dantzig, G. (2016). *Linear programming and extensions*. Princeton university press.

Das, S. and Suganthan, P. N. (2011). Differential evolution : A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1) :4–31.

Dashti, H., Conejo, A. J., Jiang, R., and Wang, J. (2016). Weekly two-stage robust generation scheduling for hydrothermal power systems. *IEEE Transactions on Power Systems*, 31(6) :4554–4564.

Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv preprint arXiv :1406.2572*.

Dayan, P. (1992). The convergence of td (λ) for general λ. *Machine learning*, 8(3-4) :341–362.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* University of Michigan.

De Jong, K. A. and Sarma, J. (1993). Generation gaps revisited. In *Foundations of genetic algorithms*, volume 2, pages 19–28. Elsevier.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4) :311–338.

Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons.

Deb, K. and Goldberg, D. E. (1989). An investigation of niche and species formation in genetic function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 42–50.

Deb, K. and Goyal, M. (1996). A combined genetic adaptive search (geneas) for engineering design. *Computer Science and informatics*, 26 :30–45.

Deb, K., Mohan, M., and Mishra, S. (2003). A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions. *KanGAL report*, 2003002 :1–18.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2) :182–197.

Delaunay, B., Vide, S., Lamémoire, A., and De Georges, V. (1934). Bulletin de l'académie des sciences de l'urss. *Classe des sciences mathématiques et naturelles*, 6 :793–800.

Dick, T. (2015). *Policy Gradient Reinforcement Learning Without Regret*. PhD thesis, University of Alberta.

Domingos, P. and Pazzani, M. (1997). On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, 29(2-3) :103–130.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system : a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1) :53–66.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system : optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1) :29–41.

Dorigo, M. and Stützle, T. (2019). Ant colony optimization : overview and recent advances. *Handbook of metaheuristics*, pages 311–351.

Dozat, T. (2016). Incorporating nesterov momentum into adam.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE.

Eiben, A. E. and Smit, S. K. (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. *Swarm and Evolutionary Computation*, 1(1) :19–31.

Eiben, Á. E., van Kemenade, C. H., and Kok, J. N. (1995). Orgy in the computer : Multi-parent reproduction in genetic algorithms. In *European Conference on Artificial Life*, pages 934–945. Springer.

Eslami, M., Shareef, H., Khajehzadeh, M., and Mohamed, A. (2012). A survey of the state of the art in particle swarm optimization. *Research Journal of Applied Sciences, Engineering and Technology*, 4(9) :1181–1197.

Everitt, B. (1998). The cambridge dictionary of statistics cambridge university press. *Cambridge, UK Google Scholar*.

Ferber, J. (1997). *Les systèmes multi-agents : vers une intelligence collective*. InterEditions.

Fleury, G. (1995). Applications de méthodes stochastiques inspirées du recuit simulé à des problèmes d'ordonnancement. *Automatique-productique informatique industrielle*, 29(4-5) :445–470.

Fliege, J., Vaz, A. I. F., and Vicente, L. N. (2019). Complexity of gradient descent for multiobjective optimization. *Optimization Methods and Software*, 34(5) :949–959.

Fonseca, C. M., Fleming, P. J., et al. (1993). Genetic algorithms for multiobjective optimization : Formulationdiscussion and generalization. In *Icga*, volume 93, pages 416–423. Citeseer.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017a). Noisy networks for exploration. *arXiv preprint arXiv :1706.10295*.

Fortunato, M., Blundell, C., and Vinyals, O. (2017b). Bayesian recurrent neural networks. *arXiv preprint arXiv :1704.02798*.

Fox Jr, W. W. (1970). An exponential surplus-yield model for optimizing exploited fish populations. *Transactions of the American Fisheries Society*, 99(1) :80–88.

Franceschini, R. (2017). *Approche formelle pour la modélisation et la simulation à évènements discrets de systèmes multi-agents*. PhD thesis, Université de Corse Pasquale Paoli.

Froese, R., Thorson, J. T., and Reyes Jr, R. (2014). A bayesian approach for estimating length-weight relationships in fishes. *Journal of Applied Ichthyology*, 30(1) :78–85.

Fu, M. C. (2002). Optimization for simulation : Theory vs. practice. *INFORMS Journal on Computing*, 14(3) :192–215.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR.

Gavin, H. P. (2019). The levenberg-marquardt algorithm for nonlinear least squares curve-fitting problems. *Department of Civil and Environmental Engineering, Duke University*, pages 1–19.

Geem, Z. W., Kim, J. H., and Loganathan, G. V. (2001). A new heuristic optimization algorithm : harmony search. *simulation*, 76(2) :60–68.

Geman, D. (1985). Bayesian image analysis by adaptive annealing. *IEEE Transactions on Geoscience and Remote Sensing*, 1 :269–276.

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget : Continual prediction with lstm.

Gil, J. M., Montes, J. F. A., Alba, E., and Aldana-Montes, J. (2018). Optimizing ontology alignments by using genetic algorithms.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning.

Goldberg, D. E. et al. (1990). A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, 4(4) :445–460.

Goldberg, D. E., Richardson, J., et al. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications : Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ : Lawrence Erlbaum.

Graham, M. (1935). Modern Theory of Exploiting a Fishery, and Application to North Sea Trawling. *ICES Journal of Marine Science*, 10(3) :264–274.

Greensmith, E., Bartlett, P. L., and Baxter, J. (2004). Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5(Nov) :1471–1530.

Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. (2012). A survey of actor-critic reinforcement learning : Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6) :1291–1307.

Gunantara, N. (2018). A review of multi-objective optimization : Methods and its applications. *Cogent Engineering*, 5(1) :1502242.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR.

Hackett, S., Schlager, E., and Walker, J. (1994). The role of communication in resolving commons dilemmas : experimental evidence with heterogeneous appropriators. *Journal of Environmental Economics and Management*, 27(2) :99–126.

Haggag, S., Desokey, F., and Ramadan, M. (2017). A cosmological inflationary model using optimal control. *Gravitation and Cosmology*, 23(3) :236–239.

Hajek, B. (1988). Cooling schedules for optimal annealing. *Mathematics of operations research*, 13(2) :311–329.

Hajek, B. and Sasaki, G. (1989). Simulated annealing—to cool or not. *Systems & control letters*, 12(5) :443–447.

Hamdan, M. (2012). On the disruption-level of polynomial mutation for evolutionary multi-objective optimisation algorithms. *Computing and Informatics*, 29(5) :783–800.

Hamida, S. B. and Schoenauer, M. (2000). An adaptive algorithm for constrained optimization problems. In *International Conference on Parallel Problem Solving from Nature*, pages 529–538. Springer.

Hansen, E. A., Bernstein, D. S., and Zilberstein, S. (2004). Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715.

Hardin, G. (1968). The tragedy of the commons. *science*, 162(3859) :1243–1248.

Hasselt, H. V. (2010). Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1) :106–130.

Hernandez-Garcia, J. F. and Sutton, R. S. (2019). Understanding multi-step deep reinforcement learning : a systematic study of the dqn target. *arXiv preprint arXiv :1901.07510*.

Hesterberg, T. C. (1988). *Advances in importance sampling*. PhD thesis, Stanford University.

Higham, N. J. (2002). *Accuracy and stability of numerical algorithms*. SIAM.

Hodges, J. and Lehmann, E. L. (2012). Rank methods for combination of independent experiments in analysis of variance. In *Selected Works of EL Lehmann*, pages 403–418. Springer.

Hoffmann, M., Brooks, T., Da Fonseca, G., Gascon, C., Hawkins, A., James, R., Langhammer, P., Mittermeier, R., Pilgrim, J., Rodrigues, A., et al. (2008). Conservation planning and the iucn red list. *Endangered Species Research*, 6(2) :113–125.

Holland, J. H. (1992a). *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Holland, J. H. (1992b). Genetic algorithms. *Scientific american*, 267(1) :66–73.

Homaifar, A., Qi, C. X., and Lai, S. H. (1994). Constrained optimization via genetic algorithms. *Simulation*, 62(4) :242–253.

Horn, J., Nafpliotis, N., and Goldberg, D. E. (1994). A niched pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 82–87. Ieee.

Howard, R. A. (1964). Dynamic programming and markov processes.

IFREMER (2019). Les ressources halieutiques françaises : bilan 2018. `https://wwz.ifremer.fr/content/download/124503/file/DP_halieutique_ifremer.pdf`.

Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable markov decision problems. *Advances in neural information processing systems*, pages 345–352.

Jang, J.-S. (1993). Anfis : adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics*, 23(3) :665–685.

Janssen, M. (2002). *Complexity and ecosystem management : the theory and practice of multi-agent systems*. Edward Elgar Publishing.

Jaqaman, K. and Danuser, G. (2006). Linking data to models : data regression. *Nature Reviews Molecular Cell Biology*, 7(11) :813.

Jin, R., Du, X., and Chen, W. (2003). The use of metamodeling techniques for optimization under uncertainty. *Structural and Multidisciplinary Optimization*, 25(2) :99–116.

Joines, J. A. and Houck, C. R. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, pages 579–584. IEEE.

Jorge, J. (2010). *Nouvelles propositions pour la résolution exacte du sac à dos multi-objectif unidimensionnel en variables binaires*. PhD thesis, Université de Nantes.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2) :99–134.

Kang, K., Belkhale, S., Kahn, G., Abbeel, P., and Levine, S. (2019). Generalization through simulation : Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. In *2019 international conference on robotics and automation (ICRA)*, pages 6008–6014. IEEE.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

Karaboga, D. and Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied mathematics and computation*, 214(1) :108–132.

Karaboga, D. and Basturk, B. (2007). Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In *International fuzzy systems association world congress*, pages 789–798. Springer.

Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Katt, S., Oliehoek, F. A., and Amato, C. (2019). Bayesian reinforcement learning in factored pomdps. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 7–15. International Foundation for Autonomous Agents and Multiagent Systems.

Kelner, V., Capitanescu, F., Léonard, O., and Wehenkel, L. (2008). A hybrid optimization technique coupling an evolutionary and a local search algorithm. *Journal of Computational and Applied Mathematics*, 215(2) :448–456.

Kimbrough, E. O. and Vostroknutov, A. (2015). The social and ecological determinants of common pool resource sustainability. *Journal of environmental economics and management*, 72 :38–53.

Kingma, D. P. and Ba, J. (2014). Adam : A method for stochastic optimization. *arXiv preprint arXiv :1412.6980*.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598) :671–680.

Kleywegt, A. J. and Papastavrou, J. D. (1998). The dynamic and stochastic knapsack problem. *Operations Research*, 46(1) :17–35.

Knowles, J. D. (2002). *Local-search and hybrid evolutionary algorithms for Pareto optimization*. PhD thesis, University of Reading UK.

Koeck, B., Gérigny, O., Durieux, E. D. H., Coudray, S., Garsi, L.-H., Bisgambiglia, P.-A., Galgani, F., and Agostini, S. (2015). Connectivity patterns of coastal fishes following different dispersal scenarios across a transboundary marine protected area (bonifacio strait, nw mediterranean). *Estuarine, Coastal and Shelf Science*, 154 :234–247.

Konda, V. R. and Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014.

Kong, M., Tian, P., and Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional knapsack problem. *Computers & Operations Research*, 35(8) :2672–2683.

Kothari, V., Anuradha, J., Shah, S., and Mittal, P. (2012). A survey on particle swarm optimization in feature selection. In *Global Trends in Information Systems and Software Applications*, pages 192–201. Springer.

Koziel, S. and Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary computation*, 7(1) :19–44.

Lai, T. L. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1) :4–22.

Lange, S., Gabel, T., and Riedmiller, M. (2012). Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer.

Lawson, C. L. and Hanson, R. J. (1995). *Solving least squares problems*. SIAM.

Le Manacha, F., Durab, D., Perecd, A., Riutorte, J.-J., Lejeunec, P., Santonif, M.-C., Culiolif, J.-M., and Paulyg, D. (2011). Preliminary estimate of total marine fisheries catches in corsica. *Fisheries Centre Research Reports*, 19(3).

Le Tuyen, P., Vien, N. A., Layek, A., and Chung, T. (2018). Deep hierarchical reinforcement learning algorithm in partially observable markov decision processes. *arXiv*, pages arXiv–1805.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553) :436–444.

Lee, D.-T. and Schachter, B. J. (1980). Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3) :219–242.

Legaré, F., Graham, I. D., O'Connor, A. M., Dolan, J. G., and Bélanger-Ducharme, F. (2003). Prise de décision partagée : traduction et validation d'une échelle de confort décisionnel du médecin. *Pédagogie médicale*, 4(4) :216–222.

Leslie, P. H. (1945). On the use of matrices in certain population mathematics. *Biometrika*, 33(3) :183–212.

Leslie, P. H. (1948). Some further notes on the use of matrices in population mathematics. *Biometrika*, 35(3/4) :213–245.

Li, H. and Zhang, Q. (2009). Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *IEEE Transactions on evolutionary computation*, 13(2) :284–302.

Li, J., Zheng, W. X., Gu, J., and Hua, L. (2017). Parameter estimation algorithms for hammerstein output error systems using levenberg–marquardt optimization method with varying interval measurements. *Journal of the Franklin Institute*, 354(1) :316–331.

Li, Y., Wang, Y., Chen, J., Jiao, L., and Shang, R. (2015). Overlapping community detection through an improved multi-objective quantum-behaved particle swarm optimization. *Journal of Heuristics*, 21(4) :549–575.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2) :498–516.

Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the complexity of solving markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc.

Liu, J. S., Chen, R., and Logvinenko, T. (2001). A theoretical framework for sequential importance sampling with resampling. In *Sequential Monte Carlo methods in practice*, pages 225–246. Springer.

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., and Han, J. (2019). On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv :1908.03265*.

Lotka, A. J. (1956). Elements of mathematical biology. *Elements of physiological biology*.

Luong, N. C., Hoang, D. T., Gong, S., Niyato, D., Wang, P., Liang, Y.-C., and Kim, D. I. (2019). Applications of deep reinforcement learning in communications and networking : A survey. *IEEE Communications Surveys & Tutorials*, 21(4) :3133–3174.

MacQueen, J. et al. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.

Mahdavi, M., Fesanghary, M., and Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied mathematics and computation*, 188(2) :1567–1579.

Mahfoud, S. W. (1995). *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign.

Mahi, M., Baykan, Ö. K., and Kodaz, H. (2015). A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. *Applied Soft Computing*, 30 :484–490.

Mahmud, M., Kaiser, M. S., Hussain, A., and Vassanelli, S. (2018). Applications of deep learning and reinforcement learning to biological data. *IEEE transactions on neural networks and learning systems*, 29(6) :2063–2079.

Maringer, D. and Kellerer, H. (2003). Optimization of cardinality constrained portfolios with a hybrid local search algorithm. *Or Spectrum*, 25(4) :481–495.

Martelloni, P.-H. (2021). Modélisation et simulation des systèmes complexes spatialisés.utilisation de systèmes multi-agents et multi-composant pour la gestion des pêcheries.

Martin, J. and Simpson, T. (2006). A monte carlo method for reliability-based design opitmization. In *47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference 14th AIAA/ASME/AHS Adaptive Structures Conference 7th*, page 2146.

Matai, R., Singh, S., and Mittal, M. L. (2010). Traveling salesman problem : an overview of applications, formulations, and solution approaches. In *Traveling Salesman Problem, Theory and Applications*. InTech.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister : a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1) :3–30.

Mattei, M., Notton, G., Cristofari, C., Muselli, M., and Poggi, P. (2006). Calculation of the polycrystalline pv module temperature using a simple method of energy balance. *Renewable energy*, 31(4) :553–567.

Mavrovouniotis, M., Li, C., and Yang, S. (2017). A survey of swarm intelligence for dynamic optimization : Algorithms and applications. *Swarm and Evolutionary Computation*, 33 :1–17.

Maza, M. d. l. and Tidor, B. (1993). An analysis of selection procedures with particular attention paid to proportional and boltzmann selection. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 124–131. Morgan Kaufmann Publishers Inc.

Mehmood, Y., Aziz, N., Riaz, F., Iqbal, H., and Shahzad, W. (2018). Pso-based clustering techniques to solve multimodal optimization problems : A survey. In *2018 1st International Conference on Power, Energy and Smart Grid (ICPESG)*, pages 1–6. IEEE.

Mei, J., Xiao, C., Szepesvari, C., and Schuurmans, D. (2020). On the global convergence rates of softmax policy gradient methods. In *International Conference on Machine Learning*, pages 6820–6829. PMLR.

Mendes, R., Kennedy, J., and Neves, J. (2004). The fully informed particle swarm : simpler, maybe better. *IEEE transactions on evolutionary computation*, 8(3) :204–210.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6) :1087–1092.

Michalewicz, Z. and Janikow, C. Z. (1991). Handling constraints in genetic algorithms. In *ICGA*, pages 151–157.

Michalewicz, Z. and Nazhiyath, G. (1995). Genocop iii : A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 2, pages 647–651. IEEE.

Michie, D. and Chambers, R. A. (1968). Boxes : An experiment in adaptive control. *Machine intelligence*, 2(2) :137–152.

Minsky, M. L. (1967). *Computation : finite and infinite machines*. Prentice-Hall, Inc.

Mirjalili, S. and Lewis, A. (2016). The whale optimization algorithm. *Advances in engineering software*, 95 :51–67.

Mirjalili, S., Mirjalili, S. M., and Lewis, A. (2014). Grey wolf optimizer. *Advances in engineering software*, 69 :46–61.

Mladenović, N., Dražić, M., Kovačevic-Vujčić, V., and Čangalović, M. (2008). General variable neighborhood search for the continuous optimization. *European Journal of Operational Research*, 191(3) :753–770.

Mnih, A. and Rezende, D. (2016). Variational inference for monte carlo objectives. In *International Conference on Machine Learning*, pages 2188–2196. PMLR.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540) :529–533.

Moles, C. G., Mendes, P., and Banga, J. R. (2003). Parameter estimation in biochemical pathways : a comparison of global optimization methods. *Genome research*, 13(11) :2467–2474.

Monahan, G. E. (1982). State of the art—a survey of partially observable markov decision processes : theory, models, and algorithms. *Management science*, 28(1) :1–16.

Montana, D. J. and Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767.

Moré, J. J. (1978). The levenberg-marquardt algorithm : implementation and theory. In *Numerical analysis*, pages 105–116. Springer.

Naseri, G. and Koffas, M. A. (2020). Application of combinatorial optimization strategies in synthetic biology. *Nature communications*, 11(1) :1–14.

Ni, Y. and Sandal, L. K. (2019). Seasonality matters : A multi-season, multi-state dynamic optimization in fisheries. *European Journal of Operational Research*, 275(2) :648–658.

Nilsson, C. (2003). Heuristics for the traveling salesman problem. *Linkoping University*, pages 1–6.

Niu, B., Duan, Q., Tan, L., Liu, C., and Liang, P. (2015). A population-based clustering technique using particle swarm optimization and k-means. In *International Conference in Swarm Intelligence*, pages 145–152. Springer.

Nomura, T. and Shimohara, K. (2001). An analysis of two-parent recombinations for real-valued chromosomes in an infinite population. *Evolutionary Computation*, 9(3) :283–308.

Nothias, L.-F., Petras, D., Schmid, R., Dührkop, K., Rainer, J., Sarvepalli, A., Protsyuk, I., Ernst, M., Tsugawa, H., Fleischauer, M., et al. (2020). Feature-based molecular networking in the gnps analysis environment. *Nature Methods*, 17(9) :905–908.

Oliehoek, F. A., Amato, C., et al. (2016). *A concise introduction to decentralized POMDPs*, volume 1. Springer.

Omran, M. G. and Mahdavi, M. (2008). Global-best harmony search. *Applied mathematics and computation*, 198(2) :643–656.

Ong, Y.-S., Nair, P. B., and Lum, K. Y. (2006). Max-min surrogate-assisted evolutionary algorithm for robust design. *IEEE Transactions on Evolutionary Computation*, 10(4) :392–404.

Ostrom, E. (2008). The challenge of common-pool resources. *Environment : Science and Policy for Sustainable Development*, 50(4) :8–21.

Padua, R. N. and Ontoy, D. S. (2010). The use of a kernel ecological system in a multi species predator-prey model and climate change impact on biodiversity. *Asian Journal of Biodiversity*, 1(1).

Papadrakakis, M., Tsompanakis, Y., Lagaros, N. D., and Friagiadakis, M. (2004). Reliability based optimization of steel frames under seismic loading conditions using evolutionary computation. *Journal of Theoretical and Applied Mechanics*, 42 :585–608.

Papoudakis, G., Christianos, F., Rahman, A., and Albrecht, S. V. (2019). Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv preprint arXiv :1906.04737*.

Parsopoulos, K. E. and Vrahatis, M. N. (2002). Particle swarm optimization method in multiobjective problems. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 603–607.

Parsopoulos, K. E., Vrahatis, M. N., et al. (2002). Particle swarm optimization method for constrained optimization problems. *Intelligent Technologies–Theory and Application : New Trends in Intelligent Technologies*, 76(1) :214–220.

Pauwels, B. (2016). *Optimisation sans dérivées sous incertitudes appliquées à des simulateurs coûteux*. PhD thesis, Université Paul Sabatier-Toulouse III.

Pearl, J. (1984). Heuristics : intelligent search strategies for computer problem solving.

Pella, J. J. and Tomlinson, P. K. (1969). A generalized stock production model. *Inter-American Tropical Tuna Commission Bulletin*, 13(3) :416–497.

Penas, D. R., Gómez, A., Fraguela, B. B., Martín, M. J., and Cervino, S. (2019). Enhanced global optimization methods applied to complex fisheries stock assessment models. *Applied Soft Computing*, 77 :50–66.

Peng, R. and Vempala, S. (2021). Solving sparse linear systems faster than matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 504–521. SIAM.

Pétrowski, A. et al. (1997). An efficient hierarchical clustering technique for speciation. *Evolution. Technical report, Institute National des Telecommunications, Evry, France, Technique Report.*

Petrowski, A. et al. (1997). A new selection operator dedicated to speciation. In *ICGA*, volume 97, pages 144–151.

Picek, S., Jakobovic, D., and Golub, M. (2013). On the recombination operator in the real-coded genetic algorithms. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 3103–3110. IEEE.

Poggi, B. (2014). *Développement de concepts et outils d'aide à la décision pour l'optimisation via simulation*. PhD thesis, SPE UMR CNRS 6134 ; Université de Corse.

Poiron-Guidoni, N., Bisgambiglia, P.-A., and Bisgambiglia, P. (2020a). Deep hierarchical reinforcement learning in a markov game applied to fishery management decision making. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1371–1378. IEEE.

Poiron-Guidoni, N., Bisgambiglia, P.-A., and Bisgambiglia, P. (2020b). Utilisation de l'optimisation robuste pour palier au problème d'équifinalité d'un modèle bio-économique de pêcherie. In *Les journées Francophones de la Modélisation et de la Simulation (JFMS).*, pages 70–80, Cargèse (Corse). Cépadus.

Poiron-guidoni, N., Bisgambiglia, P.-A., and Bisgambiglia, P.-A. (2020). A probabilistic optimization approach to deal with uncertainties in model calibration. *IEEE CEC 2020*.

Poiron-Guidoni, N., Poggi, B., Bisgambiglia, P.-A., and Bisgambiglia, P. (2018). Optimisation robuste via simulation appliquée à la gestion durable des ressources. In *Actes des Journées DEVS Francophones (JDF2018) : Théorie et applications, Workshop RED.*, pages 91–100, Cargèse (Corse). Cépadus. 00000.

Powell, D. and Skolnick, M. M. (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In *Proceedings of the 5th International conference on Genetic Algorithms*, pages 424–431. Morgan Kaufmann Publishers Inc.

Puchinger, J. and Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization : A survey and classification. In *International Work-Conference on the Interplay Between Natural and Artificial Computation*, pages 41–53. Springer.

Qin, A. K., Huang, V. L., and Suganthan, P. N. (2008). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2) :398–417.

Qin, A. K., Huang, V. L., and Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2) :398–417.

Qu, B.-Y., Liang, J. J., Wang, Z., Chen, Q., and Suganthan, P. N. (2016). Novel benchmark functions for continuous multimodal optimization with comparative results. *Swarm and Evolutionary Computation*, 26 :23–34.

Rafalimanana, T. (2003). Les crevettes pénéides exploitées sur la côte ouest de madagascar : variabilités spatio-temporelles des paramètres biologiques et dynamique des populations. *Thčse Doct., Dép. Halieutique UPR MESH ENSA Rennes–France*.

Rahnamayan, S., Tizhoosh, H. R., and Salama, M. M. (2007). A novel population initialization method for accelerating evolutionary algorithms. *Computers & Mathematics with Applications*, 53(10) :1605–1614.

Raidl, G. R. (1998). An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on*, pages 207–211. IEEE.

Rakshit, P., Konar, A., and Das, S. (2017). Noisy evolutionary optimization algorithms–a comprehensive survey. *Swarm and Evolutionary Computation*, 33 :18–45.

Rana, S., Jasola, S., and Kumar, R. (2013). A boundary restricted adaptive particle swarm optimization for data clustering. *International journal of machine learning and cybernetics*, 4(4) :391–400.

Rankinen, K., Karvonen, T., and Butterfield, D. (2006). An application of the glue methodology for estimating the parameters of the inca-n model. *Science of the total environment*, 365(1-3) :123–139.

Rao, S. S. and Rao, S. S. (2009). *Engineering optimization : theory and practice*. John Wiley & Sons.

Ray, T. and Smith, W. (2006). A surrogate assisted parallel multiobjective evolutionary algorithm for robust engineering design. *Engineering Optimization*, 38(8) :997–1011.

Rechenberg, I. (1973). Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution.

Reinelt, G. (1994). *The traveling salesman : computational solutions for TSP applications*. Springer-Verlag.

Ridha, H. M., Gomes, C., Hizam, H., Ahmadipour, M., Heidari, A. A., and Chen, H. (2021). Multi-objective optimization and multi-criteria decision-making methods for optimal design of standalone photovoltaic system : A comprehensive review. *Renewable and Sustainable Energy Reviews*, 135 :110202.

Robbins, H. (1985). Some aspects of the sequential design of experiments. In *Herbert Robbins Selected Papers*, pages 169–177. Springer.

Rodríguez, P., Bautista, M. A., Gonzalez, J., and Escalera, S. (2018). Beyond one-hot encoding : Lower dimensional target embedding. *Image and Vision Computing*, 75 :21–31.

Roopnarine, P. (2013). Ecology and the tragedy of the commons. *Sustainability*, 5(2) :749–773.

Rosen, D. M., Kaess, M., and Leonard, J. J. (2012). An incremental trust-region method for robust online sparse least-squares estimation. In *2012 IEEE International Conference on Robotics and Automation*, pages 1262–1269. IEEE.

Rossi, R. J. (2018). *Mathematical statistics : an introduction to likelihood based inference*. John Wiley & Sons.

Rubin, P. A. (1984). Generating random points in a polytope. *Communications in Statistics-Simulation and Computation*, 13(3) :375–396.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv :1609.04747*.

Ruiz-Vanoye, J. A., Pérez-Ortega, J., Díaz-Parra, O., Frausto-Solís, J., Huacuja, H. J. F., Cruz-Reyes, L., et al. (2011). Survey of polynomial transformations between np-complete problems. *Journal of computational and applied mathematics*, 235(16) :4851–4865.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088) :533.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England.

Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on evolutionary computation*, 4(3) :284–294.

Russo, D., Van Roy, B., Kazerouni, A., Osband, I., and Wen, Z. (2017). A tutorial on thompson sampling. *arXiv preprint arXiv :1707.02038*.

Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3) :660–674.

Salas, E., Rosen, M. A., and DiazGranados, D. (2010). Expertise-based intuition and decision making in organizations. *Journal of management*, 36(4) :941–973.

Samet, H. (2006). *Foundations of multidimensional and metric data structures*. Morgan Kaufmann.

Sandgren, E. and Cameron, T. M. (2002). Robust design optimization of structures through consideration of variation. *Computers & structures*, 80(20-21) :1605–1613.

Sareni, B. (1999). *Méthodes d'optimisation multimodales associées à la modélisation numérique en électromagnétisme*. PhD thesis, Ecole Centrale de Lyon.

Schaefer, M. B. (1954). Some aspects of the dynamics of populations important to the management of the commercial marine fisheries. *Inter-American Tropical Tuna Commission Bulletin*, 1(2) :23–56.

Schaefer, M. B. (1959). Biological and Economic Aspects of the Management of Commercial Marine Fisheries. *Transactions of the American Fisheries Society*, 88(2) :100–104.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv :1707.06347*.

Schulz, K., Beven, K., and Huwe, B. (1999). Equifinality and the problem of robust calibration in nitrogen budget simulations.

Schwefel, H.-P. (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

Sea Around Us (2020). *Sea Around Us*. http ://www.seaaroundus.org.

Seibert, J. and McDonnell, J. J. (2002). On the dialog between experimentalist and modeler in catchment hydrology : Use of soft data for multicriteria model calibration. *Water Resources Research*, 38(11) :23–1.

Sen, Z. (2016). *Spatial modeling principles in earth sciences*. Springer.

Sevaux, M. and Le Quéré, Y. (2003). Solving a robust maintenance scheduling problem at the french railways company. Technical report, Technical report, University of Valenciennes.

Shah, M. and Sharma, U. (2003). Optimal harvesting policies for a generalized gordon–schaefer model in randomly varying environment. *Applied Stochastic Models in Business and Industry*, 19(1) :43–49.

Shcherbina, O., Neumaier, A., Sam-Haroud, D., Vu, X.-H., and Nguyen, T.-V. (2002). Benchmarking global optimization and constraint satisfaction codes. In *International Workshop on Global Optimization and Constraint Satisfaction*, pages 211–222. Springer.

Shelton, C. R. (2001). Importance sampling for reinforcement learning with multiple objectives.

Shima, T., Rasmussen, S. J., Sparks, A. G., and Passino, K. M. (2006). Multiple task assignments for cooperating uninhabited aerial vehicles using genetic algorithms. *Computers & Operations Research*, 33(11) :3252–3269.

Siarry, P. (2014). *Métaheuristiques*. Editions Eyrolles.

Sibson, R. (1973). Slink : an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1) :30–34.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587) :484–489.

Silverman, B. W. (2018). *Density estimation for statistics and data analysis*. Routledge.

Sinclair, M. and Ashkanasy, N. M. (2005). Intuition : myth or a decision-making tool ? *Management learning*, 36(3) :353–370.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3) :123–158.

Smit, S. K. and Eiben, A. E. (2009). Comparing parameter tuning methods for evolutionary algorithms. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 399–406. IEEE.

Smith, A. E. (1993). Genetic optimization using a penalty function. In *Proc. 5th Int. Conf. on Genetic Algorithms*, pages 499–505.

Soler-Dominguez, A., Juan, A. A., and Kizys, R. (2017). A survey on financial applications of metaheuristics. *ACM Computing Surveys (CSUR)*, 50(1) :1–23.

Sorensen, K. (2001). Tabu searching for robust solutions, mic'2001. In *4th Metaheuristic International Conference*, pages 707–712.

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3) :221–248.

Stanley, R. P. (1980). Decompositions of rational convex polytopes. *Ann. Discrete Math*, 6(6) :333–342.

Stein, P. (1966). A note on the volume of a simplex. *The American Mathematical Monthly*, 73(3) :299–301.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4) :341–359.

Stutzle, T. and Hoos, H. (1997). Max-min ant system and local search for the traveling salesman problem. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314. IEEE.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1) :9–44.

Sutton, R. S. (1996). Generalization in reinforcement learning : Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Sutton, R. S. and Singh, S. P. (1994). On step-size and bias in temporal-difference learning. In *Proceedings of the Eighth Yale Workshop on Adaptive and Learning Systems*, pages 91–96. Citeseer.

Tahara, T., Gavina, M. K. A., Kawano, T., Tubay, J. M., Rabajante, J. F., Ito, H., Morita, S., Ichinose, G., Okabe, T., Togashi, T., et al. (2018). Asymptotic stability of a modified lotka-volterra model with small immigrations. *Scientific reports*, 8(1) :7029.

Talpaert, V., Sobh, I., Kiran, B. R., Mannion, P., Yogamani, S., El-Sallab, A., and Perez, P. (2019). Exploring applications of deep reinforcement learning for real-world autonomous driving systems. *arXiv preprint arXiv :1901.01536*.

Tanabe, R. and Ishibuchi, H. (2019). A review of evolutionary multimodal multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 24(1) :193–200.

Tang, J. and Wang, Y. (2015). An adjustable robust optimisation method for elective and emergency surgery capacity allocation with demand uncertainty. *International Journal of Production Research*, 53(24) :7317–7328.

Tato, A. and Nkambou, R. (2018). Improving adam optimizer.

Tessema, B. and Yen, G. G. (2006). A self adaptive penalty function based algorithm for constrained optimization. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 246–253. IEEE.

Thathachar, M. and Sastry, P. S. (1985). A new approach to the design of reinforcement schemes for learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, (1) :168–175.

Thomas, A., Barriere, S., Broseus, L., Brooke, J., Lorenzi, C., Villemin, J.-P., Beurier, G., Sabatier, R., Reynes, C., Mancheron, A., et al. (2019). Gecko is a genetic algorithm to classify and explore high throughput sequencing data. *Communications biology*, 2(1) :1–8.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4) :285–294.

Thomsen, R. (2004). Multimodal optimization using crowding-based differential evolution. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volume 2, pages 1382–1389. IEEE.

Tian, Z., Zhang, X., Jin, X., Zhou, X., Si, B., and Shi, X. (2018). Towards adoption of building energy simulation and optimization for passive building design : A survey and a review. *Energy and Buildings*, 158 :1306–1316.

Tjørve, K. M. and Tjørve, E. (2017). The use of gompertz models in growth analyses, and new gompertz-model approach : An addition to the unified-richards family. *PloS one*, 12(6).

Traoré, M. K. and Muzy, A. (2006). Capturing the dual relationship between simulation models and their context. *Simulation Modelling Practice and Theory*, 14(2) :126–142.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Van Hoorebeke, D. (2008). L'émotion et la prise de décision. *Revue française de gestion*, (2) :33–44.

Van Laarhoven, P. J., Aarts, E. H., and Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations research*, 40(1) :113–125.

Van Roy, B., Bertsekas, D. P., Lee, Y., and Tsitsiklis, J. N. (1997). A neuro-dynamic programming approach to retailer inventory management. In *Decision and Control, 1997., Proceedings of the 36th IEEE Conference on*, volume 4, pages 4052–4057. IEEE.

Van Seijen, H., Van Hasselt, H., Whiteson, S., and Wiering, M. (2009). A theoretical and empirical analysis of expected sarsa. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on*, pages 177–184. IEEE.

Van Toan Dao, H. Q. N., Maigne, L., Breton, V., and Hill, D. R. (2014). Numerical reproducibility, portability and performance of modern pseudo random number generators.

Verhulst, P.-F. (1845). Recherches mathématiques sur la loi d'accroissement de la population. *Nouveaux memoires de l'academie royale des sciences*, 18 :1–41.

Versmisse, D. (2008). *Gestion de la complexité formelle et opérationnelle des systèmes complexes Application aux anthroposystèmes marins.* PhD thesis, Université du Littoral Côte d'Opale.

Versmisse, D., Macher, C., Ramat, E., Soulié, J.-C., and Thébaud, O. (2007). Developing a bioeconomic simulation tool of fisheries dynamics : a case study. Post-Print hal-00368955, HAL.

Vidal, J.-P. (2005). *Assistance au calage de modèles numériques en hydraulique fluviale–Apports de l'intelligence artificielle*. PhD thesis, Institut National Polytechnique de Toulouse-INPT.

Von Bertalanffy, L. (1938). A quantitative theory of organic growth (inquiries on growth laws. ii). *Human biology*, 10(2) :181–213.

Von Bertalanffy, L. (1956). General system theory. *General systems*, 1 :1–10.

von der Osten, F. B., Kirley, M., and Miller, T. (2017). Sustainability is possible despite greed-exploring the nexus between profitability and sustainability in common pool resource systems. *Scientific reports*, 7(1) :1–12.

Voronoi, G. (1907). Nouvelles applications des parametres continusa la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *J. reine angew. Math*, 133(97-178) :14.

Voss, R., Quaas, M. F., Schmidt, J. O., Stoeven, M. T., Francis, T. B., Levin, P. S., Armitage, D. R., Cleary, J. S., Jones, R. R., Lee, L. C., et al. (2018). Quantifying the benefits of spatial fisheries management–an ecological-economic optimization approach. *Ecological Modelling*, 385 :165–172.

Voyant, C., Notton, G., Kalogirou, S., Nivet, M.-L., Paoli, C., Motte, F., and Fouilloy, A. (2017). Machine learning methods for solar radiation forecasting : A review. *Renewable Energy*, 105 :569–582.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval Research Logistics (NRL)*, 6(2) :131–140.

Wang, H., Wang, W., Xiao, S., Cui, Z., Xu, M., and Zhou, X. (2020). Improving artificial bee colony algorithm using a new neighborhood selection mechanism. *Information Sciences*, 527 :227–240.

Wang, J.-S. and Li, S.-X. (2019). An improved grey wolf optimizer based on differential evolution and elimination mechanism. *Scientific reports*, 9(1) :1–21.

Wang, L., Cai, Q., Yang, Z., and Wang, Z. (2019a). Neural policy gradient methods : Global optimality and rates of convergence. *arXiv preprint arXiv :1909.01150*.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

Wang, Z.-J., Zhan, Z.-H., Lin, Y., Yu, W.-J., Wang, H., Kwong, S., and Zhang, J. (2019b). Automatic niching differential evolution with contour prediction approach for multimodal optimization problems. *IEEE Transactions on Evolutionary Computation*, 24(1) :114–128.

Wang, Z.-J., Zhan, Z.-H., Lin, Y., Yu, W.-J., Yuan, H.-Q., Gu, T.-L., Kwong, S., and Zhang, J. (2017). Dual-strategy differential evolution with affinity propagation clustering for multimodal optimization problems. *IEEE Transactions on Evolutionary Computation*, 22(6) :894–908.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4) :279–292.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.

Watson, D. F. (1981). Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2) :167–172.

White III, C. C. and White, D. J. (1989). Markov decision processes. *European Journal of Operational Research*, 39(1) :1–16.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3) :229–256.

Wirth, C., Akrour, R., Neumann, G., Fürnkranz, J., et al. (2017). A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136) :1–46.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1) :67–82.

Wong, K.-C., Wu, C.-H., Mok, R. K., Peng, C., and Zhang, Z. (2012). Evolutionary multimodal optimization using the principle of locality. *Information Sciences*, 194 :138–170.

Yang, X.-S. (2010). Appendix a : test problems in optimization. *Engineering optimization*, pages 261–266.

Yanıkoğlu, İ., Gorissen, B. L., and den Hertog, D. (2019). A survey of adjustable robust optimization. *European Journal of Operational Research*, 277(3) :799–813.

Yao, X. and Liu, Y. (1996). Fast evolutionary programming. *Evolutionary Programming*, 3 :451–460.

Yu, W., Li, B., Jia, H., Zhang, M., and Wang, D. (2015). Application of multi-objective genetic algorithm to optimize energy efficiency and thermal comfort in building design. *Energy and Buildings*, 88 :135–143.

Yuan, Y.-x. (2000). A review of trust region algorithms for optimization. In *Iciam*, volume 99, pages 271–282.

Zakaria, A., Ismail, F. B., Lipu, M. H., and Hannan, M. A. (2020). Uncertainty models for stochastic optimization in renewable energy applications. *Renewable Energy*, 145 :1543–1571.

Zavriev, S. (1993). On the global optimization properties of finite-difference local descent algorithms. *Journal of Global Optimization*, 3(1) :67–78.

Zhang, L.-m. (2015). An improved adaptive harmony search algorithm.

Zhang, Q. and Li, H. (2007). Moea/d : A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6) :712–731.

Zhang, Y., Wang, S., and Ji, G. (2015). A comprehensive survey on particle swarm optimization algorithm and its applications. *Mathematical Problems in Engineering*, 2015.

Zhang, Z. (2018). Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pages 1–2. IEEE.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms : a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4) :257–271.

TABLE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ALGORITHMES