Conception d'heuristiques d'optimisation pour les problèmes de grande dimension : application à l'analyse de données de puces à ADN

Vincent Gardeux

UNIVERSITÉ DE PARIS-EST CRÉTEIL

ÉCOLE DOCTORALE (ED 532)

MATHÉMATIQUES ET SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION (MSTIC)

THÈSE DE DOCTORAT

SPÉCIALITÉ INFORMATIQUE

intitulée

Conception d'heuristiques d'optimisation pour les problèmes de grande dimension. Application à l'analyse de données de puces à ADN.

soutenue publiquement le 30 novembre 2011 par

Vincent GARDEUX

devant le jury composé de

M. Jin-Kao HAO	Professeur des Universités, Université d'Angers	**Rapporteur**
M. Eric TAILLARD	Professeur, HEIG-VD, Suisse	**Rapporteur**
M. Gérard BERTHIAU	Professeur des Universités, Université de Nantes	**Examinateur**
M. René NATOWICZ	Professeur associé, ESIEE, Marne-la-Vallée	**Examinateur**
M. Jean-Philippe URBAN	Professeur des Universités, Université de Haute Alsace, Mulhouse	**Examinateur**
M. Rachid CHELOUAH	Professeur associé, EISTI, Cergy	**Co-directeur de thèse**
M. Patrick SIARRY	Professeur des Universités, Université de Paris-Est Créteil	**Directeur de thèse**

Résumé

Cette thèse expose la problématique récente concernant la résolution de problèmes de grande dimension. Nous présentons les méthodes permettant de les résoudre ainsi que leurs applications, notamment pour la sélection de variables dans le domaine de la fouille de données.
Dans la première partie de cette thèse, nous exposons les enjeux de la résolution de problèmes de grande dimension. Nous nous intéressons principalement aux méthodes de recherche linéaire, que nous jugeons particulièrement adaptées pour la résolution de tels problèmes. Nous présentons ensuite les méthodes que nous avons développées, basées sur ce principe : CUS, EUS et EM323. Nous soulignons en particulier la très grande vitesse de convergence de CUS et EUS, ainsi que leur simplicité de mise en œuvre. La méthode EM323 est issue d'une hybridation entre la méthode EUS et un algorithme d'optimisation unidimensionnel développé par F. Glover : l'algorithme 3-2-3. Nous montrons que ce dernier algorithme obtient des résultats d'une plus grande précision, notamment pour les problèmes non séparables, qui sont le point faible des méthodes issues de la recherche linéaire.
Dans une deuxième partie, nous nous intéressons aux problèmes de fouille de données, et plus particulièrement l'analyse de données de puces à ADN. Le but est de classifier ces données et de prédire le comportement de nouveaux exemples. Dans un premier temps, une collaboration avec l'hôpital Tenon nous permet d'analyser des données privées concernant le cancer du sein. Nous développons alors une méthode exacte, nommée δ-test, enrichie par la suite d'une méthode permettant la sélection automatique du nombre de variables. Dans un deuxième temps, nous développons une méthode heuristique de sélection de variables, nommée ABEUS, basée sur l'optimisation des performances du classifieur DLDA. Les résultats obtenus sur des données publiques montrent que nos méthodes permettent de sélectionner des sous-ensembles de variables de taille très faible, ce qui est un critère important permettant d'éviter le sur-apprentissage.

Mots clés : *Heuristiques, Optimisation combinatoire, Problèmes de grande dimension, Recherche linéaire, Fouille de données, Analyse de données d'expression géniques, Puces à ADN, Prédiction*

Abstract

This PhD thesis explains the recent issue concerning the resolution of high-dimensional problems. We present methods designed to solve them, and their applications for feature selection problems, in the data mining field.
In the first part of this thesis, we introduce the stakes of solving high-dimensional problems. We mainly investigate line search methods, because we consider them to be particularly suitable for solving such problems. Then, we present the methods we developed, based on this principle : CUS, EUS and EM323. We emphasize, in particular, the very high convergence speed of CUS and EUS, and their simplicity of implementation. The EM323 method is based on an hybridization between EUS and a one-dimensional optimization algorithm developed by F. Glover : the 3-2-3 algorithm. We show that the results of EM323 are more accurate, especially for non-separable problems, which are the weakness of line search based methods.
In the second part, we focus on data mining problems, and especially those concerning microarray data analysis. The objectives are to classify data and to predict the behavior of new samples. A collaboration with the Tenon Hospital in Paris allows us to analyze their private breast cancer data. To this end, we develop an exact method, called δ-test, enhanced by a method designed to automatically select the optimal number of variables. In a second time, we develop an heuristic, named ABEUS, based on the optimization of the DLDA classifier performances. The results obtained from publicly available data show that our methods manage to select very small subsets of variables, which is an important criterion to avoid overfitting.

Keywords : *Heuristics, Combinatorial optimization, High-dimensional problems, Line search, Data mining, Gene expression analysis, DNA Microarray analysis, Prediction*

Remerciements

Je tiens à remercier en tout premier lieu Patrick Siarry et Rachid Chelouah qui m'ont donné l'opportunité de faire cette thèse après mon diplôme d'ingénieur. Tout au long de ces trois années, ils ont su orienter mes recherches et m'encourager, tout en me laissant une grande liberté dans les directions que je souhaitais prendre. Ils ont toujours été disponibles pour répondre à mes questions, à une vitesse défiant toute loi physique ! Pour leur confiance et leur infinie patience, je les remercie vivement.

J'adresse également toute ma gratitude au professeur Eric Taillard de l'école HEIG-VD en Suisse et au professeur Jin-Kao Hao, de l'université d'Angers qui me font l'honneur d'être rapporteurs de cette thèse. Je les remercie vivement pour l'intérêt qu'ils ont porté à mon travail, ainsi que leurs conseils précieux qui m'ont permis d'enrichir grandement ce manuscrit.
Merci également aux autres membres du jury qui ont accepté de juger ce travail : Gérard Berthiau, Jean-Philippe Urban et en particulier René Natowicz, professeur associé de l'école d'ingénieurs ESIEE à Marne-la-Vallée. Il m'a permis d'orienter mon travail sur un domaine qui me touche particulièrement à cœur, la génomique, et m'a appris énormément, sans compter son temps.

Je remercie sincèrement Houcine Senoussi, directeur de l'EISTI Cergy et Nesim Fintz, directeur général et fondateur de l'EISTI, qui m'ont permis d'obtenir un poste de professeur associé afin de financer cette thèse, tout en me laissant la possibilité de la réaliser dans les meilleures conditions possibles.

J'aimerais également remercier mes collègues de l'EISTI, pour l'ambiance très conviviale qu'ils ont su faire régner sur mon lieu de travail. Et plus particulièrement Hervé de Milleville pour ses précieux conseils dans le domaine des statistiques et des mathématiques, ainsi que Maria Malek et Jean-Paul Forrest qui ont pris du temps pour me conseiller lors de la rédaction de ce manuscrit. Je souhaite également remercier Stefan Bornhofen, premier compagnon de thèse, dont la bonne humeur constante est toujours un remontant.

Enfin, je tiens à remercier ma famille qui m'a toujours soutenu et aidé dans mes choix, et pour leurs vifs encouragements durant ces trois années d'études.

À ma chère Julie, qui m'a constamment soutenu et a permis la réalisation de cette thèse. Je la remercie ici de son amour et lui dédie ce travail.

Table des matières

Introduction générale

L'optimisation combinatoire couvre un large éventail de techniques - certaines datant même de quelques siècles - et est utilisée dans de nombreux domaines de la vie courante. Elle peut être définie comme la recherche d'une solution optimale, parmi un ensemble de solutions candidates, pour un problème donné. Elle fait toujours l'objet de recherches intensives et s'applique aujourd'hui à de nombreux domaines : finance (minimisation de coût, maximisation de profit), transport (planification), surveillance (détection de dysfonctionnement, prévention), biologie (analyse de données, modélisation), etc.
Pour certains de ces problèmes, la solution optimale peut être trouvée par des méthodes exactes. Cependant, ces méthodes ne peuvent s'appliquer qu'à certains problèmes particuliers et sous certaines conditions. En effet, une grande partie des problèmes nécessiteraient une recherche exhaustive de l'ensemble des solutions possibles, ce qui n'est pas toujours faisable en un temps raisonnable.

Dans la première partie de cette thèse, nous présentons les métaheuristiques, qui sont des méthodes permettant d'obtenir une valeur approchée de la solution optimale en un temps polynomial. Elles s'appliquent à tout type de problème d'optimisation et l'engouement à leur égard ne cesse de se développer depuis une vingtaine d'années. On peut citer par exemple les algorithmes génétiques, le recuit simulé, la recherche tabou, les colonies de fourmis ou bien les essaims particulaires, sans parler des nouvelles techniques qui voient encore le jour.
Ces méthodes sont appréciées pour leur facilité de mise en place, mais aussi par la qualité des solutions qu'elles sont capables de trouver en un temps "assez court". Cependant, elles se heurtent à un problème bien connu en informatique : la "malédiction de la dimension". Effectivement, un certain nombre de problèmes actuels (notamment les problèmes de fouille de données) sont constitués d'un nombre impressionnant de variables à optimiser (parfois plusieurs milliers), alors que les heuristiques sont initialement mises au point sur des problèmes d'une dizaine de variables. On admet dans la littérature que l'on peut qualifier de "problème à haute dimension" un problème dont le nombre de variables est supérieur à 100. Ces problèmes nécessitent une approche différente. De fait, la plupart des méthodes citées ne sont pas robustes à l'augmentation de la dimension, et ne convergent plus en un temps raisonnable, lorsque ce nombre devient trop important.

Dans ce manuscrit, nous nous sommes concentrés sur l'intérêt d'utiliser des méthodes de relaxation pour résoudre ce type de problème. Leur principe est de séparer un problème multidimensionnel en sous-problèmes unidimensionnels. Les algorithmes que nous avons développés reposent sur des méthodes très simples, dites de *line search*, afin de résoudre ces problèmes unidimensionnels. Leur point fort est d'avoir une convergence très rapide et un faible coût calculatoire, qui s'avèrent être des atouts considérables pour des problèmes de grande dimension. Dans un second temps, nous verrons que ces méthodes rencontrent un obstacle majeur, lors de la résolution de problèmes non séparables, du fait de leur définition. Nous verrons alors que l'on peut adapter ces méthodes de *line search*, afin d'améliorer la résolution de ce type de fonctions.

Dans une deuxième partie, nous nous sommes intéressés à des applications dans le domaine de l'analyse de données, et plus particulièrement en bioinformatique. Assurément, l'évolution des technologies récentes d'acquisition de données relatives à l'ADN ouvre des perspectives très prometteuses pour la compréhension des phénomènes biologiques. Dans cette étude, nous avons collaboré avec l'hôpital Tenon, l'école d'ingénieurs Esiee-Paris, et l'Université fédérale brésilienne de l'état de Minas Gerais à Belo Horizonte, dans le cadre d'un programme de coopération franco-brésilien CAPES-COFECUB, 2008-2011. Nous avons travaillé sur un modèle de prédiction de la réponse à un traitement chimiothérapique pour le cancer du sein, pour lequel nous disposions de données cliniques provenant de puces à ADN (ou biopuces), une technologie qui permet de mesurer simultanément l'expression d'un très grand nombre de gènes au sein d'un échantillon biologique.

L'analyse des données acquises par ces biopuces a pour but d'en extraire de la connaissance, ou bien de créer des modèles permettant de structurer les informations qu'elles contiennent. Dans la littérature, ces méthodes de fouille de données regroupent des méthodes statistiques, des modèles de prédiction, de classification, etc. Dans notre cas, nous nous sommes particulièrement intéressés aux modèles de prédiction pour l'apprentissage supervisé à deux classes.
Dans la littérature, il existe un grand nombre de modèles de prédiction ayant déjà fait leurs preuves (analyse discriminante linéaire, arbres de décisions, réseaux de neurones, etc.). Or, la plupart de ces modèles de prédiction ne peuvent pas être appliqués directement sur nos données. En effet, la spécificité de notre problème vient du fait que nous disposons de très peu de cas d'études (de patientes) relativement au nombre de variables (le nombre de gènes). Nous avons donc pris le parti de focaliser notre travail sur les méthodes de sélection de variables, en prétraitement à l'utilisation de ces classifieurs.

Les méthodes de sélection de variables visent à sélectionner, parmi un ensemble de variables, le plus petit sous-ensemble de variables pertinentes représentant le problème. Ces méthodes peuvent être statistiques (les plus répandues), ou bien peuvent provenir de l'optimisation d'une fonction objectif. Dans ce cas, nous sommes en présence d'un problème de grande dimension et les algorithmes que nous avons présentés en première partie peuvent être appliqués. Dans cette optique, nous avons conçu deux méthodes de sélection d'attributs basées sur les algorithmes développés durant la première partie de cette thèse.

Nos méthodes ont notamment été adaptées pour résoudre des problèmes discrets. Le problème de sélection de variables est alors considéré comme un problème d'optimisation bi-objectif que nous résoudrons par une méthode d'agrégation des objectifs. Cet algorithme sera par la suite amélioré en une méthode exacte, dans la lignée des méthodes de sélection par filtre.
Dans un deuxième temps, nous développerons une méthode permettant d'optimiser les performances d'un classifieur particulier, appartenant à la classe des méthodes de sélection dites *wrapper*. Nous comparerons alors les résultats de ce type d'approche à ceux des méthodes par filtre, afin de dresser un bilan de leurs efficacités respectives.

Cette thèse a été préparée conjointement au sein de l'université de Paris-Est Créteil, dans le Laboratoire Images, Signaux et Systèmes Intelligents (LiSSi, E.A. 3956), et à l'EISTI Cergy (École Internationale des Sciences du Traitement de l'Information), dans le Laboratoire de Recherche en Informatique et Systèmes (L@RIS). Ces deux laboratoires travaillent en particulier sur des problèmes de recherche opérationnelle, et notamment sur les méthodes d'optimisation, dans le cadre d'applications en traitement d'images ou dans le domaine médical. Ce travail a été proposé et encadré par Patrick Siarry, professeur et directeur de l'équipe Traitement de l'Image et du Signal. Il a également été co-encadré par Rachid Chelouah, professeur associé et directeur du département informatique de l'EISTI.

La partie applicative, réalisée dans le domaine de la bioinformatique, a été encadrée par René Natowicz, professeur associé de l'école d'ingénieur Esiee-Paris.

La suite de ce manuscrit est structurée en quatre chapitres : les deux premiers chapitres sont consacrés au développement de méthodes d'optimisation pour les problèmes à grande dimension, tandis que les deux derniers sont consacrés à leur application dans le cadre de la prédiction de la réponse à un traitement chimiothérapique, via des données de puces à ADN.

Le premier chapitre présente un état de l'art des méthodes d'optimisation de manière générale, puis plus particulièrement des méthodes s'attachant aux problèmes à haute dimension. Nous ferons en particulier une description du fameux problème de la "malédiction de la dimension", ainsi qu'une brève introduction à la théorie de la complexité. Ces constatations nous permettront d'introduire l'intérêt de développer des heuristiques adaptées aux problèmes à haute dimension.

Dans le second chapitre, nous présenterons les différents algorithmes d'optimisation développés durant cette thèse. Nous verrons en quoi ils sont adaptés aux problèmes de grande dimension et nous les comparerons à différentes méthodes récentes de la littérature, via différents *benchmarks* classiques.

Le troisième chapitre fera un nouvel état de l'art, cette fois-ci pour décrire le contexte dans lequel se place l'application en bioinformatique, et plus précisément le domaine de l'onco-pharmacogénomique. Nous verrons les principes de base de la génomique, ainsi que les méthodes d'acquisition de données via les puces à ADN. Puis nous ferons un récapitulatif des différentes méthodes d'analyse de données existantes et placerons notre problème dans le contexte de l'apprentissage automatique supervisé. Nous étudierons enfin les différentes méthodes de sélection de variables, sur lesquelles porteront les méthodes développées dans le dernier chapitre.

Dans le quatrième et dernier chapitre, nous présenterons les travaux effectués dans le domaine de la fouille de données, et plus particulièrement pour la sélection de variables dans un problème d'apprentissage supervisé. Nous verrons comment le formaliser en tant que problème d'optimisation et nous les résoudrons via des algorithmes d'optimisation développés dans le deuxième chapitre. Nous comparerons ensuite les résultats à ceux d'autres méthodes récentes de la littérature, sur les mêmes jeux de données.

En conclusion, nous récapitulerons nos principales contributions, puis nous suggérerons quelques pistes pour améliorer les performances des algorithmes déjà développés.

Chapitre Un

Généralités sur les méthodes d'optimisation

1.1 Introduction

L'optimisation se définit comme la sélection du meilleur élément (appelé optimum) parmi un ensemble d'éléments autorisés (appelé espace de recherche), en fonction d'un critère de comparaison. C'est un domaine issu des mathématiques et aujourd'hui largement utilisé en informatique, et plus particulièrement en recherche opérationnelle.

Dans les cas les plus courants, l'optimisation d'un problème sans contraintes revient à minimiser ou maximiser la valeur d'une fonction mathématique (appelée fonction objectif) dans un espace de recherche (noté $\mathbb{S}$). Si l'on pose $f : \mathbb{S}^n \to \mathbb{Q}$ la fonction objectif à maximiser (respectivement à minimiser) à valeurs dans $\mathbb{Q}$, le problème revient alors à trouver l'optimum $x^* \in \mathbb{S}^n$ tel que $f(x^*)$ soit maximal (respectivement minimal).

Historiquement, les premières méthodes d'optimisation provenaient de résolutions mathématiques, que l'on retrouve par exemple dans les travaux de Fermat, Lagrange et Hamilton. Des méthodes algorithmiques itératives ont également été proposées par Newton et Gauss. Les premières méthodes dites de "programmation linéaire" ont été introduites par Kantorovich [Kantorovich 60] peu de temps avant qu'elles ne soient redécouvertes et améliorées par Dantzig [Dantzig 63]. Ces méthodes étaient originellement dédiées à la planification de programmes militaires pendant la seconde guerre mondiale. D'où le terme "programmation", qui ne fais pas référence au départ à une programmation informatique, mais plutôt aux termes planification et ordonnancement. Ces méthodes ont ensuite été étendues à tout type d'optimisation linéaire.

Aujourd'hui, les applications concernent par exemple l'optimisation d'un trajet, de la forme d'un objet, du rendement d'un appareil, du choix d'investissements, etc. Ces quelques exemples montrent qu'il existe pléthore d'applications possibles, dans des domaines très différents.

Après une présentation du contexte amenant à l'utilisation de ces différentes méthodes d'optimisation, nous verrons dans ce chapitre les outils mathématiques, ainsi que les classes d'algorithmes permettant la résolution de ces problèmes. Puis nous verrons comment les comparer, et nous nous attacherons à mettre en valeur l'importance de nouvelles méthodes pour le cas des problèmes de grande taille. Nous conclurons sur les méthodes actuelles qu'il est intéressant d'approfondir.

1.2 Théorie de la complexité

La théorie de la complexité [Papadimitriou 03] [Arora 09] s'intéresse à l'étude formelle de la difficulté des problèmes en informatique. En effet, pour chaque problème posé, il est toujours possible de trouver une multiplicité d'algorithmes permettant de le résoudre. On se pose alors la question fondamentale de savoir quel algorithme est le plus performant. Pour les comparer, on pourrait calculer le temps mis par un algorithme pour s'exécuter, une fois implémenté sur une machine. Cependant, cette approche est dépendante des caractéristiques de la machine ou du langage utilisé. Pour définir plus rigoureusement ce qu'est l'efficacité d'un algorithme, on doit donc considérer une approche complètement indépendante de l'expérimentation matérielle. L'approche de Donald Knuth [Knuth 73] a alors été de calculer une mesure de la complexité d'un algorithme, afin d'obtenir un ordre de grandeur du nombre d'opérations élémentaires nécessaires pour que l'algorithme fournisse la solution du problème à l'utilisateur.

Théoriquement, tout problème d'optimisation peut être résolu par une énumération complète de toutes les solutions possibles de l'espace de recherche. De telles méthodes de résolution sont d'ailleurs appelées **méthodes énumératives** [Spielberg 79]. Cependant, sur certains problèmes, une énumération exhaustive amène à des algorithmes de complexité exponentielle (c'est-à-dire en $O(e^n)$), inutilisables en pratique.

Même si, pour certains problèmes, on connaît des algorithmes efficaces à complexité polynomiale (c'est-à-dire en $O(n^p)$), ce n'est pas généralisable à l'ensemble des problèmes. Nous pouvons donc conjecturer que certains problèmes sont par nature plus difficiles que d'autres, et par conséquent que leur résolution requiert des algorithmes de complexité plus élevée. Il est alors intéressant de définir une classification permettant de regrouper les problèmes ayant un même niveau de difficulté.

Il existe de nombreuses classes de problèmes dans la littérature, nous ne mentionnerons ici que les plus représentatives : les classes P et NP [Sipser 92] [Fortnow 09]. On dit qu'un problème est de classe P, s'il peut être résolu, de manière exacte, par un algorithme de complexité polynomiale. On peut citer par exemples les problèmes bien connus de tri de tableaux ou bien de recherche d'un sous-graphe connexe. La classe NP, quant à elle, regroupe les problèmes dont on peut vérifier que n'importe quelle proposition est bien une solution du problème, en un temps polynomial. On peut donc facilement en déduire que $P \subset NP$. La question qui fait encore débat aujourd'hui est de savoir si $NP \subset P$. Ceci prouverait qu'il existe des algorithmes polynomiaux permettant de résoudre n'importe quel type de problèmes. Or, la conjecture actuelle est plutôt de dire que $P \neq NP$. Ce qui impliquerait qu'il n'existe pas d'algorithmes polynomiaux pour résoudre les problèmes NP. De tels problèmes sont alors catégorisés comme difficiles et, à l'heure actuelle, leur résolution requiert des algorithmes à complexité exponentielle.

Une dernière classe de problèmes existe, et est particulièrement importante, il s'agit des problèmes NP-complets [Karp 72] [Johnson 85]. Ce sont les problèmes NP les plus difficiles, leur particularité est que tout problème NP peut être transformé en un problème NP-complet en un temps polynomial. Ces problèmes constituent donc le "noyau dur" des problèmes NP, si bien que si l'on trouvait un algorithme polynomial permettant de résoudre un seul problème NP-complet, on pourrait en déduire un autre pour tout problème NP. Parmi les problèmes NP-complets les plus connus, on peut citer : la clique maximum, le problème du sac à dos ou bien le problème SAT [Cook 71]. Ce dernier problème est particulièrement important, car Cook a démontré que tout problème pouvait se réduire à un problème de type SAT par une transformation en temps polynomial.

Pour ces problèmes d'optimisation difficile, seuls des algorithmes à complexité exponentielle ont été trouvés. Ainsi, en pratique, les temps de calcul des machines ne permettent pas de les résoudre et ne le permettront jamais. Par exemple, le problème du voyageur de commerce consiste à chercher le plus court chemin hamiltonien (circuit qui relie tous les sommets une seule fois) dans un graphe. L'algorithme naïf serait alors de déterminer la longueur de tous les chemins possibles et de prendre le plus court. Or, si l'on suppose que l'on a n sommets, on a alors $(n-1)!/2$ circuits possibles. Ainsi, si l'on prend par exemple $n = 250$, on obtient environ 10^{490} chemins possibles, plus que d'atomes dans l'univers. Même si l'on prend $n = 30$, aucun ordinateur, aussi puissant soit-il, ne pourra résoudre le problème avant plusieurs milliards d'années de calculs.

Cependant, pour des "petites" valeurs de n, ce problème, pourtant difficile, peut être résolu de manière exacte. L'explosion combinatoire ne survient qu'à partir d'une certaine dimension. On peut néanmoins diminuer la combinatoire à l'aide de méthodes arborescentes ou de la programmation dynamique mais, dans la plupart des cas, il faudra utiliser des méthodes approchées : les **heuristiques**. La figure 1.1 montre comment les différentes méthodes d'optimisation peuvent être classées en fonction du problème posé.

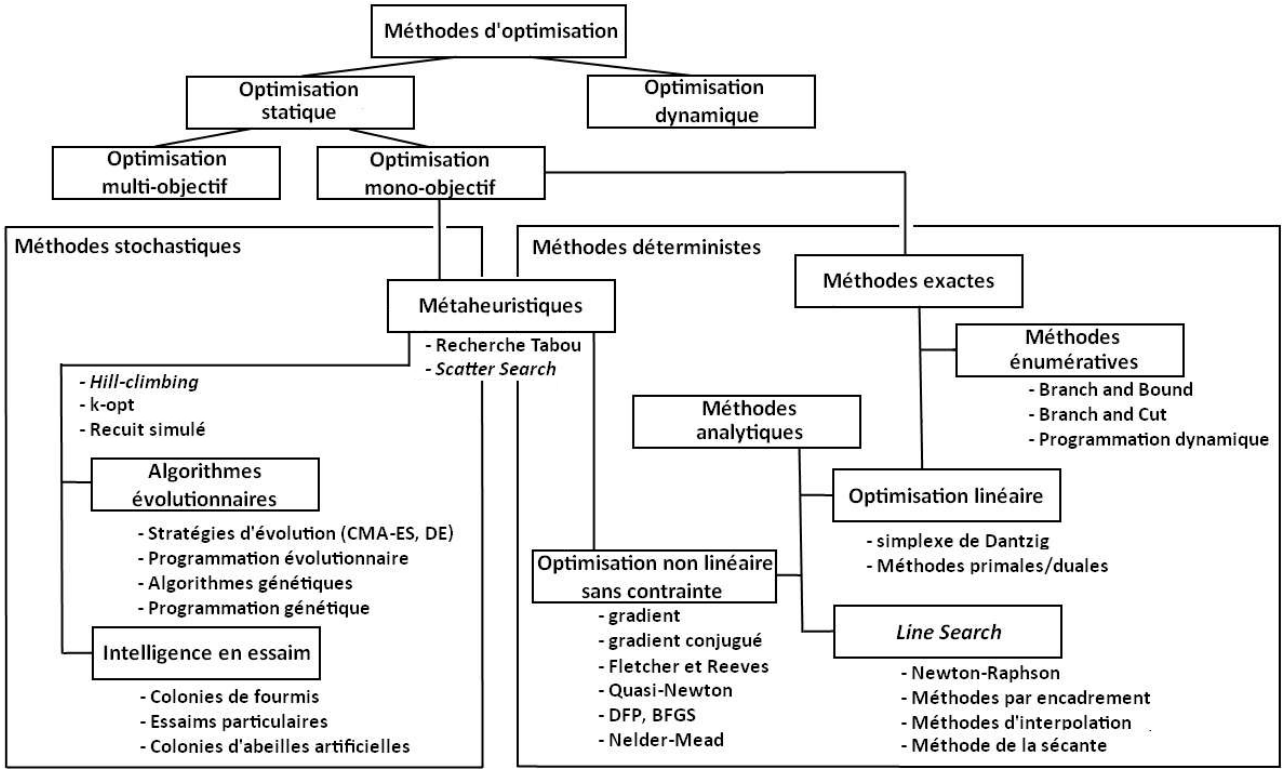


Figure 1.1 – Une classification possible des différents types de méthodes d'optimisation.

Dans la suite, nous détaillerons certaines de ces méthodes, afin d'avoir une approche globale des techniques d'optimisation existantes.

1.3 Méthodes exactes

Le terme de **méthodes exactes** [Dudzinski 87] [Kohl 95] regroupe l'ensemble des méthodes permettant d'obtenir la solution optimale d'un problème, en un temps "raisonnable". Elles s'opposent aux heuristiques, que nous verrons dans la section 1.7, car les méthodes exactes permettent d'obtenir théoriquement la solution optimale (à un epsilon près en raison de la précision numérique des calculs), et non une solution approchée.

Dans cette famille d'algorithmes, nous pouvons citer par exemple les méthodes de *backtracking*, aussi appelées méthodes de retour sur trace. Sans oublier les méthodes plus mathématiques, telles que la programmation linéaire, avec par exemple la méthode du simplexe de Dantzig [Dantzig 63].

D'autres méthodes exactes reposent sur le principe "diviser-et-régner". On décompose le problème en plus petits sous-problèmes, plus faciles à résoudre, et on combine les résultats jusqu'à arriver à résoudre le problème initial. Cette manière d'obtenir la solution optimale globale à partir des solutions optimales des sous problèmes est issue du principe d'optimalité de Bellman [Bellman 57] qui postule que "toute politique optimale est composée de sous-politiques optimales". Il est important de souligner que ce principe n'est applicable que dans le cadre de l'hypothèse de Markov [Howard 60], qui suppose qu'à chaque étape de décomposition du problème on ne puisse pas remettre en cause la solution optimale de chaque sous problème. Par exemple, lorsque l'on cherche le plus court chemin entre deux points d'un graphe, si le chemin le plus court entre A et B passe par C, alors le tronçon de A à C est le chemin le plus court entre A et C. Dans ce cas l'hypothèse de Markov est vérifiée. Cependant, lorsque l'on cherche le plus long chemin sans boucle entre deux points, si le chemin le plus long entre A et B passe par C, alors le tronçon de A à C n'est pas forcément le chemin le plus long entre A et C. Dans ce cas l'hypothèse de Markov ne s'applique pas et l'on ne peut pas résoudre le problème de cette manière.

Les méthodes qui en découlent sont assez nombreuses, on peut par exemple citer la méthode par séparation et évaluation (*branch and bound*) [Land 60] et ses dérivées (*branch and cut*, etc.) ou bien la programmation dynamique. L'analyse des propriétés du problème permet d'éviter l'énumération de larges classes de mauvaises solutions. Dans l'idéal, seules les solutions potentiellement bonnes sont donc énumérées. Pour être efficaces, ces méthodes exactes d'énumération implicite utilisent néanmoins des heuristiques (pour choisir les branchements, pour calculer des bornes, etc...). Nous pouvons également citer l'algorithme A* [Hart 68], qui permet de trouver un chemin entre deux sommets d'un graphe, et qui est très souvent utilisé dans le domaine de l'intelligence artificielle.

De par leur nature, les méthodes exactes ne peuvent donc s'appliquer qu'à des problèmes spécifiques [Sprecher 94] [Wolf 99].

1.4 Méthodes unidimensionnelles

Ces méthodes d'optimisation visent à traiter des fonctions objectifs à une seule variable. Par la suite, nous supposerons que nous sommes dans le cas d'une minimisation, nous devons donc minimiser $f(\lambda)$, avec $a < \lambda < b$, a et b étant les bornes de l'espace de recherche.

1.4.1 Méthodes par encadrement

Les méthodes unidimensionnelles les plus connues sont les techniques par encadrement, telles que la *golden section search* [Kiefer 53] [Press 07]. L'algorithme général de ces méthodes considère un intervalle initial $[\alpha, \beta]$ (par exemple $[a, b]$) et calcule $f(\alpha)$ et $f(\beta)$. On pose ensuite $\lambda_1 = \alpha + (1 - \tau)(\beta - \alpha)$ et $\lambda_2 = \alpha + \tau(\beta - \alpha)$ (dans le cadre de la *golden section search*, $\tau = 2/(1 + \sqrt{(5)}) \simeq 0{,}618$). On a donc $\alpha < \lambda_1 < \lambda_2 < \beta$, en fonction de l'ordre de $f(\lambda_1)$, et $f(\lambda_2)$, on peut donc calculer la nouvelle section :
– $[\alpha, \lambda_2]$ si $f(\lambda_1) < f(\lambda_2)$
– $[\lambda_1, \beta]$ sinon
L'algorithme se répète jusqu'à ce que la taille de la section atteigne une valeur seuil. Il est à noter qu'une autre version conserve l'information d'un triplet de points, plutôt que d'une paire. Ces méthodes par encadrement sont très facilement applicables car elles ne posent aucune hypothèse sur la fonction objectif. Elles sont particulièrement performantes dans le cas de fonctions unimodales.

1.4.2 Méthode de Newton-Raphson

Une méthode unidimensionnelle très connue est la méthode de Newton-Raphson [Newton 11]. Celle-ci démarre d'un point initial λ_0 et calcule le nouveau point par la relation :

$$\lambda_{i+1} = \lambda_i - \frac{f'(\lambda_i)}{f''(\lambda_i)} \tag{1.4.1}$$

L'inconvénient principal de cette méthode est qu'elle nécessite une fonction objectif doublement dérivable, telle que $\forall \lambda_i f''(\lambda_i) > 0$. Elle est cependant particulièrement utile dans le cas de l'optimisation d'une fonction localement convexe.

1.4.3 Méthode des sécantes

La méthode des sécantes [Wolfe 59] est une version modifiée de la méthode de Newton, elle ne nécessite plus d'hypothèse sur la dérivée seconde $f''(\lambda_i)$, qu'elle remplace par son approximation par différences finies $\frac{f'(\lambda_i) - f'(\lambda_{i-1})}{\lambda_i - \lambda_{i-1}}$.

1.5 Recherche linéaire

On appelle *line search* (ou *recherche linéaire*) [Grippo 86] [Moré 94] la classe des méthodes d'optimisation utilisant les méthodes unidimensionnelles comme principe de résolution de problèmes multidimensionnels. Les méthodes unidimensionnelles sont alors utilisées comme *routine* [Wächter 06].

Le principe de ces méthodes est de partir d'une solution x que l'on veut améliorer, et de calculer une direction d le long de laquelle on va chercher l'optimum. On obtient alors une fonction unidimensionnelle à optimiser par les techniques que nous avons vues précédemment. La recherche linéaire revient alors à trouver la valeur de α afin de minimiser $f(x + \alpha d)$.

Ces méthodes sont notamment utilisées dans les algorithmes du gradient, du gradient conjugué ou bien de l'algorithme de quasi-Newton.

1.6 Méthodes déterministes

Cette famille de méthodes d'optimisation regroupe l'ensemble des techniques de résolution de problème n'utilisant pas de concept stochastique. Elles peuvent se diviser en deux classes principales :

- **Les méthodes directes** : Se servent d'hypothèses sur la fonction objectif à optimiser. Par exemple, la fonction peut être continue et dérivable en tout point de l'espace de recherche. On peut ainsi calculer la direction de descente de la plus forte pente pour orienter la recherche.
- **Les méthodes indirectes** : Utilisent des principes mathématiques pour résoudre des systèmes d'équations linéaires, non linéaires, avec ou sans contraintes, etc.

Une méthode directe très connue est la méthode du gradient [Hestenes 52], que nous allons voir plus en détail.

1.6.1 Algorithme du gradient

Cette méthode est destinée à optimiser une fonction réelle continue et dérivable. C'est une technique de recherche locale déterministe. L'algorithme part d'une solution x_k qu'il veut améliorer, en plongeant vers le minimum local le plus proche. Pour cela, il calcule $\nabla f(x_k)$ le gradient de la fonction objectif au point x_k. Il obtient alors la direction de la plus forte pente $-\nabla f(x_k)$ sur laquelle il va avancer d'un pas α_k, dont l'optimum est estimé par une méthode de *line search* (cf section 1.5). La solution est ensuite remplacée par $x_{k+1} = x_k - \alpha_k \nabla f(x_k)$, et l'on recommence l'opération, jusqu'à ne plus pouvoir optimiser la solution.

Cette méthode converge globalement assez rapidement vers l'optimum mais nécessite de connaître la dérivée de la fonction objectif, ce qui est rarement le cas dans des problèmes d'optimisation réels et n'a de sens que pour les problèmes continus. De plus, dans certaines circonstances, elle nécessitera de nombreuses itérations avant d'obtenir une précision suffisante. Elle a alors un comportement similaire à une succession de rebonds sur les bords de l'optimum local. Pour éviter cela, de nombreuses améliorations ont été proposées, comme par exemple la méthode du gradient conjugué [Hestenes 52], qui permet d'accélérer la convergence de l'algorithme en l'évitant de partir en "zigzag". Ce dernier a par la suite été amélioré par les méthodes de Fletcher-Reeves [Fletcher 64] et Polak-Robiere [Polak 69].

1.7 Heuristiques et métaheuristiques

Nous avons vu dans la section 1.6 qu'une méthode d'optimisation pouvait être déterministe ou stochastique. Nous retrouvons également une opposition dans les termes "exacte" et "heuristique". En effet, l'utilisation de méthodes exactes n'est pas toujours possible pour un problème donné, par exemple à cause de temps de calcul trop important ou bien d'une séparation du problème impossible. Dans ces cas, nous utiliserons des méthodes approchées, appelées **heuristiques**. Il convient néanmoins de souligner qu'une méthode heuristique peut être déterministe ou stochastique.

Le mot heuristique vient du grec ancien eurisko ("je trouve") et qualifie tout ce qui sert à la découverte, à l'invention et à la recherche. Ces méthodes exploitent au mieux la structure du problème considéré dans le but de trouver une solution approchée, de qualité "raisonnable", en un temps aussi faible que possible. Typiquement, elles trouvent une solution approchée à

un problème NP en temps polynomial. On peut citer des heuristiques très simples comme les algorithmes gloutons [Cormen 90] [DeVore 96] ou les approches par amélioration itérative [Basili 75]. Le principe des méthodes gloutonnes est de faire une succession de choix optimaux localement, jusqu'à ce que l'on ne puisse plus améliorer la solution, et ce, sans retour en arrière possible. Ce principe, assez générique, doit être adapté en fonction de la structure du problème. Les heuristiques sont guidées par des spécificités liées au problème posé et en sont donc dépendantes.

Le terme **métaheuristique** a été inventé par Fred Glover [Glover 86] lors de la conception de la recherche tabou :

> *La recherche avec tabou peut être vue comme une "métaheuristique", superposée à une autre heuristique. L'approche vise à éviter les optimums locaux par une stratégie d'interdiction (ou, plus généralement, de pénalisation) de certains mouvements.*

Les métaheuristiques désignent un cadre général de résolution de problèmes NP [Reeves 93] [Glover 03] [Dréo 06]. Leur fonctionnement, au contraire des heuristiques, est donc indépendant du problème traité. Théoriquement, l'utilisation des métaheuristiques n'est pas vraiment justifiée et les résultats théoriques sont plutôt mauvais. Il existe quelques théorèmes de convergence pour la méthode du recuit simulé [Hajek 88] ou la méthode tabou [Faigle 92] [Glover 02] qui prouvent que la probabilité de trouver la solution optimale augmente avec le temps. On a alors la garantie de trouver la solution optimale si le temps tend vers l'infini. Or, ceci est inutilisable en pratique, puisque ce temps est supérieur au temps nécessaire pour énumérer l'ensemble des solutions de l'espace de recherche. Néanmoins, les performances pratiques de ces techniques sont excellentes.

Lorsque l'on veut résoudre un problème d'optimisation, on recherche la meilleure solution possible à ce problème, c'est-à-dire l'**optimum global**. Cependant, il peut exister des solutions intermédiaires, qui sont également des optimums, mais uniquement pour un sous-espace restreint de l'espace de recherche : on parle alors d'**optimums locaux**. Cette notion est illustrée sur la figure 1.2.

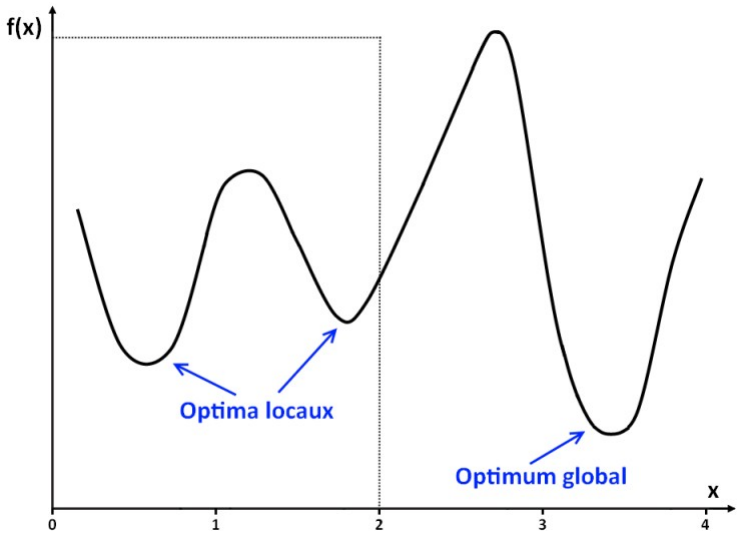


Figure 1.2 – Différence entre un optimum global et des optimums locaux.

Dans ce graphique, nous illustrons le cas d'une minimisation de la fonction objectif f à une variable (notée x). Si l'on considère le problème dans l'espace de recherche $x \in [0,4]$, on a

alors deux optimums locaux et un optimum global. Cependant, si l'on considère l'espace de recherche restreint à $x \in [0,2]$, on remarque que le premier optimum local devient l'optimum global. Tout dépend donc de l'espace de recherche dans lequel on se place. Le piège classique des métaheuristiques est de converger vers un optimum local, et de s'y trouver bloqué. Il est alors difficile de savoir si cet optimum est global, car toutes les solutions proches de celui-ci dégradent la fonction objectif.

Nous pouvons alors définir le **voisinage** d'une solution x, comme l'ensemble des solutions "proches" de x.
Considérons que l'espace de recherche est un espace métrique E, avec une mesure de distance d. On dit alors que l'ensemble V est le voisinage du point x s'il existe une boule ouverte B de centre x et de rayon r, contenue dans V, telle que :

$$B(x;r) = \{y \in E,\ \ d(y,p) < r\} \tag{1.7.1}$$

On peut alors définir un optimum local (relativement au voisinage V) comme la solution x^* de $\mathbb{S}$ (l'espace de recherche) telle que $f(x^*) \leq f(x), \forall x \in V(x^*)$.

L'**optimisation globale** fait référence à la recherche des optimums globaux de la fonction objectif. Les méthodes dites d'optimisation globale ont donc pour but de trouver l'optimum global de la fonction objectif, tout en évitant de rester bloquées dans les optimums locaux, elles s'opposent en cela aux **méthodes locales** [Aarts 97]. Il faut toutefois prendre en considération l'ambiguïté du terme de **recherche locale**. En effet, le terme de recherche locale est utilisé pour expliquer le principe des méthodes qui parcourent l'espace de recherche par améliorations successives d'une ou plusieurs solutions initiales. Ainsi, une méthode de recherche locale peut être une méthode d'optimisation globale, en ce sens qu'elle cherche l'optimum global de la fonction objectif, mais en parcourant l'espace de recherche par voisinages successifs de la solution courante.

Les méthodes d'optimisation peuvent alors être partagées en deux catégories : les **méthodes locales**, qui permettent de déterminer un minimum local (et donc différentes des méthodes de recherche locale) et les méthodes de **recherche globale**, qui s'efforcent de déterminer un optimum global. Or ces méthodes ne s'excluent pas mutuellement. Afin d'améliorer les performances d'une recherche, plusieurs auteurs [Glover 89a] ont pensé à combiner ces deux approches. L'algorithme résultant possède alors deux phases : l'**exploration** (aussi appelée "diversification") et l'**exploitation** (également appelée "intensification"). L'exploration permet à l'algorithme de rechercher de nouvelles solutions dans l'espace de recherche. Lors de cette phase, on cherchera des solutions dans des espaces encore non explorés afin de trouver d'éventuels optimums locaux. C'est cette phase qui permet à l'algorithme de ne pas être bloqué dans des optimums locaux, mais c'est également celle-ci qui est la plus difficile, car l'espace de recherche peut être très vaste, et donc impossible à parcourir en totalité. La phase d'exploitation, quant à elle, utilise les résultats obtenus lors de la phase d'exploration, afin de sélectionner le sous-espace de recherche le plus prometteur, et de "plonger" vers l'optimum local le plus proche. C'est dans cette phase que l'on essayera d'obtenir la meilleure précision possible pour une solution.

Les métaheuristiques basées sur ce principe cherchent donc le meilleur compromis entre l'exploration de l'espace de recherche et l'exploitation des résultats [March 91]. Pour cela, elles peuvent par exemple avoir un paramètre ajustant l'importance des phases d'exploration et d'exploitation. Une autre possibilité très souvent utilisée est d'intégrer dans une méthode d'optimisation deux méthodes distinctes, dont chacune sera adaptée à la résolution d'une des deux phases, on parle alors de **méthodes hybrides** [Roberts 91] [Renders 96].

1.8 Méthodes de recherche locale

Les méthodes de recherche locale [Aarts 97], aussi appelées algorithmes de descente ou d'amélioration itérative, partent d'une solution initiale, et ont pour but de l'améliorer. Le principe général de ces méthodes est le suivant : à partir d'une solution initiale x, dont on connaît la valeur de la fonction objectif $f(x)$, on cherche la meilleure solution x' dans le voisinage de x. Si l'on ne parvient pas à améliorer x, alors on s'arrête, sinon on remplace x par x', et on recommence.

Différentes méthodes de recherche locale existent dans la littérature, nous avons vu la méthode du gradient, qui est une méthode déterministe, mais qui nécessite de trop lourdes hypothèses sur la fonction objectif. Il existe un grand nombre de méta heuristiques de recherche locale ne demandant aucune hypothèse sur la fonction objectif, nous allons décrire les plus classiques.

1.8.1 Algorithme de descente

Cette méthode de recherche locale est l'une des plus simples de la littérature, elle est également appelée *hill climbing* dans les problèmes de maximisation. Son principe consiste, à partir d'une solution initiale, à choisir à chaque itération un voisin qui améliore strictement la fonction objectif. Il existe plusieurs moyens de choisir ce voisin, soit par le choix aléatoire d'un voisin parmi ceux qui améliorent la solution courante (*first improvement*), soit en choisissant le meilleur voisin qui améliore la solution courante (*best improvement*). Dans tous les cas, le critère d'arrêt est atteint lorsque plus aucune solution voisine n'améliore la solution courante.

Cet algorithme est également une méthode d'optimisation locale, c'est-à-dire qu'elle converge vers l'optimum local le plus proche et ne peut plus en sortir. Elle converge assez rapidement, mais nécessite des améliorations pour obtenir une précision intéressante. Elle est initialement destinée aux problèmes discrets et doit donc être adaptée dans le cadre de problèmes continus.

Une version améliorée de cet algorithme consiste à faire des redémarrages lorsqu'un optimum local est trouvé, en repartant d'une nouvelle solution générée aléatoirement. On parle alors d'algorithme de descente avec relance (*random-restart hill climbing*). L'algorithme n'est donc plus considéré comme une méthode locale, car l'exploration de l'espace de recherche se fait par les relances. Cependant, les solutions initiales étant générées aléatoirement, on ne tire aucune information des optimums locaux déjà trouvés. Il serait plus intéressant par exemple de ressortir de l'optimum local pour visiter un optimum local voisin. C'est le principe de la méthode de recherche tabou que nous verrons par la suite.

1.8.2 Algorithme k-opt

La méthode 2-opt est un algorithme de recherche locale proposé par Croes en 1958 [Croes 58] pour résoudre le problème du voyageur de commerce en améliorant une solution initiale. Son principe est très simple : à chaque itération elle permute aléatoirement la valeur de deux composantes du vecteur solution. Elle était initialement utilisée pour supprimer les "croisements" dans un chemin (une solution possible) effectué par le voyageur de commerce du problème éponyme.

Cette méthode a ensuite été étendue en 3-opt. Le principe est alors de supprimer 3 connexions aléatoirement dans le chemin du voyageur de commerce, puis de reconnecter le réseau de toutes

les manières possibles. La meilleure solution est conservée et le processus est ensuite répété pour différents sous-ensembles de 3 connexions.

Cette méthode a ensuite été généralisée en k-opt, c'est-à-dire en cherchant à permuter k variables à chaque itération. Mais, dans le cas général, le 3-opt est rarement dépassé. Il a d'ailleurs mené à la méthode de Lin-Kernighan [Lin 73], qui est l'un des algorithmes les plus efficaces pour la résolution du problème du voyageur de commerce.

La méthode k-opt est généralement utilisée dans des problèmes discrets d'ordonnancement et de planification de trajectoire, comme les problèmes de tournées de véhicules.

1.9 Méthodes d'optimisation locale

Les méthodes d'optimisation locale sont des algorithmes d'optimisation dont le but est de converger vers l'optimum local le plus proche. Elles s'opposent donc aux méthodes d'optimisation globale qui cherchent à explorer le plus possible l'espace des solutions afin de trouver l'optimum global.

1.9.1 Algorithme de Nelder-Mead

La méthode de Nelder-Mead [Nelder 65] est la méthode d'optimisation locale la plus courante. C'est une méthode directe à solutions multiples, c'est-à-dire que, lors de ses itérations, elle n'améliore pas seulement une solution, mais un ensemble de solutions candidates. Dans le cadre de cet algorithme, on appellera cet ensemble un "polytope", qui est une représentation géométrique de $(n+1)$ points (ou solutions), n étant la dimension du problème.

Les solutions qui composent ce polytope sont initialement choisies par le tirage aléatoire d'un point x_1 dans l'espace de recherche, les autres points x_i sont ensuite choisis de manière à former une famille, généralement orthogonale, telle que $x_{i+1} = x_1 + \lambda e_i, \forall i \in [1,n]$, où les e_i sont des vecteurs unitaires linéairement indépendants, et λ une constante adaptée à la caractéristique du problème (domaine de variation des différentes composantes). Dans certaines applications, il est néanmoins possible de choisir des λ_i différents pour chaque vecteur de direction.

À chaque itération de l'algorithme, les points x_i sont ordonnés de manière à avoir $f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$ (dans le cas d'une minimisation). De nouveaux points sont calculés en utilisant des transformations géométriques élémentaires (réflexion, contraction, expansion et rétrécissement). À chaque itération, le plus "mauvais" point (en termes de valeur de la fonction objectif) du polytope est remplacé par le meilleur des nouveaux points calculés. Le polytope se transforme donc au fur et à mesure des itérations. Il s'étend et se contracte, s'adaptant ainsi à l'allure de la fonction et convergeant vers l'optimum. L'algorithme s'arrête quand la différence entre le meilleur et le plus mauvais point du polytope devient inférieure à un certain seuil. On peut voir les différents effets géométriques sur une fonction à 3 dimensions (donc avec un polytope à 4 points), dans la figure 1.3. Nous supposons ici que le point x_4 est le moins bon, il sera donc remplacé par x'_4, si ce dernier point a une meilleure valeur de la fonction objectif.

Dans la suite, nous allons détailler davantage le fonctionnement de cette méthode. On commence par calculer un nouveau point x_G, centre de gravité des points $(x_1,...,x_n)$ du polytope (on exclut volontairement le point x_{n+1}, ayant les moins bonnes performances pour la fonction objectif).

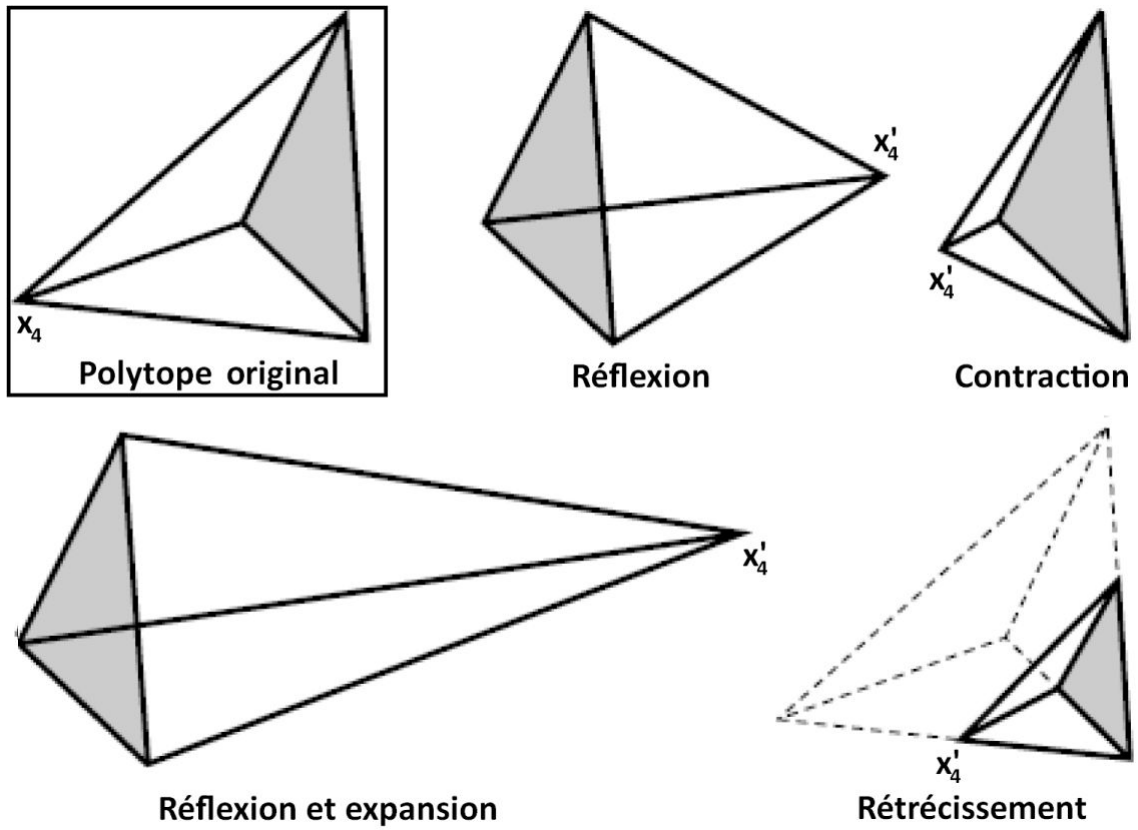


Figure 1.3 – Différents effets géométriques de la méthode du polytope de Nelder-Mead

Puis le simplexe commence sa transformation par une phase de **réflexion** : on calcule alors le nouveau point $x_r = (1+\alpha)x_G - \alpha x_{n+1}$, réflexion de x_{n+1} par rapport à x_G. On a alors deux possibilités :

- $f(x_r) < f(x_n)$: On commence alors la phase d'**étirement** (c'est-à-dire **réflexion** + **expansion**). On calcule alors le nouveau point $x_e = \gamma x_r + (1-\gamma)x_G$, et on remplace x_{n+1} par le meilleur des deux points x_r ou x_e.
- $f(x_n) \leq f(x_r)$: On commence alors la phase de **contraction**. Celle-ci peut être interne ou externe :
  - **contraction interne :** Si $f(x_{n+1}) \leq f(x_r)$, on calcule alors le nouveau point $x_c = \rho x_{n+1} + (1-\rho)x_G$, et on remplace x_{n+1} par x_c s'il est meilleur.
  - **contraction externe :** Si $f(x_r) < f(x_{n+1})$, on calcule alors le nouveau point $x_c = \rho x_r + (1-\rho)x_G$, et on remplace x_{n+1} par x_c s'il est meilleur.

Pour chacun de ces deux cas, la transformation géométrique marque la fin de l'itération. Sauf dans le cas (assez rare) où la contraction échoue à améliorer x_{n+1}. Dans ce cas, on commence une phase de **rétrécissement** en modifiant tous les points du polytope par similitude par rapport au meilleur point x_1. On remplace donc chaque x_i par $x_1 + \sigma(x_i - x_1)$.
Lorsque l'itération est terminée, on trie à nouveau les points pour avoir $f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$ et on commence une nouvelle itération, tant que la condition d'arrêt n'est pas vérifiée.

Les coefficients α, γ, ρ et σ sont les paramètres de la méthode décrivant les transformations géométriques. Par défaut, on fixe $\alpha = 1$, $\gamma = 2$ et $\rho = \sigma = \frac{1}{2}$.

1.10 Méthodes d'optimisation globale

1.10.1 Recherche stochastique (méthode de Monte-Carlo)

La méthode de Monte-Carlo a été développée par N. Metropolis et S. Ulam [Metropolis 49], alors qu'ils travaillaient sur le projet de la bombe atomique. Elle aurait été nommée ainsi en hommage au casino de Monte-Carlo, où l'oncle d'Ulam aurait souvent perdu son argent.

Cette méthode est la plus simple parmi l'ensemble des méthodes de recherche stochastique. Elle commence par générer une solution aléatoirement dans l'espace de recherche, puis évalue sa valeur par la fonction objectif. Ensuite, elle recommence et compare les deux résultats. Si le nouveau résultat est meilleur, elle le garde et ainsi de suite jusqu'à ce qu'un critère d'arrêt soit vérifié.

Utilisée telle quelle, cette méthode n'est pas très performante, elle fait uniquement une exploration de l'espace de recherche, sans essayer d'améliorer la qualité des solutions trouvées, ni d'utiliser les informations de ces solutions.

Un autre algorithme qui en découle est la **marche aléatoire**, qui cherche une nouvelle solution dans le voisinage de la première, plutôt que dans l'espace de recherche entier. Au contraire de la recherche stochastique, cette dernière est donc une méthode de recherche locale.

1.10.2 Recuit simulé

La métaheuristique du recuit simulé [Kirkpatrick 83] provient d'une analogie avec le processus de recuit utilisé en métallurgie, lui-même reposant sur les lois de thermodynamique énoncées par Boltzmann. La méthode de recuit en métallurgie consiste à alterner des cycles de chauffe et de refroidissement lent. Elle doit marquer des paliers de température suffisamment longs pour que le matériau atteigne l'"équilibre thermodynamique". Le but étant d'obtenir une structure régulière du matériau à l'état solide, comme dans les cristaux, et ainsi minimiser son énergie.

Dans le cadre d'un problème d'optimisation, la fonction objectif à minimiser est alors assimilée à l'énergie du système. On introduit également un paramètre fictif, la "température" T du système, que l'on fait décroître au fur et à mesure des itérations, afin de simuler le refroidissement. Dans le recuit simulé, en effet, T ne fait que décroître. Si l'on considère une solution x, on pourra alors l'assimiler à un état du système qui aura pour énergie $f(x)$.

Présentons le cas d'une minimisation de la fonction objectif. On démarre alors l'algorithme avec une solution unique x que l'on cherche à améliorer. On perturbe cette solution afin d'obtenir une nouvelle solution x' dans le voisinage de la première. Ensuite, on calcule l'écart des valeurs de la fonction objectif pour ces deux solutions : $\Delta f = f(x') - f(x)$. On se retrouve alors dans deux cas possibles :

- $\Delta f \leq 0$, la nouvelle solution est meilleure que la solution initiale, on la remplace donc : $x = x'$.
- $\Delta f > 0$, la nouvelle solution est moins bonne que la solution initiale. Cependant, on a tout de même la possibilité de la remplacer avec une probabilité d'acceptation $exp(\frac{-\Delta f}{T})$. Ce critère d'acceptation étant défini par Metropolis [Metropolis 53].

Si la nouvelle solution améliore le critère que l'on cherche à optimiser, on dit alors que l'on fait baisser l'énergie du système. L'acceptation d'une solution dégradant la fonction objectif permet

donc d'explorer une plus grande partie de l'espace des solutions et tend à éviter de se retrouver piégé dans un optimum local.

La température T du système influe directement sur la probabilité d'accepter une solution plus mauvaise. À une température élevée, la probabilité d'acceptation tendant vers 1, presque tous les changements seront acceptés. L'algorithme est alors équivalent à une marche aléatoire, que nous avons vue dans la section précédente. Cependant, la température étant diminuée lentement au fur à mesure du déroulement de l'algorithme, les mauvaises solutions seront de moins en moins souvent acceptées, et l'algorithme convergera donc vers l'optimum local le plus proche.

Le rôle de la température T au cours du processus de recuit simulé est très important. Une trop forte décroissance de température risque de piéger l'algorithme dans un optimum local. Un compromis souvent utilisé pour adapter la décroissance de la température à l'évolution du processus consiste à utiliser une variation logarithmique. Il existe également des méthodes permettant d'adapter la décroissance de la température au fur et à mesure des itérations, afin d'atteindre des seuils optimaux.

Il est intéressant de relever que le recuit simulé est une méthode de recherche locale (qui procède par voisins successifs), mais qui recherche l'optimum global de la fonction objectif, et est donc également une méthode d'"optimisation globale". Nous constatons donc encore une fois l'ambiguïté existant entre ces deux termes.

1.10.3 Recherche tabou

Cette méthode d'optimisation est la première à avoir porté le nom de "métaheuristique". Elle a été développée par Fred Glover en 1986 [Glover 86] [Glover 89a] [Glover 89b]. Elle peut être classée comme méthode de recherche locale à mémoire adaptative [Taillard 98]. En effet, sa caractéristique principale est de mémoriser des solutions, ou des caractéristiques de solutions, visitées durant la recherche, afin d'éviter de rester piégé dans des optimums locaux. Des algorithmes à mémoire adaptative pourront effectivement utiliser une procédure créant de nouvelles solutions à partir des informations mémorisées.

La recherche tabou consiste à guider une méthode heuristique de recherche locale, afin d'explorer l'espace de recherche au delà de l'optimalité locale. La caractéristique de mémoire adaptative crée en effet un comportement de recherche plus flexible et permet, par exemple, de choisir une solution **détériorant la fonction objectif** pour sortir d'un optimum local (si aucune solution voisine n'améliore la solution courante). La mémoire, appelée **liste tabou**, contient la liste des solutions (ou des zones) récemment visitées, et permet d'éviter de cycler et de retomber en permanence dans l'optimum local duquel on vient de sortir. Ce procédé simple permet alors de sortir de l'optimum local et de se diriger vers d'autres régions de l'espace des solutions. Pour résumer, on peut dire qu'à chaque itération, on choisit le meilleur voisin non tabou, même si celui-ci dégrade la fonction-objectif.

Différentes améliorations de cette méthode ont été mises au point. Dans le cas continu notamment [Chelouah 97], la sauvegarde d'une liste tabou de solutions n'est pas efficace. On peut alors considérer que la liste ne contient pas des solutions, mais des transformations, ou bien des voisinages entiers. De nombreuses autres améliorations sont possibles, la plus courante est l'ajout d'un **critère d'aspiration**. Celui-ci détermine si un élément de la liste tabou peut quand même être utilisé (on dit qu'on lève le statut tabou), car il remplirait certaines conditions désirées. Par exemple si un mouvement tabou conduit à une valeur de la fonction objectif

qui serait meilleure que celles obtenues jusqu'ici. D'autres critères d'aspiration plus complexes peuvent être envisagés, mais leur sur-utilisation a l'inconvénient de détruire en partie la protection offerte par la liste tabou vis-à-vis du cyclage.

Une autre amélioration intéressante de la recherche tabou est l'approfondissement des notions d'exploration et d'exploitation décrites dans la section 1.7. Ces phases sont alors appelées intensification et diversification, et se basent principalement sur l'utilisation d'une mémoire à long terme.

- L'**intensification** peut s'appliquer de différentes manières. L'algorithme peut, par exemple, mémoriser les caractéristiques communes des meilleures solutions trouvées lors de la recherche, afin de les combiner. Une autre application est de mémoriser une liste des meilleures solutions rencontrées, et de revenir vers cette "zone prometteuse" en l'améliorant. En effet, une hybridation classique de la recherche tabou est d'ajouter une heuristique d'optimisation locale pour améliorer les meilleures solutions visitées.
- La **diversification** dirige la recherche vers des zones inexplorées. En effet, l'inconvénient principal de la recherche tabou est qu'elle est basée sur un algorithme de recherche locale, qui explore donc l'espace de recherche de manière relativement limitée. Pour pallier à cela, on peut par exemple modifier temporairement la fonction objectif pour favoriser des mouvements non encore effectués ou bien pénaliser des mouvements souvent répétés. On peut également envisager de relancer la recherche aléatoirement à une position nouvelle, afin de couvrir une plus large partie de l'espace de recherche.

1.10.4 Algorithmes évolutionnaires

Les algorithmes évolutionnaires [Bäck 95] sont une famille d'algorithmes issues de la théorie de l'évolution par la sélection naturelle, énoncée par Charles Darwin en 1859 [Darwin 59]. L'évolution naturelle a en effet permis de créer des systèmes biologiques très complexes. Le principe fondamental étant que les individus les mieux adaptés à leur environnement survivent et peuvent se reproduire, laissant une descendance qui transmettra leurs gènes. Cette conclusion étant évidemment impossible si tous les individus avaient le même bagage génétique, on suppose donc que des variations non dirigées (mutations) du matériel génétique des espèces peuvent apparaître aléatoirement.

La clef étant l'adaptation des individus face à la pression de l'environnement, l'analogie avec l'optimisation devient claire. Cette adaptation peut alors être assimilée à une optimisation des individus afin qu'ils soient de mieux en mieux adaptés à leur environnement, au fur et à mesure des générations (correspondant aux itérations de l'algorithme). Nous pourrons alors définir les algorithmes évolutionnaires comme des méthodes faisant évoluer un ensemble de solutions appelé "population". Les solutions, appelées "individus", sont représentées par leur génotype, qui s'exprime sous la forme d'un phénotype. Afin d'évaluer la performance d'un individu, on associe au phénotype la valeur de la fonction objectif (ou fonction d'évaluation). Celle-ci se distingue de la fonction *fitness* qui représente l'évaluation d'un individu quant à sa survie dans la population. Cette méthode permet de s'assurer que les individus performants seront conservés, alors que les individus peu adaptés seront progressivement éliminés de la population. Dans son algorithme génétique binaire [Holland 75], Holland distingue clairement la fonction d'évaluation de la fonction *fitness*. Cependant, en général, on ne fait plus cette distinction et la fonction *fitness* est alors assimilée à la fonction objectif.

Un algorithme évolutionnaire se décompose en plusieurs étapes, chacune d'elles étant associée à un opérateur décrivant la façon de manipuler les individus :
- **Évaluation** : On calcule la *fitness* de chaque individu, par rapport à son phénotype.
- **Reproduction** :
  - **Croisement** : On croise une partie des individus de la population en "mélangeant" leurs génotypes selon l'opérateur de croisement défini. On obtient ainsi un ensemble de nouveaux individus appelés *offsprings* (descendants).
  - **Mutation** : Les descendants sont mutés, c'est-à-dire que l'on modifie aléatoirement une partie de leur génotype, selon l'opérateur de mutation.
- **Sélection** : On sélectionne une partie des descendants en fonction de leur *fitness*, afin de former une nouvelle population (typiquement de la même taille qu'au début de l'itération) qui formeront les "survivants" pour la prochaine génération. L'opérateur de sélection le plus simple consiste à prendre les meilleurs individus de la population, en fonction de leur *fitness*.

Dans le cadre des métaheuristiques, l'exploration de l'espace de recherche est alors réalisé par les opérateurs de mutation et assure la diversification des individus de la population (et donc des solutions). L'exploitation, quant à elle, est assurée par les opérateurs de croisement, qui recombinent les solutions, afin de les améliorer en conservant leurs meilleures caractéristiques. Dans les années 60 et 70, quatre grandes écoles utilisant ce principe général de façon différente furent développées indépendamment. Ces différentes familles d'algorithmes évolutionnaires ne différent que par la structure du génotype des individus ou par les opérateurs utilisés :
- **Les stratégies d'évolution** (ES) [Rechenberg 65] : Cette famille de méthodes est chronologiquement la première parmi tous les algorithmes évolutionnaires. Dans ces algorithmes, l'opérateur de mutation utilise une distribution normale pour ajouter une valeur aléatoire à chaque composante du génotype. Différentes variantes existent, en fonction du nombre de descendants que l'on veut générer et/ou garder à chaque itération. Soit μ le nombre d'individus de la population. À chaque itération, ρ individus sont sélectionnés et croisés afin de générer λ descendants. On reconstitue ensuite la population de départ en supprimant les individus les moins performants. Si l'on supprime uniquement les moins bons descendants, on note alors l'algorithme (μ/ρ,λ)-ES. Sinon, on peut supprimer les moins bons individus de l'ensemble total, après ajout des descendants, on note alors l'algorithme ($\mu/\rho + \lambda$)-ES. À l'origine, ces algorithmes n'utilisaient pas d'opérateur de croisement, et étaient alors notés (μ,λ)-ES ou ($\mu + \lambda$)-ES. Cet opérateur a ensuite été adopté afin d'éviter le piégeage dans des optimums locaux. Un exemple particulièrement connu est la méthode *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) dont la caractéristique principale est l'auto-adaptation de la matrice de covariance de la distribution normale utilisée pour la mutation.
- **La programmation évolutionnaire** (EP) [Fogel 66] : Un an après l'apparition des stratégies d'évolution, cette famille voit le jour dans le cadre de nouveaux concepts d'apprentissage en intelligence artificielle, pour la création d'automates à états finis. Chaque individu de la population est vu comme appartenant à une espèce distincte. Lors de la phase de reproduction, chaque individu génère donc un descendant.
- **Les algorithmes génétiques** (GA) [Holland 75] : Ce sont les algorithmes évolutionnaires les plus connus et les plus répandus. Ils ont notamment subi un grand élan lors de la parution du livre de Goldberg [Goldberg 89]. Ces algorithmes se détachent en grande partie par la représentation des données du génotype, initialement sous forme d'un vecteur binaire et plus généralement sous forme d'une chaîne de caractères.
- **La programmation génétique** (GP) [Koza 90] : C'est la dernière famille d'algorithmes évolutionnaires à avoir vu le jour. Elle a longtemps été assimilée à un sous-groupe des algorithmes génétiques, car elle n'en diffère que par la représentation des données. Le génotype est effectivement représenté sous la forme d'un arbre plutôt qu'une chaîne.

Parmi toutes ces familles d'algorithmes évolutionnaires, quelques algorithmes, particulièrement reconnus pour être efficaces, se détachent. Nous pouvons par exemple citer les algorithmes à **évolution différentielle** (*Differential Evolution*) [Storn 97] qui sont issus de la famille des stratégies d'évolution. La principale nouveauté apportée par cette méthode est son opérateur de mutation. En effet, à chaque itération, plutôt que de faire une reproduction des individus deux à deux, chaque individu est d'abord muté, puis croisé avec son mutant. La phase de mutation implique que, pour chaque individu de la population x, un nouvel individu x' est généré, en ajoutant à ses composantes une différence pondérée de deux autres individus pris aléatoirement dans la population. Le descendant final x'' est ensuite généré après croisement entre l'individu initial x et son mutant x'. La phase de sélection intervient juste après, par compétition entre l'individu père x et son descendant x'', le meilleur étant conservé pour la génération suivante. Ce processus est répété pour chaque individu de la population initiale, et même donc à la création d'une nouvelle population de taille identique.

D'autres méthodes sont également qualifiées d'algorithmes évolutionnaires, bien qu'elles n'en utilisent pas tous les concepts. C'est le cas par exemple des **algorithmes à estimation de distribution** (*Estimation of Distribution Algorithms* ou EDA), qui ont été proposés en 1996 par Mühlenbein et Paaß [Mühlenbein 96]. Ils ont une base commune avec les algorithmes évolutionnaires, mais s'en distinguent car le cœur de la méthode qui consiste à estimer les relations entre les différentes variables du problème. Ils utilisent pour cela une estimation de la distribution de probabilité, associée à chaque point de l'échantillon. La représentation de la population comme un ensemble de solutions est alors remplacée par une distribution de probabilité des choix disponibles pour chaque variable du vecteur solution (chaque chromosome d'un individu). Ces algorithmes n'emploient pas d'opérateurs de croisement ou de mutation, l'échantillon étant directement construit à partir des paramètres de distribution, estimés à l'itération précédente.

Les algorithmes évolutionnaires permettent donc la mise en place de stratégies d'optimisation efficaces, et sont très abondamment utilisés dans la littérature. Les méthodes les plus récentes, et probablement les plus efficaces à ce jour, sont les algorithmes à évolution différentielle, et la méthode CMA-ES.

1.10.5 Algorithmes d'intelligence en essaim

Ces méthodes d'optimisation, de la même manière que les algorithmes évolutionnaires, proviennent d'analogies avec des phénomènes biologiques naturels. La particularité des algorithmes issus de l'intelligence en essaim (ou *swarm intelligence*) [Bonabeau 99] est d'utiliser une population d'agents (alors appelée essaim) au lieu d'une simple population de solutions. Chaque agent de l'essaim pourra agir indépendamment des autres, il aura son propre système de décision, gouverné par un ensemble de règles. Un tel système est appelé système multi-agent [Ferber 99] [Weiss 99], et aboutit à l'émergence d'un comportement pour l'essaim entier (ici le comportement sera la convergence vers une solution du problème d'optimisation). On rejoint alors le domaine de la vie artificielle, qui postule qu'un comportement complexe peut émerger de règles simples, par la seule interaction des agents entre eux et avec leur environnement. Le système ainsi formé sera donc "auto-organisé" et avec un système de contrôle totalement décentralisé. Ce qui revient à dire que l'on ne prévoit pas le comportement qu'aura l'essaim, celui-ci "émergera" des règles simples imposées aux agents.
Les algorithmes de ce type les plus connus sont les méthodes de colonies de fourmis et d'optimisation par essaim particulaire. D'autres méthodes plus récentes commencent néanmoins à percer, on peut notamment citer les colonies d'abeilles artificielles [Pham 06].

1.10.5.1 Algorithmes de colonies de fourmis

L'idée initiale de cette méthode provient de l'observation du comportement des fourmis lorsqu'elles cherchent de la nourriture [Deneubourg 90]. En effet, celles-ci parviennent à trouver le chemin le plus court entre leur nid et une source de nourriture, sans pour autant avoir des capacités cognitives individuelles très développées. Après cette étude, un principe étonnant fut révélé : les fourmis communiquent indirectement via leur environnement (on parle alors de "stigmergie"), en déposant des phéromones sur le sol pour éclairer leurs comparses sur le chemin qu'elles ont déjà parcouru. Ainsi, lorsqu'elles reviennent au nid avec de la nourriture, les autres fourmis pourront suivre la piste de phéromones pour retrouver la source de nourriture. Mais, plus important, les phéromones s'évaporant avec le temps, les chemins les plus courts seront forcément ceux qui auront une intensité de phéromones plus forte. Ce qui, au final, mènera automatiquement les fourmis vers le chemin le plus court.

L'algorithme de colonie de fourmis artificielles (*Ant Colony Optimization* ou ACO) a été proposé par Marco Dorigo [Dorigo 92] lors de sa thèse de doctorat. Il était initialement destiné à la recherche de chemins optimaux dans un graphe. Il reprend la notion de système multi-agent, chaque agent étant une fourmi. Une fourmi parcourt alors le graphe de manière aléatoire, mais avec une probabilité plus importante de suivre une arête du graphe, en fonction de la quantité de phéromones déposée dessus. Lorsque le graphe est entièrement parcouru, elle laisse sur le chemin qu'elle a pris une quantité de phéromones proportionnelle à la longueur de ce chemin. La piste la plus courte sera donc de plus en plus renforcée, et donc de plus en plus attractive. L'exploration de l'espace de recherche se fait via un phénomène d'évaporation se produisant à chaque itération, et retirant une partie des phéromones présentes dans l'environnement. Sans ce dernier phénomène, l'algorithme serait facilement piégé dans un optimum local.

Dans les premières itérations, les phéromones sont déposées uniformément sur les arêtes du graphe et les fourmis se déplacent donc aléatoirement. Mais, au fur et à mesure des itérations, le phénomène d'évaporation et l'"intensification" des fourmis sur les arêtes des chemins les plus courts, fait disparaître les phéromones des arêtes les moins intéressantes. Ce processus continue jusqu'à ce que le plus court chemin (trouvé par les fourmis, ce n'est pas forcément l'optimum global) apparaisse comme une piste de phéromones.

L'algorithme s'est depuis diversifié, et permet maintenant de résoudre d'autres types de problèmes d'optimisation. Par exemple, une fourmi est assimilée à une solution possible du problème, et peut se déplacer dans l'espace de recherche [Socha 04].

1.10.5.2 Optimisation par essaim particulaire

Cette méthode est inspirée des travaux de C. Reynolds [Reynolds 87] et de Heppner et Grenander [Heppner 90]. Ces chercheurs se sont intéressés au mouvement de bancs de poissons ou de vols d'oiseaux, et ont créé des modèles mathématiques permettant de les simuler. Le modèle le plus connu est sans doute BOIDS, développé par C. Reynolds dans [Reynolds 87].

La méthode d'optimisation par essaim particulaire (*Particle Swarm Optimization* ou PSO) [Kennedy 95] est inspirée de ce modèle en y intégrant une dimension sociale importante, sur laquelle nous reviendrons. L'essaim particulaire correspond à une population d'agents appelés "particules". Chaque particule est modélisée comme une solution du problème ; elle possède une position (le vecteur solution) et une vitesse. De plus, les particules ont une mémoire leur permettant de connaître la meilleure position qu'elles ont trouvée au cours de leur "vie" (notée

p_{best}) et la meilleure position atteinte par les particules "voisines" (notée g_{best}). La notion de voisinage ne correspond pas ici au voisinage géométrique donc nous avions parlé, nous le définirons plus en détail dans la suite. À chaque itération, chaque particule se déplace dans l'espace de recherche en suivant un vecteur, calculé comme somme pondérée des vecteurs représentant sa vitesse courante (notée v_i), ainsi que sa p_{best} et sa g_{best} (cf équation (1.10.1) et figure 1.4). Sa nouvelle vitesse v_{i+1} est alors mise à jour, comme différence entre son ancienne x_i et sa nouvelle position x_{i+1} :

$$v_{i+1} = \omega * v_i + c_1 * r_1 * (p_{best} - x_i) + c_2 * r_2 * (g_{best} - x_i) \tag{1.10.1}$$

en désignant par ω le paramètre d'inertie, c_1 un paramètre déterminant l'influence de la composante cognitive et c_2 un paramètre déterminant l'influence de la composante sociale.

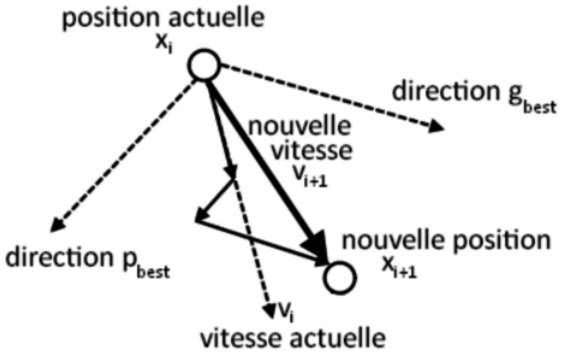


Figure 1.4 – Calcul de la nouvelle position et de la nouvelle vitesse d'une particule de l'algorithme PSO

Le point fondamental de la technique PSO, et qui est malheureusement souvent mis en aparté, est le moyen de communication utilisé par les particules afin de s'échanger leur p_{best}. Au contraire des colonies de fourmis, les particules ne communiquent pas par stigmergie, mais directement entres elles à l'aide d'un réseau social. La notion de voisinage n'existe alors plus comme proximité géométrique, un "voisin" d'une particule est un nœud voisin dans le graphe de son réseau social.
Différentes topologies de réseaux sociaux sont alors envisageables pour les particules (centralisée, distribuée, en étoile) et modifient le comportement de ces dernières.

De la même manière que pour les algorithmes de colonie de fourmis, chaque agent (ici des particules) est soumis à des règles de déplacement très simples, qui mènent cependant l'essaim à converger rapidement vers un optimum. C'est d'ailleurs un des points forts de cette méthode. En effet, elle s'avère particulièrement efficace pour trouver rapidement un optimum local, tout en ne nécessitant pas beaucoup de paramétrage. Cependant, elle peut se retrouver bloquée dans cet optimum, mais la modification de la topologie du réseau social permet souvent de résoudre le problème de convergence précoce.

De nombreuses variantes et hybridations ont été proposées, la plus populaire est sans doute Tribes [Clerc 03], qui supprime l'ensemble des paramètres de cette méthode, en y ajoutant la notion de "tribu".

1.11 Optimisation multiobjectif

Les métaheuristiques développées précédemment permettent de résoudre des problèmes mono-objectif. C'est-à-dire que la fonction objectif est définie pour maximiser ou minimiser un unique critère. On peut par exemple minimiser le coût d'un produit dans le cadre d'une production industrielle, ou bien maximiser les profits faits par une entreprise. Cependant, les problèmes réels ne sont pas toujours aussi "simples". En effet, il arrive parfois que l'on ait besoin de définir, pour un même problème, plusieurs objectifs à optimiser. Par exemple, on peut avoir besoin de minimiser le coût d'un produit tout en maximisant sa qualité. Dans ce cas, on a deux objectifs à optimiser en même temps, sachant qu'ils peuvent être contradictoires. Effectivement, dans notre exemple, le fait de minimiser le coût du produit aura un impact sur sa qualité, et inversement.

L'optimisation multiobjectif [Collette 02] fait partie des familles de méthodes d'optimisation combinatoire. Elle trouve ses origines au cours du XIXème siècle dans les travaux en économie d'Edgeworth [Edgeworth 85] et de Pareto [Pareto 96]. On peut formaliser un problème d'optimisation multiobjectif par la minimisation (ou maximisation) d'une solution $x = (x_1,...,x_n)$ par m fonctions objectifs, chacune optimisant un critère du problème (cf équation (1.11.1)).

$$f(x) = (f_1(x), f_2(x), ..., f_m(x)) \tag{1.11.1}$$

On dit alors que $f(x)$ est le vecteur de fonctions objectifs et f_i les objectifs à optimiser (ou critères de décision) avec $m \geq 2$ le nombre d'objectifs. On peut d'ailleurs ajouter au problème un ensemble de contraintes à satisfaire, de la même manière que pour les problèmes d'optimisation linéaire. On doit alors distinguer la notion d'espace de recherche, représentant l'ensemble des valeurs pouvant être prises par les variables, et la notion d'**espace réalisable**, sous-espace des variables satisfaisant les contraintes. L'image de l'espace de recherche par la fonction objectif f est appelée **espace des objectifs** (ou espace des critères). Ces notions sont représentées dans la figure 1.5.

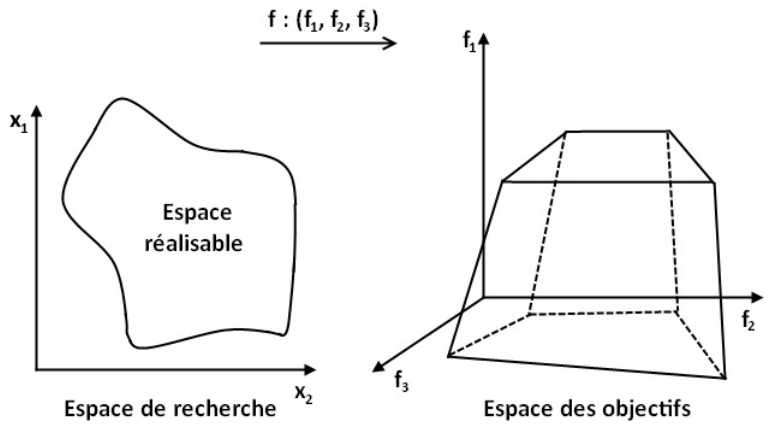


Figure 1.5 – Représentation de l'espace de recherche et de son image par la fonction multiobjectif.

On remarque assez rapidement que ce genre de problème n'a pas une seule solution optimale, mais potentiellement plusieurs, en fonction des objectifs que l'on suppose plus ou moins importants. Il est alors assez difficile d'établir une définition de ce que doit être l'optimum. On fait alors intervenir un **décideur**, qui va choisir, parmi les solutions possibles, celle qui lui semble la meilleure.

Le décideur peut intervenir dans différentes phases de l'optimisation. Les méthodes que nous allons présenter peuvent alors être classées en trois catégories (cf figure 1.6) :
- méthodes a priori : Le décideur intervient en amont de la méthode d'optimisation. Il peut par exemple pondérer les objectifs afin de décrire leurs importances.
- méthodes a posteriori : Le décideur intervient en aval de l'optimisation. Il dispose alors d'un ensemble de solutions et peut choisir celle qui lui semble la plus intéressante.
- méthodes interactives : Tout au long de la recherche, le décideur peut intervenir pour écarter ou favoriser des solutions.

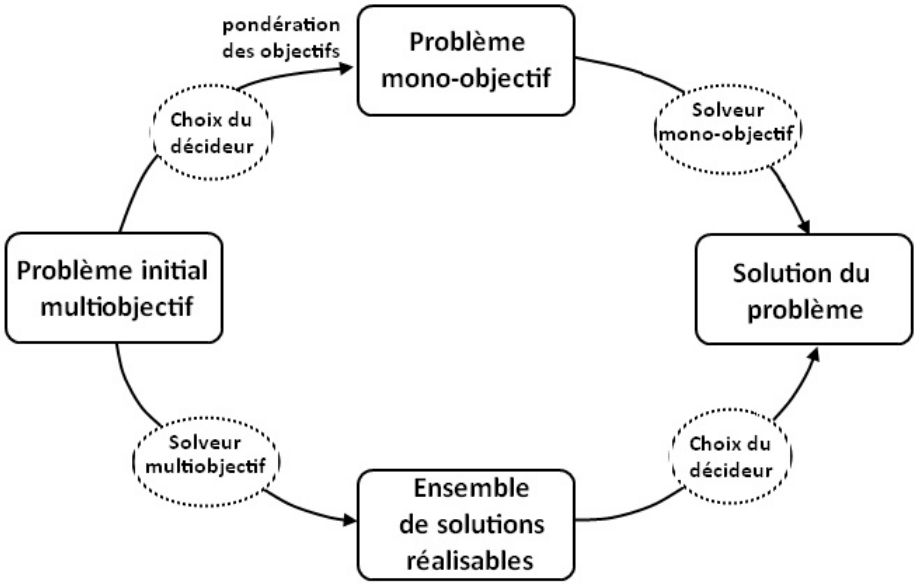


Figure 1.6 – Différents types de résolution d'un problème multiobjectif, en fonction de l'intervention du décideur.

L'optimisation de la fonction objectif $f(x)$, regroupant tous les critères, doit se faire par des méthodes différentes de celles que nous avons vu précédemment. Nous les détaillons dans la suite.

1.11.1 Méthodes agrégées

Ces méthodes reposent sur la connaissance du problème par le décideur. Il sait comment attribuer différentes importances aux critères à optimiser. On dit alors que le décideur optimise une "fonction d'utilité" $f = f(f_1,...,f_m)$.

Les méthodes d'agregation les plus simples utilisent une fonction de mise à l'échelle de chaque critère, afin de pouvoir les additionner (**modèle additif**) ou bien les multiplier (**modèle multiplicatif**). Les critères à optimiser peuvent alors être regroupés dans une unique fonction objectif.

Une autre technique très utilisée est la moyenne pondérée. Elle consiste à additionner tous les objectifs en leurs affectant un coefficient. Cette pondération représente l'importance relative que le décideur attribue à chaque objectif.

$$f(x) = \omega_1 * f_1(x) + \omega_2 * f_2(x) + ... + \omega_m * f_m(x) = \sum_{i=1}^{m} \omega_i * f_i(x) \tag{1.11.2}$$

Cette méthode a cependant plusieurs inconvénients. Tout d'abord, les poids doivent être attribués par le décideur avant la phase d'optimisation. Il ne sait donc pas à l'avance le type de résultat qu'il peut obtenir, ce qui rend cette pondération difficile. De plus, dans le cas où le front de Pareto (que nous définirons dans la section suivante) est non convexe, certaines solutions peuvent ne pas être accessibles en utilisant cette méthode.

Il existe également d'autres méthodes agrégées :
- Le *Goal programming* [Charnes 57] ou le min-max [Coello 95] : Dans ces deux méthodes, le décideur fixe un but G_i à atteindre pour chaque objectif f_i. L'optimisation revient alors à minimiser la somme des erreurs entre la valeur de chaque fonction objectif et les buts à atteindre.
- La méthode e-contrainte [Ritzel 94] : Cette méthode optimise un des objectifs en considérant les autres comme des contraintes.

L'avantage des méthodes agrégées est leur simplicité d'utilisation. En effet, l'optimisation multiobjectif est transformée en optimisation mono-objectif, que l'on sait résoudre par des méthodes d'optimisation classiques. Leur inconvénient principal est que le décideur doit avoir des connaissances approfondies du problème pour pouvoir paramétrer correctement la fonction objectif. De plus, il est souvent difficile, pour un décideur, de comprendre le sens de chacun de ces paramètres. Ce genre de méthodes requiert donc un nombre de tests important pour déterminer l'influence de chaque objectif.

1.11.2 Méthodes basées sur l'équilibre de Pareto

Ces méthodes sont les plus utilisées en optimisation multiobjectif. Elles reposent sur le postulat de V. Pareto [Pareto 96] : "Il existe un équilibre tel que l'on ne peut pas améliorer un critère sans détériorer au moins un des autres critères". En effet, comme nous l'avons déjà remarqué, on ne peut pas définir "la" valeur optimale d'un problème d'optimisation multiobjectif. Il existe donc un ensemble de valeurs optimales, que l'on appelle "Pareto-optimales", selon le critère de dominance au sens de Pareto :

Definition 1.11.1. *Soient* $\{f_i, i \in [1,m]\}$ *un ensemble de critères à minimiser, et* x *et* x' *deux solutions de l'espace réalisable. On dira que* x *domine* x' *au sens de Pareto si,* $\forall i \in [1,m]$, $f_i(x) \leq f_i(x')$, *avec au moins une inégalité stricte.*

Ce critère de dominance est la base des méthodes dites de Pareto. Une solution dite Pareto-optimale est une solution non dominée, c'est-à-dire qu'aucune autre solution de l'espace réalisable ne domine cette solution, selon le critère de dominance. On obtient ainsi un ensemble de solutions qui forment une frontière d'optimalité, que l'on nomme **front de Pareto** ou **surface de compromis** (cf figure 1.7).
Les solutions placées sur le front de Pareto ne peuvent pas être comparées, aucune n'étant systématiquement meilleure que les autres sur tous les objectifs. C'est le décideur qui aura pour rôle de choisir la solution à retenir.

De nombreux algorithmes ont été proposés pour résoudre ce type de problème, les plus populaires sont issus des algorithmes évolutionnaires que nous avons vus précédemment. En effet, au début des années 90, un nombre important d'algorithmes évolutionnaires ont été suggérés pour résoudre des problèmes d'optimisation multiobjectif. On peut par exemple citer les algorithmes *Multi Objective Genetic Algorithm* (MOGA) [Fonseca 93], *Niched Pareto Genetic Algorithm* (NPGA) [Horn 94] ou *Non dominated Sorting Genetic Algorithm* (NSGA) [Srinivas 95].

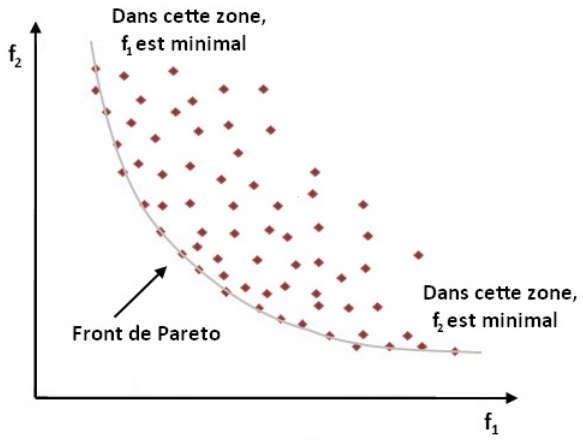


Figure 1.7 – Front de Pareto d'une fonction bi-objectifs.

Ces travaux ont notamment montré que des opérateurs supplémentaires étaient nécessaires pour convertir un algorithme évolutionnaire mono-objectif (EA) en un algorithme évolutionnaire multiobjectif (MOEA). Les deux composantes principales étant :
- affecter la *fitness* aux individus de la population en fonction d'un tri des solutions non dominées ;
- préserver la diversité parmi les solutions étant sur le même front.

Cependant, la convergence de ces algorithmes était souvent très lente. Pour pallier au problème, la notion d'élitisme fut introduite [Zitzler 00]. Son principe est de conserver dans la population, ou dans une mémoire adjointe, les meilleurs résultats trouvés, afin de les combiner et éventuellement de les améliorer.

Actuellement, la méthode la plus populaire est sans doute la méthode NSGA-II, qui a été présentée dans [Deb 00]. NSGA-II intègre un opérateur de sélection très différent de celui de NSGA [Srinivas 95], basé sur la distance de *crowding*, qui se calcule en fonction du périmètre formé par les points les plus proches d'une solution sur chaque objectif. De plus, NSGA et NSGA-II intègrent la notion d'élitisme, leur permettant une convergence rapide. Comparativement à NSGA, NSGA-II obtient de meilleurs résultats, sur toutes les instances présentées dans les travaux de K. Deb.

1.11.3 Méthodes non agrégées et non Pareto

D'autres méthodes n'utilisent aucun des deux principes énoncés précédemment. Elles possèdent un processus de recherche qui traite séparément les objectifs.
Par exemple, la méthode *Vector Evaluated Genetic Algorithm* (VEGA) [Schaffer 85] utilise le principe des algorithmes génétiques vus précédemment. La seule différence porte sur l'opérateur de sélection. La population initiale est divisée en m sous-populations (m étant le nombre d'objectifs), chacune de ces sous-populations étant sélectionnée en fonction d'un des objectifs. La population est ensuite reconstruite et l'on applique les opérateurs génétiques de croisement et de mutation. Ainsi, seuls les individus répondant le mieux aux différents objectifs sont conservés.

1.12 Cas particulier : les problèmes de grande dimension

1.12.1 Contexte

Comme nous l'avons vu en introduction de cette thèse, on admet dans la littérature que l'on peut qualifier de problème de grande dimension (ou problème à grande échelle) un problème dont le nombre de variables de décision est supérieur à 100. La plupart des méthodes heuristiques classiques s'effondrent lorsque l'on travaille en grande dimension, et ce phénomène est communément appelé "*curse of dimensionality*" [Bellman 61].

La "malédiction de la dimension" explique que les problèmes de grande dimension sont très difficiles à résoudre, même en utilisant des métaheuristiques, et ce pour différentes raisons :
- plus il y a de variables et plus le bruit est important. Ce qui implique que l'on augmente également les erreurs ;
- il n'y a pas assez d'observations possibles par rapport au nombre total de possibilités pour avoir une bonne estimation de la solution.

De plus, les caractéristiques d'un problème peuvent changer avec l'échelle. Par exemple, la fonction de Rosenbrock [Rosenbrock 60] est unimodale (un seul optimum local) pour des problèmes à 2 dimensions, mais devient multimodale (plusieurs optimums locaux) au-delà. Ceci implique qu'une métaheuristique efficace pour résoudre un problème de faible dimension peut devenir totalement inefficace en grande dimension, car elle fait face à un tout nouveau type de problème [Shang 06].

Pour illustrer cela, nous pouvons supposer que notre espace de recherche est de dimension 1, on peut donc l'assimiler à une droite. Lorsque l'on cherchera un optimum, il nous faudra chercher des solutions sur cette droite. Pour explorer cet espace de recherche, on pourrait par exemple le diviser en p segments et générer une solution au centre de chaque segment. On aurait ainsi une approximation des solutions réparties uniformément dans l'espace de recherche. Ce qui peut s'avérer important dans le cas de fonctions complexes, afin de favoriser l'exploration de l'espace de recherche, et ainsi d'éviter de tomber dans des optimums locaux.
Supposons maintenant que l'on veut explorer de la même manière un espace de dimension 2. Il faudra alors générer une grille de $p \times p$ cases, et générer une solution au centre de chaque case. Par récurrence, on peut généraliser ce principe, il faudra donc p^n solutions à générer si l'on veut conserver la même densité de solutions dans l'espace de recherche. On remarque donc que l'exploration de l'espace de recherche augmente exponentiellement avec la dimension, rendant la recherche trop coûteuse en temps de calcul, sitôt que la dimension est trop élevé.

Ce problème est un obstacle majeur en optimisation, puisque de nombreuses métaheuristiques ne sont pas du tout robustes à l'augmentation de la dimension, comme par exemple la méthode de Monte-Carlo ou bien l'algorithme du simplexe de Nelder-Mead. Le travail récent de Hvattum [Hvattum 09] compare la vitesse de convergence de simples métaheuristiques de recherche locale sur différentes fonctions unimodales (un seul optimum). Une partie de ces résultats a été regroupée dans le tableau 1.1 pour quatre méthodes de recherche locale : le simplexe de Nelder-Mead [Nelder 65], la méthode *Multi-Directional Search* [Torczon 89], l'algorithme de Rosenbrock [Rosenbrock 60] et la méthode de Solis et Wets [Solis 81]. Ce tableau représente les résultats de ces quatre méthodes de recherche locale sur la fonction *Sphere*. Cette fonction sera définie plus en détail par la suite, mais elle est considérée comme l'une des fonctions objectifs les plus simples à résoudre, de par sa nature séparable (les variables sont indépendantes) et unimodale.

n	Nelder-Mead	*Multi-directional search*	Algorithme de Rosenbrock	Solis et Wets
2	67,2	31,0	25,3	55,3
4	283,2	126,6	64,9	89,0
8	(0,9)	524,2	156,0	172,3
16	(0,2)	2385,0	371,8	373,5
32	(0,0)	10535,4	1082,8	784,8
64	(0,0)	46657,0	(0,9)	1602,2
128	(0,1)	(0,0)	(0,7)	3515,3
256	(0,0)	(0,0)	(0,3)	7401,3
512	(0,0)	(0,0)	(0,0)	15770,9

Tableau 1.1 – Nombre moyen d'évaluations de la fonction objectif *Sphere*, pour n variables. Source : [Hvattum 09]

L'expérience est renouvelée 10 fois et la moyenne du nombre nécessaire d'évaluations de la fonction objectif est calculée et portée dans le tableau. Les chiffres entre parenthèses correspondent à la proportion de simulations ayant correctement trouvé l'optimum global parmi les 10 essais. La simulation s'arrête si 50000 évaluations de la fonction objectif ont été atteintes, ou bien si un x est trouvé tel que $f(x) < \varepsilon$, où $\varepsilon = 0.001$. On remarque que la vitesse de convergence chute rapidement à mesure que la dimension augmente. Il faut de plus en plus d'évaluations de la fonction objectif pour que les métaheuristiques de recherche locale atteignent l'optimum avec une précision suffisante. On remarque en particulier que la méthode de Nelder-Mead, pourtant très souvent utilisée, devient inefficace très rapidement. La méthode de Solis et Wets obtient les meilleurs résultats sur cette fonction, cependant la seule étude de la convergence ne permet pas de conclure sur l'efficacité globale d'une méthode. En particulier, cette méthode n'obtient pas d'aussi bon résultats sur les autres fonctions proposées par [Hvattum 09].

En voyant ces résultats, pourtant obtenus uniquement via une recherche locale sur une fonction unimodale, donc ne nécessitant aucune exploration de l'espace de recherche, on peut rapidement conclure à la nécessité de développer des métaheuristiques robustes à l'augmentation de la dimension.

1.12.2 Algorithmes existants pour les problèmes de grande taille

Cette thèse fut commencée en 2008, période où les problèmes de grande taille n'étaient pas encore beaucoup référencés dans la littérature. Les méthodes existantes à cette époque étaient, pour la plupart, des adaptations de métaheuristiques standards qui, de par leur structure, étaient assez robustes pour ce type de problèmes.

La méthode *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) [Auger 05], présentée dans la section 1.10.4, a été adaptée pour les problèmes de grande taille [Ros 08], grâce à une "astuce" lui permettant d'accélérer sa convergence. L'amélioration consiste à contraindre la matrice de covariance à être diagonale, afin de gérer chaque dimension indépendamment. Ceci réduit la complexité du modèle et donc réduit les performances sur les problèmes non séparables mais, en accélérant la convergence, permet à la méthode résultante, sep-CMA-ES, d'améliorer les résultats pour des dimensions supérieures à 100, surtout pour les problèmes séparables.

Une autre méthode issue des algorithmes évolutionnaires a été adaptée afin de résoudre des problèmes de grande taille. Il s'agit de la *Differential Evolution* (DE) [Storn 97] présentée dans 1.10.4. Son point faible étant son paramétrage difficile, de nombreux papiers visent à réduire le nombre de paramètres, ou bien à les rendre auto-adaptatifs [Bäck 02]. Plusieurs adaptations de cette méthode ont été faites afin de la rendre robuste aux problèmes de grande taille. Nous pouvons par exemple citer le travail de Gao [Gao 07], qui ajoute une méthode d'initialisation, visant à disperser les individus de la population dans l'espace de recherche, ainsi qu'une méthode de recherche locale, afin d'accélérer la convergence. Une autre variante, nommée jDEdynNP-F [Brest 08], utilise une technique permettant d'auto-ajuster deux des paramètres (nommés CR et F) lors de l'optimisation. Elle utilise également une méthode pour réduire graduellement la taille de la population durant le processus d'optimisation, ce qui permet à l'algorithme de converger plus rapidement.

Des études ont également été menées sur l'algorithme *Particle Swarm Optimization* (PSO). Nous pouvons par exemple citer la variante nommée *Efficiency Population Utilization Strategy for PSO* (EPUS-PSO) [Hsieh 08]. Cet algorithme introduit trois nouvelles notions, afin d'accélérer la convergence de PSO :
- une gestion de la population dynamique, qui lui permet d'ajouter ou de supprimer des particules, pour améliorer l'exploration de l'espace de recherche, en l'adaptant au problème ;
- un système de partage des informations stochastiques, permettant aux particules de s'échanger leurs p_{best} ;
- une stratégie de recherche améliorée, appelée *Searching Range Sharing*, permettant aux solutions de ne pas être bloquées dans des optimums locaux. Cette stratégie utilise des méthodes de recherche locale sur les particules, en perturbant leur solution sur différents sous-espaces de recherche.

Plus récemment, une nouvelle adaptation de PSO pour ce genre de problème a été proposée [García-Nieto 10]. Dans cet article, on constate que l'ajout d'une procédure de "restart" pour les particules, ainsi qu'une vitesse adaptative, rendent l'algorithme plus robuste à l'augmentation de la dimension.

D'autres métaheuristiques ont également été adaptées pour les problèmes de grande taille. C'est le cas des algorithmes à estimation de distribution, avec l'algorithme LSEDA-gl [Wang 08]. Celui-ci a également choisi de réduire la complexité de son modèle probabiliste en considérant les variables indépendamment.

Cependant, lors de la conférence CEC'08 axée sur ces problèmes, la méthode ayant obtenu les meilleurs résultats était une méthode totalement nouvelle : *Multiple Trajectory Search* MTS [Tseng 08]. Cette méthode utilise un ensemble de solutions, qu'elle génère uniformément sur l'espace de recherche. Le reste de l'algorithme consiste à effectuer des recherches locales sur chacune de ces solutions. Trois algorithmes de recherche locale peuvent être utilisés :
- le premier, nommé LS1, fait une recherche le long de chaque dimension prise séparément, de la première à la dernière ;
- LS2 fait la même chose que LS1, mais en restreignant au quart la longueur de l'espace de recherche de chaque dimension ;
- LS3 a un comportement un peu différent. Il considère trois "petits" mouvements le long de chaque dimension et détermine ensuite de manière stochastique le mouvement qui sera réellement effectué sur cette dimension. Ainsi, une seule évaluation de la fonction objectif est faite après avoir parcouru toutes les dimensions

Pour connaître la recherche locale la plus adaptée parmi les trois, on les teste successivement, et celle qui parvient à optimiser le mieux la solution est retenue.

Ce dernier algorithme, pourtant relativement simple, obtient des résultats réellement compétitifs sur la plupart des fonctions objectifs proposées.

1.13 Problèmes de test

1.13.1 Présentation

Comme nous l'avons vu précédemment, il est très important de pouvoir comparer différents algorithmes d'optimisation, afin de pouvoir choisir le plus performant pour un problème donné. Afin d'avoir une base commune, mais également pour avoir un ensemble représentatif d'une grande partie des problèmes que l'on peut retrouver dans la réalité, des protocoles de tests génériques ont été mis en place. Ces jeux de tests, appelés *benchmarks*, sont utilisés pour évaluer les performances et les capacités de convergence des algorithmes d'optimisation. Ils regroupent différentes fonctions objectifs et des protocoles de test afin de pouvoir comparer les résultats obtenus par la communauté scientifique. Davantage d'informations concernant les *benchmarks* utilisés dans ce manuscrit sont consignées en annexe (cf annexe A.1).

Les plus répandus sont ceux élaborés dans le cadre de conférences. En effet, un grand nombre de méthodes ont alors pu être testées sur ces problèmes, ce qui facilite leur comparaison. On peut citer par exemple les *benchmarks* issus des conférences *Congress on Evolutionary Computation* (CEC) [Suganthan 05] ou *Black Box Optimization Benchmarking* (BBOB) [Hansen 09]. Dans ces *benchmarks*, les optimums globaux sont connus à l'avance. À la fin d'une simulation, il est donc possible d'évaluer l'erreur commise par l'algorithme. Les *benchmarks* sur lesquels nous travaillons sont dédiés à la "*black-box optimization*", c'est-à-dire que nous n'avons aucune information sur la nature de la fonction objectif, nous ne pouvons donc pas calculer sa dérivée. Nous proposons une solution à la "boîte noire" qu'est la fonction objectif inconnue, et celle-ci nous renvoie une valeur associée.

Pour tous ces *benchmarks*, les critères d'évaluation sont les suivants :
- Mesure quantitative de la performance : Plutôt que d'utiliser le temps d'exécution, qui est dépendant de la machine sur laquelle l'algorithme est lancé, on calcule le nombre d'évaluations de la fonction objectif qu'il a fallu faire pour atteindre une précision donnée. On peut inversement fixer le nombre d'évaluations de la fonction objectif et calculer l'erreur obtenue. De plus, pour éviter d'obtenir un résultat biaisé, on lance l'algorithme plusieurs fois sur la même fonction objectif et on calcule une moyenne.
- Invariance : On vérifie que l'algorithme est robuste aux translations ou à l'augmentation de la dimension pour une même fonction objectif.
- Paramètres : Cette partie est indépendante des *benchmarks*, mais néanmoins très importante lors de l'évaluation d'un algorithme. Elle est malheureusement rarement pris en considération. Le temps de paramétrage peut parfois être très important, limitant considérablement l'intérêt de la méthode. Certains algorithmes incluent d'ailleurs des procédés de paramétrage automatique pour simplifier cette démarche.

1.13.2 Théorème du "*No free lunch*"

Un point important est que la comparaison des performances d'algorithmes d'optimisation ne peut pas être mesurée par le nombre de problèmes qu'ils résolvent le mieux. Effectivement, le théorème du "*no free lunch*" [Wolpert 96] montre qu'il ne peut pas exister une méthode résolvant tous les types de problèmes. Ainsi, lorsque l'on compare deux algorithmes d'optimisation avec toutes les fonctions objectifs possibles, les performances des deux algorithmes seront semblables en moyenne. Ceci a pour conséquence directe de rendre non pertinente la construction d'un *benchmark* "parfait", qui contiendrait tous les types de fonctions possibles, afin de tester si un algorithme est meilleur pour toutes les fonctions. C'est pourquoi, lorsque l'on teste les performances d'un algorithme, nous devons vérifier le type de problème qu'il résout le mieux, afin de caractériser les problèmes auxquels il s'applique.

1.13.3 Catégorisation des fonctions objectifs

Les fonctions proposées par les *benchmarks* peuvent être catégorisées en trois parties :
- Les fonctions unimodales : ce sont, la plupart du temps, les fonctions considérées comme "faciles". Elles n'ont qu'un seul optimum local (qui est donc l'optimum global). Une simple méthode d'optimisation locale permet de les résoudre.
- Les fonctions "faiblement multimodales" : ces fonctions possèdent quelques optimums locaux, mais en densité relativement faible par rapport à la taille de l'espace de recherche. Elles sont plus difficiles à résoudre que les premières, car il faut explorer l'espace de recherche suffisamment pour trouver le bon optimum.
- Les fonctions "hautement multimodales" : ces fonctions sont souvent les plus difficiles à résoudre, car elles contiennent un "grand" nombre d'optimums locaux. Il est alors difficile de trouver celui qui est l'optimum global.

Une autre caractéristique importante des fonctions contenues dans les *benchmarks* est leur séparabilité. On dit en effet qu'une fonction est séparable si l'on peut la minimiser (ou la maximiser) en minimisant (respectivement maximisant) ses composantes prises séparément. Les fonctions non séparables sont donc plus difficiles à optimiser, car leurs variables peuvent être interdépendantes, ce qui implique que l'on ne peut pas traiter chaque composante séparément.

1.13.4 Exemple : la fonction *Sphere*

La fonction *Sphere* [Rechenberg 73] [De Jong 75], dont nous avons déjà parlé, correspond à l'équation suivante :

$$f(x) = \sum_{i=1}^{n} {x_i}^2 \tag{1.13.1}$$

en désignant par n la dimension du vecteur solution x. Dans les *benchmarks*, les fonctions objectifs sont souvent représentées graphiquement dans le cas où $n = 2$: la figure 1.8 montre notre fonction *Sphere*. L'espace de recherche est ici limité à $[-6{,}6]$ pour chaque x_i, la troisième composante est donc l'image de x par notre fonction objectif ($f(x)$).

Un descriptif complet de toutes les fonctions objectifs utilisées dans ce manuscrit est fourni dans l'annexe A.1.

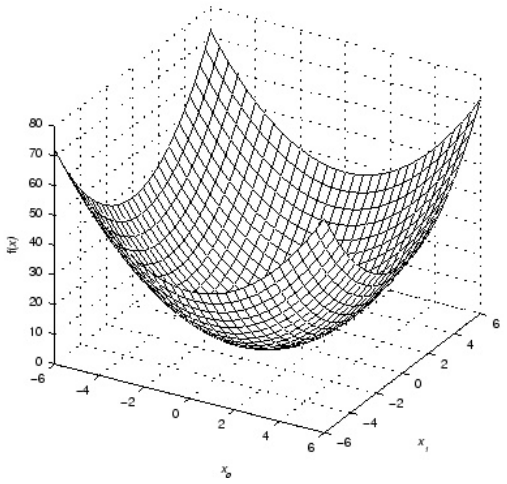


Figure 1.8 – Représentation de la fonction *Sphere* pour n = 2

1.13.5 *Benchmarks* pour les problèmes de grande dimension

L'engouement récent pour les problèmes de grande taille a abouti à la création de *benchmarks* spécialisés. Par exemple, à deux reprises, la conférence CEC a proposé des *benchmarks* dans ce domaine [Tang 07] [Tang 09]. De même, la conférence ISDA a consacré une session spéciale à ces types de problèmes en 2009 [Herrera 10a], ce qui a abouti à la création d'un autre *benchmark*, ainsi que de plusieurs nouvelles méthodes d'optimisation adaptées à ces problèmes. La particularité de ces *benchmarks* est qu'ils ne contiennent que des fonctions dont on peut adapter la taille (c'est-à-dire la dimension). Il est alors possible d'évaluer les algorithmes d'optimisation avec des dimensions allant de 1 à 1000, afin de vérifier leur robustesse.

1.14 Conclusion

Les méthodes d'optimisation existent donc sous diverses formes, chacune étant adaptée à un type de problème particulier. Certains problèmes difficiles ne peuvent d'ailleurs pas être résolus de manière optimale par une machine. Pour pallier ce problème, des techniques ont été présentées, et notamment des méthodes de la famille des métaheuristiques. Ces algorithmes d'optimisation utilisent un processus de recherche stochastique combiné avec des méthodes d'intensification, permettant d'obtenir une solution approchée du problème en un temps "raisonnable". Il en existe un très grand nombre, et nous nous sommes attachés à présenter les plus représentatifs et les plus courants.

En particulier, nous avons vu que ces métaheuristiques, bien que très efficaces pour des problèmes réputés difficiles, étaient sujettes à la "malédiction de la dimension", c'est-à-dire qu'elles peuvent devenir rapidement inefficaces, sitôt que la dimension devient trop importante. Ce problème est assez récent, car il concerne principalement des applications en *Data* ou en *Web Mining*, qui se sont principalement développés depuis la propagation d'Internet et le développement des nouvelles technologies.
Nous avons alors présenté des méthodes récentes permettant de résoudre ces problèmes d'optimisation très spécifiques. En particulier, nous avons montré les particularités qui les rendaient robustes à l'augmentation de la dimension.

Dans le chapitre suivant, nous verrons trois nouvelles méthodes développées durant cette thèse et adaptées particulièrement aux problèmes de grande taille.

Chapitre Deux

Élaboration d'algorithmes d'optimisation adaptés aux problèmes de grande dimension

2.1 Introduction

Nous avons vu dans le chapitre précédent que les métaheuristiques actuelles n'étaient pas toujours adaptées aux problèmes de grande dimension. Cependant, il est possible d'augmenter leur robustesse en modifiant certaines parties de leur fonctionnement.

Dans les exemples d'amélioration que nous avons vus, la plupart des méthodes d'optimisation modifient leur structure de façon à augmenter leur vitesse de convergence. En effet, la *"curse of dimensionality"* implique que les temps de convergence de ces algorithmes augmentent exponentiellement avec la dimension.

Les stratégies d'adaptation des algorithmes, afin d'améliorer leur convergence, ne sont pas toujours triviales. Les méthodes sep-CMA-ES [Ros 08] et LSEDA-gl [Wang 08] simplifient l'algorithme initial en traitant les dimensions séparément. D'autres méthodes ajoutent des processus de recherche locale [Gao 07] [Hsieh 08], ou bien réduisent la complexité du modèle initial [Brest 08]. L'algorithme MTS [Tseng 08], quant à lui, introduit une nouvelle technique fondée sur trois algorithmes de recherche locale, chacun d'eux traitant les dimensions séparément.

Pour adapter les algorithmes d'optimisation aux problèmes de grande dimension, les méthodes employées visent à réduire la complexité de l'algorithme, ou bien à limiter sa capacité d'exploration. Nous pouvons donc conjecturer, d'après le théorème du "*No Free Lunch*" énoncé précédemment, qu'une métaheuristique d'optimisation en grande dimension ne pourra résoudre ces problèmes qu'en contrepartie d'une perte dans les capacités initiales de l'algorithme. Par exemple, le fait de traiter les dimensions séparément est une forme de simplification du modèle, les métaheuristiques résultantes auront alors plus de difficultés à résoudre des problèmes plus "difficiles" (multimodaux ou non séparables), même en faible dimension. Dans ce cas, il faudrait faire un compromis entre la complexité du modèle et l'efficacité qu'aura l'algorithme en grande dimension.

Ce chapitre présente trois méthodes issues des réflexions présentées dans cette introduction. Elles seront comparées à d'autres méthodes d'optimisation en haute dimension afin d'estimer leurs performances.

2.2 HeurisTest : un outil de *benchmarking* pour l'optimisation

Avant de présenter les méthodes d'optimisation que nous avons réalisées, il convient de définir le contexte dans lequel l'ensemble des simulations ont été menées dans ce chapitre. Nous avons élaboré un outil de *benchmarking* permettant de tester des méthodes d'optimisation sur des fonctions objectifs. Cet outil a été réalisé au début de cette thèse et a été nommé **HeurisTest**, nous allons le détailler brièvement dans cette partie.

HeurisTest est une application en Java, conçue pour tester différentes méthodes d'optimisation sur un ensemble de *benchmarks*. Elle fournit un grand nombre d'outils afin de permettre à un utilisateur d'implémenter sa méthode d'optimisation et de la comparer à des méthodes existantes (cf figure 2.1). En effet, l'utilisateur n'a qu'une seule classe Java à écrire. Il pourra ensuite la charger dans l'application, qui la compilera et l'exécutera. Certaines métaheuristiques ont déjà été implémentées et validées par différents tests de la littérature :
- Algorithme de Nelder-Mead [Nelder 65] ;
- Recherche dispersée (*Scatter Search*) [Glover 77] ;
- Recherche tabou [Glover 86] ;
- Optimisation par essaims particulaires (PSO) [Kennedy 95] ;
- *MTS* [Tseng 08].

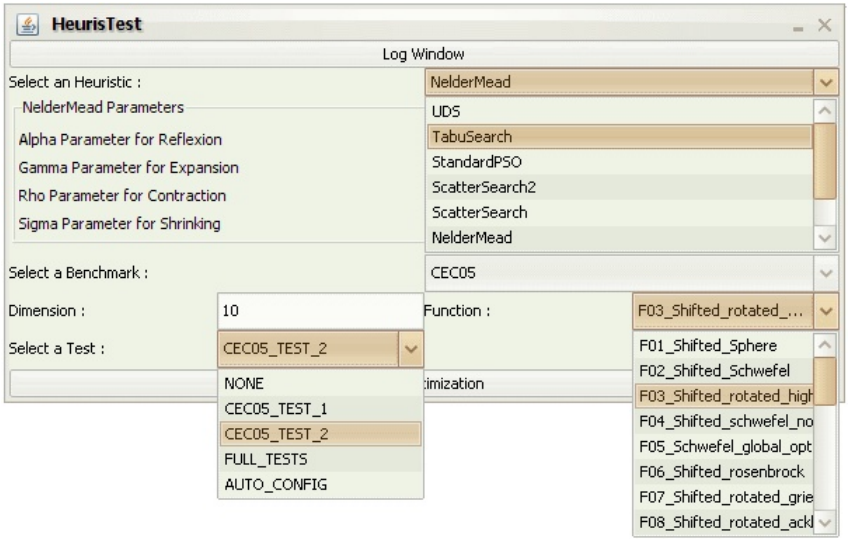


Figure 2.1 – Écran de contrôle de l'application HeurisTest.

Un poster décrivant cette application a été présenté lors de la conférence ITT'09 [Gardeux 09b]. Son principe repose sur le modèle "Simulation Optimisation" proposé par [April 03]. Il sépare le problème d'optimisation en deux blocs principaux : une méthode d'optimisation et un modèle de simulation. L'approche assimile le modèle de simulation à une boîte noire permettant d'évaluer des fonctions objectifs pour une entrée donnée, c'est-à-dire une solution dont on veut connaître la valeur par la fonction objectif. La partie optimisation utilise les réponses engendrées par le modèle de simulation pour prendre des décisions, comme la sélection de la prochaine solution à analyser. Cette partie optimise la fonction objectif pas à pas, car elle ne peut lancer qu'une itération de l'algorithme à chaque étape. Le principe général de cette méthode est récapitulé dans la figure 2.2.

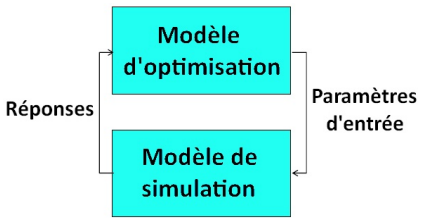


Figure 2.2 – Approche de type "Boîte Noire" utilisée dans le modèle *Simulation-Optimization*

HeurisTest permet à l'utilisateur de choisir une méthode d'optimisation à tester ainsi qu'une fonction objectif (issue d'un *benchmark*). À la fin de la simulation, l'outil produit les graphes de convergence, ainsi que des statistiques détaillées pour évaluer les performances de la méthode d'optimisation. Cet outil est complètement modulaire, il est donc possible d'intégrer dynamiquement de nouvelles heuristiques d'optimisation, sans avoir à recompiler tout le code.

Les protocoles expérimentaux permettent de considérer plusieurs tests pour chaque couple (fonction objectif, méthode) et de calculer une moyenne de l'ensemble de ces tests. Les graphes de convergence sont produits pour chaque test effectué et une moyenne logarithmique est calculée sur l'ensemble de la simulation, afin d'avoir une vue globale de la convergence de la méthode. Les graphes ne sont pas statiques et peuvent être zoomés, défilés, exportés en un fichier lisible par SciLab, etc.

Par exemple, nous avons testé les algorithmes PSO et Nelder-Mead sur la fonction objectif de *Rosenbrock* pour $n = 10$ (n : dimension du problème) et nous avons obtenu les graphes de convergence de la figure 2.3.

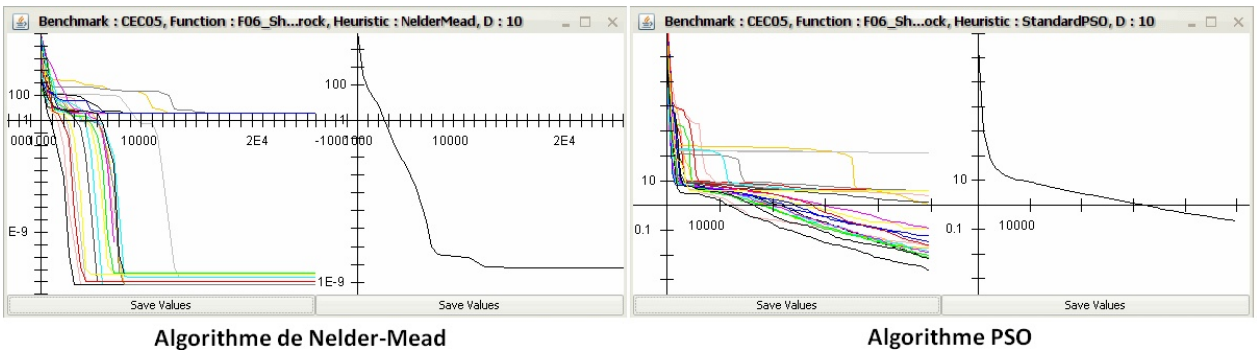


Figure 2.3 – Courbes de convergence de deux algorithmes d'optimisation sur la fonction *Shifted Rosenbrock*.

La simulation a été lancée 25 fois et on peut voir la courbe de convergence de chaque test sur le graphe de gauche pour chaque algorithme, le graphe de droite représentant la moyenne logarithmique des 25 tests.

C'est à partir de cette plateforme que seront effectués tous les tests et toutes les simulations de ce chapitre. Elle est disponible en téléchargement via [Gardeux 09a].

2.3 Méthode CUS

Après l'observation des algorithmes adaptés aux problèmes de grande dimension, nous avons décidé d'implémenter nos propres algorithmes pour ce type de problème. L'idée était d'isoler les caractéristiques principales de ces algorithmes et de les regrouper afin d'obtenir une méthode épurée. L'adaptation d'algorithmes existants n'auraient en effet pas permis de saisir les composantes principales qui font qu'un algorithme sera plus ou moins robuste à l'augmentation de la dimension. Nous voulions obtenir l'algorithme le plus simple possible, afin de s'en servir comme base pour la réalisation d'algorithmes plus élaborés par la suite.

À cet effet, nous avons pris les deux caractéristiques principales des méthodes d'optimisation de grande dimension que nous avions étudiées :
– convergence rapide,
– traitement séparé des dimensions.
Ces caractéristiques sont d'ailleurs liées, puisque le fait de traiter les dimensions séparément réduira la combinatoire et, dans le cas de fonctions simples (unimodales et séparables par exemple), permettra une convergence rapide.

L'idée fondatrice a alors été d'adopter le principe de décomposition du problème, dans sa version la plus extrême. Nous décomposons donc le problème de dimension n en n problèmes unidimensionnels. La fonction objectif est alors optimisée sur les n dimensions du problème, prises séparément. Ces techniques sont alors dites de *line search* (cf section 1.5).
Des méthodes plus récentes utilisent également des techniques de *line search*, par exemple dans les travaux de Grosan et Abraham [Grosan 07], ou bien dans l'algorithme MTS [Tseng 08] décrit précédemment.

2.3.1 Principe

La procédure *Classic Unidimensional Search* (CUS) [Gardeux 09c] [Gardeux 10] est très simple à implémenter et reprend les principes énoncés précédemment. Elle utilise une méthode de relaxation, qui consiste à relâcher des contraintes sur le problème, afin de rendre sa résolution plus aisée. Ici, la méthode de relaxation se traduit par le fait que nous optimisons les dimensions séparément. Si nous notons $(e_1, e_2, ..., e_n)$ les vecteurs unitaires de l'espace de recherche, on obtient un ensemble de n directions de recherche. Avec cette formalisation, nous utilisons une méthode d'optimisation unidimensionnelle qui s'applique sur chaque direction e_i. Elle optimise la fonction objectif le long de la première direction, jusqu'à ce qu'elle atteigne un optimum. Puis réitère avec la seconde direction, et de la même manière parcourt toutes les directions. Nous pouvons voir le principe général de cette méthode dans le pseudo-code de la figure 2.4.

Dans la suite, on note $h = h_1, ..., h_n$ un vecteur de "pas". Chaque composante de ce vecteur sera initialisée comme la différence entre la borne supérieure et la borne inférieure de l'espace de recherche. On aura donc $h_i = U_i - L_i, \forall i \in [1,n]$, avec $U = U_1, ..., U_n$ la borne supérieure et $L = L_1, ..., L_n$ la borne inférieure de l'espace de recherche. Chaque composante pouvant être bornée différemment, les variables U et L seront considérées comme des vecteurs.

Initialiser la solution $x = (x_1,...,x_n)$ aléatoirement
répéter
  pour $i = 1$ **à** n **faire**
    Utiliser un algorithme unidimensionnel pour optimiser $f(x)$ sur chaque direction e_i
  fin pour
jusqu'à ce que x ne puisse plus être amélioré

Figure 2.4 – Méthode de relaxation : principe général.

Ensuite, à chaque itération, CUS parcourt toutes les dimensions dans l'ordre. Pour optimiser f sur chaque dimension i, on considère deux nouvelles solutions "x *up*" (notée x^u) et "x *down*" (notée x^d), construites à partir de la solution courante x :

$$x^u = x + h_i e_i = (x_1, x_2, ..., x_i + h_i, ..., x_n) \tag{2.3.1a}$$

$$x^d = x - h_i e_i = (x_1, x_2, ..., x_i - h_i, ..., x_n) \tag{2.3.1b}$$

Si une solution est générée en dehors de l'espace de recherche, elle est corrigée et placée sur le bord le plus proche. Étant donnée l'initialisation du vecteur h lors de la première itération, l'algorithme commencera donc par tester les bords de l'espace de recherche.

L'algorithme évalue ensuite la valeur de la fonction objectif pour ces deux nouvelles solutions. Dans la suite, on considérera qu'une solution x est meilleure qu'une solution x' si $f(x) < f(x')$ dans le cas d'une minimisation, ou $f(x) > f(x')$ dans le cas d'une maximisation.
Trois cas possibles peuvent alors se présenter :
– x est meilleur que x^u et x^d. Dans ce cas, on ne fait rien et on passe à la dimension suivante ;
– x^d est meilleur que x et x^u. Dans ce cas on pose $s = -1$, cela nous permettra de continuer la recherche dans cette direction ;
– x^u est meilleur que x et x^d. Dans ce cas on pose $s = 1$.
Si l'on est dans un des deux derniers cas, on pose $x = x + s * h_i * e_i$. Puis on évalue la valeur de la fonction objectif pour la nouvelle solution $x + s * h_i * e_i$. Si elle est meilleure que x, alors on pose à nouveau $x = x + s * h_i * e_i$. On continue ainsi à parcourir cette direction, jusqu'à ce que l'on arrive sur un bord de l'espace de recherche, ou bien que l'on ne parvienne plus à améliorer la solution courante. Le principe de parcours est illustré par les courbes de la figure 2.5.

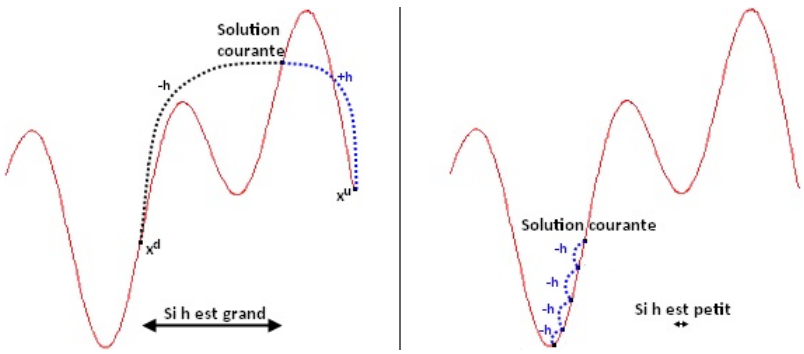


Figure 2.5 – Optimisation d'une fonction unidimensionnelle par la méthode de *line search* utilisée par CUS.

On remarque que, lorsque h est petit, l'algorithme converge vers l'optimum local le plus proche. En revanche, si h est grand, il pourra "sauter" certains optimums locaux.

Lorsque le parcours d'une composante est terminé, on passe à la composante suivante, et ainsi de suite, jusqu'à ce que toutes les dimensions aient été parcourues. Le parcours successif de toutes les composantes du vecteur solution constitue une itération de l'algorithme. À ce moment, avant de commencer l'itération suivante, on augmente la granularité (ou la précision) de la recherche, en diminuant la valeur de h ($h = h * r$). La variable r est l'un des paramètres de cet algorithme. Plus r sera grand, et plus l'algorithme convergera lentement, parcourant l'espace de recherche de manière plus approfondie. Nous fixons ce paramètre à 0,5 par défaut.
Le pseudo-code d'une itération de la méthode CUS est détaillé dans la figure 2.6.

Procédure CUS
début
  pour $i = 1$ **à** n **faire**
    $x^u = x + h_i e_i$
    $x^d = x - h_i e_i$
    (on met à jour x comme la meilleure des 3 solutions)
    $x = arg \ \ min(f(x), f(x^u), f(x^d))$
    tant que l'on améliore la solution x
      $x = x \pm h_i e_i$ dans la direction de la meilleure solution
    fin
  fin pour
  $h = h * r$
fin

Figure 2.6 – Une itération de l'algorithme CUS.

À chacune de ces itérations, on améliore la solution x d'une précision fixée par h. Pour simplifier, on peut dire qu'à la fin de chaque itération, on a trouvé un optimum local restreint à la précision h.

Le critère d'arrêt de cet algorithme est atteint lorsqu'une certaine précision (notée h_{min}) est atteinte par les composantes de h. Par défaut, nous fixerons cette valeur à 10^{-15}. Ce critère peut évidemment être modifié en fonction de la précision que l'on cherche à atteindre. En effet, dans le cadre de l'algorithmique théorique, nous ne sommes pas limités et nous pouvons donc atteindre une précision quelconque ; cependant, ce n'est jamais le cas en pratique. Le choix de 10^{-15} provient de raisonnements purement techniques que l'on peut aborder du point de vue de la représentation des nombres sur la machine. En effet, dans la plupart des langages de programmation, lorsque la différence de deux nombres atteint une précision suffisamment importante, le programme ne sait plus les différencier. Par exemple, les valeurs 10,000000000000001 et 10 seront considérées égales, même si en réalité elles différent de 10^{-15}. Ce genre de problème survient souvent lorsque l'on arrive à des précisions inférieures à 10^{-12} ; d'où le choix de ce critère d'arrêt nous permettant de conserver une petite marge.

L'algorithme CUS, tel qu'il est défini jusqu'ici, est une méthode de recherche locale. Il converge donc vers l'optimum local le plus proche, et s'arrête. Pour effectuer l'exploration de l'espace

de recherche, il nous a fallu trouver une méthode permettant de conserver cette vitesse de convergence, nous avons donc opté pour un redémarrage "intelligent". En effet, la méthode la plus simple serait de réinitialiser une solution aléatoirement dans l'espace de recherche et de recommencer la recherche depuis ce point. Or, cette méthode ne ferait aucune utilisation des optimums locaux déjà trouvés, ce qui serait une perte d'information. Nous avons donc opté pour une méthode générant une solution aléatoirement dans l'espace de recherche, mais de telle sorte que la nouvelle solution soit créée assez "loin" des optimums locaux déjà trouvés.
À cet effet, nous utilisons une liste L, vide au début de notre algorithme. Lorsqu'une première recherche locale est terminée (c'est-à-dire quand la précision h_{min} a été atteinte), nous ajoutons la solution courante x dans la liste L. Nous redémarrons ensuite l'algorithme en créant une nouvelle solution x, de façon qu'elle soit éloignée des solutions dans L. À cet effet, nous utilisons un algorithme de dispersion inspiré de la méthode *Scatter Search* [Marti 06] :

1. On ajoute l'optimum local x que l'on vient de trouver dans la liste L.
2. On génère aléatoirement 1000 solutions dans l'espace de recherche.
3. Pour chaque solution x' parmi les 1000, on calcule la distance $dist(x',L)$.
4. On prend comme nouvelle solution x, celle qui maximise cette distance, parmi les 1000.

La mesure de distance *dist* permet de déterminer la distance d'un point x à un ensemble S :

$$dist(x,S) = inf\{distanceEuclidienne(x,s), s \in S\} \tag{2.3.2}$$

Il est à noter que cette procédure de redémarrage n'utilise aucune évaluation de la fonction objectif, et donc ne "coûte" rien, en termes d'efficacité de l'algorithme de recherche. Cette procédure peut s'activer des nombres de fois différents, en fonction de la valeur de h_{min}. En effet, plus h_{min} est grand, et plus l'algorithme de redémarrage sera appelé, améliorant ainsi l'exploration de l'espace de recherche, en détériorant la précision des solutions (et inversement). Contrairement aux méthodes à mémoire adaptative, telle que la recherche tabou, la mémoire de CUS n'a pas de taille limitée. En effet, dans notre cas, il n'y a aucun intérêt pratique à vider cette mémoire. De plus, sa taille ne pourra jamais atteindre des dimensions trop importantes.

L'algorithme CUS est maintenant complètement défini, il regroupe donc :
- une procédure de recherche locale, permettant de converger rapidement vers un optimum local ;
- une mémoire, stockant les optimums locaux trouvés tout au long de la recherche ;
- une procédure de redémarrage, exploitant la mémoire, afin de créer de nouvelles solutions ;
- deux paramètres : h_{min} et r, que nous verrons plus en détail par la suite.

2.3.2 Étude de l'influence des paramètres

2.3.2.1 Paramètre r

Nous avons étudié l'influence des deux paramètres de l'algorithme CUS sur ses performances. À cet effet, nous avons commencé par comparer sa vitesse de convergence en fonction du paramètre r, le paramètre h_{min} étant fixé à 10^{-15}. Comme mesure de comparaison, nous avons calculé le nombre moyen d'évaluations de la fonction objectif qui ont été nécessaires pour atteindre une précision de l'ordre de 10^{-4}, sur 10 essais. Nous avons commencé par faire ce test sur la fonction *Sphere*, qui a été présentée dans la section 1.13.4. Les résultats sont consignés dans le tableau 2.1.

n	$CUS_{0,9}$	$CUS_{0,75}$	$CUS_{0,5}$	$CUS_{0,25}$	$CUS_{0,1}$	$CUS_{0,05}$	$CUS_{0,01}$
2	399,3	154,0	59,7	40,5	38,6	44,9	104,0
4	855,8	325,3	129,6	84,4	77,8	86,7	230,7
8	1806,2	670,2	276,9	178,1	158,8	169,9	425,2
16	3762,9	1420,0	567,4	376,0	322,8	345,5	828,4
32	7768,0	2910,9	1187,7	742,8	779,6	847,0	2394,9
64	16087,0	5973,1	2387,4	1481,0	1534,8	1824,5	4765,7
128	33613,8	12367,3	5087,6	3356,4	3079,2	3653,6	9572,7
256	69303,4	26110,2	10157,2	6692,2	6131,6	7355,8	19111,6
512	142791,1	53506,3	21618,4	13390,3	12287,6	14702,9	38149,7

Tableau 2.1 – Influence du paramètre r sur l'algorithme CUS (noté CUS_r). Calcul du nombre moyen d'évaluations de la fonction objectif *Sphere* pour atteindre une précision de 10^{-4}, pour n variables.

Nous avons remarqué que, plus le paramètre r est petit, et plus la convergence s'accélère, ce qui correspond au rôle que nous lui avons donné. Cependant, s'il atteint une valeur trop faible, la vitesse de convergence a tendance à ralentir. En effet, s'il est choisi trop faible, l'algorithme fait beaucoup de "petits sauts" sur chaque dimension, avant d'atteindre l'optimum, ce qui est contre-productif.

Nous avons ensuite fait le même test avec la fonction objectif *Shifted Schwefel 2.21*, détaillée dans le *benchmark* proposé par la conférence CEC'08 [Tang 07]. Cette fonction est particulièrement intéressante, car unimodale, comme la fonction *Sphere*, mais non séparable au contraire de cette dernière. Les résultats sont visibles dans le tableau 2.2. Les valeurs entre parenthèses

n	$CUS_{0,99}$	$CUS_{0,95}$	$CUS_{0,9}$	$CUS_{0,75}$	$CUS_{0,5}$	$CUS_{0,1}$
2	7104,8	1114,6	552,2	205,7	87,1	15199,6
4	10756,6	2132,0	1052,1	388,3	672,2	(0,0)
8	20376,3	4031,0	1984,6	750,4	(0,0)	(0,0)
16	38850,1	7743,7	3821,4	127375,5	(0,0)	(0,0)
32	76065,6	15189,6	17093,1	(0,0)	(0,0)	(0,0)
64	150486,1	39993,1	(0,0)	(0,0)	(0,0)	(0,0)
128	302406,4	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
256	613265,1	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)
512	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)	(0,0)

Tableau 2.2 – Influence du paramètre r sur l'algorithme CUS (noté CUS_r). Calcul du nombre moyen d'évaluations de la fonction objectif *Shifted Schwefel 2.21* pour atteindre une précision de 10^{-4}, pour n variables.

représentent la proportion d'essais (parmi les 10), ayant atteint la précision de 10^{-4} en moins d'un million d'évaluations de la fonction objectif. On remarque ici que le paramètre r a un fort impact sur la résolution de la fonction objectif. En effet, s'il est choisi trop petit, l'algorithme ne converge plus en un temps raisonnable. Ce comportement est probablement dû au fait que cette fonction est non séparable. Un parcours dimension par dimension augmente la difficulté de résolution de ce genre de problèmes.

Nous pouvons tout de même constater que pour certaines valeurs du paramètre r, CUS parvient à résoudre le problème, ce qui prouve que des méthodes issues de la *line search* sont capables de résoudre des problèmes non séparables. En effet, à chaque itération, chaque fonction unidimensionelle à optimiser peut changer à cause des modifications qui ont lieu sur les autres dimensions. Néanmoins, un algorithme traitant le problème dimension par dimension pourra tout de même trouver l'optimum, s'il recommence le parcours de chacune des dimensions.

2.3.2.2 Paramètre h_{min}

Après l'étude du paramètre r, nous nous sommes intéressés au deuxième paramètre de l'algorithme : h_{min}. Ce paramètre peut servir à augmenter le nombre de redémarrages de l'algorithme, en détériorant la précision de la solution. L'intérêt principal étant l'exploration de l'espace de recherche, nous avons testé son impact sur une fonction multimodale : la fonction de *Rosenbrock*. Cette fonction, également définie dans [Tang 07], est l'une des plus difficiles de la littérature. Elle est non séparable, et donc particulièrement difficile pour notre algorithme.

Nous avons fixé le paramètre r à 0,5 et avons étudié l'erreur entre l'optimum réel et la valeur trouvée par l'algorithme CUS au bout de 100000 évaluations de la fonction objectif. Les résultats sont consignés dans le tableau 2.3. Les valeurs entre parenthèses représentent les erreurs les plus

n	$CUS_{10^{-15}}$	$CUS_{10^{-6}}$	$CUS_{10^{-5}}$	$CUS_{10^{-4}}$	$CUS_{10^{-3}}$
2	4,3790(0,0064)	0,1641(0,0866)	0,1429(0,0094)	0,1523(0,0012)	0,1162(0,0024)
4	582,05(0,6766)	24,785(1,2542)	13,214(0,5209)	2,4634(0,0001)	0,1655(0,0039)
8	251,93(3,0215)	21,176(0,2695)	1,3624(0,0066)	0,0356(0,0005)	0,0642(0,0198)
16	909,87(32,176)	119,75(12,828)	170,85(6,8965)	8,9024(3,1163)	6,1931(0,7340)

Tableau 2.3 – Influence du paramètre h_{min} sur l'algorithme CUS (noté $CUS_{h_{min}}$). Calcul de l'erreur moyenne et du minimum atteint sur la fonction objectif *Shifted Rosenbrock*, pour n variables.

basses atteintes sur les 10 essais. On remarque, sur cette fonction, que les précisions atteintes ne sont pas du même ordre que sur le reste du *benchmark*. Cependant, nous pouvons observer que le fait d'augmenter la valeur du paramètre h_{min} permet d'obtenir de meilleurs résultats. Ce constat est dû à la méthode de redémarrage, qui permet d'explorer de manière plus large l'espace de recherche. Il faut évidemment prendre en compte que, si le paramètre h_{min} est fixé a 10^{-2}, il sera beaucoup plus compliqué d'atteindre une précision inférieure à 10^{-1}.

Ainsi, en fonction de la complexité de la fonction, adapter le paramètre h_{min} peut s'avérer intéressant pour parcourir au mieux l'espace de recherche de la fonction objectif. La fonction de *Rosenbrock* ne permet cependant pas d'obtenir de bonnes précisions lorsque l'on dépasse une dizaine de dimensions, sa complexité dépasse en effet les capacités de notre algorithme.

2.4 Méthode EUS

L'étude de la méthode CUS présentée précédemment nous a permis de constater certains problèmes liés à sa structure. Notamment un ajustement difficile du paramètre r, qui affecte la vitesse de convergence ainsi que la qualité des solutions obtenues. La méthode *Enhanced Unidimensional Search* (EUS) a alors été développée, dans le but de résoudre en partie ces problèmes.

2.4.1 Principe

Son fonctionnement est le même que l'algorithme CUS, il n'en diffère qu'en deux points :
- Lors de l'optimisation unidimensionnelle le long de chaque direction e_i, on évalue le triplet x, x^u, x^d. Dans CUS, on regarde la meilleure des 3 solutions et on continue l'optimisation dans ce sens, jusqu'à ce que l'on ne puisse plus améliorer la solution. Dans EUS, on garde le meilleur des trois, et on s'arrête, en passant directement à la direction suivante.
- Dans CUS, on réduit le vecteur h à chaque itération en le multipliant par r. Dans EUS, on ne réduit h que si, lors d'une itération, aucune direction n'a permis d'améliorer la solution courante.

Cette version est conçue pour résoudre des problèmes plus complexes, en faisant de plus petits pas dans une direction à chaque étape. En effet, changer de direction à chaque étape réduit la probabilité d'être emprisonné dans un optimum local et permet ainsi d'explorer une plus grande partie de l'espace des solutions. Le pseudo-code d'une itération de la méthode EUS est détaillé dans la figure 2.7.

Procédure EUS
début
  pour $i = 1$ **à** n **faire**
    $x^u = x + h_i e_i$
    $x^d = x - h_i e_i$
    (on met à jour x comme la meilleure des 3 solutions)
    $x = arg \ \ min(f(x), f(x^u), f(x^d))$
  fin pour
  si aucune amélioration n'a été trouvée en parcourant les n directions **alors**
    $h = h * r$
  fin si
fin

Figure 2.7 – Une itération de l'algorithme EUS.

La méthode EUS ne fait donc pas une optimisation intensive sur chaque dimension, mais simplement une approximation rapide à chaque passage. Cette modification devrait théoriquement permettre d'améliorer la convergence sur certaines fonctions non séparables, comme la fonction *Shifted Schwefel 2.21*. En effet, avec ce type de fonction, il est inutile d'optimiser chaque direction de manière précise. À chaque passage, la fonction unidimensionnelle à optimiser pour une direction fixée peut avoir été changée, relativement à l'optimisation faite sur les autres directions.

2.4.2 Comparaison avec CUS

Pour évaluer les performances de ce nouvel algorithme, nous reprenons les expériences de convergence réalisées pour l'algorithme CUS. Nous calculons donc le nombre d'évaluations de la fonction objectif requis pour atteindre une précision de 10^{-4}. Les moyennes sur 10 essais sont disponibles dans le tableau 2.4.

n	$EUS_{0,9}$	$EUS_{0,75}$	$EUS_{0,5}$	$EUS_{0,25}$	$EUS_{0,1}$	$EUS_{0,05}$	$EUS_{0,01}$
2	542,6	199,8	75,8	59,4	56,6	85,0	223,0
4	1283,4	473,0	195,4	142,6	154,6	231,4	691,4
8	2737,0	1018,6	423,4	308,2	351,4	481,0	1405,8
16	5709,8	2138,6	871,4	666,6	749,8	1031,4	2852,2
32	11860,2	4410,6	1857,0	1345,0	1645,8	2215,4	6273,0
64	24615,4	9089,0	3713,0	2804,2	3495,4	4801,0	14068,2
128	51175,4	18689,0	7937,0	6145,0	7194,6	9985,0	30669,8
256	105933,8	39425,0	15873,0	12289,0	14797,8	20685,8	65383,4
512	218113,0	80897,0	33793,0	24986,6	30721,0	42906,6	136193,3

Tableau 2.4 – Influence du paramètre r sur l'algorithme EUS (noté EUS_r). Calcul du nombre moyen d'évaluations de la fonction objectif *Sphere* pour atteindre une précision de 10^{-4}, pour n variables.

Nous remarquons que l'algorithme EUS a une convergence plus lente que CUS, ce qui paraît cohérent, au vu du fonctionnement plus contraint que nous avons défini. Des itérations supplémentaires sont donc nécessaires avant d'arriver à l'optimum. La convergence reste tout de même encore assez rapide, même en grande dimension.

Nous avons ensuite fait le même test, avec la fonction objectif *Shifted Schwefel 2.21*. Les résultats sont visibles dans le tableau 2.5.

n	$EUS_{0,9}$	$EUS_{0,75}$	$EUS_{0,5}$	$EUS_{0,25}$	$EUS_{0,1}$
2	905,8	336,6	133,0	**87.0**	**95,0**
4	2340,2	882,6	**324,2**	**246,6**	**266,6**
8	7105,0	2593,0	**961,0**	**775,4**	**954,6**
16	23114,6	**8554,6**	**2941,8**	**2711,6**	**3268,2**
32	81057,0	**30330,6**	**10202,6**	**9505,0**	**13447,4**
64	**301146,6**	**112679,4**	**37517,8**	**36071,4**	**52609,0**
128	**1160193,6**	**431617,2**	**148993,9**	**139393,0**	**203316,2**
256	**4561409,3**	**1678337,8**	**563713,1**	**554753,0**	**812852,2**
512	**1,7814529E7**	**6736897,0**	**2247681,0**	**2211533,8**	**3166209,0**

Tableau 2.5 – Influence du paramètre r sur l'algorithme EUS (noté EUS_r). Calcul du nombre moyen d'évaluations de la fonction objectif *Shifted Schwefel 2.21* pour atteindre une précision de 10^{-4}, pour n variables.

Pour plus de clarté, les solutions obtenues par EUS, qui sont de meilleure qualité que celles de CUS, ont été mises en gras. Ceci nous permet de constater, que sur cette fonction non séparable, les résultats sont bien meilleurs que ceux de CUS. En effet, même si la convergence est généralement plus lente que celle de CUS, l'algorithme EUS converge toujours vers l'optimum, quelle que soit la valeur du paramètre r. Ce n'était pas le cas pour l'algorithme CUS, dont la convergence pouvait atteindre un palier assez rapidement. L'intérêt est double car notre algorithme est robuste pour ce problème non séparable mais le paramétrage de r est sans conséquence pour l'obtention des résultats. Quelle que soit la valeur de r, la méthode converge vers l'optimum, en un temps plus ou moins long.

Pour terminer l'étude de l'algorithme EUS, nous avons refait les tests de convergence sur la fonction de *Rosenbrock*, en fonction du paramètre h_{min}. Nous avons fixé le paramètre r à 0,5 et avons étudié l'erreur entre l'optimum réel et la valeur trouvée par l'algorithme EUS au bout de 100000 évaluations de la fonction objectif. Les résultats sont consignés dans le tableau 2.6.

n	$EUS_{10^{-15}}$	$EUS_{10^{-6}}$	$EUS_{10^{-5}}$	$EUS_{10^{-4}}$	$EUS_{10^{-3}}$
2	**6,457**(**3,944E-11**)	32,39(**1,108E-7**)	20,06(**7,304E-6**)	9,762(0,002)	**0,044**(**6,45E-5**)
4	**74,25**(**9,282E-11**)	37,13(**1,456E-7**)	33,34(**6,680E-6**)	15,54(0,003)	5,197(0,027)
8	**72,41**(**1,449E-7**)	**0,398**(**1,728E-7**)	18,48(**7,487E-6**)	84,61(0,004)	0,141(0,119)
16	**27,64**(**9,984E-4**)	**74,52**(**0,003**)	**11,97**(**5,040E-4**)	89,74(**0,004**)	8,915(**0,135**)

Tableau 2.6 – Influence du paramètre h_{min} sur l'algorithme EUS (noté $EUS_{h_{min}}$). Calcul de l'erreur moyenne et du minimum atteint sur la fonction objectif *Shifted Rosenbrock*, pour n variables.

Les valeurs entre parenthèses représentent les erreurs les plus basses atteintes sur les 10 essais. On peut remarquer qu'en moyenne les résultats sont similaires à ceux obtenus par CUS avec les mêmes paramètres. Parfois ils sont un peu meilleurs, parfois un peu moins bons. Le point important concerne les minimums atteints par EUS. En effet, on remarque que, sur un nombre d'essais non négligeable, les valeurs minimales obtenues sont bien meilleures que celles obtenues par CUS. La moyenne absorbant ces résultats, il est difficile d'en voir l'impact réel. On a donc calculé, sous les mêmes conditions d'expérimentation, la valeur de la médiane (cf tableau 2.7).

n	$EUS_{10^{-15}}$	$EUS_{10^{-6}}$	$EUS_{10^{-5}}$	$EUS_{10^{-4}}$	$EUS_{10^{-3}}$
2	3,9449E-11	1,10829-7	9,0739E-6	3,9261E-3	5,4229E-2
4	2,2876E+1	5,0577E0	2,4727E+1	4,5753E+1	1,3669E-1
8	3,05615E-7	4,6186E-7	7,6647E-4	7,3402E-3	1,5521E-1
16	3,9985E0	3,9884E0	1,9963E0	3,9946E0	1,7353E-1

Tableau 2.7 – Influence du paramètre h_{min} sur l'algorithme EUS (noté $EUS_{h_{min}}$). Calcul de la médiane sur la fonction objectif *Shifted Rosenbrock*, pour n variables.

Ces résultats montrent qu'EUS trouve assez fréquemment un optimum très proche de l'optimum global. Cependant, ce n'est pas le cas pour tous les tests, ce qui aboutit à une moyenne biaisée. En effet, l'expérience étant limitée à 100000 évaluations de la fonction objectif, dans certains tests, EUS n'a pas le temps de trouver l'optimum durant l'exploration de l'espace de recherche.

Des tests complémentaires ont montré que l'algorithme EUS convergeait toujours sur la fonction objectif *Rosenbrock*, mais en un temps parfois très long. Ainsi, si on laisse tourner l'algorithme

suffisamment longtemps, il atteindra la précision souhaitée, ce qui n'était pas le cas pour CUS. Le tableau 2.8 montre ces résultats.

n	EUS
2	1,2703E5
4	1,0402E5
8	9,9759E5
16	2,519E6
32	1,8882E6
64	3,9229E6
128	6,6016E6

Tableau 2.8 – Calcul du nombre moyen d'évaluations de la fonction objectif *Shifted Rosenbrock* requis par EUS pour atteindre une précision de 10^{-4}, pour n variables.

Dans cette expérience, les paramètres ont été fixés à $h_{min} = 10^{-15}$ et $r = 0.5$. On remarque que l'algorithme converge bien vers l'optimum, même s'il faut parfois de nombreux redémarrages. Le résultat est tout de même important, car cette fonction est bien connue pour sa difficulté.

2.5 Analyse des méthodes sur un *benchmark*

Les méthodes CUS et EUS ont été présentées lors de la conférence ISDA'09 [Herrera 10a] organisée à Pise [Gardeux 09c]. Une session spéciale était organisée sur les problèmes de grande dimension donc le but était d'identifier les mécanismes clefs qui rendaient les métaheuristiques robustes sur ces problèmes. Un *benchmark* de 11 fonctions objectifs était proposé pour les problèmes de grande taille, les 6 premières étant celles de la conférence CEC'08 [Tang 07]. Les différentes caractéristiques de ces fonctions peuvent être vues dans le tableau 2.9 et dans l'annexe A.1. Une discussion plus détaillée de la fonction "*Extended* F_{10}" peut être trouvée dans [Whitley 95].

	Nom	Unimodale	Séparable
f_1	*Shifted Sphere*	X	X
f_2	*Shifted Schwefel Problem 2.21*	X	-
f_3	*Shifted Rosenbrock*	-	-
f_4	*Shifted Rastrigin*	-	X
f_5	*Shifted Griewank*	-	-
f_6	*Shifted Ackley*	-	X
f_7	*Schwefel Problem 2.22*	X	X
f_8	*Schwefel Problem 1.2*	X	-
f_9	*Extended* F_{10}	X	-
f_{10}	*Bohachevsky #1*	X	-
f_{11}	*Schaffer #2*	X	-

Tableau 2.9 – Caractéristiques des fonctions du *benchmark* proposé par la session spéciale d'ISDA'09.

Le protocole expérimental a été fixé de la même manière que pour la conférence CEC'08 [Tang 07]. Pour chaque fonction f, les expériences sont conduites en dimensions $n = 50, 100, 200$ et 500. Chaque algorithme est lancé 25 fois et l'erreur $e = f(x^*) - f(o)$ est calculée entre la meilleure solution x^* trouvée par l'algorithme et l'optimum global réel o. Le nombre maximal d'évaluations de la fonction objectif est fixé à $5000 * n$ et détermine le critère d'arrêt pour chaque test.
Les erreurs moyennes (sur les 25 tests) obtenues par les algorithmes EUS et CUS pour chaque fonction sont listées dans le tableau 2.10. Les valeurs entres parenthèses représentent la valeur du paramètre r, h_{min} étant toujours fixé à 10^{-15}.

Fn	n = 50		n = 100		n = 200		n = 500	
	CUS	EUS	CUS	EUS	CUS	EUS	CUS	EUS
f_1	4,18E-13	4,14E-13	9,40E-13	9,45E-13	2,01E-12	2,07E-12	5,40E-12	5,37E-12
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)
f_2	8,02E-8	1,90E-11	9,20E-9	9,82E-11	2,24E-9	5,53E-10	2,87E0	1,35E-4
(r)	(0,99)	(0,5)	(0,99)	(0,5)	(0,99)	(0,5)	(0,99)	(0,5)
f_3	3,01E2	1,07E1	3,65E2	9,70E1	1,49E3	4,47E2	1,34E3	4,78E2
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)
f_4	4,83E-13	6,88E-13	1,16E-12	1,47E-12	2,63E-12	3,29E-12	7,27E-12	8,52E-12
(r)	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)
f_5	2,11E-13	2,10E-13	4,73E-13	4,85E-13	1,02E-12	1,39E-12	2,89E-12	3,53E-12
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)
f_6	3,79E-13	4,35E-13	1,13E-12	1,19E-12	2,33E-12	2,27E-12	5,71E-12	5,99E-12
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)
f_7	3,38E-13	1,80E-13	7,63E-13	3,94E-13	1,57E-12	3,87E-14	4,24E-12	9,96E-14
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)
f_8	4,95E-13	5,14E-21	1,24E-2	1,39E-9	4,68E2	1,99E-1	8,11E6	6,06E3
(r)	(0,99)	(0,75)	(0,99)	(0,75)	(0,99)	(0,75)	(0,99)	(0,75)
f_9	3,65E2	3,55E2	7,32E2	7,10E2	1,44E3	1,42E3	3,65E3	3,52E3
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)
f_{10}	6,66E-27	1,72E-27	1,46E-26	3,63E-27	3,10E-26	7,85E-27	7,98E-26	2,03E-26
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)
f_{11}	1,63E2	1,52E2	4,24E2	2,21E2	4,62E2	3,49E2	1,58E3	9,43E2
(r)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)	(0,5)

Tableau 2.10 – Erreurs commises par les algorithmes EUS et CUS sur le *benchmark* proposé.

2.5.1 Méthode de classement des résultats

Afin de comparer de manière plus approfondie le comportement des deux algorithmes sur les 11 fonctions, nous mettons en place une classification basée sur ces comportements. En effet, nous remarquons ici trois résultats de types différents :

- **Type 1 :** Les fonctions f_1, f_4, f_5, f_6, f_7 et f_{10} ne posent a priori aucun problème pour les algorithmes CUS et EUS. Les précisions atteintes montrent qu'elles sont correctement résolues pour toutes les dimensions.
- **Type 2 :** Les fonctions f_2 et f_8 sont correctement résolues en faible dimension. Cependant, avec l'augmentation de la dimension du problème, les algorithmes perdent en vitesse de convergence, ils ne sont donc pas robustes sur ces fonctions.
- **Type 3 :** Les fonctions f_3, f_9 et f_{11} "résistent" complètement aux deux algorithmes, elles ne sont même pas résolues correctement en faible dimension.

Parmi les fonctions de "type 2", nous avions analysé plus particulièrement f_2 lors de l'étude de l'influence des paramètres. Sur cette fonction, nous avions vu qu'il fallait un grand nombre d'évaluations de la fonction objectif pour que les algorithmes atteignent des précisions suffisantes. Nous avions conclu que seul un paramètre de r très élevé permettait à CUS de converger, sauf en dimension 500, où la convergence s'arrêtait brutalement et atteignait un palier. À l'inverse, la méthode EUS convergeait quel que soit le paramètre r, le seul impact était sa rapidité de convergence. Ce type de fonction est très intéressant dans le cadre de l'optimisation à haute dimension, puisque sa complexité semble évoluer rapidement avec la dimension. Les algorithmes d'optimisation doivent donc être particulièrement robustes pour pouvoir les résoudre. Nos résultats sur ce type de fonction sont encourageants, notamment pour EUS, qui parvient à les résoudre correctement en un temps légèrement plus long que celui qui est permis par le *benchmark*.

Les fonctions de "type 3" sont des fonctions assez complexes. Par exemple, f_3 est une fonction connue pour sa complexité que nous avions analysée dans la partie précédente. Nous avions vu que CUS ne convergeait pas, au contraire d'EUS, qui nécessitait toutefois un temps relativement long. Le protocole expérimental définit dans le *benchmark* n'autorise pas suffisamment d'évaluations de la fonction-objectif, et donc rend la fonction résistante pour de nombreux algorithmes. Les fonctions f_9 et f_{11} ont un comportement légèrement différent. En effet, leur convergence est très lente, ou même inexistante, et ce dès la dimension 2. L'absence de convergence sur ces fonctions peut provenir d'une complexité particulièrement rebutante pour les algorithmes. Cependant, il est plus judicieux de penser que ce sont les principes de nos algorithmes qui sont en cause. En effet, pour des fonctions non séparables complexes, il peut y avoir de nombreuses corrélations entre les variables, les rendant particulièrement difficiles à résoudre pour nos algorithmes en un temps raisonnable.

Une étude de la convergence d'EUS sur les cinq premières fonctions du *benchmark* nous permet de mieux distinguer les différents "types" de résultats que nous avons obtenu. La figure 2.8 montre en effet que l'algorithme EUS a une convergence très rapide pour les fonctions f_1, f_4 et f_5, qui sont de "type 1". Il a également une convergence lente pour la fonction f_2, qui est de "type 2". Pour la fonction f_3, que nous avons catégorisée de "type 3", la convergence de EUS s'arrête assez tôt et la méthode semble bloquée dans un optimum local. En fait, nous avions vu dans la section 2.4.2 que cette convergence existait mais était excessivement lente.

Le palier atteint par les fonctions de "type 1" est dû aux problèmes de représentation des nombres dont nous avions déjà parlé (cf section 2.3.1), la machine ne peut donc pas aller plus bas que cette précision. Au vu de la précision atteinte, on peut d'ailleurs considérer que l'algorithme a atteint l'optimum.

On peut conclure que, sur l'ensemble des fonctions, EUS obtient des résultats équivalents ou meilleurs que CUS. De plus, pour que CUS converge, nous devons trop souvent paramétrer la valeur de r, tandis qu'EUS semble avoir un bon comportement de convergence pour un r fixé à 0,5. La valeur de 0,75 prise pour la fonction 8 ne permet en effet d'améliorer que très

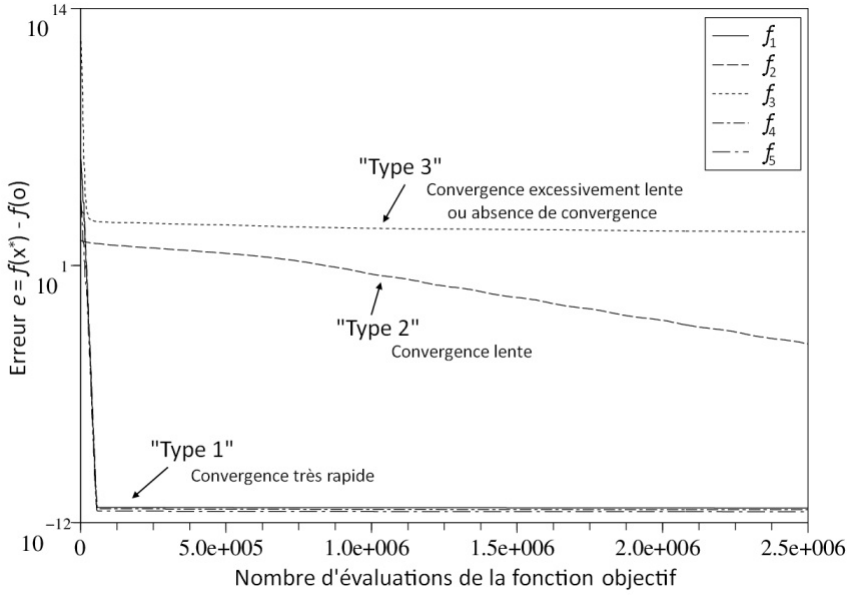


Figure 2.8 – Étude de la convergence de l'algorithme EUS, sur les 5 premières fonctions du *benchmark*.

légèrement les résultats. La seule exception étant la fonction f_4 qui nécessite une valeur de 0,1 pour converger. EUS semble donc le meilleur compromis pour la résolution de problèmes à grande dimension. Le fait de fixer ses deux paramètres à des valeurs par défaut permet de rendre EUS utilisable directement par n'importe quel utilisateur, sans nécessité de paramétrage.

2.6 Comparaison de EUS et CUS avec d'autres méthodes

2.6.1 Méthodes de la conférence CEC'08

Pour commencer, nous avons comparé nos méthodes à celles ayant obtenu les meilleurs résultats durant la conférence CEC'08. Celles-ci étaient effectivement notées par les concurrents et les trois meilleures ont été résumées dans un document-bilan fourni par Ke Tang [Tang 08]. Leurs résultats étant donnés pour $n = 1000$, nous avons complété nos expériences, afin d'obtenir le tableau 2.11.

Fn	CUS	EUS	MTS	LSEDA-gl	jDEdynNP-F
f_1	9,67E-12	9,66E-12	**0,00E0**	3,23E-13	1,14E-13
f_2	1,81E2	1,28E-1	4,72E-2	**1,04E-5**	1,95E1
f_3	2,52E3	8,24E2	**3,41E-4**	1,73E3	1,31E3
f_4	1,47E-11	9,62E-12	**0,00E0**	5,45E2	2,17E-4
f_5	5,99E-12	6,39E-12	**0,00E0**	1,71E-13	3,98E-14
f_6	1,23E-11	1,22E-11	1,24E-11	**4,26E-13**	1,47E-11

Tableau 2.11 – Erreurs obtenues sur les 6 premières fonctions objectifs de la conférence CEC'08, pour $n = 1000$

Les résultats en gras représentent les meilleurs résultats obtenus lors de la conférence CEC'08, parmi les 10 algorithmes présentés. Les trois meilleurs : MTS [Tseng 08], LSEDA-gl [Wang 08] et jDEdynNP-F [Brest 08] ont d'ailleurs été présentés et analysés dans le chapitre précédent (cf section 1.12.2).

Le point le plus intéressant est le résultat de MTS sur la fonction de *Rosenbrock*. En effet, aucune des autres méthodes présentées ne dépasse une précision de 1E3, et notre résultat en 1E2 avec EUS aurait donc été plutôt "bon". Une question se pose néanmoins sur les précisions de l'ordre de 0.00E0 obtenues par MTS sur la plupart des fonctions objectifs : correspondent-elles à des précisions faibles, assimilées à des 0 ? L'équipe de Tseng ayant arrêté leurs activités de recherche, nous n'avons pas pu obtenir davantage de précision sur ces résultats.

Nous pouvons voir que, sur les fonctions f_1, f_5 et f_6, les 5 algorithmes obtiennent des résultats équivalents. En effet, ces fonctions sont correctement résolues avec une précision inférieure à 1E-10 (ce qui caractérise un comportement de "type 1"). La fonction f_3, quant à elle, fait l'objet de mauvais résultats de manière égale pour tous les algorithmes, excepté MTS, qui parvient exceptionnellement à la résoudre avec une bonne précision, de l'ordre de 1E-4.

Les résultats qui sont plus hétérogènes, et donc les plus intéressants à analyser, sont ceux des fonctions f_2 et f_4. En effet, on remarque que sur la fonction f_2, tous les algorithmes ont un comportement de convergence "relative". Ce constat confirme le caractère de cette fonction, dont la complexité augmente proportionnellement à l'augmentation de la dimension. Notre algorithme parvient tout de même à la résoudre avec une précision de l'ordre de 1E-1, approximativement du même ordre que l'algorithme MTS. Cependant, la méthode LSEDA-gl obtient de meilleurs résultats ici, probablement du fait d'une convergence plus rapide. Si notre algorithme disposait d'un peu plus de temps (en termes de nombre d'évaluations de la fonction objectif), il atteindrait également cette précision, comme nous l'avons montré dans la section 2.4.2. La fonction f_4, quant à elle, est parfaitement résolue par nos algorithmes, ainsi que par MTS, les deux autres algorithmes ne parvenant pas à une telle précision. L'explication la plus probable étant que cette fonction est séparable, et donc beaucoup plus simple à résoudre par un algorithme traitant les dimensions séparément.

2.6.2 Méthodes de la session spéciale de la conférence ISDA'09

Dans un deuxième temps, nous avons analysé les six algorithmes présentés lors de la session spéciale de la conférence ISDA'09.

- MA-MTSLS-Chains [Molina 09] : un algorithme mémétique utilisant une méthode dite de recherche locale appelée "*LS chaining*".
- MH [García-Martínez 09] : une métaheuristique basée sur le modèle VNS(*Variable Neighbourhood Search*). Ce modèle regroupe 3 métaheuristiques : CMA-ES pour initialiser une solution dans l'espace de recherche (*generation*), *Continuous Local EA* comme recherche locale (*improvement*) et μCHC pour diversifier les solutions existantes, utilisé comme perturbateur (*shaking*).
- *Memetic Differential Evolution* (MDE-DC) [Muelas 09] : Un algorithme hybridant l'évolution différentielle (DE) avec LS1, méthode de recherche locale de l'algorithme MTS vu précédemment.
- STS [Duarte 09] : Un algorithme hybridant les méthodes à mémoire adaptative *Scatter Search*, *Tabu Search* et une version de *Tabu Search* hybridée au simplexe de Nelder-Mead.
- GODE [Wang 09] : Une méthode améliorant l'évolution différentielle (DE) avec une technique de *Generalized Opposition-Based Learning* (GOBL) [Tizhoosh 05].

Afin de comparer leur efficacité, nous avons résumé les résultats en dimension 500 dans le tableau 2.12.

n = 500	EUS	GODE	STS	MDE-DC	MH	MA-MTSLS-C
f_1	**5,37E-12**	2,29E-02	1,50E+02	**0,00E+00**	**6,00E-13**	**2,84E-13**
f_2	1,35E-04	8,90E+01	1,44E+02	3,39E-04	**3,60E-14**	7,00E+01
f_3	4,78E+02	2,69E+03	1,14E+04	**2,96E+01**	6,10E+04	7,86E+02
f_4	**8,52E-12**	2,69E+03	8,05E+01	**0,00E+00**	2,50E+03	1,07E+00
f_5	**3,53E-12**	3,18E-02	2,37E+00	**0,00E+00**	**3,50E-13**	**1,44E-13**
f_6	**5,99E-12**	1,22E+01	1,71E+00	**0,00E+00**	2,20E+01	**3,33E-13**
f_7	**9,96E-14**	**0,00E+00**	**0,00E+00**	**0,00E+00**	5,00E-02	**1,27E-58**
f_8	6,06E+03	**0,00E+00**	2,42E+05	1,10E+04	4,90E-04	6,84E+04
f_9	3,52E+03	**0,00E+00**	1,70E+01	3,25E+00	2,50E+03	3,47E+00
f_{10}	**2,03E-26**	**0,00E+00**	**0,00E+00**	**0,00E+00**	4,70E-03	**7,04E-45**
f_{11}	9,43E+02	**0,00E+00**	2,79E+01	3,27E+00	2,40E+03	1,61E+01

Tableau 2.12 – Comparaison des erreurs commises par les algorithmes de la conférence ISDA'09 sur le *benchmark* proposé, pour $n = 500$.

Pour plus de lisibilité, nous avons mis en gras les algorithmes qui obtenaient les meilleurs résultats pour chaque fonction, en supposant que les précisions inférieures à 1E-10 étaient équivalentes, afin de pallier aux problèmes techniques de représentation des nombres sur la machine. Nous pouvons remarquer que notre algorithme obtient les meilleurs résultats, conjointement avec d'autres algorithmes, pour les fonctions f_1, f_4, f_5, f_6, f_7 et f_{10}. Sur ces fonctions, l'algorithme converge parfaitement. De ce point de vue, nous résolvons 6 fonctions sur 11 tandis que MDE-DC en résout 7 sur 11, ce qui nous place au second rang. Sur la fonction f_2, nous nous plaçons second, au même niveau que MDE-DC, f_2 étant parfaitement résolue par MH. Ce dernier algorithme résout d'ailleurs assez bien la fonction f_8, ce qui est cohérent, car elle est du même type que f_2. La fonction f_3 est mal résolue par tous les algorithmes, mais MDE-DC fait un peu mieux. Les résultats de GODE sur les fonctions f_8, f_9 et f_{11} sont assez intrigants, d'autant plus que cet algorithme résout mal la fonction f_1, qui est la plus simple, ou la fonction f_2, qui est du même type que f_8. Nous pensons que ce comportement est dû à un problème lié à la première version du *benchmark*, qui n'a pas été mis à jour dans les résultats publiés pour GODE. Si nous écartons ces résultats, nous remarquons que tous les algorithmes ont des difficultés à résoudre les fonctions f_8, f_9 et f_{10}.

2.6.3 Discussion

Les résultats d'EUS sont donc meilleurs, ou du même niveau que ceux présentés lors des conférences CEC'08 ou ISDA'09 sur l'ensemble des fonctions. Les seuls algorithmes dépassant réellement les résultats d'EUS étant les algorithmes MH et LSEDA-gl sur les fonctions f_2 et f_8, et MTS sur f_3. Ces résultats sont probablement la conséquence d'une convergence plus rapide par rapport à EUS, qui converge tout de même, mais en un temps légèrement plus long, comme nous l'avions déjà vu. Nos résultats sont donc compétitifs, puisque nous nous comparons à des algorithmes parfois très complexes et contenant de nombreux paramètres, au contraire de nos algorithmes.

2.7 Algorithme SEUS

Après publication de ces résultats, M. G. Omran a repris le principe de l'algorithme EUS afin de l'améliorer [Omran 10]. En effet, après étude de la méthode EUS et de ses deux paramètres (r et h_{min}), M. G. Omran a montré que, sur certaines fonctions, le paramètre $r = 0{,}5$, n'est pas toujours le meilleur. Ainsi, il a produit une méthode visant à parcourir différents intervalles pour ce paramètre.

La méthode résultante est appelée *Stepped EUS* (SEUS), et elle obtient des résultats équivalents, ou meilleurs, sur toutes les fonctions testées. Le pseudo-code de cette méthode est représenté sur la figure 2.9.

Procédure SEUS
début
  $ratio = 0{,}1$
  tant que $ratio < 1$ **faire**
    Lancer EUS avec le paramètre $r = ratio$
    $ratio = ratio + 0{,}1$
  fin tant que
fin

Figure 2.9 – Pseudo-code de l'algorithme SEUS.

L'algorithme est ici très simple, et se résume à lancer la méthode EUS plusieurs fois, en faisant varier r de 0,1 à 0,9. Cependant, ceci montre qu'une auto-adaptation du paramètre r peut être très intéressante et mener à de meilleurs résultats. D'autres pistes peuvent également être étudiées, telles qu'un r aléatoire. Ces idées ont été reprises dans l'algorithme EM323 que nous allons présenter.

2.8 Algorithme EM323

Pour suivre les objectifs que nous nous étions fixés au début de cette thèse, nous avons réalisé deux méthodes très simples permettant de résoudre efficacement des problèmes de grande dimension : CUS et EUS. L'étape suivante a alors été de raffiner ces méthodes, afin de résoudre des fonctions plus complexes. Nous avons alors pris pour base l'algorithme EUS, qui obtient de meilleurs résultats que CUS. Dans l'optique de l'améliorer, nous avons trois pistes possibles :
- utiliser un algorithme d'exploration de l'espace de recherche efficace pour les problèmes de grande dimension, et l'hybrider à notre méthode EUS. EUS est alors considéré comme une méthode d'optimisation locale et la méthode de redémarrage serait donc abandonnée.
- améliorer le fonctionnement de notre algorithme EUS, notamment dans le parcours de ses dimensions.
- améliorer la méthode d'optimisation unidimensionnelle, afin de résoudre des fonctions plus complexes.

Nous avons pris le parti de nous concentrer sur les deux derniers points. L'idée étant d'avoir un algorithme relativement simple, mais permettant de résoudre la plus grande panoplie de fonctions possible. Le but final étant de montrer qu'un tel algorithme est capable de résoudre également des fonctions complexes non séparables.

À cet effet, nous avons combiné EUS avec une méthode d'optimisation unidimensionnelle récente, développée par Fred Glover [Glover 10]. De plus, afin d'adapter l'algorithme aux fonctions non séparables, nous avons également ajouté quelques composantes, qui seront développées par la suite.

2.8.1 *3-2-3* : un algorithme de *line search*

Nous avons décidé de chercher une autre méthode à substituer à la méthode de *line search* d'EUS, qui ne permettait pas de résoudre des fonctions trop complexes. La méthode utilisée améliorait une solution x en considérant ses deux solutions voisines, notées x^u et x^d. Le but était de trouver rapidement de nouvelles solutions candidates pour remplacer la solution courante x. À la place, nous avons sélectionné la procédure *3-2-3* développée par Fred Glover [Glover 10], qui a une structure parfaitement adaptée à l'exploitation des résultats pour notre algorithme.

2.8.1.1 Formalisation de la *line search*

Nous allons définir plus en détail le principe des méthodes de *line search* vues dans la section 1.5, qui permettent de résoudre des problèmes d'optimisation multidimensionnels par l'optimisation d'une fonction unidimensionnelle, le long d'une direction. Ce type de problème peut être exprimé comme une minimisation (ou maximisation) d'une fonction objectif f sur un segment noté LS(x', x''), passant par les points x' et x'', tels que

$$\text{LS}(x', x'') = \{x = x(\theta), x(\theta) = x' + (x'' - x')\theta \text{ pour } \theta_{min} \leq \theta \leq \theta_{max}\} \tag{2.8.1}$$

Les valeurs θ_{min} et θ_{max} sont calculées de sorte que LS(x', x'') soit compris dans une région convexe bornée définissant un ensemble de solutions réalisables. Cet espace de recherche peut correspondre à un problème unidimensionnel, ou à une relaxation mathématique d'un tel problème. Ainsi, les points x' et x'' ne sont pas forcément les extrémités du segment LS(x', x'').

2.8.1.2 Principe de l'algorithme *3-2-3*

L'algorithme *3-2-3* est une procédure d'optimisation unidimensionnelle qui trouve ses fondements dans les méthodes dites par encadrement (cf section 1.4).

2.8.1.2.1 Préparation du segment : La méthode *3-2-3* démarre par une approche communément adoptée par les procédures de *line search* non linéaires. Celle-ci consiste à identifier une succession de points $x(\theta_0)$, $x(\theta_1)$, ..., $x(\theta_s)$ le long d'une direction donnée sur un segment $LS(x', x'')$, avec $s \geq 2$ tel que

$$\theta_0 < \theta_1 < ... < \theta_s, \text{ où } \theta_0 = \theta_{min}, \theta_s = \theta_{max} \tag{2.8.2}$$

Les valeurs θ_i sont des paramètres qui identifient la localisation de x (ou plus particulièrement $x(\theta_i)$) sur le segment $LS(x',x'')$ joignant x' et x''. Nous pouvons supposer par exemple que $x' = x(0)$ et $x'' = x(1)$, avec θ_{min} et θ_{max} sélectionnés afin de satisfaire $0 \leq \theta_{min} < \theta_{max} \leq 1$.

La façon la plus simple de générer les valeurs θ_i est de diviser l'intervalle $[\theta_{min}, \theta_{max}]$ en $NCut$ sous-intervalles de même taille, tels que :

$$\theta_i = \theta_{i-1} + \Delta (= \theta_0 + i\Delta) \text{ pour } i = 1, ..., NCut, \text{ avec } \Delta = \frac{\theta_{max} - \theta_{min}}{NCut} \tag{2.8.3}$$

2.8.1.2.2 Paramètres d'entrée de *3-2-3* : La méthode *3-2-3* se concentre sur des séquences de points pris parmi les sous-segments définis en référence aux valeurs θ. Ceux-ci peuvent être de deux sortes : soit sous la forme d'une paire $(x(\theta_{i-1}), x(\theta_i))$ pour $1 \leq i \leq NCut$, soit d'un triplet $(x(\theta_{i-1}), x(\theta_i), x(\theta_{i+1}))$ pour $1 \leq i \leq NCut - 1$. Les points $x(\theta_0)$ et $x(\theta_{NCut})$ seront par conséquent les extrémités des intervalles définies par ces paires et ces triplets.

L'algorithme *3-2-3* démarre d'une façon similaire à la méthode de la *Golden Section Search* dont nous avons déjà parlé (cf section 1.4). Elle prend également en paramètre un triplet $(x(\theta_{i-1}), x(\theta_i), x(\theta_{i+1}))$, mais elle permet de considérer différents types d'intervalles, et des manières plus complexes de les gérer. Cette approche peut être particulièrement appropriée quand les distances entre les points successifs $x(\theta_i)$ sont relativement faibles, ou quand le but est d'améliorer une solution sur le segment, en se concentrant davantage sur la région autour d'un point $x(\theta_q)$.

Nous sélectionnons un triplet $(x(\theta_{q-1}), x(\theta_q), x(\theta_{q+1}))$ de telle manière que l'équation (2.8.4) soit satisfaite.

$$f(x(\theta_q)) \leqslant f(x(\theta_{q-1})) \text{ et } f(x(\theta_q)) \leqslant f(x(\theta_{q+1})) \tag{2.8.4}$$

Cette équation revient à dire que l'on sélectionne un point $x(\theta_q)$ sur le segment de telle manière que ses deux voisins obtiennent une valeur de la fonction objectif de moins bonne qualité (pour simplifier, on se place ici dans le cadre d'une minimisation).
Un tel triplet existe toujours, à moins que les optimums locaux contenus parmi les points $x(\theta_0)$, ..., $x(\theta_{NCut})$ ne soient situés aux points $x(\theta_0)$ ou $x(\theta_s)$. En écartant ce cas exceptionnel, l'algorithme *3-2-3* est appliqué pour chaque triplet satisfaisant l'équation (2.8.4). Sinon, l'algorithme *3-2-3* ne pouvant être lancé directement, un algorithme préliminaire nommé *2-1-2* est alors appelé pour créer le triplet d'entrée nécessaire à *3-2-3*.

Les trois cas de départ peuvent être résumés comme suit, où les Cas 1 et 2 ne sont considérés que dans la situation exceptionnelle où aucun triplet, satisfaisant l'équation (2.8.4), n'existe.
- Cas 1 : $f(x(\theta_0))$ est le minimum global parmi les points $x(\theta_0)$,..., $x(\theta_{NCut})$. Soit $x^a = x(\theta_0)$ et $x^c = x(\theta_1)$ (ce qui implique que $f(x^a) < f(x^c)$). On pose $x^* = x^a$ et on lance l'algorithme préliminaire *2-1-2* afin de générer un triplet (x^a,x^b,x^c). On lance ensuite l'algorithme principal *3-2-3* avec ce triplet en paramètre.
- Cas 2 : $f(x(\theta_{NCut}))$ est le minimum global parmi les points $x(\theta_0)$, ..., $x(\theta_{NCut})$. Soit $x^a = x(\theta_{NCut})$, $x^c = x(\theta_{NCut-1})$ (ce qui implique que $f(x^a) < f(x^c)$). On pose $x^* = x^a$ et on lance l'algorithme préliminaire *2-1-2* afin de générer un triplet (x^a,x^b,x^c). On lance ensuite l'algorithme principal *3-2-3* avec ce triplet en paramètre.
- Cas 3 : l'équation (2.8.4) est satisfaite pour un q donné tel que $1 \leq q \leq NCut - 1$. Soit $x^a = x(\theta_{q-1})$, $x^b = x(\theta_q)$, $x^c = x(\theta_{q+1})$ (ce qui implique que $f(x^b) \leq f(x^a)$, $f(x^c)$). On pose $x^* = x^b$ et on lance l'algorithme principal *3-2-3* avec ce triplet en paramètre.

On appelle x^* la meilleure solution obtenue lors de la recherche.

2.8.1.2.3 Algorithme préliminaire *2-1-2* : L'algorithme préliminaire est appelé "2-1-2" de par son fonctionnement. En effet, à chaque itération, il démarre avec 2 points, génère un nouveau point, et supprime un des 2 points initiaux pour terminer avec 2 points (l'un étant le nouveau point). Le pseudo-code de la figure 2.10 détaille cet algorithme préliminaire, où *maxIter* est un paramètre représentant le critère d'arrêt.

Procédure 2-1-2
$x^* = x^a$
$iter = 0$
début
  (On a et on conserve $f(x^a) < f(x^c)$)
  tant que $iter < maxIter$ **faire**
    $iter = iter + 1$
    $x^b = 0.5 * (x^a + x^c)$
    si $f(x^b) \leq f(x^a)$ **alors**
      ((x^a,x^b,x^c) est de la bonne forme pour l'algorithme *3-2-3*)
      $x^* = x^b$
      Arrêter *2-1-2* et appeler la procédure *3-2-3* avec les paramètres x^a,x^b et x^c
    sinon
      ($f(x^a) < f(x^b)$)
      Soit (x^a,x^b) le nouveau (x^a,x^c)
    fin si
  fin tant que
fin

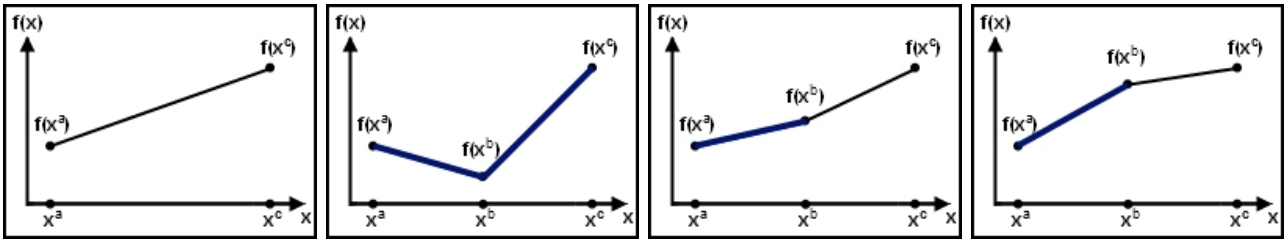
Figure 2.10 – Pseudo-code de l'algorithme préliminaire *2-1-2*.

2.8.1.2.4 Algorithme principal *3-2-3* : De la même manière, l'algorithme principal est appelé "*3-2-3*" car chaque itération démarre avec 3 points, génère 2 nouveaux points et écarte 2 de ces points, pour finir avec 3 points (incluant au moins l'un des nouveaux points). Si l'on considère ce principe du point de vue des intervalles, l'algorithme aurait d'ailleurs pu s'appeler "*2-4-2*", car chaque itération démarre avec 2 intervalles adjacents $[x^a,x^b]$, $[x^b,x^c]$, les étend pour obtenir 4 sous-intervalles adjacents $[x^a,x^{a_1}]$, $[x^{a_1},x^b]$, $[x^b,x^{b_1}]$, $[x^{b_1},x^c]$, pour finalement obtenir de nouveau deux intervalles adjacents.
Si l'on pose $d(x^a,x^b)$ et $d(x^b,x^c)$, les distances des deux intervalles de départ $[x^a,x^b]$ et $[x^b,x^c]$, alors, s'ils sont pris de même longueur, les longueurs des deux intervalles à la fin d'une itération seront diminuées de moitié. Le pseudo-code de la figure 2.11 détaille le fonctionnement de l'algorithme *3-2-3*, où *nbIter* est un paramètre représentant le critère d'arrêt.

Procédure 3-2-3
$x^* = x^b$
$iter = 0$
début
(On a et on conserve $f(x^b) \leq f(x^a)$,$f(x^c)$)
tant que $iter < nbIter$ **faire**
$iter = iter + 1$
$x^{a_1} = 0.5 * (x^a + x^b)$
$x^{b_1} = 0.5 * (x^b + x^c)$
si $f(x^b) \leq f(x^{a_1})$ **et** $f(x^b) \leq f(x^{b_1})$ **alors**
Soit (x^{a_1},x^b,x^{b_1}) le nouveau (x^a,x^b,x^c)
sinon si $f(x^{a_1}) \leq f(x^{b_1})$ **alors**
$x^* = x^{a_1}$
Soit (x^a,x^{a_1},x^b) le nouveau (x^a,x^b,x^c)
sinon
$x^* = x^{b_1}$
Soit (x^b,x^{b_1},x^c) le nouveau (x^a,x^b,x^c)
fin si
fin tant que
fin

Figure 2.11 – Pseudo-code de l'algorithme *3-2-3*

2.8.1.2.5 Illustration du fonctionnement des procédures : Les procédures *2-1-2* et *3-2-3* sont illustrées dans les figures 2.12 et 2.13, respectivement, en utilisant une représentation en 2 dimensions.



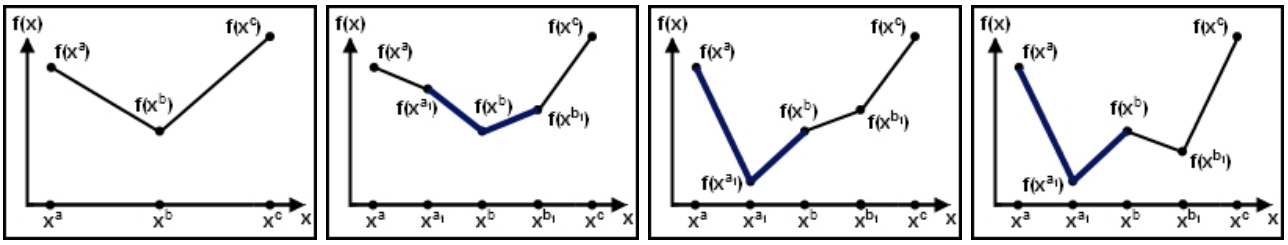
(a) Cas initial (b) Cas $f(x^b) \leq f(x^a)$ (c) Cas $f(x^a) < f(x^b)$ (d) Cas $f(x^a) < f(x^b)$

Figure 2.12 – Les différents cas possibles de la procédure *2-1-2*.

La figure 2.12(a) montre la configuration de départ de la procédure 2-1-2, où $f(x^a) < f(x^c)$. Une ligne a été tracée entre les points $f(x^a)$ et $f(x^c)$ pour clarifier leur relation, mais elle n'a pas de rôle dans la procédure elle-même. La figure 2.12(b) illustre le cas où $f(x^b) \leq f(x^a)$. Les lignes qui joignent successivement $f(x^a)$, $f(x^b)$, et $f(x^c)$ sont accentuées, afin d'indiquer que cette configuration est caractéristique de celle de la méthode *3-2-3*, terminant ainsi la procédure *2-1-2* et démarrant la méthode *3-2-3*.

Les figures 2.12(c) et 2.12(d) illustrent deux versions du même cas, où $f(x^a) < f(x^b)$. Dans ces deux exemples, la ligne joignant $f(x^a)$ et $f(x^b)$ est accentuée pour indiquer que la configuration résultante est caractéristique de la méthode 2-1-2. La procédure 2-1-2 continue donc sur cette portion accentuée du diagramme, et x^b devient le nouveau x^c.

Une troisième possibilité de ce cas (pour $f(x^a) < f(x^b)$) peut survenir si l'on a également $f(x^b) > f(x^c)$. Le traitement est alors le même que celui illustré dans les figures 2.12(c) et 2.12(d).



(a) Cas initial (b) Cas $f(x^b) \leq f(x^a)$ (c) Cas $f(x^a) < f(x^b)$ (d) Cas $f(x^a) < f(x^b)$

Figure 2.13 – Les différents cas possibles de la procédure *3-2-3*.

La figure 2.13(a) montre la configuration initiale pour la procédure 3-2-3, où $f(x^b) \leq f(x^a)$ et $f(x^b) \leq f(x^c)$. Une fois encore, les points représentant les valeurs de la fonction objectif sont connectés par une ligne à des fins illustratives. La figure 2.13 inclut également les points x^{a_1} et x^{b_1} associés à leur image par la fonction objectif $f(x^{a_1})$ et $f(x^{b_1})$.

La figure 2.13(b) illustre la situation dans laquelle $f(x^b) \leq f(x^{a_1})$ et $f(x^b) \leq f(x^{b_1})$. Dans ce cas, x^b ne change pas car il a la plus petite valeur pour f, et la séquence x^{a_1}, x^b, x^{b_1} représente alors une configuration 3-2-3 valide, comme indiqué par les lignes accentuées joignant $f(x^{a_1})$, $f(x^b)$, et $f(x^{b_1})$. L'itération suivante de la procédure 3-2-3 recommence alors, avec x^{a_1} comme nouveau x^a et x^{b_1} comme nouveau x^c.

Les figures 2.13(c) et 2.13(d) illustrent deux versions du cas où les conditions de la figure 2.13(b) ne sont pas satisfaites (c'est-à-dire $f(x^{a_1}) < f(x^b)$ ou $f(x^{b_1}) < f(x^b)$, et $f(x^{a_1}) \leq f(x^{b_1})$). Le point x^{a_1} est alors qualifié pour devenir le nouveau x^b, et dans les figures 2.13(c) et 2.13(d) nous avons donc accentué la ligne joignant $f(x^a)$, $f(x^{a_1})$ et $f(x^b)$. Cette nouvelle configuration est donc bien valide pour la prochaine itération de *3-2-3*.

Il reste le cas où aucune des conditions des figures 2.13(b) et 2.13(c) (et 2.13(d)) n'est satisfaite. Nous avons alors $f(x^{b_1}) < f(x^b)$ et $f(x^{b_1}) < f(x^{a_1})$. Cette situation est la même que celles illustrées dans les figures 2.13(c) et 2.13(d), en inversant les rôles de x^{b_1} et x^{a_1}. Nous n'avons donc pas inclus de diagramme supplémentaire pour l'illustrer.

Les diagrammes présentés permettent de comprendre la forme "basique" de la procédure *3-2-3*. Celle-ci néglige d'examiner plus en profondeur des régions potentiellement intéressantes - comme c'est le cas de la figure 2.13(d), où le triplet (x^b, x^{b_1}, x^c) peut raisonnablement être considéré comme un candidat pour devenir le prochain (x^a, x^b, x^c). Une version plus élaborée de la méthode *3-2-3*, permettant de prendre en compte ces considérations, est donnée dans [Glover 10].

2.9 Hybridation des méthodes EUS et *3-2-3* : EM323

Notre algorithme hybride incorpore une procédure de recherche globale issue de celle utilisée par la méthode EUS. Nous avons également remplacé la méthode d'optimisation unidimensionnelle d'EUS par la méthode *3-2-3*. Le nouvel algorithme résultant, nommé *Enhanced Multidimensional 3-2-3* (EM323) [Gardeux 11a], a alors pour but de résoudre des problèmes plus complexes qu'EUS seul. Nous avons également incorporé de nouvelles fonctionnalités, afin d'accélérer la convergence de ce nouvel algorithme et ainsi explorer l'espace de recherche de manière plus approfondie.

2.9.1 Parcours des dimensions en suivant une *oscillation stratégique*

EM323 commence par créer une solution initiale x aléatoirement dans l'espace de recherche. Ensuite, plutôt que de parcourir toutes les dimensions dans l'ordre, nous utilisons une approche dite de *strategic oscillation*, qui parcourt, à chaque itération, un ensemble de dimensions N_o en commençant par $N_o = \{1,...,n\}$. Puis, à la fin de l'itération, les dimensions qui n'ont pas fourni d'amélioration sont supprimées de cet ensemble. Ainsi, à l'itération suivante, N_o contient uniquement les dimensions qui ont amélioré la solution lors de l'itération précédente. Cette procédure réduit donc graduellement la taille de N_o, jusqu'à ce qu'il devienne vide. Puis, nous "oscillons", en réinitialisant $N_o = \{1,...,n\}$ et en recommençant. Ceci correspond au modèle classique appelé *one-sided oscillation*), d'autres types de modèles existent pour l'oscillation stratégique et sont discutés dans [Glover 95].)

2.9.2 Granularité (ou précision) de la recherche

Nous utilisons un vecteur $\delta = (\delta_1, ..., \delta_n)$ pour déterminer la granularité de l'algorithme EM323. Celui-ci est analogue au vecteur h utilisé avec l'algorithme EUS. En particulier, δ est utilisé pour déterminer les valeurs θ_{max} et θ_{min} sur chaque dimension i en fixant $\theta_{max} = x_i + \delta_i$ et $\theta_{min} = x_i - \delta_i$ de telle façon que θ_{max} et θ_{min} correspondent aux valeurs x_i^u et x_i^d de EUS. Chaque composante δ_i du vecteur δ est initialisée par :

$$\delta_i = (U_i - L_i) * rand \tag{2.9.1}$$

où *rand* est une valeur aléatoire sur l'intervalle [0,1]. Nous utilisons un paramètre nommé $NCut$ afin de diviser l'intervalle $[\theta_{min}, \theta_{max}]$ en $NCut$ sous-intervalles de longueurs égales. Nous examinons ainsi des triplets comme spécifiés dans l'équation (2.8.4) et lançons l'algorithme *3-2-3* (précédé par l'algorithme *2-1-2* si c'est nécessaire).

Nous faisons également une oscillation lorsque nous modifions la granularité de la recherche. En effet, dans le cas des fonctions non séparables, il peut arriver qu'avec un "grain plus fin" (c'est-à-dire une précision plus élevée) la recherche découvre un nouvel optimum local et s'y retrouve piégée. Nous allons alors permettre à l'algorithme de retourner en arrière, en ce sens qu'il pourra dégrader sa précision pour sortir de ces minimums locaux.

Pour expliquer le principe, on définit un bloc comme une série d'itérations allant de $N_o = \{1,...,n\}$ à $N_o = \emptyset$. On dira alors qu'un bloc est "fructueux" si une amélioration de la solution a eu lieu durant au moins une itération de ce bloc (ou "infructueux" si le parcours de l'ensemble des dimensions échoue à produire une amélioration). Si le bloc a été "infructueux", nous posons

$\delta = \delta * rand$ pour améliorer la granularité de la recherche. Lorsque le bloc est "fructueux" au contraire, nous posons $\delta = \delta / rand$ pour dégrader la précision et ainsi revenir à une granularité plus large.

2.9.3 Méthode EM323

L'algorithme EUS ne faisait qu'une approximation rapide de la solution x sur chaque dimension, en ne considérant que deux des solutions voisines de x : x^u et x^d. Dans notre implémentation d'EM323, nous voulions aller plus loin dans cette recherche, tout en conservant une certaine vitesse de convergence. Ainsi, nous avons pris comme compromis de n'appliquer l'algorithme 3-2-3 que sur le meilleur triplet $(x(\theta_{q-1}), x(\theta_q), x(\theta_{q+1}))$ trouvé lors de la séparation du segment en $NCut$ sous-segments. Pour les mêmes raisons, les paramètres $nbIter$ et $maxIter$ sont fixés à $nbIter = maxIter = 3$. Le pseudo-code d'une itération de la procédure EM323 est détaillé dans la figure 2.14.

Procédure EM323
$N_o = \{1,...,n\}$
début
  pour $i \in N_o$ **faire**
    améliorer x le long de la dimension i en utilisant la méthode *3-2-3*. (On obtient x^*)
    si *3-2-3* n'a pas trouvé de meilleure solution que x **alors**
      $N_o = N_o - \{i\}$
    sinon
      $x = x^*$
    fin si
  fin pour
  si $N_o = \emptyset$ **alors**
    $N_o = \{1,...,n\}$
    si $\delta < \delta_{min}$ **alors**
      appeler la procédure de redémarage
    sinon si N_o est passé de $N_o = N$ à $N_o = \emptyset$ en une seule itération **alors**
      (améliorer la granularité de la *line search*)
      $\delta = \delta * rand$ (rand représentant un nombre aléatoire réel dans l'intervalle [0,1])
    sinon
      (dégrader la granularité de la *line search*)
      $\delta = \delta / rand$ (rand représentant un nombre aléatoire réel dans l'intervalle [0,1])
    fin si
  fin si
fin

Figure 2.14 – Une itération de la méthode EM323

La valeur δ_{min} est fixée à 10^{-15} pour les mêmes raisons que celles décrites précédemment. Nous utilisons également la même procédure de redémarrage que pour l'algorithme EUS. Le deuxième paramètre d'EUS, noté r, était fixé à 0,5 ; cependant nous avions constaté que l'on pouvait, dans certains cas, obtenir de meilleurs résultats avec d'autres valeurs. Nous avons donc choisi de le rendre aléatoire dans la procédure EM323, avec l'anticipation que cela pourrait améliorer la qualité des solutions. De plus, la motivation principale est de supprimer totalement le paramètre r de notre algorithme final.

La méthode EM323 résultante incorpore donc les fonctionnalités suivantes :
- Un algorithme global de recherche issu de EUS avec :
  - Une mémoire, stockant les optimums locaux trouvés tout au long de la recherche ;
  - Une procédure de redémarrage, exploitant la mémoire, afin de générer de nouvelles solutions ;
  - Un parcours dimension par dimension en suivant une procédure d'*oscillation stratégique* ;
  - Une oscillation de la granularité de la recherche, permettant d'améliorer ou de diminuer la précision en fonction des résultats.
- Une procédure d'optimisation unidimensionnelle *3-2-3* permettant une meilleure approximation de l'optimum de la fonction restreinte à une dimension.
- Quatre paramètres : $maxIter$, $nbIter$, δ_{min} et $NCut$, dont trois peuvent être fixés : $\delta_{min} = 10^{-15}$ et $nbIter = maxIter = 3$.

2.10 Protocole expérimental

2.10.1 Comparaison des algorithmes EUS et EM323

Afin d'estimer les performances de ce nouvel algorithme par rapport à EUS, nous l'avons tout d'abord comparé aux données que nous avions sur le *benchmark* proposé par ISDA'09. Le paramètre $NCut$ est utilisé pour diviser l'espace de recherche le long de la direction à optimiser par la procédure de *line search*. Plus cette valeur est grande et meilleure sera la granularité de la recherche le long de la direction ; cependant cela ralentira la convergence. Nous avons donc choisi de fixer ce paramètre à 5, comme compromis entre une recherche assez fine et une convergence relativement rapide. Les résultats sont consignés dans le tableau 2.13.

n=500	EM323	EUS
f_1	**5,80E-12**	**5,37E-12**
f_2	2,04E+01	**1,35E-04**
f_3	1,25E+03	4,78E+02
f_4	**7,08E-12**	**8,52E-12**
f_5	1,77E-03	**3,53E-12**
f_6	**6,50E-12**	**5,99E-12**
f_7	**0,00E+00**	**0,00E+00**
f_8	3,91E+05	6,06E+03
f_9	**2,21E-06**	3,52E+03
f_{10}	**0,00E+00**	**0,00E+00**
f_{11}	**1,10E-02**	9,43E+02

Tableau 2.13 – Comparaison des erreurs commises par EM323 et EUS sur le *benchmark* de la conférence ISDA'09, pour $n = 500$.

Nous pouvons remarquer que les deux algorithmes ont le même comportement pour les fonctions f_1, f_4, f_6, f_7 et f_{10}, pour lesquelles ils convergent ; au contraire, ils ne convergent pas sur les fonctions f_3 et f_8. Ils ont donc le même comportement pour 7 fonctions sur 11. Il est tout de même important de noter qu'EUS ne convergeait sur f_4 qu'avec une valeur très spécifique de r, sinon il divergeait. Ici EM323 converge parfaitement sur cette fonction, sans que l'on ait à le paramétrer.
Sur les fonctions f_2 et f_5, EUS obtient des résultats légèrement meilleurs à ceux de EM323. Ceci est dû à une convergence moins rapide de EM323 du fait de la complexité plus importante de son modèle. Les résultats les plus étonnants étant pour la fonction f_5, qui ne semblait a priori poser aucun problème pour EUS. EM323 parvient tout de même à converger, mais il nécessite plus de temps.
Enfin, les fonctions f_9 et f_{11} posaient de nombreux problèmes, non seulement à EUS, mais également aux autres algorithmes présentés dans la conférence. Pour ces fonctions, EM323 obtient de bons résultats. En effet, sur ces fonctions, il parvient à converger avec une bonne précision, qui peut d'ailleurs être améliorée s'il dispose de plus de temps. C'est le point le plus important ici, car ces fonctions sont assez complexes et non séparables. Ce comportement montre que EM323 semble plus apte à résoudre des fonctions plus complexes.

2.10.2 Performances d'*EM323* sur un *benchmark* de 19 fonctions

2.10.2.1 Caractéristiques du *benchmark*

Nous avons testé ce nouvel algorithme sur un *benchmark* proposé pour l'article [Gardeux 11a]. Ce nouvel ensemble de problèmes de grande dimension contient les 11 fonctions déjà utilisées pour la conférence, ainsi que 8 nouvelles fonctions f_{12}-f_{19^*}.

f	Nom	Séparable	m_{ns}
f_1	*Shifted Sphere*	X	-
f_2	*Shifted Schwefel Problem 2.21*	-	-
f_3	*Shifted Rosenbrock*	-	-
f_4	*Shifted Rastrigin*	X	-
f_5	*Shifted Griewank*	-	-
f_6	*Shifted Ackley*	X	-
f_7	*Schwefel Problem 2.22*	X	-
f_8	*Schwefel Problem 1.2*	-	-
f_9	*Extended* F_{10}	-	-
f_{10}	*Bohachevsky #1*	-	-
f_{11}	*Schaffer* #2	-	-
f_{12}	Hybridation F9 & F1	-	0.25
f_{13}	Hybridation F9 & F3	-	0.25
f_{14}	Hybridation F9 & F4	-	0.25
f_{15}	Hybridation F10 & F7	-	0.25
f_{16^*}	Hybridation F9 & F1	-	0.5
f_{17^*}	Hybridation F9 & F3	-	0.75
f_{18^*}	Hybridation F9 & F4	-	0.75
f_{19^*}	Hybridation F10 & F7	-	0.75

Tableau 2.14 – Caractéristiques des fonctions du *benchmark*

Celles-ci sont des compositions hybrides créées à partir de deux fonctions, dont une non séparable. La procédure utilisée pour hybrider une fonction non séparable f_{ns} avec une autre fonction f' (pour créer $f = f_{ns} \oplus f'$) consiste en trois étapes principales :
– diviser la solution en deux composantes ;
– évaluer chaque composante avec une fonction différente ;
– combiner les résultats.
Le mécanisme permettant de diviser la solution utilise un paramètre, m_{ns}, qui caractérise la quantité de variables à évaluer par f_{ns}. Ainsi, plus la valeur de m_{ns} sera élevée, et plus la fonction résultante sera difficile à optimiser dimension par dimension, car il y aura une plus grande association entre les variables. Une explication plus détaillée des fonctions du *benchmark* est disponible dans [Herrera 10b], certaines caractéristiques de l'ensemble des fonctions du *benchmark* sont résumées dans le tableau 2.14.

Ce *benchmark* semble donc plus difficile que le premier pour nos algorithmes, car il contient de nombreuses fonctions non séparables. Pour chaque fonction f, les expériences ont été conduites en dimensions $n = 50, 100, 200, 500$ et 1000. L'algorithme est relancé 25 fois sur chaque fonction, et l'erreur moyenne est calculée. Le critère d'arrêt est atteint lorsque le nombre maximum autorisé d'évaluations de la fonction objectif est atteint ; il est fixé à $5000*n$. Les erreurs obtenues par l'algorithme EM323 pour chaque fonction du *benchmark* sont listées dans le tableau 2.15

f	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
f_1	4,80E-13	9,91E-13	2,15E-12	5,80E-12	1,18E-11
f_2	4,08E-09	3,42E-04	1,92E-01	2,04E+01	2,31E+01
f_3	6,12E+01	2,10E+02	4,47E+02	1,25E+03	1,42E+03
f_4	5,34E-13	1,15E-12	2,23E-12	7,08E-12	1,33E-11
f_5	3,05E-13	5,91E-13	3,95E-04	1,77E-03	3,94E-04
f_6	5,66E-13	1,14E-12	2,42E-12	6,50E-12	1,29E-11
f_7	0,00E+00	0,00E+00	0,00E+00	0,00E+00	-
f_8	2,08E+02	6,31E+02	3,37E+04	3,91E+05	1,46E+06
f_9	0,00E+00	3,29E-08	1,31E-08	2,21E-06	7,88E-02
f_{10}	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
f_{11}	2,16E-07	9,99E-09	1,92E-06	1,10E-02	5,43E-01
f_{12}	9,88E-08	1,51E-02	4,13E-07	3,41E-01	6,16E-01
f_{13}	1,96E+01	5,97E+01	2,97E+02	1,19E+03	1,24E+03
f_{14}	1,32E-07	3,74E-07	2,10E-01	1,51E-01	1,03E+00
f_{15}	0,00E+00	0,00E+00	0,00E+00	0,00E+00	-
f_{16^*}	2,52E-07	4,57E-07	9,40E-07	4,71E-03	1,29E-01
f_{17^*}	8,58E+01	9,60E+01	5,73E+01	2,14E+02	3,00E+02
f_{18^*}	3,12E-07	5,02E-02	2,63E-03	2,28E-01	1,66E-01
f_{19^*}	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00

Tableau 2.15 – Erreurs obtenues par EM323 sur le *benchmark* des 19 fonctions.

Il est important de noter que nous n'avons pas pu calculer les résultats pour les fonctions f_7 et f_{15} en dimension $n = 1000$, à cause des problèmes de représentation des nombres sur la machine, liés au langage de programmation que nous avons utilisé. En effet, lors du calcul de la valeur de la fonction objectif, celle-ci dépasse la taille autorisée pour une variable codée en double

précision. Elle est alors approchée à la valeur $Infinity$, borne supérieure de l'ensemble des valeurs de double précision, rendant impossible la comparaison entre les différentes solutions.

Dans un premier temps, si nous considérons ces fonctions dans le cadre d'une dimension "relativement faible" ($n = 50$), nous pouvons constater que EM323 est capable de résoudre 15 fonctions parmi les 19 avec une bonne précision ($e < 10E - 7$). Les fonctions les plus problématiques étant la fonction de *Rosenbrock* f_3 et ses hybridations f_{13} et f_{17^*}. Typiquement, ce sont les fonctions sur lesquelles notre algorithme aura un comportement de "type 3", comme défini précédemment.

De plus, la fonction f_8 résiste également à notre algorithme, mais son comportement est quelque peu différent. En effet, il semblerait que notre algorithme converge, mais trop lentement pour atteindre une précision suffisante dans le temps imparti. À titre de comparaison, la figure 2.15 montre les graphes de convergence de EUS et EM323 sur cette fonction, en dimension $n = 50$. On voit bien sur cette image que les deux algorithmes convergent, mais à des vitesses différentes. Nous pouvons donc dire que EM323 a un comportement de "type 2", malgré les mauvais résultats en faible dimension.

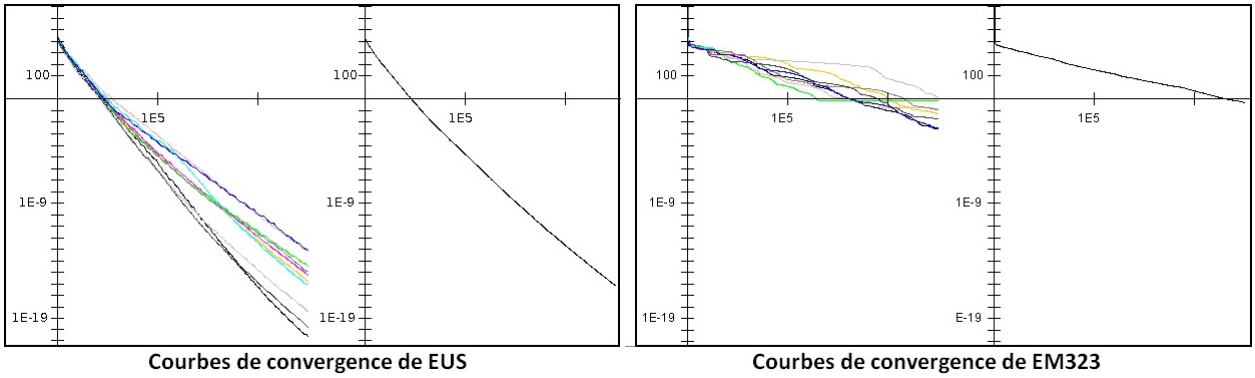


Figure 2.15 – Graphes de convergence de EUS et EM323 sur f_8 : *Schwefel Problem 1.2*, pour $n = 50$.

Si nous considérons maintenant l'ensemble des tests, nous pouvons constater que les fonctions f_1, f_4, f_6, f_7, f_{10}, f_{15} et f_{19^*} sont bien résolues tout au long de l'augmentation de la dimension. EM323 a donc un comportement de "type 1" sur celles-ci, puisqu'il parvient à les résoudre sans problème.
Enfin, d'autres fonctions sont résolues par EM323, mais sans atteindre une excellente précision en faible dimension. De plus, sur ces fonctions, la précision semble chuter proportionnellement à l'augmentation de la dimension. C'est le cas des fonctions f_2, f_5, f_9, f_{11}, f_{12}, f_{14} f_{16^*} et f_{18^*}. On pourrait donc considérer ces fonctions comme de "type 2" pour notre algorithme, mais une étude des courbes de convergence nous montre que ce n'est pas tout à fait le cas. Pour les fonctions f_5, f_{12}, f_{14}, f_{16^*} et f_{18^*}, le comportement est légèrement différent (on pourra voir par exemple le comportement de EM323 sur la fonction f_5 dans la figure 2.16). En effet, pour ces fonctions, on remarque que les résultats se détériorent brusquement lorsque la dimension atteint $n = 100$ ou $n = 200$ ($e = 1E - 12$ avant, puis $1E - 3$ à $1E + 1$ ensuite). En fait, la plupart des essais parmi les 25 réussissent à atteindre l'optimum (précision inférieure à $1E-10$) ; cependant, plus la dimension augmente, et plus l'algorithme a du mal a trouver l'optimum global et peut se retrouver bloqué dans des optimums locaux. Ce phénomène n'a lieu que sur quelques tests parmi les 25 (1 ou 2 la plupart du temps), cependant cela biaise complètement l'erreur moyenne, dont la valeur est absorbée par le résultat de l'optimum local. Nous supposons que ce phénomène

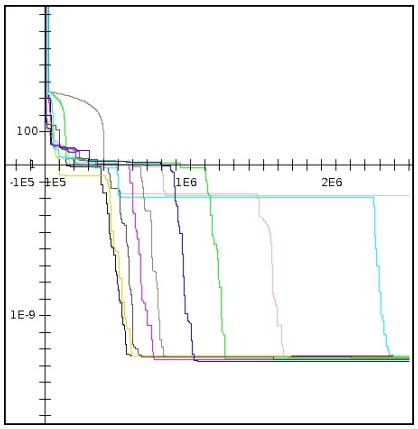


Figure 2.16 – Graphes de convergence de EM323 sur la fonction f_5 : *Shifted Griewank*, pour $n = 500$.

est dû à un manque d'exploration en haute dimension, où la procédure de redémarrage n'est pas appelée assez fréquemment, proportionnellement à la grandeur de l'espace de recherche. Nous obtenons ainsi un nouveau type de comportement de notre algorithme sur ces fonctions, que nous catégoriserons de "type 1.5". Les fonctions pour lesquelles EM323 a réellement un comportement de "type 2" sont les fonctions f_2, f_8, f_9 et f_{11}.

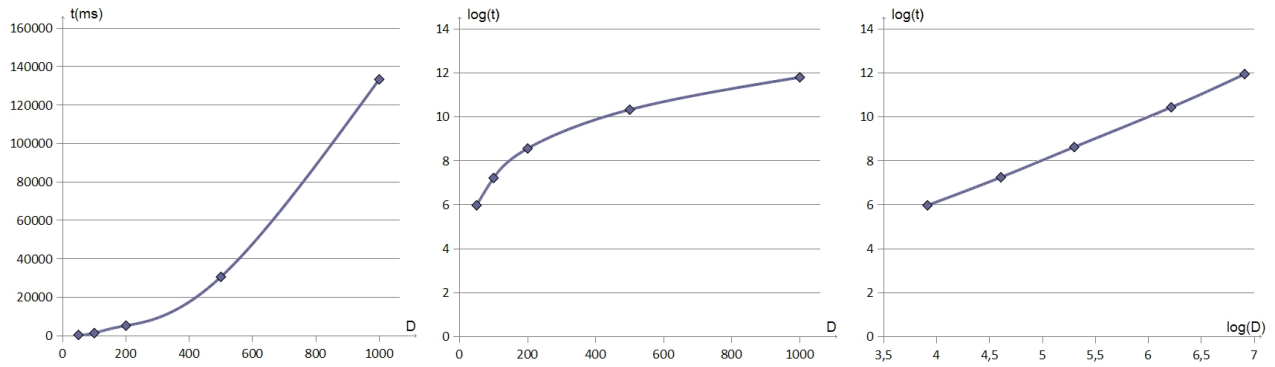
2.10.3 Étude du temps d'exécution de l'algorithme EM323

Afin d'analyser davantage le comportement de notre algorithme, nous avons observé les temps d'exécution moyens de notre algorithme parmi les 25 tests sur chaque fonction f. Le tableau 2.16 récapitule ces résultats.

f	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
f_1	0,394	1,368	5,233	30,605	133,380
f_2	0,461	1,745	6,508	35,329	162,728
f_3	0,413	1,539	5,868	30,443	147,325
f_4	1,211	4,591	16,995	90,785	379,014
f_5	1,333	5,160	19,476	113,553	452,131
f_6	1,253	4,713	18,462	110,734	452,609
f_7	0,400	1,286	4,587	28,468	-
f_8	0,351	1,140	3,977	23,515	116,735
f_9	10,888	38,809	153,926	1241,516	4806,078
f_{10}	1,602	6,219	25,820	147,766	645,547
f_{11}	12,104	47,855	191,999	1074,136	4784,208
f_{12}	3,152	12,434	50,221	292,506	1306,567
f_{13}	2,826	11,428	46,427	336,313	1396,375
f_{14}	3,331	13,399	52,675	377,015	1525,203
f_{15}	0,753	2,765	10,904	59,735	-
f_{16^*}	6,093	23,860	95,058	620,141	2227,735
f_{17^*}	8,025	30,013	119,783	863,937	3679,783
f_{18^*}	8,134	30,003	123,796	882,250	3884,616
f_{19^*}	1,471	5,190	21,826	122,500	489,828

Tableau 2.16 – Temps d'exécution moyen (en secondes) pour EM323

Les expériences ont été faites en Java (1.6) sur un ordinateur ayant les caractéristiques suivantes :
– CPU : Intel Core 2 Duo, T8100 @ 2.10GHz
– RAM : 2,00 GB
– OS : Windows XP
Nous pouvons constater que le temps d'exécution dépend de la fonction objectif et de la dimension. Pour illustrer ce phénomène, nous avons tracé le graphe $t = f(n)$ pour la fonction f_1, en désignant par t le temps d'exécution et n la dimension (cf figure 2.17(a)). Les résultats montrent que le temps d'exécution semble augmenter en fonction du carré de la dimension. Pour confirmer cette relation, nous avons tracé deux nouveaux graphes à partir des mêmes données : le premier utilise une échelle logarithmique pour t, et une échelle décimale pour n (cf figure 2.17(b)) ; l'autre est représenté avec les deux axes en échelle logarithmique (cf figure 2.17(c)). Nous avons ensuite calculé les coefficients de corrélation des deux courbes. Pour le premier graphe, nous avons trouvé $\alpha = 0{,}941$ et pour le second $\alpha = 0{,}999$, ce qui indique que la variation du temps par rapport à la dimension est une fonction puissance. Une simple régression linéaire nous permet de déterminer le degré de cette fonction puissance, nous trouvons : $log(t) = 1.942 * log(n) - 1.685$, ce qui donne $t = n^{1.94} * e^{-1.7}$. Par conséquent, nous pouvons conclure que le temps d'exécution augmente approximativement comme le carré (1,94) de la dimension pour la fonction f_1. Le même test a été effectué pour toutes les fonctions et les résultats sont sensiblement équivalents. Le temps d'exécution étant directement proportionnel au nombre d'évaluations de la fonction objectif, nous obtenons une mesure permettant d'évaluer la complexité de notre algorithme : $C = O(n^2)$. Ce résultat confirme d'ailleurs le calcul de la complexité théorique : si l'on suppose qu'une itération nécessite $\Theta(n)$ évaluations de la fonction objectif, et que les méthodes se limitent à $O(n)$ évaluations (comme défini dans le *benchmark*), la complexité est bien en $O(n^2)$.



(a) Fonction $f_1 : t = f(n)$ (b) Fonction $f_1 : log(t) = f(n)$ (c) Fonction $f_1 : log(t) = f(log(n))$

Figure 2.17 – Temps d'exécution d'EM323 pour f_1 en fonction de la dimension du problème.

2.10.4 Comparaison d'EM323 avec d'autres algorithmes

Maintenant que le comportement d'EM323 est plus clair, nous avons comparé les résultats qu'il procure sur le *benchmark* avec ceux d'autres algorithmes.

2.10.4.1 Comparaison avec trois algorithmes évolutionnaires

Afin d'avoir une première idée des performances de EM323 sur le *benchmark*, nous l'avons comparé à trois algorithmes évolutionnaires réputés pour être très efficaces. Le paramétrage

utilisé pour chacun est celui recommandé par les auteurs, à moins que des tests n'aient montré que d'autres paramètres étaient plus efficaces :

- *Differential Evolution*(DE) [Storn 97] : Cette méthode a été expliquée plus en détail dans la section 1.12.2. L'opérateur de croisement utilisé ici est rand/1/exp, car il donne de meilleures performances. Les paramètres F et CR ont été fixés respectivement à 0,5 et 0,9. Normalement, la taille de la population est définie en fonction de la dimension du problème ($10n$ ou $3n$), cependant, en haute dimension ce critère n'est pas adéquat, et une limite maximale doit être fixée. Pour les expériences, une population de 60 individus a été utilisée (des résultats similaires ont été atteints avec une population de taille 100).
- *Real-coded* CHC [Eshelman 93] : Le seuil initial est fixé à $L = 20 * n$. Le paramètre α utilisé par l'opérateur de croisement BLX-α est fixé à 0,5. La taille de la population est de 50.
- G-CMA-ES [Auger 05] : Cette méthode a gagné la compétition de la conférence CEC'05 sur l'optimisation à paramètres réels, elle a été présentée plus en détail dans la section 1.12.2. Les paramètres utilisés sont ceux suggérés par Auger et Hansen. La solution initiale est choisie aléatoirement, de manière uniforme, dans l'espace de recherche et la taille de la distribution σ est fixée à un tiers de la taille de l'espace de recherche.

Les erreurs obtenues par ces trois algorithmes pour les 19 fonctions du *benchmark* ont été listées dans les tableaux 2.17, 2.18 et 2.19. Le protocole expérimental est le même que celui utilisé pour l'algorithme EM323.

f	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
f_1	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
f_2	3,60E-01	4,45E+00	1,92E+01	5,35E+01	8,46E+01
f_3	2,89E+01	8,01E+01	1,78E+02	4,76E+02	9,69E+02
f_4	3,98E-02	7,96E-02	1,27E-01	3,20E-01	1,44E+00
f_5	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
f_6	1,43E-13	3,10E-13	6,54E-13	1,65E-12	3,29E-12
f_7	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
f_8	3,44E+00	3,69E+02	5,53E+03	6,09E+04	2,46E+05
f_9	2,73E+02	5,06E+02	1,01E+03	2,52E+03	5,13E+03
f_{10}	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
f_{11}	6,23E-05	1,28E-04	2,62E-04	6,76E-04	1,35E-03
f_{12}	5,35E-13	5,99E-11	9,76E-10	7,07E-09	1,68E-08
f_{13}	2,45E+01	6,17E+01	1,36E+02	3,59E+02	7,30E+02
f_{14}	4,16E-08	4,79E-02	1,38E-01	1,35E-01	6,90E-01
f_{15}	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00
f_{16^*}	1,56E-09	3,58E-09	7,46E-09	2,04E-08	4,18E-08
f_{17^*}	7,98E-01	1,23E+01	3,70E+01	1,11E+02	2,36E+02
f_{18^*}	1,22E-04	2,98E-04	4,73E-04	1,22E-03	2,37E-03
f_{19^*}	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00

Tableau 2.17 – Erreurs obtenues pour l'algorithme DE

Pour comparer ces algorithmes, différentes méthodes sont possibles. Nous allons commencer par présenter une méthode basée le comportement des algorithmes sur les différentes fonctions. À cet effet, nous comptons le nombre de fonctions pour lesquelles chaque algorithme a un comportement de "type 1". Puis nous faisons de même avec le "type 1.5", et ainsi de suite. Nous obtenons le tableau 2.20. Ce calcul est possible, car nous disposons des valeurs de la médiane pour ces trois algorithmes. La différence entre la médiane et la moyenne peut donc nous donner une idée des fonctions de "type 1.5", c'est-à-dire pour lesquelles l'algorithme est

f	$n = 50$	$n = 100$	$n = 200$	$n = 500$	$n = 1000$
f_1	1,67E-11	3,56E-11	8,34E-01	2,84E-12	1,36E-11
f_2	6,19E+01	8,58E+01	1,03E+02	1,29E+02	1,44E+02
f_3	1,25E+06	4,19E+06	2,01E+07	1,14E+06	8,75E+03
f_4	7,43E+01	2,19E+02	5,40E+02	1,91E+03	4,76E+03
f_5	1,67E-03	3,83E-03	8,76E-03	6,98E-03	7,02E-03
f_6	6,15E-07	4,10E-07	1,23E+00	5,16E+00	1,38E+01
f_7	2,66E-09	1,40E-02	2,59E-01	1,27E-01	3,52E-01
f_8	2,24E+02	1,69E+03	9,38E+03	7,22E+04	3,11E+05
f_9	3,10E+02	5,86E+02	1,19E+03	3,00E+03	6,11E+03
f_{10}	7,30E+00	3,30E+01	7,13E+01	1,86E+02	3,83E+02
f_{11}	2,16E+00	7,32E+01	3,85E+02	1,81E+03	4,82E+03
f_{12}	9,57E-01	1,03E+01	7,44E+01	4,48E+02	1,05E+03
f_{13}	2,08E+06	2,70E+06	5,75E+06	3,22E+07	6,66E+07
f_{14}	6,17E+01	1,66E+02	4,29E+02	1,46E+03	3,62E+03
f_{15}	3,98E-01	8,13E+00	2,14E+01	6,01E+01	8,37E+01
f_{16^*}	2,95E-09	2,23E+01	1,60E+02	9,55E+02	2,32E+03
f_{17^*}	2,26E+04	1,47E+05	1,75E+05	8,40E+05	2,04E+07
f_{18^*}	1,58E+01	7,00E+01	2,12E+02	7,32E+02	1,72E+03
f_{19^*}	3,59E+02	5,45E+02	2,06E+03	1,76E+03	4,20E+03

Tableau 2.18 – Erreurs obtenues pour l'algorithme CHC

f	$n = 50$	$n = 100$	$n = 200$	$n = 500$
f_1	0,00E+00	0,00E+00	0,00E+00	0,00E+00
f_2	2,75E-11	1,51E-10	1,16E-09	3,48E-04
f_3	7,97E-01	3,88E+00	8,91E+01	3,58E+02
f_4	1,05E+02	2,50E+02	6,48E+02	2,10E+03
f_5	2,96E-04	1,58E-03	0,00E+00	2,96E-04
f_6	2,09E+01	2,12E+01	2,14E+01	2,15E+01
f_7	1,01E-10	4,22E-04	1,17E-01	-
f_8	0,00E+00	0,00E+00	0,00E+00	2,36E-06
f_9	1,66E+01	1,02E+02	3,75E+02	1,74E+03
f_{10}	6,81E+00	1,66E+01	4,43E+01	1,27E+02
f_{11}	3,01E+01	1,64E+02	8,03E+02	4,16E+03
f_{12}	1,88E+02	4,17E+02	9,06E+02	2,58E+03
f_{13}	1,97E+02	4,21E+02	9,43E+02	2,87E+03
f_{14}	1,09E+02	2,55E+02	6,09E+02	1,95E+03
f_{15}	9,79E-04	6,30E-01	1,75E+00	2,82E+262
f_{16^*}	4,27E+02	8,59E+02	1,92E+03	5,45E+03
f_{17^*}	6,89E+02	1,51E+03	3,36E+03	9,59E+03
f_{18^*}	1,31E+02	3,07E+02	6,89E+02	2,05E+03
f_{19^*}	4,76E+00	2,02E+01	7,52E+02	2,44E+06

Tableau 2.19 – Erreurs obtenues pour l'algorithme G-CMA-ES

piégé dans un optimum local et ne peut trouver l'optimum global dans le temps imparti. Ce comportement est pris sur les fonctions f_4, f_{14}, f_{18} par l'algorithme DE ; f_3 et f_5 par G-CMA-ES ; f_1, f_3, f_5, f_{10}, f_{12}, f_{13}, f_{15}, f_{17}, f_{19} par CHC.

	EM323	DE	CMA-ES	CHC
Type 1	7	7	2	0
Type 1.5	4	3	2	9
Type 2	5	5	4	3
Type 3	3	4	11	7

Tableau 2.20 – Erreurs obtenues pour l'algorithme G-CMA-ES

Nous pouvons rapidement constater que les algorithmes G-CMA-ES et CHC n'obtiennent pas de très bons résultats sur le *benchmark*, au contraire de l'algorithme DE. Ce dernier obtient, en effet, des résultats approximativement du même ordre que ceux d'EM323. Si nous voulons classer les algorithmes, il suffit de commencer par comparer le nombre de fonctions qu'ils résolvent avec un comportement de "type 1", l'algorithme qui en résout le plus sera considéré comme meilleur. Si deux algorithmes sont à égalité (comme c'est le cas ici entre EM323 et DE), nous considérons le comportement de "type 1.5", et ainsi de suite, jusqu'au "type 3". Cette méthode montre qu'EM323 obtient des résultats légèrement meilleurs que DE, même si ce n'est pas très significatif.
Cette classification nous permet de remarquer que les algorithmes G-CMA-ES et CHC ont un grand nombre de fonctions de "type 1.5", pouvant montrer un défaut d'exploration de l'espace de recherche. Le peu de résultats de "type 1" renforce le fait que ces algorithmes ne sont pas bien adaptés aux problèmes de grande dimension (en tout cas, sous cette forme).

En général, on peut dire que notre algorithme obtient des résultats équivalents ou meilleurs que les trois autres algorithmes, sur toutes les fonctions, à part pour les fonctions f_2 et f_8, où G-CMA-ES obtient des résultats de très bonne qualité, même en haute dimension.

Pour chaque fonction du *benchmark*, nous avons voulu vérifier si les différences entre les solutions trouvées par EM323 et les autres algorithmes étaient statistiquement significatives. À cet effet, nous avons utilisé le logiciel de statistiques *R*, afin de faire un test de Wilcoxon apparié ([García 09]). Cette méthode a été ajustée pour l'appariement par la méthode de Holm. Le tableau 2.21 montre les résultats pour chaque paire d'algorithmes (EM323 et X). La *p-value* est calculée pour l'hypothèse nulle suivante : H_0 ="*Il n'y a pas de différence significative entre la distribution des solutions engendrées par EM323 et celle des solutions engendrées par* X".

Algorithmes \ $n =$		50	100	200	500	1000
DE	W^+	60	57	38	32	33
	W^-	60	63	82	88	87
	$p-value$	1,0000	0,8904	0,2293	0,1205	0,1354
CHC	W^+	187	190	174	173	138
	W^-	3	0	16	17	15
	$p-value$	1,907E-05	3,815E-06	0,0006	0,0008	0,0021
G-CMA-ES	W^+	158	159	156	143	-
	W^-	32	31	34	28	-
	$p-value$	0,0095	0,0082	0,0124	0,0104	-

Tableau 2.21 – Résultats du test de Wilcoxon pour EM323 contre trois algorithmes évolutionnaires

Ce tableau montre que les *p-values* sont toutes inférieures à 0,05 pour les algorithmes *Real-coded* CHC et G-CMA-ES sur toutes les dimensions. Par conséquent, l'hypothèse nulle est rejetée avec une probabilité d'erreur de $\alpha = 5\%$. Les valeurs W^+ étant très hautes, on peut en conclure que EM323 surpasse ces deux algorithmes quelque soit la dimension du problème. Pour l'algorithme DE, comme la *p-value* est plus grande que 0,05, nous devons accepter l'hypothèse nulle H_0 et ne pouvons donc prétendre qu'il existe une différence statistique significative entre les deux algorithmes.
Nous obtenons donc approximativement la même conclusion qu'avec l'analyse des différents types de fonctions pour chaque algorithme, cependant, avec le test de Wilcoxon, il est impossible de différencier EM323 et DE, alors que notre méthode montre tout de même une légère supériorité de EM323.

Un examen plus approfondi de ces résultats nous fait tout de même réfléchir sur la validité du test de Wilcoxon pour comparer des algorithmes d'optimisation. En effet, il se concentre sur la comparaison des erreurs moyennes, sans référence à leur différence relative. Pour voir ces effets, nous pouvons prendre par exemple la fonction f_4, pour laquelle EM323 (avec une erreur de $1E-12$) obtient des résultats de bien meilleure qualité que DE (avec une erreur de $1E-2$). Cependant, la différence calculée par le test de Wilcoxon sera très faible (de l'ordre de $1E-2$). À l'inverse, les résultats obtenus par les deux méthodes sur la fonction f_3 sont assez similaires. DE obtient des résultats très légèrement meilleurs que EM323, ce qui aboutit à une différence de l'ordre de $1E1$. Ainsi, lorsque le test de Wilcoxon va trier ces différences pour les classer en fonction de leur importance, la différence de $1E1$ sera considérée comme très importante, a contrario de la différence de $1E-2$, qui sera négligeable. Le test de Wilcoxon ne prend en compte que les différences absolues, perdant l'information que la seconde différence représente une plus grande disparité entre les deux algorithmes.
Pour remédier à ce type de distorsion, les calculs devraient être faits après une normalisation logarithmique, afin de comparer correctement les précisions de chaque algorithme, en termes de leurs erreurs relatives.

2.10.4.2 Comparaison à d'autres algorithmes

Après publication de nos résultats [Gardeux 11a], nous avons pu les comparer aux différents algorithmes s'étant confrontés au même *benchmark*. Nous avons alors récupéré les résultats de 13 méthodes différentes, pour avoir une vue plus générale sur les performances de ces algorithmes. Nous ne détaillerons pas ici chacune de ces méthodes, nous engageons donc le lecteur à se référer aux articles cités pour de plus amples informations. Les différents algorithmes sont :

Évolution différentielle :
- SOUPDE [Weber 10]
- DE-D^40+M^m (DE-D+M) [García-Martínez 10]
- GODE [Wang 10]
- GaDE [Yang 10]
- jDElscop [Brest 10]
- SaDE-MMTS (SaDE) [Zhao 10]

Algorithmes mémétiques :
- MOS [LaTorre 10]
- MA-SSW-Chains (MA-SSW) [Molina 10]

Optimisation par essaim particulaire :
- RPSO-vm [García-Nieto 10]
- Tuned IPSOLS (IPSOLS) [Montes de Oca 10]

Autres :
- EvoPROpt [Duarte 10]
- EM323 [Gardeux 11a]
- VXQR [Neumaier 10]

Nous pouvons remarquer que de nombreux papiers sont issus des algorithmes évolutionnaires, et notamment des algorithmes d'évolution différentielle, qui semblent obtenir de bons résultats en haute dimension. Les résultats de ces algorithmes en dimension 1000 ont été consignés dans la figure 2.18.

	SOUPDE	DE-D+M	GODE	GaDE	jDElscop	SaDE	MOS	MA-SSW	RPSO-vm	IPSOLS	EvoPROpt	EM323	VXQR
f_1	0,00E+00	5,07E-16	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	2,87E-15	2,72E-14	0,00E+00	4,00E-05	1,18E-11	0,00E+00
f_2	9,25E+01	7,02E+01	9,02E+01	8,93E+01	6,14E+01	4,88E+01	4,25E-01	1,39E+02	4,29E+01	6,68E-14	3,21E+01	2,31E+01	9,60E+01
f_3	9,62E+02	9,47E+02	9,70E+02	9,45E+02	8,48E+02	0,00E+00	6,15E+01	1,22E+03	3,21E+02	0,00E+00	1,12E+03	1,42E+03	6,35E+02
f_4	3,18E-01	1,23E+00	1,03E+00	0,00E+00	1,99E-01	3,21E+01	0,00E+00	1,58E+03	4,81E-14	0,00E+00	4,08E+02	1,33E-11	6,55E-13
f_5	0,00E+00	2,43E-16	0,00E+00	0,00E+00	0,00E+00	0,00E+00	0,00E+00	5,92E-04	2,14E-01	0,00E+00	3,72E-02	3,94E-04	1,31E-11
f_6	3,41E-13	1,54E-12	2,88E-13	1,66E-14	2,67E-12	0,00E+00	0,00E+00	1,46E-09	4,92E-12	0,00E+00	1,97E+00	1,29E-11	1,70E-11
f_7	3,57E-13	0,00E+00	-	0,00E+00	0,00E+00	0,00E+00	0,00E+00	6,23E-13	3,63E-15	7,13E-11	1,50E-04	-	0,00E+00
f_8	2,13E+05	9,72E+10	1,86E+05	1,77E+04	3,21E+04	1,46E+03	1,94E+05	7,49E+04	9,35E+05	1,20E+05	2,15E+05	1,46E+06	0,00E+00
f_9	7,39E-05	0,00E+00	1,70E-04	0,00E+00	4,40E-03	9,25E+01	0,00E+00	5,99E+03	1,17E+01	9,76E+00	4,07E+02	7,88E-02	1,26E+02
f_{10}	0,00E+00	2,37E-31	0,00E+00	4,62E-01	0,00E+00	0,00E+00	0,00E+00	2,09E-05	0,00E+00	0,00E+00	3,86E+02	0,00E+00	0,00E+00
f_{11}	7,44E-05	0,00E+00	1,73E-04	0,00E+00	8,58E-04	1,85E+02	0,00E+00	5,27E+01	1,10E+01	9,75E+00	3,96E+02	5,43E-01	1,58E+02
f_{12}	0,00E+00	4,43E-12	1,87E-09	3,85E-12	0,00E+00	0,00E+00	0,00E+00	9,48E-02	1,00E-09	1,37E+00	3,23E+01	6,16E-01	2,75E+02
f_{13}	7,29E+02	7,23E+02	7,31E+02	7,15E+02	6,57E+02	6,55E+02	8,80E+01	1,02E+03	1,93E+03	1,28E+00	1,13E+03	1,24E+03	5,92E+02
f_{14}	2,79E-01	5,17E-01	6,06E-01	8,82E-11	3,98E-02	1,40E+02	0,00E+00	7,33E+02	5,27E-01	1,78E+01	4,31E+02	1,03E+00	6,19E+00
f_{15}	2,69E-13	1,78E-17	-	0,00E+00	0,00E+00	0,00E+00	0,00E+00	1,16E-13	0,00E+00	1,39E-10	1,26E+02	-	2,35E+00
f_{16^*}	0,00E+00	2,23E-11	4,59E-09	2,35E-12	8,04E-01	0,00E+00	0,00E+00	2,19E+00	9,50E-01	2,41E+00	8,44E+01	1,29E-01	9,65E+01
f_{17^*}	2,31E+02	2,25E+02	2,36E+02	2,19E+02	1,72E+02	1,90E+02	2,25E+01	3,26E+02	2,82E+03	6,49E+00	6,75E+02	3,00E+02	3,09E+02
f_{18^*}	4,29E-04	3,98E-02	3,29E-05	1,30E-07	1,65E-01	8,20E+01	0,00E+00	2,58E+01	1,80E+00	2,46E+01	1,95E+02	1,66E-01	1,07E+02
f_{19^*}	9,50E-14	1,01E-31	0,00E+00	3,78E-01	0,00E+00	0,00E+00	0,00E+00	1,56E-12	0,00E+00	1,01E-10	2,03E+02	0,00E+00	1,70E-03

Figure 2.18 – Erreurs obtenues par les 13 algorithmes sur le *benchmark* des 19 fonctions

On peut voir dans ce tableau que l'algorithme MOS semble obtenir les meilleurs résultats, avec 14 fonctions sur 19 résolues parfaitement. Notre algorithme EM323 obtient particulièrement de bons résultats pour les fonctions f_9 et f_{11}. À partir de ces résultats, nous avons évalué et classé les comportements des 13 algorithmes sur les 19 fonctions (suivant le classement défini à la section 2.5.1). Comme nous n'avions pas accès aux médianes ou aux courbes de convergence pour tous les algorithmes, nous ne pouvions pas distinguer les "types 1.5" des "types 2", nous les avons donc regroupés. Les résultats sont consignés dans la figure 2.19.

	SOUPDE	DE-D+M	GODE	GaDE	jDElscop	SaDE	MOS	MA-SSW	RPSO-vm	IPSOLS	EvoPROpt	EM323	VXQR
Type 1	9	9	7	9	8	10	14	3	7	8	0	7	6
Type 1.5 & 2	7	8	9	7	9	8	3	12	8	9	10	9	6
Type 3	3	2	3	3	2	1	2	4	4	2	9	3	7

Classement	1	2	3	4	4	6	6	8	8	10	11	12	13
	MOS	SaDE	DE-D+M	GaDE	SOUPDE	jDElscop	IPSOLS	EM323	GODE	RPSO-vm	VXQR	MA-SSW	EvoPROpt

Figure 2.19 – Comportement des algorithmes sur les 19 fonctions du *benchmark*. Classement des algorithmes à partir de leur comportement.

On peut voir, grâce à ces calculs, que MOS et SaDE se détachent en obtenant des résultats excellents. Les autres algorithmes sont plus regroupés et obtiennent des résultats de bonne

qualité. En revanche, MA-SSW-Chains et EvoPROpt obtiennent de mauvais résultats pour l'ensemble des fonctions.
Les algorithmes issus de l'évolution différentielle obtiennent tous d'assez bons résultats, les meilleurs étant détenus par SaDE. Ce constat est cohérent avec l'analyse que nous avions faite sur l'algorithme DE simple, qui obtenait déjà des résultats du même ordre que notre algorithme EM323. Ce qui implique que certaines de ces améliorations n'ont que très peu d'effet sur les performances initiales de DE. Il est plus difficile de conclure pour les algorithmes mémétiques, car MOS obtient les meilleurs résultats, mais MA-SSW-Chains est avant-dernier de notre classement. Les deux méthodes basées sur PSO obtiennent également des résultats du même niveau, les meilleurs étant détenus par Tuned IPSOLS. Enfin, parmi les trois méthodes basées sur de nouveaux concepts pour l'optimisation de grande taille, les résultats sont plutôt disparates, la méthode obtenant les meilleurs résultats étant la nôtre : EM323.

2.11 Discussion générale

Les expériences montrent que EM323 produit des résultats de bonne qualité sur une grande partie des fonctions à haute dimension. Ceci est d'autant plus intéressant que la méthode est issue d'une réflexion neuve sur ce genre de problèmes. De plus, l'algorithme ne dispose que de très peu de paramètres, qu'il est possible de fixer, sans avoir de différences notoires sur les résultats en sortie.

L'hybridation de l'algorithme *3-2-3* avec notre méthode globale de recherche issue de l'algorithme EUS a permis d'adopter une stratégie différente. Elle fait varier la granularité de la recherche, mais également utilise une stratégie d'oscillation qui fait varier la séquence de dimensions sur laquelle la recherche est lancée. La méthode EM323 résultante s'avère alors efficace à résoudre de nombreuses fonctions, et même des fonctions non séparables.
Il est toutefois important de noter que, sur certaines fonctions, l'algorithme EUS obtenait un comportement de convergence plus appuyé que le nouvel algorithme EM323. C'est le cas par exemple des fonctions f_2, f_5 et f_8, qui sont légèrement moins bien résolues avec EM323. Nous pouvons donc conjecturer que l'augmentation de la complexité de la structure de notre nouvel algorithme EM323 a directement affecté sa vitesse de convergence. C'est cette convergence ralentie qui l'handicape principalement par rapport à EUS. Cependant, l'algorithme EM323 obtient d'excellentes convergences sur les fonctions f_9 et f_{11}, alors que EUS ne convergeait pas du tout sur ces fonctions. Ainsi, on peut dire que, sur la globalité, les résultats de EM323 sont nettement meilleurs.

Au même moment que notre article présentant EM323 est paru [Gardeux 11a], la conférence CEC'10 organisait une nouvelle fois une session dédiée aux problèmes de grande dimension. Certains des algorithmes présentés dans la section 2.10.4.2 ont alors été testés sur ce nouveau *benchmark* [Tang 09]. Les résultats ont ensuite été comparés et classés suivant une méthode appelée "*Formula 1 point system*". Il s'est avéré que la méthode la plus performante fut la méthode *MA-SSW-Chains*, c'est-à-dire la même méthode qui, dans notre *benchmark*, obtenait des résultats de mauvaise qualité, et se plaçait avant-dernière de notre classement.
Ces résultats, a priori surprenants, rejoignent en fait le théorème du *No Free Lunch* [Wolpert 96] énoncé dans le chapitre précédent. La méthode *MA-SSW-Chains*, réagit assez mal au *benchmark* présenté pour notre algorithme EM323. En revanche, le *benchmark* proposé par la conférence CEC'10 semble contenir des fonctions non pas plus "simples", mais dont la structure correspond mieux au mode de résolution de *MA-SSW-Chains*. Il serait donc intéressant de comparer les résultats avec ceux de notre méthode EM323, qui surpasse nettement *MA-SSW-Chains* sur le *benchmark* que nous avons testé.

2.12 Conclusion

Dans ce chapitre, nous avons tout d'abord présenté une plateforme modulaire, nommée Heuris-Test. Cette plateforme, destinée au *benchmarking*, est utilisée tout au long de cette thèse, elle nous a permis d'intégrer facilement de nouveaux algorithmes, ainsi que de nouvelles fonctions objectifs, et de produire des fichiers résultats exploitables.

Nous avons ensuite présenté deux nouveaux algorithmes : CUS et EUS, issus de nos réflexions sur les caractéristiques rendant un algorithme robuste à l'augmentation de la dimension. Ces deux algorithmes ont une structure volontairement simple car ils ont été mis au point dans le but de servir de brique de base pour de futures améliorations. Ils ont été testés sur différentes fonctions objectifs, afin d'évaluer leurs performances et de comparer leurs caractéristiques. EUS obtenant des résultats de meilleure qualité que CUS, il a été comparé ensuite à d'autres algorithmes, lors d'une session spéciale de la conférence ISDA'09 destinée aux problèmes de grande dimension. Les résultats encourageants nous ont poussés à continuer dans cette voie et à améliorer la structure de l'algorithme EUS, afin de résoudre des fonctions de haute dimension plus complexes.

Différentes caractéristiques ont été ajoutées à l'algorithme EUS, afin de le rendre plus robuste face à des fonctions objectifs non séparables. Il a notamment été hybridé avec la méthode *3-2-3* développée par Fred Glover. L'algorithme résultant, EM323, dispose entre autres d'une stratégie d'oscillation pour le parcours de ses dimensions. Il a été comparé à l'algorithme EUS, et l'on a montré que, malgré une convergence moins rapide, il réussissait à résoudre des fonctions objectifs d'une plus grande complexité, notamment certaines fonctions complexes non séparables. EM323 a ensuite été comparé à trois algorithmes évolutionnaires connus pour leurs performances. Cette étude a été menée sur un *benchmark* de 19 fonctions, dont certaines étaient volontairement non séparables. Une comparaison statistique a montré que EM323 obtenait des résultats équivalents ou meilleurs sur l'ensemble des fonctions. Une brève comparaison avec 12 autres algorithmes destinés aux problèmes de grande dimension a montré qu'EM323 obtenait des résultats de bonne qualité, notamment pour deux fonctions du *benchmark*.

Dans ce chapitre, nous avons également présenté un système de classement du comportement des algorithmes sur les fonctions-objectifs ("types 1, 1.5, 2 et 3"). L'intérêt de ce système est de pouvoir classer rapidement les algorithmes les plus efficaces sur un *benchmark*. En effet, certaines procédures statistiques, telles que le test de Wilcoxon, même si elles sont plus rigoureuses, ne sont pas toujours adaptées pour comparer ces types de problèmes. Cette classification permet en plus d'isoler les comportements spécifiques de certains algorithmes. En effet, si tous les algorithmes ont un comportement de "type 2" sur une fonction, mais qu'un algorithme a un comportement meilleur ("type 1" ou "type 1.5"), alors il est intéressant d'isoler la procédure particulière permettant de "vaincre" la difficulté posée par la fonction.

Les pistes d'amélioration d'EM323 impliquent de tester d'autres méthodes de *line search* ou d'explorer des directions de recherche qui diffèrent des directions données par les dimensions. Une idée serait, par exemple, d'utiliser des algorithmes évolutionnaires, apparemment bien adaptés à ce genre de problèmes, pour combiner les solutions obtenues, ou pour générer des vecteurs de directions.

Chapitre Trois

Classification des données de puces à ADN

3.1 Introduction

Les applications de l'optimisation de grande dimension sont très nombreuses en *data mining* et en *web mining*. Dans cette partie, nous nous concentrerons sur le domaine du *data mining*, et plus particulièrement la création de modèles de prédiction à partir de données issues de puces à ADN. Cette technologie permet de récupérer le niveau d'expression des gènes d'un échantillon biologique. Une fois ces données récupérées, elles peuvent être analysées afin d'en tirer des informations sur l'échantillon testé.
Le but de notre étude, est de créer des modèles permettant de prédire le comportement de nouveaux exemples. Pour cela, nous utilisons les données existantes pour créer un modèle, que nous appliquons ensuite sur de nouveaux échantillons. Ce domaine d'application est appelé apprentissage automatique, et fait partie des techniques d'intelligence artificielle.
Les applications en oncologie et plus particulièrement en onco-pharmacogénomique permettent, à partir d'un échantillon prélevé sur un patient, de prédire par exemple le traitement chimiothérapique le plus efficace à lui administrer.

Cependant, la quantité de données récupérées par la technologie des puces à ADN est très conséquente. L'analyse directe de ces données pourra donc s'avérer très lente ou trop complexe, voire impossible par les méthodes traditionnelles. C'est dans ce cadre que nous utiliserons les méthodes d'optimisation à haute dimension élaborées dans la première partie de cette thèse. Leur but sera de filtrer l'ensemble des données afin de sélectionner un sous-ensemble des variables les plus pertinentes pour notre problème.

Ce chapitre est consacré au traitement des données issues de techniques de bioinformatique et plus particulièrement de techniques de fouilles de données. Nous présenterons dans un premier temps les méthodes d'acquisition de ces données, puis nous verrons un état de l'art des techniques de fouille de données. Enfin, nous conclurons sur l'intérêt des méthodes d'optimisation dans le cadre de ce problème.

3.2 Présentation du problème

Dans un premier temps, nous avons étudié des données fournies par l'hôpital Tenon à Paris. Elles sont composées de données sur des patientes atteintes du cancer du sein. En effet, lorsqu'un patient est atteint d'une tumeur cancéreuse, le médecin doit choisir le traitement à lui administrer. Cette prescription ne va pas forcément donner de bons résultats, or le médecin n'a a priori aucun moyen de savoir si ce traitement va être efficace ou non. C'est seulement après plusieurs mois que l'on peut juger de l'efficacité de ce dernier. Mais dans cette pathologie,

chaque jour compte, d'autant que dans la recherche contre le cancer du sein par exemple, seules 30% des patientes réagissent positivement aux traitements. Dans l'étude qui va suivre, nous verrons que la prédisposition d'un patient à pouvoir être traité par une chimiothérapie est une information qui pourra être récupérée (au moins en partie) dans son **génome**, c'est-à-dire dans son code génétique.

Le génome d'un être vivant est contenu dans chacune de ses cellules. Le support de cette information est l'**ADN** (Acide DésoxyriboNucléique), qui est lui-même composé de 4 **bases azotées** (ou nucléotides) : Adénine, Cytosine, Guanine, Thymine. L'ordre dans lequel sont placés ces nucléotides permet de définir un code à 4 lettres. L'ADN est regroupé en différents **chromosomes**, eux-même constitués de plusieurs **gènes**. Un gène est donc une petite portion du code génétique, placé sur un chromosome. Chaque gène code pour une ou plusieurs protéines qui pourront être produites par la cellule (cf figure 3.1).

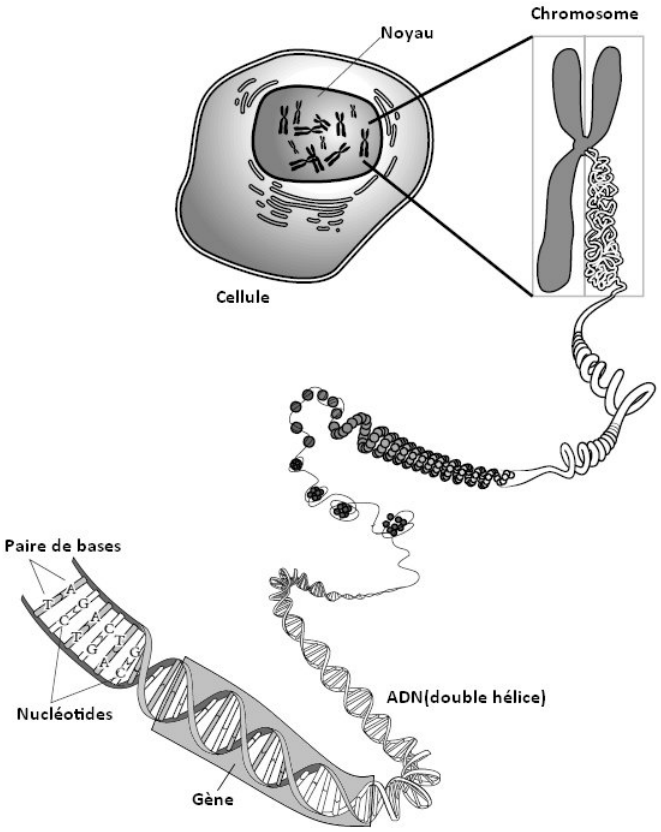


Figure 3.1 – Cellule, Chromosome et ADN. Source : *National Human Genome Research Institute*.

Ainsi, si l'on savait décoder le génome de la patiente, on pourrait prévoir avant même de le lui administrer, si le traitement chimiothérapique sera efficace ou non. De nombreuses équipes ont travaillé sur le séquençage du génome humain [Venter 01] et c'est le *Human Genome Project* (HGP) [Bentley 00] qui a terminé cette tâche en avril 2003. Il est actuellement à disposition, via une base de données publique sur Internet. Cependant, le génome humain, même s'il est actuellement connu, ne nous permet pas de faire des prédictions sur l'efficacité d'un traitement chimiothérapique, car nous ne connaissons pas la fonction d'une grande partie de ces gènes. De plus, même si certains caractères simples sont déterminés par un seul gène (comme le groupe sanguin), dans la plupart des cas, un caractère observable dépend d'un ensemble de gènes, et

éventuellement de l'interaction avec l'environnement (comme le poids du patient). De plus, même si toutes les cellules d'un organisme partagent le même génome, certains gènes ne sont exprimés que dans certaines cellules, à certaines périodes de la vie de l'organisme, ou sous certaines conditions. Un être humain dispose de plus de 20000 gènes, et il est difficile de savoir ceux qui sont les plus pertinents pour notre problème. Il faudrait donc, pour pouvoir prédire la réponse d'un patient donné à un traitement chimiothérapique, obtenir l'expression génique d'une grande partie de son génome.

3.3 Acquisition des données : les puces à ADN

Les **puces à ADN** ou biopuces (également appelées *DNA microarrays* ou *GeneChip*) [Heller 02] relèvent d'une biotechnologie qui date des années 1990 et dont les premières utilisations en recherche remontent à 1995 [Schena 95]. Cette technologie est donc assez récente et permet d'analyser le niveau d'expression des gènes dans une cellule, un tissu, un organe, un organisme, ou encore un mélange complexe à un moment donné. Une puce à ADN peut évaluer simultanément l'expression de milliers de gènes, voire d'un génome entier.

Une puce à ADN est une surface (verre, silicium ou plastique) sur laquelle sont fixées des molécules d'ADN, classées et compartimentées sur une grille. Elle peut donc être assimilée à une matrice de n lignes et p colonnes, chaque "case" (appelée sonde ou *probe*) étant un compartiment qui contiendra la séquence d'un gène donné.
Pour obtenir cette séquence, les deux brins d'ADN du gène que l'on veut tester sont séparés (à haute température). En effet, une molécule d'ADN est constituée de deux brins complémentaires (brin codant et brin transcrit), chaque base azotée ayant elle-même une base complémentaire (l'adénine est le complémentaire de la thymine et la cytosine de la guanine). Chaque gène étant une portion de l'ADN, il est également constitué de deux brins complémentaires.

Lors de la création de la puce à ADN, on dépose dans chaque case une multitude de brins codants (des milliers) du gène correspondant à cet emplacement. Ce brin est fixé par de la polylysine, qui maintient l'ADN dans son compartiment par des interactions électrostatiques. On obtient alors une grille contenant les brins codants de $n \times p$ gènes différents, rangés sur la grille. La phase de préparation étant terminée, notre puce à ADN est prête à recevoir un échantillon biologique à tester.

La phase suivante repose sur le principe d'hybridation moléculaire [Stekel 03]. En effet, dans des conditions très strictes (température de 60°C, entre 10 et 17 heures d'incubation, etc.), un fragment d'ADN simple brin peut reconnaître son brin complémentaire (ADNc) parmi des millions d'autres, pour reformer de l'ADN double brin (cf figure 3.2). C'est ce processus qui est à la base de la technique des puces à ADN.

Pour tester notre échantillon biologique, nous commençons donc par en extraire l'ARN messager (ARNm). Les ARNm sont ensuite transformés en ADN complémentaires (ADNc) par transcription inverse. Ces fragments d'ADNc sont alors marqués par fluorescence ou par radioactivité et déposés sur la puce à ADN. La puce est alors incubée une nuit dans les conditions d'hybridation moléculaire. Lorsque l'expérience est terminée, la grille est nettoyée afin de supprimer les fragments marqués qui ne se seront pas hybridés avec ceux de la puce.

Nous calculons ensuite la quantité de gènes qui se sont hybridés avec leur séquence complémentaire dans chaque compartiment, en mesurant leur intensité lumineuse (ou radioactive). En

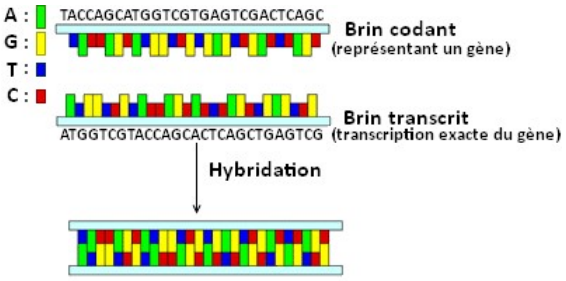


Figure 3.2 – Représentation schématique de 2 brins d'ADN. Hybridation moléculaire.

effet, si une case n'est pas fluorescente par exemple, c'est qu'aucun fragment marqué de l'échantillon biologique testé ne s'est attaché à son brin complémentaire. C'est donc que l'échantillon ne contenait pas ce gène particulier. Cependant, les résultats ne sont pas binaires, les gènes de l'échantillon peuvent s'exprimer de manière plus ou moins forte. Nous y associerons donc une valeur numérique qui sera plus ou moins grande en fonction de la quantité de gènes hybridés dans la case, c'est-à-dire de la quantité présente au départ dans l'échantillon testé (cf figure 3.3). L'ensemble des données sera ensuite normalisé pour éliminer les différences liées aux variations de la quantité de départ, de bruit de fond, ou bien aux différents biais entrant en jeu dans la réaction d'hybridation.

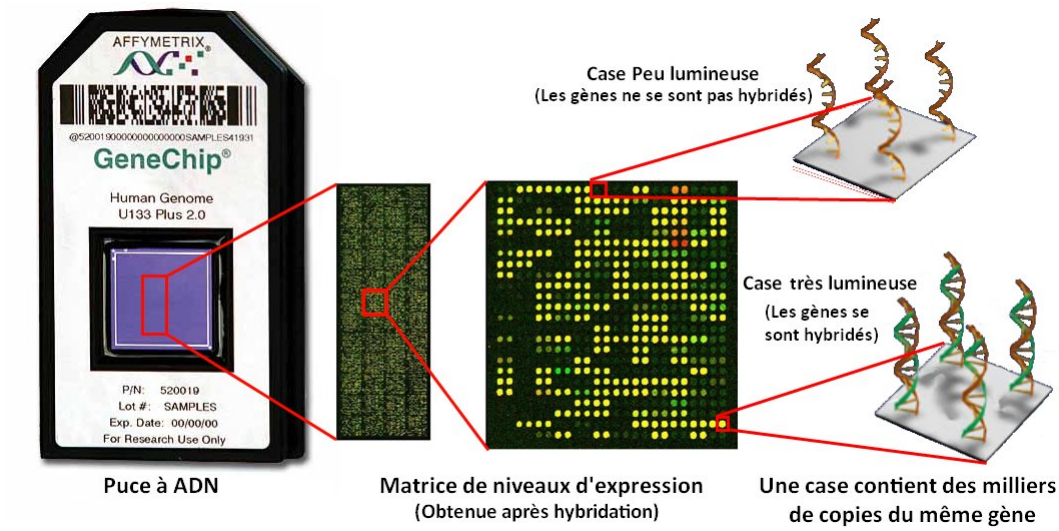


Figure 3.3 – Détail d'une puce à ADN. Source : Institut Pasteur, Affymetrix, Bioteach.

Cette technologie connaît aujourd'hui un essor considérable (notamment dans le domaine de la cancérologie) puisque les techniques d'analyse de données permettent d'en déduire la fonction d'un gène, voire de créer des modèles de prédiction et des outils de diagnostic [Baldi 01] [Golub 99] [Dudoit 02]. Par exemple, on a pu découvrir que le gène appelé BRCA1 permettait d'identifier les risques de certains cancers du sein chez la femme. Une femme pourrait donc théoriquement faire un test pour savoir si elle possède ce gène ou non, et donc connaître une éventuelle prédisposition au cancer du sein.

Les données fournies par les puces à ADN sont normalisées et présentées sous la forme d'une liste de valeurs d'expression (cf tableau 3.1).

	Patient
Gène 1	Niveau d'expression du gène 1
Gène 2	Niveau d'expression du gène 2
...	...
Gène n	Niveau d'expression du gène n

Tableau 3.1 – Représentation des données fournies par la puce à ADN.

Lorsque l'on travaille avec des puces à ADN, le constructeur de la puce n'associe pas directement le nom d'un gène à chaque ligne du tableau. Dans les données fournies, les différents gènes sont repérés par un identifiant. Il faut donc utiliser le glossaire d'association du constructeur, qui liste tous les identifiants des sondes de la puce, afin d'obtenir le nom du gène qui y est associé.

3.4 Fouille de données

Une fois ces données acquises, il faut les analyser et en extraire l'information [Berrar 03]. En informatique, lorsque les données sont conséquentes, le traitement des données peut se faire par différentes méthodes, dont la **fouille de données** (ou *Data Mining*) [Fayyad 96] [Hand 01] [Tan 05] [Witten 05]. La fouille de données est une discipline à l'intersection des statistiques, de l'informatique et de l'intelligence artificielle [Glymour 97]. Les méthodes qui en sont issues peuvent extraire, parmi un ensemble de données souvent très conséquent, de nouvelles connaissances. Le but étant de reconnaître, parmi l'ensemble des données, des régularités ou bien des modèles permettant de décrire ces données. Dans un second temps, ces modèles permettront par exemple de classifier ces données ou bien de faire des prédictions sur de nouvelles données.

La fouille de données est utilisée en informatique décisionnelle [Turban 10], marketing [Berry 97], détection d'anomalies [Lee 98], etc. comme un outil aidant à la prise de décision. Elle regroupe différentes techniques qui sont représentées sur le schéma de la figure 3.4.

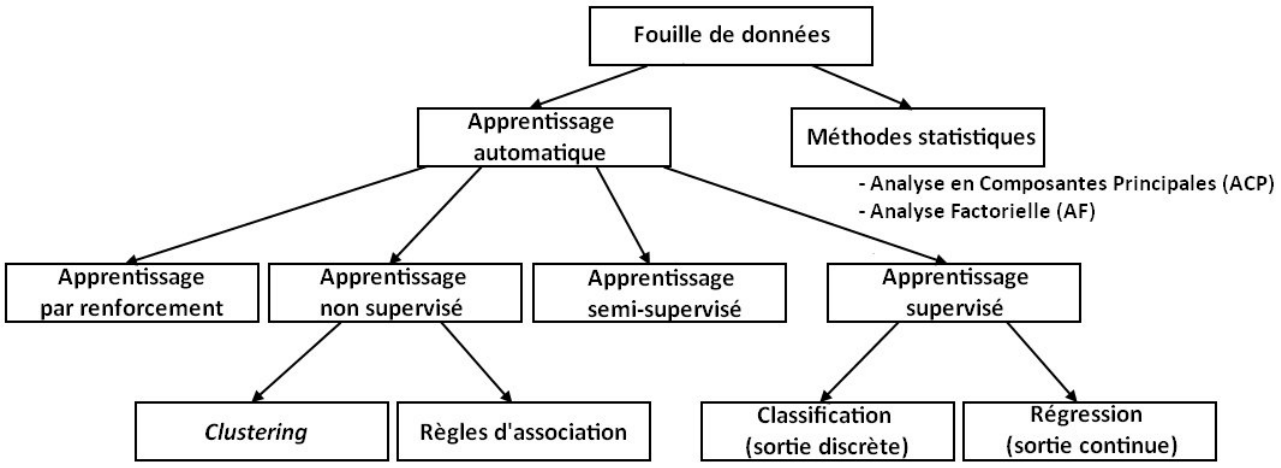


Figure 3.4 – Schéma des différentes techniques issues de la fouille de données.

Les techniques de fouille de données nécessitent un certain nombre d'étapes afin de produire des modèles performants. Elles peuvent se résumer en trois grandes phases :

1. **Exploration :** On commence par préparer les données, afin de les alléger le plus possible, tout en conservant leur information. Cette étape est souvent assez négligée mais elle est néanmoins très importante dans le processus de fouille de données. Elle est souvent utilisée sur des données récupérées de manière automatique (par le Web dans le cadre de *Web mining* par exemple). Il arrive souvent que les données récupérées ne soient pas contrôlées et que les données contiennent des valeurs erronées (par exemple un âge négatif), ou bien des combinaisons impossibles (par exemple, Sexe : Homme, Enceinte : Oui). Le fait d'analyser ces données sans avoir écarté ces problèmes peut mener à des modèles non pertinents. On peut dégager plusieurs parties différentes dans ce processus :
   - Nettoyage : On retire les données redondantes (doublons) ou erronées. On peut également compléter les données manquantes par des méthodes statistiques.
   - Transformation : On normalise les données, afin de les mettre sur une même échelle pour faciliter leur comparaison.
   - Partitionnement : On crée plusieurs sous-ensembles à partir des données fournies, typiquement un ensemble d'apprentissage et un ensemble de test (et éventuellement un ensemble de validation).
   - Sélection d'attributs (*Feature Selection* en anglais) : On l'utilise dans le cas de problèmes où l'on a beaucoup de variables. Elle permet de réduire le nombre de variables à celles qui sont les plus pertinentes.
2. **Création de modèle :** On teste différents modèles sur les données et on compare leur performance (qui explique le mieux le problème et qui a les résultats les plus stables possibles parmi tous les échantillons d'apprentissage). Le but étant de trouver celui qui est le plus adapté au problème.
3. **Déploiement :** On applique le meilleur modèle sélectionné à l'étape précédente à de nouvelles données, afin d'obtenir les meilleurs résultats possibles.

Les deux composantes principales de la fouille de données, sont des techniques issues de la statistique multivariée, ou bien des techniques d'apprentissage automatique (ou *Machine Learning*). Les méthodes statistiques telles que l'Analyse en Composantes Principales (ACP) ou l'Analyse Factorielle (AF), permettent de trouver les structures qui lient les données entres elles. Nous nous intéresserons plus particulièrement aux techniques d'apprentissage automatique.

3.4.1 Apprentissage automatique

Les systèmes dit à **apprentissage automatique** [Vapnik 95] [Mitchell 97] [Bishop 06], sont souvent utilisés dans le contexte du *data mining*. Ces méthodes visent à construire des modèles génériques à partir de données fournies, alors vues comme des **exemples** illustrant les relations entre des variables observées. Le but est alors d'utiliser ces exemples pour en déduire des caractéristiques liant les données entres elles. La difficulté repose sur le fait que les données de l'ensemble d'apprentissage ne contient qu'un nombre fini d'exemples. Nous ne disposons donc pas de l'ensemble de tous les comportements possibles, en fonction de toutes les entrées possibles. L'apprenant doit alors généraliser à partir de ces exemples, afin de pouvoir s'adapter si de nouveaux cas se présentent. Le terme "automatique" indique que l'homme ne crée pas de règles d'apprentissage, c'est la machine qui les déduit des données. Dans certains cas, l'algorithme peut continuer son apprentissage, en adaptant son modèle si de nouvelles données

n'entrent pas dans les règles pré-établies, on parle alors d'apprentissage incrémental (à opposer à l'apprentissage non incrémental).

Parmi l'ensemble des données, il faut distinguer les cas d'études (ou candidats) des variables (ou attributs). Dans notre cadre d'expérimentation, les variables seront les gènes, c'est-à-dire l'ensemble des caractéristiques pouvant différer en fonction des cas d'études. Les candidats pourront alors être représentés comme un vecteur de ces variables, représentant un exemple possible. Dans le cas des données fournies par l'hôpital Tenon, ce seront les patientes. Nous appellerons **ensemble d'apprentissage** (ou "population d'apprentissage"), les candidats (alors appelés exemples) utilisés pour générer le modèle d'apprentissage. De même, nous appellerons **ensemble de test** les candidats sur lesquels nous désirons appliquer notre modèle d'apprentissage. Un ensemble de validation pourra être utilisé lors de l'apprentissage (comme sous population de l'ensemble d'apprentissage), afin de valider le modèle et d'éviter le sur-apprentissage (que nous définirons dans la suite).

En fonction du type de problème que l'on se pose, on peut avoir à mettre en place différents types d'apprentissage :
- **Apprentissage supervisé** : cette technique a pour but la création d'un modèle reliant des données d'apprentissage à un ensemble de valeurs de sortie (un comportement). La particularité étant que l'on connaît le comportement des exemples d'apprentissage.
- **Apprentissage par renforcement** [Kaelbling 96] : les données en entrée sont les mêmes que pour l'apprentissage supervisé, cependant l'apprentissage est guidé par l'environnement sous la forme de récompenses ou de pénalités données en fonction de l'erreur commise lors de l'apprentissage.
- **Apprentissage non supervisé** [Barlow 89] : vise à créer un modèle structurant l'information. La différence ici est que les comportements (ou catégories) des données d'apprentissage ne sont pas connus, c'est ce que l'on cherche à trouver. On parle alors de "*clustering*" dont le but est de classifier les données d'entrée.
- **Apprentissage semi-supervisé** [Chapelle 06] : les données d'entrée sont constituées d'exemples étiquetés et non étiquetés. Ce qui peut être très utile quand on a ces deux types de données, car cela permet de ne pas en laisser de côté et d'utiliser toute l'information.

Dans notre cas, nous nous concentrerons sur l'apprentissage supervisé. En effet, les données fournies par les puces à ADN contiennent les niveaux d'expression de tous les gènes. Nous disposons également d'un fichier descriptif pour chaque patient, qui nous procure les données cliniques permettant de savoir à quelle classe ils appartiennent.

3.4.1.1 Apprentissage supervisé

Supposons qu'un jeu de données d'apprentissage contienne N variables le décrivant (dans notre cas, ce seront des gènes). Nous disposons alors, pour un exemple donné, de deux types d'informations : un vecteur de valeurs $X = (x_1,...,x_N)$ prises par chaque variable, et une valeur de sortie Y appelée signal supervisé (qui peut être une classe si l'on prend un problème de classification). Si nous formalisons le problème d'apprentissage décrit précédemment, nous pouvons le représenter comme un ensemble de couples entrée-sortie (X_i,Y_i), avec $i \in [1,n]$, n étant le nombre d'exemples disponibles. On appelle alors fonction d'apprentissage la fonction notée $g : X \rightarrow Y$ qui associe un signal supervisé à chaque vecteur d'entrée. Le but d'un algorithme d'apprentissage supervisé sera donc d'approcher cette fonction, uniquement à partir des exemples d'apprentissage. On pourra ensuite évaluer les performances de notre fonction d'apprentissage en se basant sur l'erreur commise par le signal de sortie, par rapport aux valeurs réelles.

En fonction du signal supervisé que l'on veut obtenir, on peut distinguer deux types de problèmes :
- Régression : lorsque le signal supervisé que l'on cherche à estimer est une valeur dans un ensemble continu de réels.
- Classification : lorsque l'ensemble des valeurs de sortie est discret. Ceci revient à attribuer une classe (aussi appelée étiquette ou *label*) pour chaque vecteur d'entrée.

Nous nous intéresserons plus particulièrement aux problèmes de classification. En effet, les données cliniques fournies dans notre cas nous permettent de savoir si la patiente est répondeuse ou non répondeuse au traitement. Nous sommes donc bien ici dans le cas d'un problème de classification avec deux valeurs de classe : **Répondeuse** ou **Non Répondeuse** au traitement.

3.5 Classification supervisée

3.5.1 Application

Les techniques d'apprentissage supervisé permettent de construire des modèles à partir d'exemples d'apprentissage. Ces modèles peuvent ensuite être utilisés dans différentes applications, telles que la prédiction ou la reconnaissance.

Les techniques dites de ***predictive modelling*** (ou *predictive data mining*) [Weiss 98] analysent un ensemble de données et en extraient un ensemble de règles, formant un modèle prédictif, afin d'essayer de prédire, avec la plus grande précision, le comportement de nouvelles données. Les techniques dans le domaine de la prédiction sont les plus classiques, car on peut directement les utiliser dans des applications concrètes. C'est d'ailleurs sur ce domaine d'application que nous nous concentrerons dans la suite de cette thèse.

3.5.2 Problèmes linéaires et non linéaires

Les méthodes de classification supervisée peuvent être basées sur des hypothèses probabilistes (classifieur naïf bayésien), sur des notions de proximité (k plus proches voisins) ou sur des recherches dans des espaces d'hypothèses (arbres de décisions, etc.).

En fonction du problème, il faut pouvoir choisir le classifieur approprié, c'est-à-dire celui qui sera à même de séparer au mieux les données d'apprentissage. On dit qu'un problème est linéairement séparable si les exemples de classes différentes sont complètement séparables par un hyperplan (appelé hyperplan séparateur, ou séparatrice). Ce genre de problème se résout par des classifieurs assez simples, qui ont pour but de trouver l'équation de l'hyperplan séparateur. Mais le problème peut également être non séparable comme illustré dans la figure 3.5. Dans ce cas il

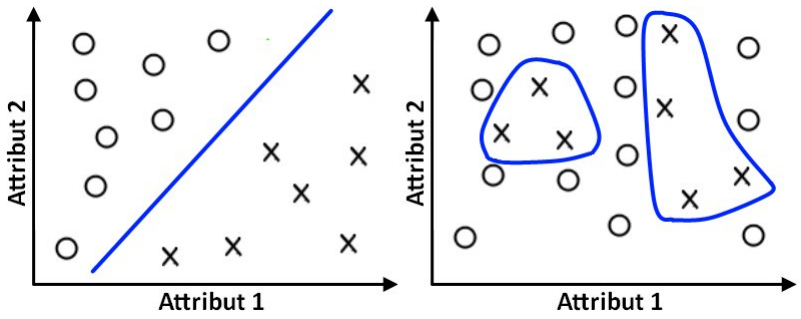


Figure 3.5 – Gauche : Problème linéairement séparable (Frontière linéaire). Droite : Problème non linéairement séparable

faut utiliser d'autres types de classifieurs, souvent plus longs à paramétrer, mais qui obtiennent des résultats plus précis. Cependant, un classifieur construisant un hyperplan séparateur peut tout de même obtenir de bons résultats, en ignorant certaines données singulières.

Il est également intéressant de noter qu'un problème initialement non linéairement séparable peut s'avérer séparable avec l'ajout d'un nouvel attribut (cf figure 3.6). D'où l'intérêt d'un choix judicieux de ces attributs. C'est ce principe qui est utilisé par le classifieur *Support Vector Machine* (SVM) que nous verrons dans la suite (cf section 3.5.5.3).

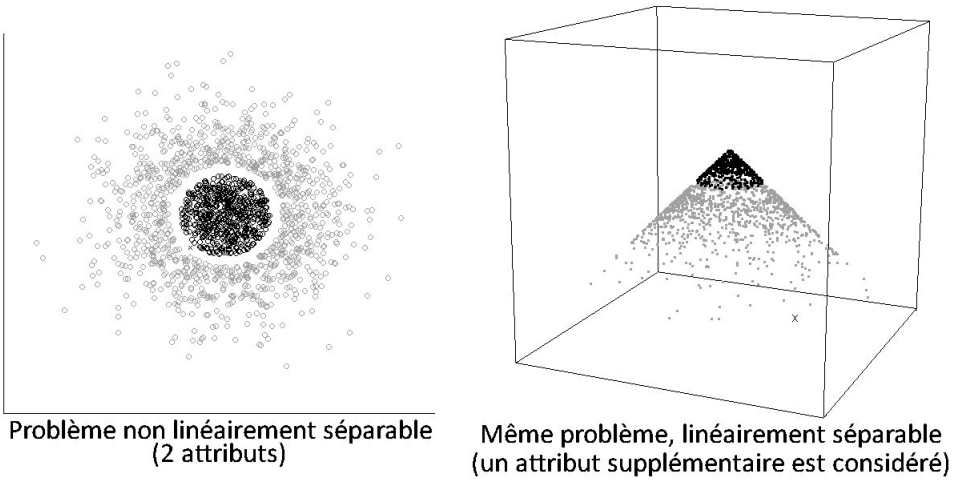


Figure 3.6 – Même problème considéré avec 2 ou 3 attributs.

Lorsque l'on a affaire à un problème linéairement séparable, la sortie peut être calculée comme la combinaison linéaire des attributs d'entrée. Les attributs sont donc pondérés par un vecteur de poids et la difficulté du problème revient alors à trouver les poids qui font correspondre la sortie du modèle à la sortie réelle. C'est le principe utilisé en régression linéaire pour des sorties continues. Mais il peut également être utilisé dans le cadre d'une classification supervisée.
La figure 3.7 montre un problème de classification de données concernant deux types d'iris.

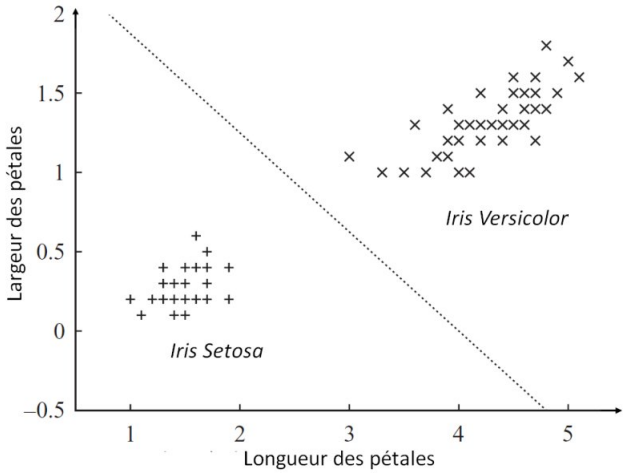


Figure 3.7 – Classification des Iris Setosa et Iris Versicolor par une frontière de décision.
Source : Fichiers de données IRIS de Fisher.

Ces données concernent deux attributs : la longueur et la largeur des pétales de la fleur. C'est un problème d'apprentissage supervisé à deux classes. Dans ce cas, l'hyperplan séparateur est assimilé à une droite appelée frontière de décision. Elle sépare les exemples des deux classes, ce qui définit l'endroit où la décision change d'une classe à l'autre. Dans notre exemple, l'équation de la frontière de décision représentée est la suivante :

$$2.0 - 0.5 * PETAL - LENGTH - 0.8 * PETAL - WIDTH = 0 \tag{3.5.1}$$

Pour faire une prédiction sur un exemple de test, il suffit de calculer le résultat de l'équation précédente, en prenant en paramètre la valeur des attributs correspondants. Si la valeur obtenue est positive, alors on peut prédire qu'il s'agit d'un iris Setosa, s'il est négatif, on prédit qu'il s'agit d'un iris Versicolor.

L'équation de la droite est déterminée par le classifieur utilisé. Il est à noter que pour la figure 3.7, on pourrait trouver plusieurs équations différentes permettant de séparer complètement les deux classes (il en existe une infinité).

Dans la suite, nous nous attacherons à présenter les méthodes de classification les plus connues, ainsi que celles que nous utiliserons dans cette thèse.

3.5.3 Classifieurs à mémoire

L'intérêt de ces classifieurs est qu'ils ne nécessitent aucune phase d'apprentissage. On peut directement déduire la classe d'un nouvel exemple à partir de l'ensemble d'apprentissage.

3.5.3.1 k plus proches voisins

Le classifieur des k plus proches voisins (*k-Nearest Neighbor* ou k-NN en anglais), est l'un des algorithmes de classification les plus simples. Un exemple est classifié par vote majoritaire de ses k "voisins" (par mesure de distance), c'est-à-dire qu'il est prédit de classe C si la classe la plus représentée parmi ses k voisins est la classe C. Un cas particulier est le cas où $k = 1$, l'exemple est alors affecté à la classe de son plus proche voisin.

L'opérateur de distance le plus souvent utilisé est la distance euclidienne, cependant, en fonction du problème, on peut également utiliser les distances de Hamming, de Mahalanobis, etc. Comme tous les classifieurs *memory-based*, cette méthode ne nécessite aucun apprentissage. Un nouvel exemple se présentant, il suffit de calculer sa distance à tous les exemples présents dans l'ensemble d'apprentissage et de garder les k premiers pour prendre une décision.

Sur la figure 3.8, on peut voir que le choix du k est très important. En effet, si k = 1,2,3 l'exemple à prédire (noté " ?") serait classifié comme étant de la classe "×", mais si $k = 5$, il serait classifié comme étant de la classe "O". On évite d'ailleurs de choisir des valeurs de k paires, pour éviter les cas d'égalité.

3.5.3.2 Classifieur naïf bayésien

La classification naïve bayésienne repose sur l'hypothèse que les attributs sont fortement (ou naïvement) indépendants. Elle est basée sur le théorème de Bayes qui ne s'applique que sous cette hypothèse.

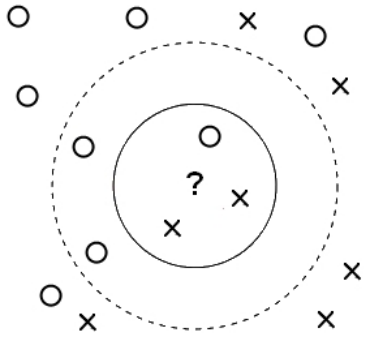


Figure 3.8 – Schéma d'une classification par la méthode k-NN

Theorem 3.5.1. (Théorème de Bayes) $P(x|y) = \frac{P(y|x)P(x)}{P(y)}$

Avec $P(x|y)$ la probabilité conditionnelle d'un événement x sachant qu'un autre événement y de probabilité non nulle s'est réalisé.

Dans le cas d'une classification, on pose H l'hypothèse selon laquelle un "vecteur d'attributs X (représentant un patient) appartient à une classe C", et l'on suppose que l'on cherche à estimer la probabilité $P(H|X)$, c'est-à-dire la probabilité que l'hypothèse H soit vraie, considérant X. Ainsi, si l'on a un nouvel exemple $x = (x_1,...,x_n)$ dont on veut trouver la classe, on va chercher la probabilité maximale d'appartenance à cette classe.

$$P(x)^* = arg\ max\ P(x_1,...,x_n|H) * P(H) \tag{3.5.2}$$

Cette équation est directement déduite du théorème de Bayes. En effet, comme l'objectif est de faire une maximisation et que le dénominateur ne dépend pas de x, on peut le supprimer.

Dans l'équation (3.5.2), $P(H)$ peut être estimé à partir des données d'apprentissage (nombre d'individus appartenant à la classe C / nombre total d'individus), il ne reste donc qu'à estimer $P(x_1,...,x_n|H)$. Cependant, cette probabilité pourrait être beaucoup trop compliquée à estimer, si l'on considère le nombre de descriptions possibles. C'est alors que l'on utilise l'hypothèse d'indépendance, qui nous permet de décomposer la probabilité conditionnelle en un produit de probabilités conditionnelles. Le classifieur devient alors :

$$Classe(x) = arg\ max \prod_{i=1}^{n} P(x_i|H) * P(H) \tag{3.5.3}$$

Ce classifieur est souvent utilisé, car très simple d'emploi. Cependant, à cause de l'hypothèse naïve d'indépendance des attributs, il est très sensible à leur corrélation.

3.5.4 Classifieurs basés sur des modèles

3.5.4.1 Arbres de décision

Les arbres de décision sont un outil très populaire de classification. Leur principe repose sur la construction d'un arbre de taille limitée. La racine constitue le point de départ de l'arbre et

représente l'ensemble des données d'apprentissage. Puis ces données sont segmentées en plusieurs sous-groupes, en fonction d'une variable discriminante (un des attributs). Par exemple, dans la figure 3.9, la première variable discriminante est la température corporelle. Elle divise la population en deux sous-groupes : les personnes dont la température est supérieure à 37°C et les autres. Le processus est ensuite réitéré au deuxième niveau de l'arbre, ou les sous-populations sont segmentées à leur tour en fonction d'une autre valeur discriminante (dans l'exemple, c'est la toux). Le processus est ainsi réitéré jusqu'à ce que l'on arrive a une feuille de l'arbre, correspondant à la classe des exemples qui ont suivi cette décomposition.

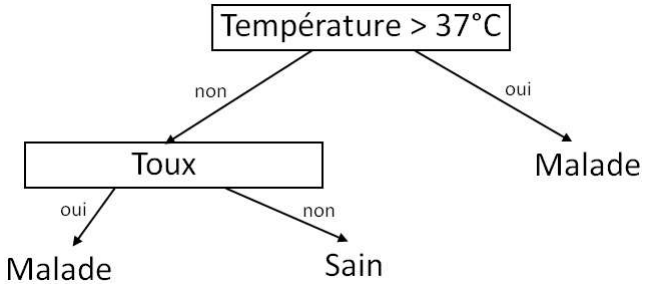


Figure 3.9 – Schéma d'un arbre de décision.

Une fois l'arbre construit à partir des données d'apprentissage, on peut prédire un nouveau cas en le faisant descendre le long de l'arbre, jusqu'à une feuille. Comme la feuille correspond à une classe, l'exemple sera prédit comme faisant partie de cette classe. Dans l'exemple, on peut en déduire qu'une personne qui a une température < 37°C et qui a de la toux est prédite comme malade, tandis qu'une personne qui a une température < 37°C mais pas de toux est considérée comme saine.

Lors de la création de l'arbre, la première question qui vient à l'esprit est le choix de la variable de segmentation sur un sommet. Pourquoi par exemple avons-nous choisi la variable "température" à la racine de l'arbre ? Il nous faut donc une mesure afin d'évaluer la qualité d'une segmentation et ainsi de sélectionner la meilleure variable sur chaque sommet. Ces algorithmes s'appuient principalement sur les techniques issues de la théorie de l'information, et notamment la théorie de Shannon [Shannon 48]. Un des paramètres importants est la taille limitée que l'on veut fixer à l'arbre. En effet, si la taille est trop grande, on créera une feuille pour chaque exemple et on aura alors un sur-apprentissage des données (cette notion sera définie plus tard). Si la taille est trop petite, en revanche, on aura un sous-apprentissage des données.

Il existe différentes procédures permettant la création d'arbres de décisions, nous pouvons citer par exemple les méthodes CART ou ID3. La méthode ID3 a été créée par Quinlan [Quinlan 86]. Il a depuis proposé un grand nombre d'améliorations utilisant des heuristiques pour améliorer le comportement de son système. Son approche a pris un tournant important lorsqu'il présenta la méthode C4.5 [Quinlan 93] qui est une référence incontournable dans le domaine des arbres de décision. Une autre évolution de cet algorithme, nommée C5.0, permet d'améliorer la méthode C4.5 principalement en générant des arbres de taille réduite.

L'intérêt des arbres de décision est en premier lieu leur lisibilité. En effet, il est très simple de comprendre les décisions de l'arbre une fois celui-ci créé, ce qui n'est pas toujours le cas pour les autres classifieurs que nous verrons. D'autre part, l'algorithme de création des arbres de décision fait automatiquement la sélection d'attributs jugés pertinents, et ce, même sur des volumes de données importants.

3.5.5 Classifieurs construisant des hyperplans séparateurs

3.5.5.1 Analyse discriminante linéaire

L'analyse discriminante linéaire (*Linear Discriminant Analysis* ou LDA) [Fukunaga 90] a pour objectif de trouver les vecteurs discriminants optimaux (transformation) en maximisant le ratio entre la distance interclasse et la distance intraclasse, afin de discriminer au maximum les différentes classes. Cette technique permet à la fois de faire de la classification linéaire, mais également de réduire le nombre d'attributs.

On représente les données d'entrée par une matrice $A = (a_{ij}) \in \mathbb{R}^{n \times N}$, où chaque colonne (notée a_k avec $1 \leq k \leq N$) correspond à un exemple (pour n attributs et N exemples) et chaque ligne correspond à un attribut. Le but est alors de projeter ces données sur un espace vectoriel $\mathcal{G}$ de plus petite dimension ($l < n$) par une transformation linéaire $G \in \mathbb{R}^{n \times l}$. Les données projetées sont alors représentées par un vecteur $y = y_1,...,y_l$, tel que chaque composante y_k est définie par la transformation :

$$G : a_k \in \mathbb{R}^n \rightarrow y_k = G^T * a_k \in \mathbb{R}^l \tag{3.5.4}$$

Cette transformation est calculée afin que la structure des classes de l'espace vectoriel original soit préservée dans l'espace réduit.

Soit p le nombre de classes de notre problème. On suppose donc que les données sont partitionnées en p sous-ensembles (appelés *clusters*). Chaque exemple appartenant à une classe, on peut décomposer les colonnes de la matrice A en p sous matrices $B_i \in \mathbb{R}^{n \times b_i}$, avec $1 \leq i \leq p$. On note :
- $\overline{s}$ le centre de gravité de l'ensemble des exemples
- $\overline{s_i}$ le centre de gravité de la $i^{\text{ème}}$ classe
- b_i la taille de la sous matrice B_i correspondant à la $i^{\text{ème}}$ classe
- S_i la matrice de covariance de la $i^{\text{ème}}$ classe

L'idée est de rendre les *clusters* les plus compacts possible, c'est-à-dire que les éléments qui les composent doivent être groupés, mais chaque *cluster* doit être éloigné des autres. Pour quantifier la qualité d'un *cluster*, on définit deux matrices :
- $S_w = \sum_{i=1}^{p} b_i S_i$ la matrice intraclasse
- $S_b = \sum_{i=1}^{p} b_i (\overline{s_i} - \overline{s})(\overline{s_i} - \overline{s})^T$ la matrice interclasse

La trace de S_w mesure alors la cohésion intraclasse, tandis que la trace de S_b mesure la séparation interclasse. Après projection suivant la transformation G, ces matrices deviennent :
- $S_w^L = G^T S_w G$
- $S_b^L = G^T S_b G$

Le but de LDA est alors de trouver la transformation optimale G qui maximise la trace de la matrice S_b^L, tout en minimisant la trace de la matrice S_w^L, ce qui peut être résumé en ce problème d'optimisation, dont on note G^* la solution :

$$G^* = maxtrace((S_w^L)^{-1} S_b^L) = maxtrace((G^T S_w G)^{-1} G^T S_b G) \tag{3.5.5}$$

Ce problème est résolu de manière exacte [Fukunaga 90], en faisant une décomposition spectrale sur la matrice $(S_w)^{-1} S_b$, afin de trouver une base de vecteurs propres qui formeront G. Chaque colonne de G sera alors appelée vecteur discriminant.

La principale limitation est qu'au moins une des deux matrices S_w ou S_b doit être inversible. La complexité théorique de cette méthode est en $O(Nnt + t^3)$ avec $t = \min(N,n)$ [Cai 08], notamment à cause de la décomposition spectrale, ce qui limite son utilisation à des ensembles de données de taille réduite. De plus, si le nombre d'observations (d'exemples) est limité comparé au nombre d'attributs, comme dans le cas de la classification par puces à ADN, les performances de LDA peuvent sérieusement se dégrader [Thomaz 05] [Tebbens 07]. Il existe néanmoins de nombreuses variantes à cette méthode, qui visent à réduire l'impact de ces problèmes (*Orthogonal LDA* [Ye 05], *Null Space LDA* [Chen 00] [Huang 02], *Regularized LDA* [Guo 07], etc.).

3.5.5.2 Analyse discriminante linéaire diagonale

Cette méthode est une extension de LDA, elle est souvent notée DLDA, pour *Diagonal LDA*. Elle présuppose que toutes les classes ont la même matrice de covariance S_i, et que celle-ci est diagonale. Les corrélations entre les attributs sont donc ignorées.

Cette méthode obtient généralement de meilleurs résultats que LDA en faible dimension, car elle évite l'instabilité de ses résultats dû au choix de ses matrices de covariance. Cependant, elle souffre des mêmes désavantages lorsque le nombre d'attributs est trop élevé [Xu 09].

Il est intéressant de noter que, malgré ces problèmes, il a été suggéré que les algorithmes de classification DLDA [Dudoit 02], SVM [Brown 00], et *Regularized LDA* [Guo 07] obtenaient de meilleurs résultats que les autres méthodes, pour des données provenant de puces à ADN.

3.5.5.3 Machine à vecteurs de support

Les machines à vecteurs de support ou séparateurs à vastes marges (notées SVM pour *Support Vector Machines*) [Gunn 98] ont été introduites lors de la conférence COLT en 1992 [Boser 92]. Elles s'appliquent aussi bien à des problèmes linéairement séparables que non séparables. En effet, l'idée principale des SVM est de reconsidérer le problème dans un espace de dimension supérieure, éventuellement de dimension infinie. Dans ce nouvel espace, il est alors probable qu'il existe un hyperplan séparateur linéaire. Si c'est le cas, les SVM cherchent parmi l'infinité des hyperplans séparateurs, celui qui maximise la marge entre les classes (cf figure 3.10).

On applique au vecteur d'entrée x une transformation non-linéaire ϕ, afin de décrire les données dans un autre espace. L'espace d'arrivée $\phi(x)$ est alors appelé **espace de redescription**. On cherche alors l'hyperplan séparateur optimal, d'équation $h(x) = \alpha\phi(x) + \beta$, dans l'espace de redescription. Pour cela, on cherche les équations des hyperplans parallèles qui passent par les vecteurs supports, c'est-à-dire les attributs les plus proches de la frontière interclasse. On en déduit l'équation de l'hyperplan optimal, équidistant de ces hyperplans.

En pratique, on ne connaît pas la transformation ϕ, on construit donc plutôt directement la fonction h, appelée fonction noyau. Le théorème de Mercer explicite les conditions que h doit satisfaire pour être une fonction noyau : elle doit être symétrique et semi-définie positive.
Lorsque l'on utilise les SVM, le premier paramétrage consiste donc à choisir une fonction noyau, parmi de nombreuses candidates :
- linéaire
- polynomial
- gaussien
- etc.

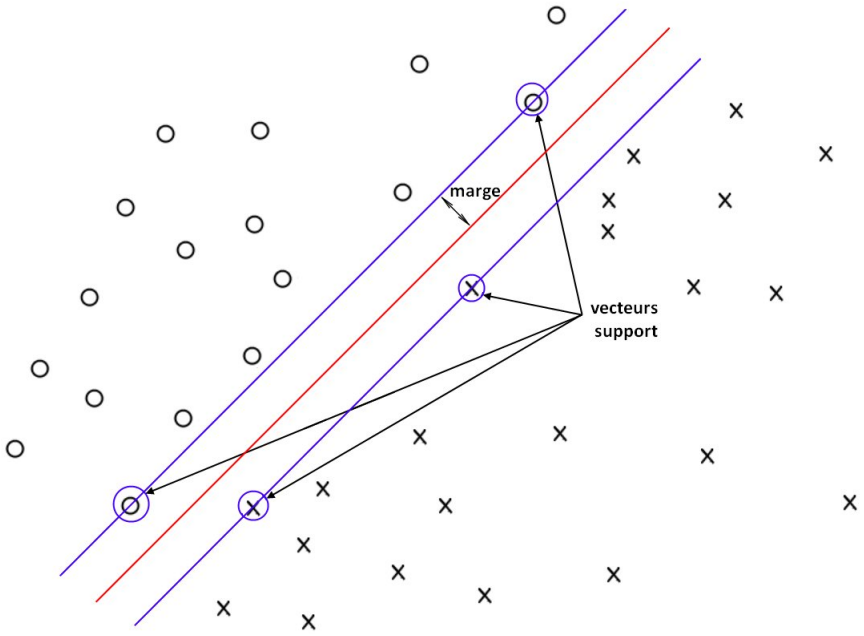


Figure 3.10 – Principe du Séparateur à Vaste Marge (SVM)

Le classifieur SVM peut résoudre des problèmes non séparables linéairement, et le choix de sa fonction noyau le rend très flexible, lui permettant de s'adapter à de nombreux problèmes. Le point négatif de ce classifieur est justement le temps que l'utilisateur doit passer à choisir la fonction noyau à utiliser, puis à la paramétrer.

3.5.5.4 Réseau de neurones

Les réseaux de neurones artificiels sont construits à partir du modèle biologique des neurones. Leurs fondements proviennent des premiers travaux sur les réseaux de neurones [Lettvin 59] qui créèrent un modèle simplifié de neurone biologique, appelé neurone formel. On peut représenter un neurone formel comme un automate à n entrées et une sortie.

Le premier modèle de réseau de neurones complet a été nommé Perceptron [Rosenblatt 58] (cf figure 3.11). Dans ce modèle, une fonction de combinaison associe les entrées du neurone

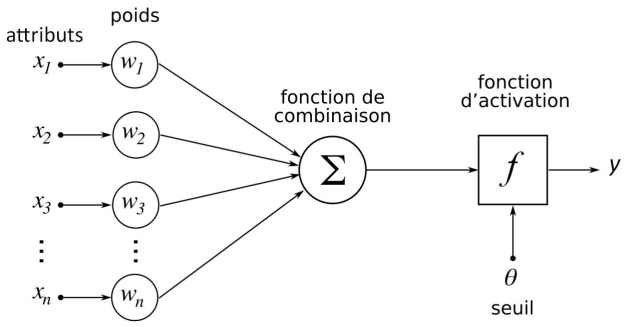


Figure 3.11 – Schéma du perceptron

(par exemple en les sommant), puis compare ce résultat à une valeur seuil, via une fonction d'activation. Typiquement, la fonction d'activation renvoie une valeur binaire, relativement au

fait qu'elle ait été activée ou pas. Les paramètres importants de ce modèle sont les coefficients synaptiques (c'est-à-dire les poids), le seuil, et la façon de les ajuster lors de l'apprentissage. En effet, il faut choisir un mécanisme permettant de les calculer et de les faire converger vers une valeur assurant une classification aussi proche que possible de l'optimale, lors de la phase d'apprentissage.

On peut représenter la sortie du réseau par l'équation suivante (avec f la fonction d'activation) :

$$y = f(\sum_{i=1}^{n}(w_i x_i) - \theta) \tag{3.5.6}$$

Dans sa première version, le neurone formel était implémenté avec une fonction à seuil (pour la fonction d'activation), mais de nombreuses versions existent (fonction linéaire par morceaux, sigmoïde, gaussienne, etc.), pouvant ainsi fournir plusieurs valeurs de sortie.

Cependant, le perceptron ne peut pas prédire correctement les problèmes non linéaires, il faudra attendre les travaux de Werbos sur le perceptron multicouches afin de généraliser le modèle du perceptron. Le modèle du perceptron multicouches [Rumelhart 86] reprend le principe du perceptron, en ajoutant plusieurs vecteurs de poids (appelées couches cachées), afin d'augmenter les combinaisons possibles (cf figure 3.12). Les neurones d'une couche sont reliés aux neurones des couches adjacentes par des liaisons pondérées. Ainsi, le poids de chacune de ces liaisons est l'élément clef du fonctionnement du réseau. On peut considérer que la fonction d'activation est la même pour chaque couche, mais qu'elle varie d'une couche à l'autre. Le perceptron multicouche est un classifieur permettant de résoudre des problèmes non linéaires. Il souffre cependant des mêmes problèmes de paramétrage que les autres classifieurs non linéaires, car il faut paramétrer le nombre de couches, la fonction d'activation, etc.

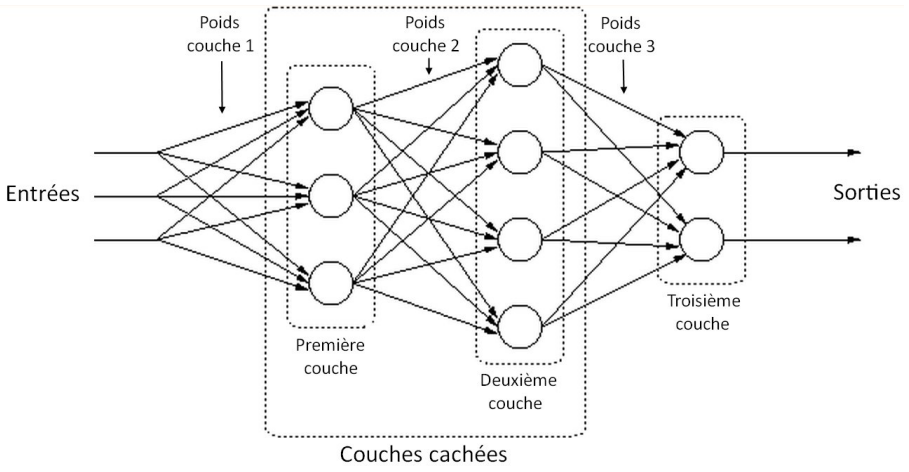


Figure 3.12 – Schéma du perceptron multi couches. Source : [Tertois 03]

Il est à noter que les réseaux de neurones ne sont pas utilisés uniquement dans le cadre de l'apprentissage ou de la classification [Bishop 95], en effet, ils sont également utilisés en optimisation combinatoire [Hopfield 85] [Cochocki 93].

3.6 Phénomène de sur-apprentissage

Ce phénomène est l'un des plus importants lorsque l'on travaille dans le domaine de l'apprentissage. En effet, il faut que le classifieur apprenne suffisamment des données d'apprentissage pour pouvoir prédire de nouveaux exemples. Cependant il ne faut pas qu'il "apprenne trop", sinon il colle parfaitement aux données d'apprentissage et perd de sa souplesse et de sa généralisation lorsqu'on lui présente de nouveaux exemples. En effet, il arrive souvent que les exemples de la base d'apprentissage comportent des valeurs approximatives ou bruitées. Si l'on oblige le modèle de prédiction à répondre de façon quasi parfaite relativement à ces exemples, il devient biaisé par des valeurs erronées. Par exemple, imaginons qu'on présente au réseau des couples $(x_i,f(x_i))$ situés sur une droite d'équation $y = ax + b$, mais bruités de sorte que les points ne soient pas exactement sur la droite. S'il y a un bon apprentissage, le modèle de prédiction sera de la forme $ax + b$ pour toute valeur de x présentée. En revanche, s'il y a sur-apprentissage (*overfitting* en anglais), le réseau répond un peu plus que $ax + b$ ou un peu moins, car chaque couple $(x_i, f(x_i)$ positionné en dehors de la droite va influencer la décision.

La théorie de la **régularisation statistique** introduite par Vladimir Vapnik (appelée théorie de Vapnik-Chervonenkis) [Vapnik 95] permet d'anticiper, d'étudier et de réguler les phénomènes liés au sur-apprentissage. La méthode la plus simple pour éviter ce phénomène est de partager la population d'apprentissage en deux sous-ensembles. Le premier sert à l'apprentissage et le deuxième sert à l'évaluation de l'apprentissage. Tant que l'erreur obtenue sur le deuxième ensemble diminue, on peut continuer l'apprentissage, sinon on l'arrête (cf figure 3.13).

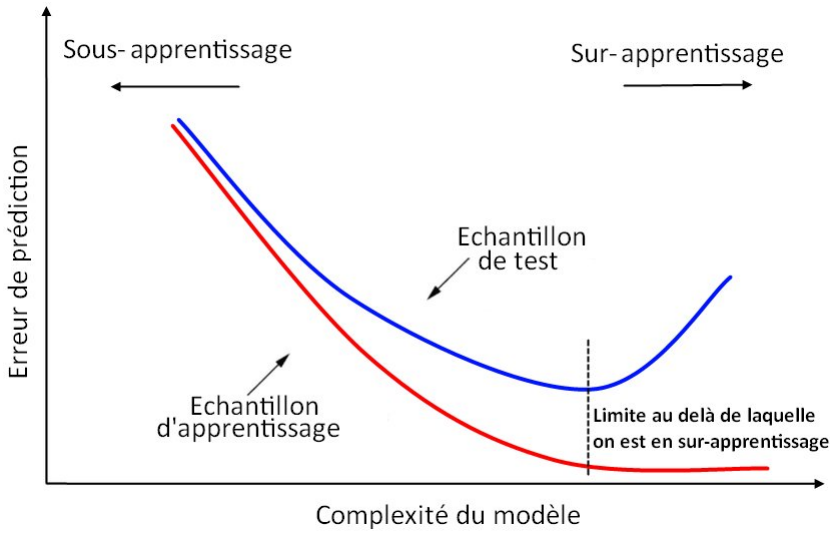


Figure 3.13 – Erreur de prédiction sur les ensembles d'apprentissage et de test, en fonction de la complexité du modèle.

En classification, un bon moyen pour éviter le sur-apprentissage est de limiter le nombre d'attributs. En effet, plus le modèle sera créé à partir d'un nombre limité de variables et plus il sera général. C'est pour cela, en partie, que nous nous intéresserons aux méthodes de sélection d'attributs.

3.7 Sélection d'attributs

Dans le cadre de la bioinformatique (et plus particulièrement l'étude de puces à ADN), l'originalité du problème vient du fait que nous avons un très grand nombre de variables (de gènes) pour un très petit nombre de cas d'études. En effet, les protocoles de test par puce à ADN coûtent encore assez cher et ne sont pas très répandus ; nous disposons donc d'assez peu de données à étudier. De même, une grande partie des gènes du génome humain ayant une fonction non encore découverte, nous avons un très grand nombre de variables avec lesquelles travailler.

3.7.1 Principe

La sélection d'attributs (ou *Feature Selection*) [Jain 97] [Guyon 03] s'est développée dans de nombreux domaines d'applications de la fouille de données [Yang 97], et notamment dans le domaine de la bioinformatique [Xing 01]. Le but de la sélection d'attributs est de choisir, parmi toutes les variables du problème, un sous-ensemble de taille réduite contenant les variables les plus pertinentes concernant le problème.

Les raisons de son utilisation sont multiples :
- La construction d'un modèle de prédiction peut s'avérer très lourde, voire impossible, si le nombre de variable est trop élevé (*curse of dimensionality*). Nous avions vu par exemple que l'analyse discriminante linéaire était un classifieur qui n'obtenait pas de bons résultats si le nombre de variables était trop élevé ;
- Améliorer la qualité de la solution en supprimant les variables qui sont sources de bruits [Xu 09] ;
- Éviter le sur-apprentissage (comme il a été dit précédemment), en améliorant les capacités de généralisation ;
- Améliorer la visibilité et l'interprétation du résultat.

Pour savoir quelles variables sont à écarter, il faut s'attacher à "mesurer" leur pertinence. Une variable est considérée comme pertinente si elle varie de manière systématique en fonction de la classe de l'exemple [Kohavi 97]. On peut d'ailleurs distinguer les variables fortement pertinentes (indispensables pour représenter le problème), faiblement pertinentes (utiles pour la compréhension de certains exemples) ou non pertinentes (non liées au problème et donc inutiles). De plus, si une variable est pertinente, on doit ensuite vérifier qu'elle ne soit pas corrélée avec d'autres (alors dite redondante). Des variables sont redondantes si elles apportent exactement la même information, il ne faut alors en garder qu'une seule. Pour calculer la corrélation de deux vecteurs $x = \{x_1,...,x_n\}$ et $y = \{y_1,...,y_n\}$, on peut utiliser le coefficient de corrélation de Bravais-Pearson (cf équation (3.7.1)).

$$r_P = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sigma_x \sigma_y} \tag{3.7.1}$$

En désignant par $\bar{x}$, $\bar{y}$ les moyennes et σ_x, σ_y les écart-types des vecteurs x et y. Ainsi, si r_P est nul, les deux vecteurs ne sont pas corrélés, tandis qu'ils le seront d'autant plus que $|r_P|$ est proche de 1.

Certains classifieurs n'ont pas besoin de pré-traitement des données par des méthodes de sélection d'attributs, car ils l'incluent dans leur procédé de création du modèle de prédiction. C'est le cas par exemple des arbres de décision, qui rejettent automatiquement les attributs

les moins pertinents lors de la création de l'arbre. À part ces cas d'exception, les méthodes de sélection d'attributs peuvent être séparées en deux grandes catégories : les méthodes par filtre et les méthodes *wrapper* [Inza 04].

3.7.2 Méthodes par filtre

Ces méthodes attribuent une valeur (ou score) à chaque variable en fonction d'un critère statistique. Les variables peuvent ainsi être classées afin de sélectionner celles qui ont le meilleur score. L'intérêt de ces méthodes est que l'on peut calculer ce score indépendamment des autres variables, ce qui les rend d'une complexité raisonnable. En revanche, elles ne prennent pas en compte les interactions entre les variables et donc peuvent retourner des sous-ensembles contenant des variables corrélées.

Afin de formaliser le problème, on pose $x = x_1,...,x_n$ un attribut dont les valeurs varient parmi n exemples. On note $\bar{x}$ la moyenne et σ l'écart-type des valeurs prises par x sur l'ensemble des n exemples. Nous nous plaçons dans le contexte de l'apprentissage supervisé, nous supposons donc que nos données sont réparties en p classes et notons $\bar{x}_i$ et σ_i la moyenne et l'écart-type des valeurs prises pour l'attribut x sur l'ensemble des n_i exemples appartenant à la classe i (tel que $1 \leq i \leq p$).

Nous allons présenter quelques critères classiques de la littérature. Afin de simplifier les équations, nous nous plaçons dans le cadre des problèmes à deux classes ($p = 2$).

t-test de Welsh
Le t-test est probablement le test le plus connu et le plus utilisé. Il en existe différentes variantes qui dépendent de la répartition des classes et de la variance de chaque classe. Le t-test de Welsh est le plus général, mais il pose tout de même l'hypothèse d'une distribution normale des exemples.

$$t(x) = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}}} \tag{3.7.2}$$

F-Test (ou test de Fisher) [Duncan 55]
Ce test repose sur la distribution de Fisher, qui consiste à comparer la différence entre deux variances. On peut remarquer que, dans le cas de deux classes, $F(x) = t(x)^2$

$$F(x) = \frac{(\bar{x}_1 - \bar{x}_2)^2}{\frac{{\sigma_1}^2}{n_1} + \frac{{\sigma_2}^2}{n_2}} \tag{3.7.3}$$

Rapport Signal sur Bruit (*Signal-to-Noise Ratio* noté SNR ou S/N) [Golub 99]
Ce critère (aussi appelé *P-measure*) provient des équations utilisées dans le traitement du signal. Il mesure à quel point la qualité d'un signal a été bruitée.

$$P(x) = \frac{\bar{x}_1 - \bar{x}_2}{\sigma_1 + \sigma_2} \tag{3.7.4}$$

Ici, les attributs ayant de grandes valeurs de $|P(x)|$ seront fortement corrélés à la séparation des classes.

Test du χ^2
Le test du χ^2 est à l'origine utilisé pour évaluer l'indépendance de variables discrètes. Cependant, il est possible d'appliquer ce test sur des variables à valeurs continues, en discrétisant les données [Liu 95].

3.7.3 Méthodes *wrapper*

D'autres méthodes de sélection d'attributs se basent sur l'optimisation d'une fonction objectif afin de trouver un sous-ensemble optimal [Blum 97]. L'espace de recherche est assimilé à l'ensemble des sous-ensembles d'attributs possibles. Une solution du problème est un ensemble d'attributs parmi les 2^n sous-ensembles d'attributs possibles (si l'ensemble est de taille n). Ce problème est considéré comme NP-difficile [Davies 94] et peut être résolu par des heuristiques. Classiquement, le problème peut être assimilé à une optimisation au moins bi-objectif. Le premier objectif étant de minimiser le nombre d'attributs dans le sous-ensemble et le deuxième étant un critère quelconque à optimiser.

Ces méthodes peuvent par exemple optimiser les performances d'un classifieur spécifique, elles sont alors appelées *wrapper gene selection* [Kohavi 97], et s'opposent aux méthodes par filtre (*filter gene selection*) que nous venons de voir [Inza 04]. Si ce critère est assimilé à la précision du classifieur, on peut utiliser la population d'apprentissage afin de construire le modèle de prédiction, mais également pour tester les performances du prédicteur. Par exemple, en séparant l'ensemble d'apprentissage en deux sous-ensembles : le premier servira à la construction du modèle de prédiction par le classifieur choisi, et le deuxième fournira une estimation de la précision du classifieur ainsi construit, qui sera le résultat de la fonction objectif associée au sous-ensemble d'attributs testé.

Les deux méthodes d'optimisation les plus utilisées dans ce domaine sont des méthodes de recherche locale avec politique d'acceptation immédiate d'un mouvement améliorant, qui permettent de pallier au problème lié à la grande dimension. La première est appelée *Forward Selection* (Sélection ascendante, cf figure 3.14). Elle démarre d'une solution ne contenant aucun attribut et parcourt l'ensemble des attributs en les ajoutant à la solution initiale. Lorsqu'elle les a tous parcourus, elle garde le meilleur, l'ajoute à la solution optimale, et recommence jusqu'à ce qu'un critère d'arrêt soit atteint. La deuxième méthode, appelée *Backward Elimination* (Élimination descendante, cf figure 3.15) fonctionne de la même façon, mais en partant d'une solution contenant tous les attributs et en les retirant tour à tour, pour finalement retirer celui qui obtient les moins bons résultats.

Il est à noter que d'autres heuristiques peuvent être utilisées, telles que les algorithmes génétiques [Yang 98], le recuit simulé [Meiri 06], les colonies de fourmis [Sivagaminathan 07], les essaims particulaires [Wang 07], etc.

3.8 Critères de performance

L'évaluation de la performance d'un classifieur se fait par le taux de bonnes classifications. Ainsi, par exemple, si pour 100 exemples de tests, 89 ont été prédits correctement par notre modèle de prédiction, on pourra dire que ce modèle a une précision de 89% (souvent écrit 0.89). Mais la précision n'est pas le seul critère à prendre en compte, notamment pour les problèmes à deux classes.

En effet, lorsque l'on travaille avec des modèles de prédiction binaires, de nouveaux critères peuvent entrer en jeu, notamment dans le domaine médical. On ne s'intéresse pas seulement au nombre de prédiction correctes, mais on veut également savoir si une prédiction a été faite positive alors que l'exemple montrait réellement un résultat négatif (on parle de faux positif) ou l'inverse (on parle alors de faux négatif), comme le montre le tableau 3.2.

Fonction ForwardSelection
début
Soit $P = \emptyset$ l'ensemble solution des attributs sélectionnés
Soit Q l'ensemble des attributs possibles
tant que la condition d'arrêt n'est pas vérifiée **faire**
pour chaque attribut $a \in Q$ **faire**
$P' = P \cup \{a\}$
Créer le modèle de prédiction avec les variables de P' et estimer ses performances
fin pour
$P = P \cup \{a^*\}$ où a^* est l'attribut ayant obtenu les meilleures performances
$Q = Q - \{a\}$
fin tant que
retourner P
fin

Figure 3.14 – Pseudo code de la méthode d'optimisation "Forward Selection"

Fonction BackwardElimination
début
Soit P l'ensemble des attributs possibles
tant que la condition d'arrêt n'est pas vérifiée **faire**
pour chaque attribut $a \in P$ **faire**
$P' = P - \{a\}$
Créer le modèle de prédiction avec les variables de P' et estimer ses performances
fin pour
$P = P - \{a^*\}$ où a^* est l'attribut ayant obtenu les moins bonnes performances
fin tant que
retourner P
fin

Figure 3.15 – Pseudo code de la méthode d'optimisation "Backward Elimination"

On peut alors définir les termes suivants :
- Sensibilité : C'est la probabilité que la prédiction soit C lorsque l'exemple est de classe C. On la calcule par l'équation : $Se = \frac{VP}{VP+FN}$.
- Spécificité : C'est la probabilité que la prédiction soit $\overline{C}$ lorsque l'exemple est de classe $\overline{C}$. On la calcule par l'équation : $Sp = \frac{VN}{VN+FP}$.
- Valeur Prédictive Positive : C'est la probabilité que l'exemple soit de classe C lorsque la prédiction est C. On la calcule par l'équation : $VPP = \frac{VP}{VP+FP}$.
- Valeur Prédictive Négative : C'est la probabilité que l'exemple soit de classe $\overline{C}$ lorsque la prédiction est $\overline{C}$. On la calcule par l'équation : $VPN = \frac{VN}{VN+FN}$.

Ces valeurs sont primordiales lors d'une classification binaire et sont malheureusement souvent

	C	$\overline{C}$
Prédit C	Vrai Positif (VP)	Faux Positif (FP)
Prédit $\overline{C}$	Faux Négatif (FN)	Vrai Négatif (VN)

Tableau 3.2 – Résultats détaillés d'une prédiction binaire (tableau de confusion).

délaissées au profit de la précision. Elles permettent de déterminer si le modèle est performant à prédire une caractéristique spécifique dans une population donnée.

La sensibilité et la spécificité d'une classification doivent toujours être données ensemble. Par exemple, un test avec une sensibilité de 95% n'est pas forcément un bon résultat si sa spécificité n'est que de 5%. Cela voudrait simplement dire que la classification prédit 95% des résultats comme étant de la classe C, sans corrélation avec la réalité. On peut d'ailleurs définir l'indice de Youden : $I_{Youden} = Se + Sp - 1$. Cet indice révèle l'efficacité de la prédiction : s'il est négatif ou nul, la classification est inefficace. Elle est d'autant plus efficace qu'il se rapproche de 1. Dans notre exemple, le I_{Youden} vaudrait 0, révélant que la prédiction est inefficace.

3.9 Méthodes de validation

Pour avoir une estimation correcte de l'erreur de classification, il faut recourir à un ensemble d'exemples qui n'ont pas servi pour l'apprentissage : il s'agit de l'ensemble de test. Cet ensemble doit également contenir des exemples étiquetés (dont on connaît les classes), afin de comparer les prédictions faites par le modèle à la valeur réelle de la classe.

Il est généralement obtenu en réservant une partie des exemples supervisés initiaux et en ne les utilisant pas pour la phase d'apprentissage. Cependant, lorsqu'on dispose de peu d'exemples, comme c'est le cas dans le traitement des données d'expression de puces à ADN, il est pénalisant de laisser de côté une partie des exemples pendant la phase d'apprentissage. De plus, si l'on fixe l'ensemble d'apprentissage et l'ensemble de test, notre résultat sera biaisé au choix de ces deux ensembles, et rien ne nous dit qu'un nouvel ensemble de test obtiendra les mêmes résultats.

Les techniques de validation croisée (ou *cross validation*) [Stone 77] [Kohavi 95], permettent d'obtenir une estimation des performances du prédicteur, en exploitant la totalité du jeu de données. Ceci est obtenu en faisant plusieurs tests sur différents ensembles d'apprentissage et de test, et en faisant la moyenne des résultats. En effet, si l'on obtient une bonne précision en moyenne, ainsi qu'un écart-type faible, notre méthode de prédiction pourra être considérée comme robuste.

3.9.1 Validation croisée : *k-Fold*

L'algorithme de validation croisée k-*Fold* consiste à segmenter aléatoirement les exemples du jeu de données initial en k sous-ensembles disjoints numérotés de 1 à k. On va ensuite écarter le 1er bloc qui nous servira d'ensemble de test, et utiliser les k - 1 autres blocs afin de constituer l'ensemble d'apprentissage. Puis, il suffit de suivre le protocole classique de construction du modèle de prédiction sur l'ensemble d'apprentissage, en validant ses performances sur l'ensemble de test. On obtient donc une estimation des performances du prédicteur calculé sur les blocs 2 à k et dont on a validé les performances sur le 1er bloc.

On recommence ensuite cette opération en réservant chaque bloc successivement comme bloc de test, et on calcule la moyenne et l'écart-type des résultats sur les k expériences.

De manière générale, on choisit $k = 10$ si le jeu de données est suffisamment grand.

3.9.2 Validation croisée : *Leave One Out*

Cette méthode est dérivée de la méthode de validation *k-Fold*, en prenant $k = n$, n étant le nombre d'exemples. A chaque itération, on va donc faire l'apprentissage sur tous les exemples moins un, et tester sur un seul exemple, afin de vérifier s'il est prédit correctement.

Cette méthode est plus coûteuse, en terme de temps de calcul, que la validation croisée *k-fold*, car elle nécessite beaucoup plus de tests. Cependant, elle sera intéressante pour des jeux de données sur lesquels on ne dispose que de peu d'exemples. Ou alors si l'on a besoin d'une estimation plus précise des performances du modèle de prédiction.

3.9.3 Validation croisée : *Repeated Random SubSampling*

Cette méthode est également dérivée de la méthode de validation *k-Fold* et y est même assimilée par erreur dans de nombreuses publications. L'idée est également de découper aléatoirement la population, classiquement en 2/3 pour l'ensemble d'apprentissage et 1/3 pour l'ensemble de test (qui est appelé *3-fold* dans la littérature, par abus de langage). Une fois la prédiction faite et les performances du prédicteur estimées, l'ensemble initial est reconstitué. On recommence ensuite cette procédure autant de fois que l'on veut et on calcule la moyenne et l'écart-type des performances sur l'ensemble des tests.

L'intérêt principal de cette méthode est que l'on peut définir la proportion des ensembles d'apprentissage et de validation indépendamment du nombre d'itérations. Son principal désavantage est que l'on ne peut pas prévoir si tous les exemples seront testés.

3.9.4 *Bootstrapping*

Le *bootstrapping* [Efron 83] est une technique très souvent utilisée dans le cadre de jeux de données contenant peu d'exemples. Lorsque l'on utilise le principe de validation croisée, on sélectionne aléatoirement les exemples dans le jeu de données initial afin de constituer les ensembles d'apprentissage et de test, en prenant garde à ne pas prendre deux fois le même exemple. Le *bootstrapping* nous permet de faire un tirage avec remise, c'est-à-dire d'autoriser de prendre plusieurs fois le même exemple, et ce sans dénaturer la distribution statistique des exemples [Efron 79].

Cette technique permet donc de transformer l'ensemble de données initial afin d'utiliser au mieux les exemples dont on dispose.

3.10 Conclusion

Les puces à ADN constituent une technologie récente permettant l'analyse simultanée de l'expression de nombreux gènes. Elles permettent d'établir le profil génétique d'une personne, d'une tumeur, ou plus généralement d'un échantillon biologique.
L'analyse des données fournies par ces biopuces est faite par des méthodes issues de la fouille de données [Hastie 01]. C'est un problème d'apprentissage automatique supervisé, c'est-à-dire que l'on essaie de prédire le comportement de nouveaux cas, en apprenant à partir de cas d'apprentissage. L'originalité de ce problème vient du fait que dans les données brutes, il y a très peu de cas d'étude (d'exemples d'apprentissage) en comparaison du nombre de gènes.

Le but est donc de limiter ce nombre de gènes, en utilisant des techniques de sélection de variables. Ces techniques peuvent être de deux types : par filtre ou *wrapper*. Les méthodes par filtre utilisent un critère statistique afin de classer les variables, tandis que les méthodes *wrapper* optimisent les performances d'un classifieur particulier. Ces méthodes permettent de créer des prédicteurs de faible taille, pouvant ensuite être utilisés en routine clinique. En effet, ces prédicteurs seront d'autant plus fiables et d'autant moins coûteux que le nombre d'attributs (qui sont des gènes ici) est faible.

Nous avons vu dans ce chapitre, que des méthodes d'optimisation étaient utilisées dans le cadre de la sélection d'attributs, et notamment pour les techniques *wrapper*. Ces méthodes ne sont pas toujours adaptées à la dimension du problème, car le nombre de variables peut varier de plusieurs milliers à des dizaines de milliers.
Dans la suite de ce manuscrit, nous montrerons l'intérêt des méthodes d'optimisation de grande dimension pour ce type de problème. Dans ce but, nous appliquerons les méthodes que nous avons présentées dans les chapitres précédents.

Chapitre Quatre

Élaboration de nouvelles méthodes de sélection d'attributs

4.1 Introduction

Dans ce dernier chapitre, nous présentons différentes méthodes issues de nos algorithmes d'optimisation développés dans le chapitre 2. Ceux-ci sont appliqués et comparés sur différents jeux de données d'expression géniques mesurées par puces à ADN, pour des problèmes d'apprentissage supervisé, à deux classes.

Le nombre de variables (ici : les sondes géniques) étant très élevé, nous avons concentré notre étude sur la sélection des variables, préalable à la construction des prédicteurs. Un prédicteur est caractérisé par un ensemble de variables et un modèle de classification. Sa construction nécessite, d'une part, d'effectuer une sélection de variables pertinentes pour la classification et, d'autre part, de minimiser le nombre de variables, afin d'éviter le sur-apprentissage des données. Pour cela, nous avions besoin d'un algorithme efficace pour résoudre des problèmes de grande dimension. Nous avons essentiellement repris le principe de notre algorithme EUS, que nous avons *discrétisé* pour l'appliquer à ce type de problèmes.

Dans une première partie, nous présenterons l'algorithme résultant, nommé BEUS (*Binary* EUS) et son application sur un jeu de données issu d'un essai clinique conduit à l'Institut Gustave Roussy, Villejuif, et au MD Anderson Cancer Center, Houston, Texas, USA. Ensuite, nous montrons que cet algorithme discrétisé peut être reformulé en une méthode analytique, nommée δ-test, par laquelle l'optimisation bi-objectif est résolue de manière exacte en temps linéaire. Cette démonstration permet d'ailleurs d'expliquer le principe général des méthodes de sélection par filtre.
L'algorithme BEUS, et la méthode δ-test équivalente, calculent un sous-ensemble de gènes optimal pour une taille fixée ; cependant, la taille optimale du sous-ensemble dépendra du modèle de classification. Nous proposons, dans un deuxième temps, un processus automatique de choix du nombre de variables à sélectionner, processus dénommé *SelectKStar*.

Enfin, nous présentons une nouvelle version de l'algorithme BEUS, dénommée ABEUS (*Aggregated* BEUS) qui permet de résoudre la fonction bi-objectif en intégrant la minimisation d'un des deux objectifs dans le corps de la procédure. Elle sera principalement utilisée comme méthode *wrapper*.

Tous ces résultats sont comparés à d'autres méthodes de la littérature et sont discutés dans une dernière partie.

4.2 Prédiction en onco-pharmacogénomique

Dans un premier temps, nous nous sommes concentrés sur un jeu de données portant sur des patientes atteintes du cancer du sein. Dans l'essai clinique dont les données sont issues, toutes les patientes avaient reçu un traitement de chimiothérapie préopératoire. Avant d'administrer le traitement, les niveaux d'expression des gènes ont été mesurés sur tissu tumoral, puis, après le traitement, la classe de la patiente, répondeuse ou non répondeuse au traitement, fut déterminée à la chirurgie (réponse pathologique complète ou non.) L'objectif de l'essai clinique est de fournir des données pour la construction de prédicteurs de la réponse à la chimiothérapie préopératoire. De tels prédicteurs permettraient de ne donner le traitement qu'à bon escient, sans en priver aucune patiente répondeuse au traitement.

Les niveaux d'expression des gènes ont été mesurés par puce à ADN Affymetrix U133A. Une telle puce à ADN mesure les niveaux d'expression de 22283 gènes [1]. L'étude fournie par l'hôpital Tenon contient deux jeux de données obtenus dans les mêmes conditions expérimentales, l'un provenant du laboratoire de recherche en cancérologie de Houston et l'autre de l'Institut Gustave Roussy (IGR) de Villejuif.

Lors de la construction de prédicteurs, après sélection des variables pertinentes, un point important est le choix du classifieur. En effet, comme nous l'avons vu dans la section 3.5, il existe de nombreux classifieurs : LDA, DLDA, SVM, k-NN, etc. Ces classifieurs ne sont efficaces qu'avec un nombre restreint de variables. De plus, nous avons vu qu'une trop grande quantité de variables peut aboutir à un phénomène de sur-apprentissage, qui obérerait leur utilisation thérapeutique, puisqu'en cas de sur-apprentissage, aucune confiance ne pourrait être attribuée à la prédiction retournée sur un nouveau cas. Enfin, un des buts essentiels d'une telle étude est la création d'un prédicteur utilisable en routine clinique. Plus faible sera le nombre de gènes dont les niveaux d'expression doivent être mesurés, plus faible sera le coût du prédicteur.

Nous avons donc pris le parti de nous concentrer sur la phase de sélection de variables, afin de sélectionner les attributs les plus pertinents en prétraitement de la construction du modèle du classifieur.

4.2.1 Méthode de sélection d'attributs

Le but de la sélection d'attributs a été décrit dans la section 3.7. Nous devons donc sélectionner un sous-ensemble de variables (les plus pertinentes) parmi l'ensemble des n variables disponibles (ici : $n = 22283$). L'espace de recherche est l'ensemble des sous-ensembles des variables. Pour formaliser le problème, nous notons $2^{\mathcal{P}}$ l'ensemble de tous les sous-ensembles de variables, sa taille est de 2^n. Le problème peut donc bien être caractérisé comme de grande dimension, puisque, dans notre cas, $n = 22283$, ce qui rend impossible toute démarche de sélection procédant par énumération.

Comme nous l'avons vu, il existe principalement deux types d'approches de sélection de variables, les méthodes par filtre, qui classent les variables en fonction d'un critère statistique, et la sélection par optimisation des performances d'un classifieur (méthodes *wrapper*). Nous opterons pour une méthode intermédiaire, qui optimise une fonction objectif non liée à un clas-

1. Pour la clarté de la présentation, nous parlerons de gènes plutôt que de sondes, bien qu'il n'y ait pas équivalence : un gène peut être représenté par plusieurs sondes sur les puces à ADN.

sifieur particulier. Cependant, nous verrons par la suite que ce type de méthodes est en fait lié aux méthodes par filtre, qui correspondent également à l'optimisation d'une fonction objectif.

Nous commençons alors par formaliser le problème afin d'en déduire une représentation des solutions et la fonction objectif à optimiser.

4.2.2 Représentation d'une solution

Une solution réalisable S est un sous-ensemble de variables. Nous avons choisi de la représenter par une chaîne binaire de longueur n, de terme général $S_i = 1$ si et seulement si la i-ème variable est élément de la solution réalisable.

La solution S peut également être représentée par un sous-ensemble de taille $|S|$ contenant uniquement les variables sélectionnées. Cependant, pour des raisons d'efficacité algorithmique, cette représentation est difficilement utilisable dans un contexte d'optimisation. La représentation par chaîne binaire a donc été préférée, et permet d'obtenir cette deuxième représentation par un simple parcours de la chaîne binaire.

4.2.3 Choix de la fonction objectif

Nous considérons le problème de sélection de variables comme un problème combinatoire d'optimisation d'une fonction objectif f sur un espace de recherche $2^{\mathcal{P}}$. La principale difficulté en recherche opérationnelle est de définir la fonction objectif, qui va nous permettre d'obtenir la meilleure solution au problème posé. Dans le cadre de la sélection de variables, nous avons plusieurs choix possibles :

- $f(S)$ = performances (précision, sensibilité, ...) d'un classifieur construit avec le sous-ensemble de variables S (méthode dite *wrapper gene selection*, vue dans la section 3.7.3).
- $f(S)$ = un critère statistique. On peut par exemple minimiser la corrélation entre les variables pour éviter la redondance. Une méthode très connue utilise ce type de critère : la méthode LASSO [Tibshirani 96], qui considère le problème sous la forme d'une régression linéaire et minimise ensuite le critère des moindres carrés.

Comme nous l'avons vu, la première méthode est souvent utilisée dans la littérature [Inza 04] ; cependant, elle restreint la méthode de sélection de variable à un classifieur particulier. En effet, un sous-ensemble de variables minimisant l'erreur en sortie pour un modèle particulier (par exemple SVM) n'a aucune raison de minimiser cette erreur pour un autre modèle (par exemple LDA). De plus, les temps de calcul pour la sélection des variables sont très importants, puisque chaque évaluation de la fonction objectif demande la construction d'un prédicteur et la classification de différents exemples. Enfin, rien ne justifie que les variables trouvées soient statistiquement cohérentes avec le problème (pour les critères du t-test par exemple). En effet, la maximisation de la fonction objectif implique que la méthode cherchera à obtenir un taux de classification parfait de tous les exemples, sans prendre en compte la pertinence de chacune des variables. Ce type d'optimisation peut conduire rapidement à un sur-apprentissage, lors de la sélection des variables pertinentes. Dans notre étude, nous étudierons ces deux types de fonctions objectifs.

Afin de sélectionner le critère statistique à optimiser, nous avons étudié les différents classifieurs existants : LDA (*Linear Discriminant Analysis*), DLDA (*Diagonal LDA*), SVM (*Support Vector Machine*), que nous avons présentés dans la section 3.5. Ces méthodes construisent un modèle de

classification avec, pour objectif premier, de séparer au mieux les différentes classes. Nous avons donc pris le parti de sélectionner les variables de façon à séparer au mieux les différentes classes, c'est-à-dire en maximisant la distance interclasse. Ce prétraitement devrait donc permettre aux différents classifieurs de disposer de données préalablement séparées, facilitant la construction de leur modèle.

Dans ces travaux, nous nous sommes attachés au traitement des problèmes à deux classes, correspondant aux données génomiques dont nous disposions. Dans ce cas, nous avons défini la distance interclasse comme la distance entre les centres de gravité des deux ensembles de cas, relativement aux variables sélectionnées. Soient R et R' les deux ensembles d'exemples, et soit $S = \{var_1, var_2, ..., var_k\}$ un sous-ensemble représentant k variables sélectionnées. Chaque exemple peut être restreint à ces k variables et représenté comme un vecteur de $\mathcal{R}^k$ tel que chaque dimension i corresponde à la valeur de la variable var_i pour cet exemple. Les deux classes sont représentées par leurs centres de gravité notés $\overline{R(S)}$ et $\overline{R'(S)}$. La distance interclasse $d(S)$, du sous-ensemble de variables S considéré, est la distance euclidienne entre les deux centres de gravité :

$$d(S) = distanceEuclidienne\left(\overline{R(S)}, \overline{R'(S)}\right) \tag{4.2.1}$$

Si l'on ne considère que ce critère de distance, le maximum est atteint pour la solution réalisable $S = \mathcal{P}$, ensemble de toutes les variables (la distance est strictement croissante en fonction du nombre de variables.) Nous devons donc fixer un second objectif : la minimisation de la taille du sous-ensemble de variables.
Le résultat est une fonction bi-objectif ayant deux critères contradictoires à optimiser :

1. minimisation de la taille du sous-ensemble ;
2. maximisation de la distance interclasse.

Nous avons vu dans la section 1.11 qu'il existait différentes méthodes pour résoudre des problèmes multiobjectif. Dans notre cas, nous avons choisi l'agrégation des objectifs en une unique fonction objectif. Cette fonction $f_w(S)$ est définie comme une combinaison linéaire convexe des deux objectifs ci-dessus :

$$f_w(S) = w \times d(S) + (1 - w) \times (1 - |S|) \tag{4.2.2}$$

où $w \in [0,1]$ représente le poids donné à chaque objectif, $d(S)$ la distance interclasse et $|S|$ la taille du sous-ensemble considéré.

Formellement, pour chaque valeur du paramètre w, l'optimisation recherche un sous-ensemble de gènes optimal $S^*(w) = \arg\max_{S \in 2^{\mathcal{P}}} f_w(S)$. De grandes valeurs du paramètre w conduisent à des optimums favorisant la distance interclasse, au détriment de la taille du sous-ensemble de gènes, et par conséquent au détriment de la robustesse du prédicteur. À l'inverse, de petites valeurs de w conduiront à de petits sous-ensembles de gènes, au détriment de la distance, donc des performances du prédicteur.

4.2.4 Méthode d'optimisation

La fonction objectif ayant été choisie, il s'agit à présent de choisir un algorithme d'optimisation permettant de résoudre ce type de problèmes. Les méthodes classiques pour ce genre d'optimisation sont les méthodes de *forward selection* et de *backward elimination* présentées dans la section 3.7.3. Elles ont par exemple été utilisées pour minimiser le taux d'erreur du classifieur

SVM [Rakotomamonjy 03]. De nombreuses autres méthodes d'optimisation ont été testées dans la littérature, telles que les algorithmes génétiques [Jarvis 05], ou bien les méthodes d'optimisation par essaims particulaires [Shen 04]. Le problème principal de ces méthodes est qu'elles ne sont pas toujours adaptées pour résoudre des problèmes de si grande dimension. La vitesse de convergence est d'ailleurs directement influencée par la complexité de la fonction objectif. Par conséquent, si l'on considère que cette dernière est basée sur le taux d'erreur d'un classifieur comme SVM, la construction du modèle de prédiction, puis la prédiction en validation croisée, peuvent rendre la recherche très lente.

Pour remédier au problème, de nombreuses méthodes présélectionnent une partie des variables qui semblent les plus pertinentes, afin de réduire l'espace de recherche. Cette pré-sélection peut être faite à partir de méthodes par filtre, telles que le t-test de Welsh, qui s'exécutent très rapidement et permettent d'éliminer les variables dont la distribution des valeurs est sans lien avec la classification des exemples d'apprentissage. Un problème essentiel est le seuil de *p-value* au-dessus duquel on considère que la variable est sans lien avec la classification : un seuil faible réduira fortement le nombre de variables, mais en rejetant des variables pertinentes pour la classification.

Ce genre de méthodes a par exemple été utilisé dans [Hernandez 08] ; elles permettent de passer d'un problème de grande dimension à un problème plus classique d'optimisation, afin d'obtenir des résultats en un temps "raisonnable". Cependant, cette présélection des variables, parce qu'elle limite a priori l'espace des solutions réalisables, ne peut garantir l'optimalité de la solution trouvée. De plus, les méthodes par filtre ne prennent pas en compte des corrélations entre les variables. Ainsi, une variable ayant un faible score individuel peut tout de même s'avérer pertinente, si elle est choisie en même temps qu'une autre. C'est alors que l'on voit l'intérêt d'utiliser des méthodes d'optimisation dédiées aux problèmes de grande dimension. En effet, ces méthodes vont permettre d'optimiser la fonction objectif sur l'ensemble de l'espace de recherche. Leur convergence permet d'obtenir rapidement un optimum local.

Nous avons décidé d'utiliser l'algorithme EUS présenté dans la section 2.4, qui est destiné aux problèmes de grande dimension. Cependant, une solution étant représentée par une chaîne binaire, nous avons affaire à un domaine de recherche discret. Or la méthode EUS a été élaborée pour résoudre des problèmes continus, et elle nécessite donc une adaptation.

4.2.4.1 Discrétisation de la méthode EUS : DEUS et BEUS

Pour discrétiser l'algorithme EUS, nous avons étudié son fonctionnement pas à pas. Il s'avère que le point gênant la discrétisation de l'algorithme est la recherche du voisinage des solutions, qu'il considère sur chaque dimension. En effet, pour chaque composante i, EUS calcule deux solutions voisines de la solution $S = \{S_1,...,S_i,...,S_n\}$: $S^+ = \{S_1,...,S_i + h_i,...,S_n\}$ et $S^- = \{S_1,...,S_i - h_i,...,S_n\}$. Il choisit ensuite la meilleure parmi l'ensemble $\{S, S^+, S^-\}$, c'est-à-dire celle ayant la meilleure valeur de la fonction objectif, avant de procéder de même sur la composante suivante. Dans le cas discret, ce vecteur h doit être un vecteur de variables entières. De plus, à chaque fois qu'une itération ne parvient pas à améliorer une solution, il faut réduire la valeur de chaque h_i jusqu'à ce qu'elle atteigne un seuil $h_{min} = 1$. Le pseudo-code d'une itération de la méthode *Discrete* EUS (DEUS) est détaillé dans la figure 4.1, en désignant par $E(h_i \times r)$ la partie entière de $h_i \times r$.

Dans notre cas, le problème est binaire, le vecteur h n'a donc plus de raison d'être, puisqu'il n'existe que deux valeurs possibles pour chaque composante. Ainsi, l'algorithme résultant *Bi-*

Procédure DEUS
(Soit S la solution courante, représentée par un vecteur discret)
début
  pour $i = 1$ **à** n **faire**
    $S^+ = S + h_i e_i$
    $S^- = S - h_i e_i$
    (on met à jour S comme la meilleure des 3 solutions)
    $S = arg \ \ min(f(S), f(S^+), f(S^-))$ (dans le cas d'une minimisation)
  fin pour
  si aucune amélioration de la solution n'a été trouvée en parcourant les n directions **alors**
    $h_i = E(h_i \times r)$ pour tous les $i \in [1,n]$
  fin si
fin

Figure 4.1 – Une itération de l'algorithme DEUS

nary EUS (BEUS) [Gardeux 11c] calcule, sur chaque dimension i, une seule solution voisine de la solution $S = \{S_1,...,S_i,...,S_n\}$: $S' = \{S_1,...,\bar{S}_i,...,S_n\}$. L'optimisation se termine lorsqu'une itération entière ne parvient pas à améliorer la solution S, ce qui donne à l'algorithme une convergence très rapide. Le pseudo-code d'une itération de la méthode BEUS est détaillé dans la figure 4.2.

Procédure BEUS
(Soit S la solution courante, représentée par un vecteur binaire)
début
  Initialiser aléatoirement S
  faire
    $fit = f(S)$
    pour $i = 1$ **à** n **faire**
      $S_i = 1 - S_i$
      $fit' = f(S)$
      si (fit' est moins bon que fit) **alors**
        $S_i = 1 - S_i$
      fin si
    fin pour
  tant que une meilleure solution est trouvée durant une itération complète
fin

Figure 4.2 – Pseudo-code de l'algorithme BEUS

L'algorithme d'optimisation consiste donc à ajouter une variable σ au sous-ensemble de variables courant S, si $f(S \cup \{\sigma\})$ est meilleur que $f(S)$. De même, la variable σ sera retirée du sous-

ensemble de variables courant S, si $f(S \setminus \{\sigma\})$ est meilleur que $f(S)$. L'arrêt de l'itération est obtenu lorsque l'on atteint une solution stable S^*, ensemble pour lequel la valeur de la fonction bi-objectif ne peut plus être améliorée par ajout ou retrait d'une variable.
Il s'agit d'une recherche locale avec politique du premier voisin améliorant. On peut dire que ce principe rejoint les méthodes de *forward selection* et de *backward elimination*, mais il en diffère par les points suivants :
- la solution initiale est générée aléatoirement ;
- les algorithmes de *forward selection* et de *backward elimination* parcourent toutes les dimensions avant de modifier le sous-ensemble solution. BEUS peut potentiellement le modifier à chaque composante parcourue, ce qui améliore considérablement la convergence ;
- la condition d'arrêt de notre algorithme est fixée et conduit à un optimum local, qu'une simple perturbation ne pourra plus améliorer.

Différentes améliorations peuvent être envisagées :
- explorer les dimensions dans un ordre aléatoire, plutôt que séquentiel ;
- ajouter ou retirer des sous-ensembles de variables, plutôt que chaque variable séparément ;
- redémarrer la procédure plusieurs fois et mémoriser les optimums locaux pour ensuite les combiner et tenter de les améliorer.

Certaines de ces idées seront développées par la suite. La deuxième idée peut s'avérer intéressante, mais elle dépasse le cadre de travail de cette thèse.

4.2.5 Application de notre algorithme BEUS sur un jeu de données

Afin de tester les performances de notre algorithme de sélection de variables, nous l'avons appliqué sur le jeu de données de l'essai clinique conduit à l'*Institut Gustave Roussy* et au *MD Anderson Cancer Center*. Ces données ont été séparées selon le centre dans lequel les patientes ont été traitées, Houston ou Villejuif. (cf tableau 4.1). Des données cliniques complémentaires

Nom	Houston	Villejuif
Puce à ADN	Affymetrix U133A	Affymetrix U133A
Type	Cancer du sein	Cancer du sein
Nb gènes	22283	22283
Nb patientes	82	51
Nb répondeuses	21	13
Nb non répondeuses	61	38

Tableau 4.1 – Caractéristiques des deux jeux de données Houston et Villejuif.

nous permettent de connaître la classe de chaque patiente, c'est-à-dire si elle est répondeuse au traitement (PCR : *Pathologic Complete Response*) ou non répondeuse (No-PCR).

4.2.5.1 Normalisation des données

Après une première analyse de ces données, on remarque que les expressions de chaque gène peuvent varier sur des plages très différentes. Par exemple, sur le jeu de données de Houston, le gène ESR1 (205225_at) varie dans l'intervalle [62,09 ; 6864,57], tandis que le gène CDC14

(205288_at) varie dans [0 ; 20,8]. Ce comportement peut conduire à des biais lors de la comparaison de l'importance de chaque gène. De plus, lors de l'analyse de données issues de puces à ADN, les effectifs sont souvent plus importants vers les faibles intensités. Ainsi, une normalisation logarithmique permet de de représenter les variations de l'expression de chaque gène sur l'intervalle [0, 1], et recentre la distribution des valeurs tout en la rendant plus uniforme.

Soit $\sigma = \{\sigma_1,...,\sigma_i,...,\sigma_p\}$ le vecteur à normaliser, il correspond à l'ensemble des niveaux d'expression pris par un unique gène σ parmi les p patientes. Soient σ_{max} et σ_{min} les extremums du vecteur σ. Chaque composante $\dot{\sigma}_i$ du vecteur normalisé $\dot{\sigma}$ est calculée comme suit :

$$\dot{\sigma}_i = \frac{log(\sigma_i) - log(\sigma_{min})}{log(\sigma_{max}) - log(\sigma_{min})} \tag{4.2.3}$$

où *log* correspond à la fonction logarithme.

Dans le cas où des valeurs sont inférieures ou égales à 0, les données sont translatées de $|\sigma_{min}|+1$, avant de faire la normalisation. Pour éviter tout biais dans la construction du prédicteur et la prédiction sur l'ensemble de test, nous avons choisi de ne normaliser l'ensemble d'apprentissage qu'en prétraitement au processus de sélection de variables. Le classifieur construit donc son modèle à partir des données d'apprentissage brutes, restreintes aux variables sélectionnées, puis prédit le comportement des données de test brutes.

4.2.5.2 Protocole expérimental

Le protocole d'optimisation étant fixé, il nous reste à évaluer la qualité de nos sous-ensembles de variables sélectionnées. Pour cela, nous avons choisi le classifieur DLDA vu dans la section 3.5.5.2. Ce classifieur a en effet été plusieurs fois cité comme l'un des plus performants sur les problèmes de prédiction en génomique [Dudoit 02], juste devant la méthode k-NN. De plus, la méthode DLDA est connue pour avoir des résultats ayant une forte sensibilité [Miller 05]. Ceci est particulièrement important, car une valeur élevée de sensibilité implique un faible nombre de faux négatifs (cf section 3.8). Dans notre cas, il est effectivement crucial de pouvoir prédire avec une grande précision une patiente répondeuse au traitement, si elle l'est réellement (vrai positif), car dans le cas contraire (faux négatif), cela reviendrait à ne pas donner le traitement à une personne qui y serait sensible et donc qui tirerait un bénéfice de ce traitement.

4.2.5.3 Biais de sélection

Il est très important que les données utilisées pour évaluer les performances du classifieur soient distinctes de celles utilisées pour la sélection des gènes et pour construire le modèle du classifieur [Dupuy 07]. Dans le cas contraire, on utilise indirectement une partie de la population de test pour faire notre sélection de variables ; c'est ce que l'on appelle le **biais de sélection** [Ambroise 02] [Simon 03]. La sélection de variables doit être faite en utilisant uniquement l'ensemble d'apprentissage. Ainsi, si l'on effectue une validation croisée et que l'on calcule sa performance en même temps que l'on fait la sélection de variables, il ne faut pas utiliser le résultat comme estimation des performances du prédicteur ; il faut réitérer la sélection de variables sur tous les sous-ensembles d'apprentissage. Les figures 4.3 et 4.4 illustrent les protocoles expérimentaux à utiliser pour éviter le biais de sélection. Dans la figure 4.3, la taille de l'ensemble initial est fixée à P patientes et N gènes. Lors de la séparation, on obtient deux nouveaux ensembles : un ensemble d'apprentissage de P_1 patientes et un ensemble de test de P_2 patientes

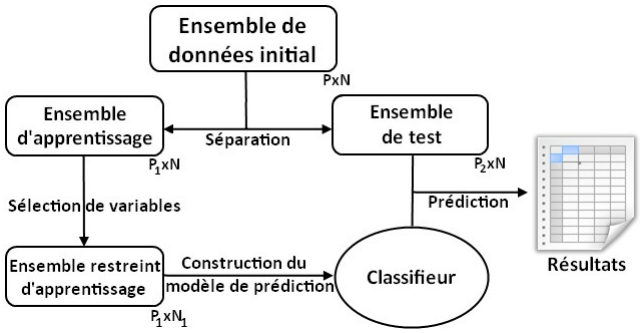


Figure 4.3 – Protocole expérimental dans le cas d'une seule prédiction.

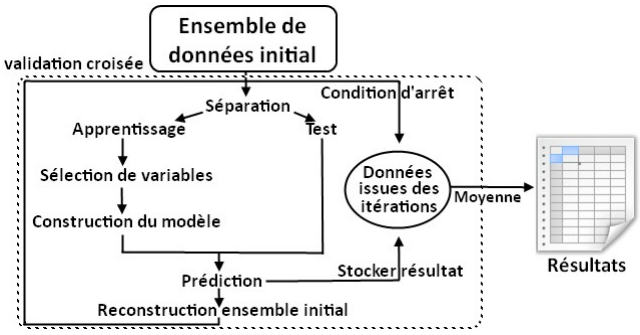


Figure 4.4 – Protocole expérimental dans le cas d'une validation croisée.

(avec $P = P_1 + P_2$). La sélection de variables est ensuite effectuée sur l'ensemble d'apprentissage seul, ce qui nous donne un nouvel ensemble d'apprentissage, restreint aux N_1 gènes sélectionnés. On crée ensuite le modèle de prédiction du classifieur choisi à partir de cette population d'apprentissage restreinte. L'estimation des performances de notre prédicteur est alors faite sur l'ensemble de test que nous avions laissé de côté.
Dans la figure 4.4, on reprend le principe de la figure 4.3, que l'on répète jusqu'à atteindre un critère d'arrêt. Ainsi, l'ensemble initial est séparé, puis recomposé, lorsque la prédiction a été faite. Le résultat final est calculé à partir des résultats intermédiaires ; de manière générale, il correspond à une moyenne.

4.2.5.4 Résultats

Dans notre première expérience, nous avons commencé par suivre le protocole expérimental utilisé dans [Hess 06] et [Natowicz 08], afin d'avoir un point de référence pour notre méthode. Nous avons donc suivi le même protocole que celui décrit dans la figure 4.3, en séparant les 133 cas de patientes en un ensemble d'apprentissage de 82 cas (correspondant aux données de Houston) et en un ensemble de test de 51 cas (correspondant aux données de Villejuif). Chacun de ces ensembles contient 1/3 de patientes classées comme PCR et 2/3 classées No-PCR.

Nous avons alors appliqué la méthode BEUS décrite dans la section 4.2.4.1 sur la fonction objectif f_w (cf équation (4.2.2)). Le tableau 4.2 compare les performances de notre méthode à celles de [Hess 06] et [Natowicz 08]. Ces résultats ont été présentés lors de la conférence ROADEF 2011 [Gardeux 11b].

Les colonnes "BEUS DLDA 31" et "BEUS DLDA 11" représentent les performances des prédicteurs calculés par l'algorithme que nous proposons ; le paramètre w a été fixé afin d'obtenir des

	BEUS DLDA 31	BEUS DLDA 11	t-test DLDA 31	BI Majorité 30
Distance interclasse	5175,45	**3670,45**	1383,30	3210,28
Précision	0,863	**0,882**	0,765	0,863
Sensibilité	0,846	**0,923**	0,923	0,923
Spécificité	0,868	**0,868**	0,711	0,842
VPP	0,688	**0,706**	0,522	0,667
VPN	0,943	**0,971**	0,964	0,970

Tableau 4.2 – Comparaison des performances des différents prédicteurs.

sous-ensembles de 31 et 11 sondes. La colonne "t-test DLDA 31" indique les performances du meilleur prédicteur de [Hess 06], dont les 31 sondes sélectionnées étaient celles de plus faible *p-value* à un t-test et dont les niveaux d'expression avaient été pondérés par des coefficients calculés par DLDA. La colonne "BI Majorité 30" indique les performances du prédicteur calculé dans [Natowicz 08], dont les performances ont été évaluées par vote majoritaire non pondéré. Ce prédicteur avait été obtenu par sélection des 30 sondes bi-informatives de plus grande valuation. La signification des critères de performance (Précision, Sensibilité, Spécificité, VPP et VPN) est détaillée dans la section 3.8.

4.2.5.5 Discussion

Nous pouvons constater que notre méthode de sélection de variables construit des prédicteurs de meilleure qualité que les meilleurs prédicteurs trouvés dans la littérature pour les mêmes données. Notre résultat principal est que ces performances sont obtenues avec trois fois moins de gènes (et deux fois moins que [Natowicz 10]). Par ailleurs, tout comme pour l'algorithme EUS [Gardeux 10], nous avons constaté que la méthode BEUS convergeait rapidement (en quelques itérations).

Lors de l'analyse de nos résultats, nous avons pu remarquer qu'ils étaient parfaitement stables, c'est-à-dire que, pour un w fixé, l'algorithme retourne toujours le même sous-ensemble de gènes. Cette stabilité renforce l'idée que la fonction objectif est unimodale et séparable. Nous avons alors examiné la possibilité de créer un algorithme d'optimisation exact permettant de trouver cet unique optimum. Après calculs, nous avons trouvé que la différence entre les valeurs de la fonction objectif dépend uniquement de la contribution à la distance interclasse de chaque variable prise séparément. Cette contribution peut d'ailleurs être calculée indépendamment des autres variables contenues dans le sous-ensemble.

4.3 Méthode exacte de sélection de variables : δ-test

Les conclusions obtenues lors de l'étude de la méthode BEUS sur les données de Houston et Villejuif nous ont orientés vers une formalisation mathématique du problème.

4.3.1 Principe

Soit S un sous-ensemble de gènes (ou plus généralement de variables) et s un gène tel que $s \notin S$. Nous cherchons alors sous quelle condition l'ajout du gène s à S améliore le résultat de la fonction f_w (décrite dans l'équation (4.2.2)). Pour répondre à cette question, considérons la

différence

$$\delta_w(s) = f_w(S \cup \{s\}) - f_w(S) \tag{4.3.1}$$

La contribution à la distance interclasse amenée par le gène s ne dépend pas de l'ensemble S auquel il est ajouté. De plus, comme l'ajout de ce gène n'augmente la taille de S que d'une unité, la valeur de $\delta_w(s)$ est

$$\delta_w(s) = w \times (\ d(S \cup \{s\}) - d(S)\ ) - (1-w) = w \times \sqrt{(\ \overline{s} - \overline{s}'\ )^2} - (1-w) \tag{4.3.2}$$

où $\overline{s}$ et $\overline{s}'$ sont les valeurs moyennes des expressions du gène s, mesurées respectivement pour les classes PCR et No-PCR. On a alors

$$\delta_w(s) > 0 \text{ ssi } \sqrt{(\ \overline{s} - \overline{s}'\ )^2} > \frac{1-w}{w} \tag{4.3.3}$$

Par conséquent, pour une valeur fixée du paramètre w, l'ensemble de gènes optimal $S^*(w)$ est

$$S^*(w) = \{s \in \mathcal{P} \text{ tel que } \delta(s) > \frac{1-w}{w}\} \tag{4.3.4}$$

où $\delta(s) = \sqrt{(\ \overline{s} - \overline{s}'\ )^2}$.

De plus, comme $\frac{1-w}{w}$ est croissant sur l'intervalle [0,1], les ensembles $S^*(w)$ sont inclusifs en fonction du paramètre w :

$$w < w' \Rightarrow S^*(w) \subseteq S^*(w') \tag{4.3.5}$$

Cette propriété nous permet, au lieu de calculer les ensembles $S^*(w)$, de calculer les ensembles $S(k)$ des k meilleurs gènes par ordre décroissant du critère $\delta(s)$. C'est en fait le principe des méthodes par filtre, telles que les méthodes de t-test (cf équation (3.7.2)) ou SNR (cf équation (3.7.4)) présentées dans la section 3.7.2. Nous avons appelé ce nouveau critère de sélection de variables δ**-test** [Gardeux 11c]. Il est défini comme suit :

$$\delta(x) = |\bar{x}_1 - \bar{x}_2| \tag{4.3.6}$$

avec $\bar{x}_1$ et $\bar{x}_2$ définis de la même manière que dans la section 3.7.2.

L'intérêt de notre approche est double. En effet, dans un premier temps, nous avons donné une explication rigoureuse de l'utilisation de notre critère de sélection de variables : il correspond à l'optimisation de la fonction bi-objectif f_w. C'est un résultat notable, car nous ne disposons pas toujours de telles preuves pour les critères définis dans la littérature. De plus, au contraire des autres critères, le δ-test ne prend pas en compte la variance ou l'écart-type comme caractéristique discriminante.

Notre méthode peut être testée via cette *applet* en ligne : `http://gardeux-vincent.eu/DeltaTest.php`.

4.3.2 Application sur les données Houston et Villejuif

4.3.2.1 Premières constatations

Pour vérifier que notre nouvelle méthode de sélection par filtre obtenait les mêmes résultats que ceux obtenus par l'algorithme BEUS, nous l'avons appliquée sur le même protocole expérimental. Les résultats sont concordants avec ceux consignés dans le tableau 4.2, mais en

un temps considérablement plus faible (de l'ordre de la seconde, au lieu de la minute). Il est particulièrement intéressant de comparer nos résultats à ceux de [Hess 06], car les auteurs ont conçu leurs prédicteurs en sélectionnant les k gènes ayant les plus petites *p-value* à un t-test. On remarque que ce critère obtient des résultats de bien moins bonne qualité que les nôtres.

Contrairement aux autres critères statistiques (comme le t-test), notre méthode se focalise sur la contribution de chaque variable à la distance interclasse. Il serait alors intéressant de comprendre pourquoi notre méthode obtient de meilleurs résultats. Une première supposition est que notre critère δ-test ne fait aucune hypothèse sur la distribution statistique des niveaux d'expression (il est **non paramétrique**), contrairement au t-test. En particulier, le critère du t-test est soumis à l'hypothèse que la distribution des valeurs est normale, ce qui n'est pas le cas dans la plupart des données réelles.

Il est important de noter que, même si la méthode δ-test de sélection de variables est exacte, le processus global de prédiction, prenant également en compte la création du modèle de prédiction par le classifieur, ne l'est pas. Le sous-ensemble de sondes sélectionné par la méthode δ-test est optimal vis-à-vis de la fonction bi-objectif que nous avons fixée, mais ne l'est pas forcément pour le problème de prédiction posé, pour le classifieur utilisé, ou bien pour les ensembles d'apprentissage et de tests utilisés.

4.3.2.2 Importance de la normalisation

Une étude supplémentaire a été conduite, afin de vérifier l'importance de la normalisation dans notre test. Nous avons utilisé le même protocole expérimental deux fois, avec et sans normalisation des données d'apprentissage. Nous avons fixé différents nombres de gènes à sélectionner, et nous avons obtenu les résultats consignés dans le tableau 4.3.

Nb Sondes	1	2	3	9	10	11	12	20	30	40	50
Données d'apprentissage normalisées											
Précision	0,745	0,706	0,725	0,843	0,863	0,882	0,882	0,863	0,863	0,843	0,843
Sensibilité	0,923	0,769	0,769	0,769	0,846	0,923	0,923	0,923	0,846	0,769	0,769
Spécificité	0,684	0,684	0,711	0,868	0,868	0,868	0,868	0,842	0,868	0,868	0,868
VPP	0,500	0,455	0,476	0,667	0,688	0,706	0,706	0,667	0,688	0,667	0,667
VPN	0,963	0,897	0,900	0,917	0,943	0,971	0,971	0,969	0,943	0,917	0,917
Données d'apprentissage non normalisées											
Précision	0,667	0,667	0,824	0,804	0,784	0,784	0,784	0,824	0,843	0,824	0,863
Sensibilité	0,769	0,769	0,846	0,846	0,769	0,769	0,769	0,769	0,769	0,692	0,769
Spécificité	0,632	0,632	0,816	0,789	0,789	0,789	0,789	0,842	0,868	0,868	0,895
VPP	0,417	0,417	0,611	0,579	0,556	0,556	0,556	0,625	0,667	0,643	0,714
VPN	0,889	0,889	0,939	0,938	0,909	0,909	0,909	0,914	0,917	0,892	0,919

Tableau 4.3 – Performances obtenues par la méthode δ-test + DLDA. Ensemble d'apprentissage : Houston. Ensemble de test : Villejuif.

Nous remarquons immédiatement que la normalisation de la population d'apprentissage avant application du critère δ-test permet d'améliorer nettement les résultats. Les gènes sélectionnés ne sont pas les mêmes, bien que l'intersection ne soit pas vide.
Nous avons ensuite vérifié l'impact de la normalisation sur le critère du t-test de Welsh. Nous obtenons les résultats consignés dans le tableau 4.4.

Nb Sondes	1	2	3	9	10	11	12	20	30	40	50
Données d'apprentissage normalisées											
Précision	0,627	0,627	0,647	0,765	0,745	0,765	0,765	0,765	0,745	0,784	0,784
Sensibilité	1,000	0,846	0,769	0,692	0,769	0,769	0,769	0,923	0,923	0,923	0,923
Spécificité	0,500	0,552	0,605	0,789	0,737	0,763	0,763	0,711	0,684	0,737	0,737
VPP	0,406	0,393	0,400	0,529	0,500	0,526	0,526	0,522	0,500	0,545	0,545
VPN	1,000	0,913	0,885	0,882	0,903	0,906	0,906	0,964	0,963	0,966	0,966
Données d'apprentissage non normalisées											
Précision	0,627	0,627	0,647	0,706	0,706	0,745	0,725	0,745	0,765	0,784	0,804
Sensibilité	1,000	0,769	0,769	1,000	1,000	1,000	1,000	1,000	0,923	0,846	0,846
Spécificité	0,500	0,579	0,605	0,605	0,605	0,658	0,631	0,658	0,711	0,763	0,789
VPP	0,406	0,385	0,400	0,464	0,464	0,500	0,481	0,500	0,522	0,550	0,579
VPN	1,000	0,880	0,885	1,000	1,000	1,000	1,000	1,000	0,964	0,935	0,938

Tableau 4.4 – Performances obtenues par la méthode t-test de Welsh + DLDA. Ensemble d'apprentissage : Houston. Ensemble de test : Villejuif.

Tout d'abord, nous pouvons constater que nous retrouvons les résultats obtenus dans [Hess 06], sur les données non normalisées. La normalisation n'améliore pas énormément les résultats, elle les détériore même légèrement pour les cas de 30 et 50 gènes. De manière générale, on peut dire qu'elle les améliore tout de même, mais de manière beaucoup moins évidente que pour le critère du δ-test. Notre critère obtient en effet des résultats très nettement meilleurs lorsque la normalisation est effectuée, quel que soit le nombre de gènes sélectionnés dans le prédicteur.

4.3.2.3 Validation croisée

Afin de vérifier la robustesse de notre méthode (sélection d'un ensemble de sondes, puis prédiction par DLDA), nous l'avons appliquée sur l'ensemble des 133 patientes (données de Houston et de Villejuif fusionnées). Nous avons ensuite appliqué la méthode de validation croisée appelée *Repeated random subsampling* (vue dans la section 3.9.3), en respectant le protocole décrit dans la figure 4.4. Nous séparons donc l'ensemble des 133 patientes aléatoirement en 2/3 pour former l'ensemble d'apprentissage et 1/3 pour former l'ensemble de test. Puis, nous sélectionnons les variables et construisons le modèle du DLDA sur l'ensemble d'apprentissage et estimons les performances de notre prédicteur sur les données de test. Ce processus est répété 1000 fois et les résultats finaux sont calculés par moyenne sur les 1000 tests. Ils sont consignés dans le tableau 4.5.

Nb Probes	1	2	3	9	10	11	12	20	30	40	50
Précision	0,634	0,672	0,703	0,723	0,735	0,745	0,752	0,760	0,749	0,757	0,764
Sensibilité	0,809	0,795	0,798	0,791	0,797	0,777	0,768	0,795	0,746	0,735	0,729
Spécificité	0,575	0,629	0,670	0,700	0,715	0,734	0,746	0,749	0,751	0,765	0,775
VPP	0,391	0,425	0,456	0,479	0,485	0,499	0,516	0,518	0,501	0,528	0,520
VPN	0,899	0,899	0,905	0,906	0,913	0,906	0,901	0,915	0,898	0,890	0,896

Tableau 4.5 – Performances obtenues par validation croisée "*Repeated random subsampling*" sur les 133 patientes, avec la méthode δ-test.

On peut constater que les résultats sont de moins bonne qualité que ceux obtenus précédemment, avec des ensembles bien séparés dès le début. Ce comportement est tout à fait normal, car le processus de validation croisée peut engendrer des ensembles d'apprentissage et de test très hétérogènes en fonction des itérations. Notre méthode conserve tout de même un bon ratio de prédictions correctes. Cependant, pour améliorer les résultats, il serait intéressant d'adapter le nombre de gènes à sélectionner en fonction de la population d'apprentissage.

Nous avons également calculé les résultats du t-test de Welsh dans les mêmes conditions expérimentales, afin de les comparer aux nôtres. Ces résultats sont consignés dans le tableau 4.6.

Nb Probes	1	2	3	9	10	11	12	20	30	40	50
Précision	0,615	0,662	0,675	0,698	0,705	0,710	0,719	0,721	0,727	0,729	0,731
Sensibilité	0,821	0,780	0,791	0,771	0,791	0,787	0,788	0,791	0,786	0,767	0,788
Spécificité	0,544	0,622	0,635	0,673	0,676	0,684	0,695	0,697	0,708	0,716	0,711
VPP	0,383	0,408	0,427	0,448	0,449	0,459	0,476	0,474	0,469	0,468	0,483
VPN	0,898	0,894	0,898	0,895	0,907	0,905	0,903	0,906	0,909	0,904	0,907

Tableau 4.6 – Performances obtenues par validation croisée "*Repeated random subsampling*" sur les 133 patientes, avec la méthode du t-test de Welsh.

On constate que les précisions sont meilleures avec notre méthode, quel que soit le nombre de gènes sélectionnés. Ce résultat est d'autant plus fort qu'il correspond à une valeur moyenne sur 1000 essais, ce qui prouve sa robustesse.

4.3.2.4 Estimation de notre prédicteur à 11 gènes.

Un résultat qui peut s'avérer intéressant pour la création d'une routine clinique serait l'efficacité des gènes sélectionnés sur ce problème particulier (prédiction de la réponse chimiothérapique). Afin de vérifier l'efficacité des 11 gènes que nous avons sélectionnés (sur l'ensemble d'apprentissage de Houston seul), nous avons réalisé une validation croisée sur les 133 patientes que nous avions (ensembles de données de Houston et Villejuif), mais sans répéter le processus de sélection de variables à chaque itération. C'est-à-dire que nous avons seulement utilisé les 11 gènes du prédicteur créé lors des expériences précédentes. Nous avons utilisé la même procédure de validation croisée que dans la partie précédente (*repeated random subsampling*). Les résultats ont été résumés dans le tableau 4.7.

Précision	0,821	± 0,046
Sensibilité	0,850	± 0,095
Spécificité	0,811	± 0,061
VPP	0,606	± 0,090
VPN	0,940	± 0,034

Table 4.7 – Validation croisée sur les 133 patientes en utilisant le prédicteur à 11 gènes trouvé sur Houston seul.

Les résultats sont plutôt bons pour une validation croisée, surtout après avoir choisi aléatoirement les ensembles d'apprentissage et de test parmi les 133 patientes. Nous pouvons donc dire que notre prédicteur de 11 gènes est robuste, et que les résultats ne varient pas, quelle que soit la construction du classifieur DLDA.

4.3.3 Discussion

Afin de vérifier la qualité des sondes sélectionnées par la méthode δ-test, nous avons représenté les 30 premières dans le tableau 4.8.

Gène	Sonde	δ-test	*p-value* t-test	[Hess 06]	[Natowicz 08]
ESR1	205225_at	0,102	5,261E-6	-	-
BTG3	213134_x_at	0,090	2,956E-5	12	1
BTG3	205548_s_at	0,088	3,307E-5	15	2
MELK	204825_at	0,083	1,224E-4	14	16
METRN	219051_x_at	0,076	1,705E-6	22	11
GAMT	205354_at	0,075	2,768E-7	18	-
MAPT	203929_s_at	0,074	2,312E-8	1	-
AGR2	209173_at	0,073	5,451E-7	-	-
SOX11	204913_s_at	0,073	0,008	-	-
AI348094	212956_at	0,072	2,037E-5	-	-
SCUBE2	219197_s_at	0,071	4,736E-5	13	-
MLPH	218211_s_at	0,070	3,663E-5	-	-
IGFBP4	201508_at	0,069	6,888E-7	30	-
GATA3	209604_s_at	0,066	2,001E-4	-	3
CAI2	214164_x_at	0,064	4,704E-5	-	18
CALML5	220414_at	0,063	0,012	-	-
RARRES1	221872_at	0,062	0,053	-	-
CAI2	215867_x_at	0,061	4,418E-5	-	17
DNAJC12	218976_at	0,060	1,436E-5	-	-
THRAP2	212207_at	0,059	2,563E-8	5	5
RARRES1	206391_at	0,059	0,042	-	-
ELF5	220625_s_at	0,058	0,006	-	-
KRT7	209016_s_at	0,057	0,002	-	8
GABRP	205044_at	0,056	0,013	-	-
IL6ST	212195_at	0,055	1,902E-4	-	-
RARRES1	206392_s_at	0,055	0,072	-	-
MAPT	203930_s_at	0,055	2,201E-8	2	-
CXCR4	211919_s_at	0,054	0,002	-	-
SKP2	203625_x_at	0,054	0,002	-	-
CHI3L2	213060_s_at	0,053	0,008	-	-

Tableau 4.8 – 30 meilleurs gènes de la sélection, triés suivant le critère δ-test.

Dans la 1ère colonne est renseigné le nom du gène (en suivant la nomenclature HUGO) correspondant à la référence de la sonde Affymetrix qui figure dans la 2ème colonne. La valeur obtenue par le gène sur le critère δ-test est renseignée dans la 3ème colonne. La 4ème colonne donne les valeurs de la *p-value* du t-test utilisé dans [Hess 06]. Les deux dernières colonnes se réfèrent aux classements effectués dans [Hess 06] et [Natowicz 08], elles nous permettent de voir rapidement les sondes en commun, ainsi que leurs importances dans ces classifications.

Nous pouvons constater que notre test met en évidence certains gènes considérés comme importants dans la littérature : ESR1 (classé premier), MAPT (classé septième) et BTG3 (classé second et troisième, pour deux *probes* différents). Ces gènes sont en effet connus par leur redondance d'apparition, ou plus précisément par la connaissance de leur rôle biologique, et notre méthode les classe dans les premiers rangs. On remarque également qu'à part le gène RARRES1 (sélectionné deux fois), tous les autres gènes ont une p-value au t-test inférieure à 5%, ce qui montre leur pertinence.

La comparaison avec les méthodes de [Hess 06] et [Natowicz 08] nous permet de constater une intersection non vide de 14 sondes parmi 30 (et 11 gènes). C'est un fait assez rare pour être relevé.

4.4 Sélection automatique du nombre de variables

4.4.1 Étude des performances en fonction du nombre de variables

Un des principaux inconvénients de notre méthode (ainsi que des méthodes utilisées dans [Hess 06] et [Natowicz 08]), et des méthodes par filtre en général, est que l'on ne sait pas à l'avance le nombre optimal de gènes à sélectionner. Ce comportement est problématique, puisque les résultats utilisent alors l'information de l'ensemble de test pour valider le nombre de gènes à utiliser, ce qui biaise les résultats. Pour pallier ce problème, il faudrait pouvoir adapter le nombre de gènes à sélectionner en fonction de la population d'apprentissage seule.

Nous avons donc réalisé une expérience qui sélectionne k variables ($k \in [1,50]$) puis construit le modèle du classifieur DLDA sur l'ensemble d'apprentissage. On estime ensuite les performances du prédicteur, non pas sur la population de validation, mais sur la population d'apprentissage. Nous pouvons alors étudier les variations de prédiction de la méthode DLDA sur la population d'apprentissage, sans toucher à l'ensemble des données de test. Les résultats ont été représentés dans la figure 4.5.

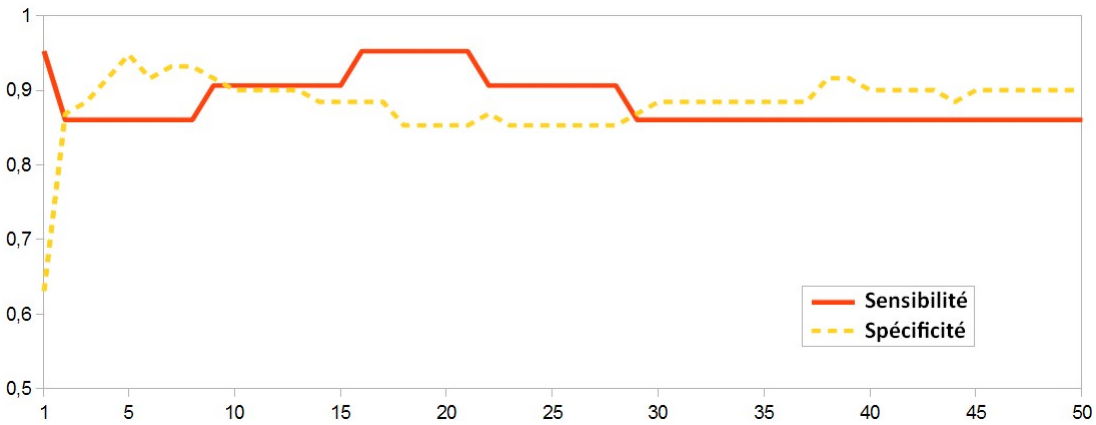


Figure 4.5 – Graphes des variations de la spécificité et de la sensibilité en fonction du nombre de gènes sélectionnés. Ensemble d'apprentissage : Houston. Ensemble de test : Houston.

L'étude des variations des courbes de sensibilité et de spécificité nous permet de repérer quatre phases :

1. Pour quelques gènes, on a un grand écart entre la spécificité et la sensibilité, ce qui montre que le classifieur ne dispose pas de suffisamment d'informations pour correctement prédire les exemples.
2. De 3 à 9 gènes, la spécificité est supérieure à la sensibilité. Cette phase ne nous intéresse pas spécialement, puisque la sensibilité a un rôle plus important que la spécificité dans notre étude.
3. De 10 à 28 gènes, la sensibilité est élevée et supérieure à la spécificité.
4. Au-dessus de 28 gènes, la sensibilité se dégrade en faveur de la spécificité. Cette dernière redevient supérieure à la sensibilité, qui se stabilise.

Ensuite, nous avons fait la même expérience, mais en estimant les performances de DLDA sur les données de l'ensemble de test (ici fixé aux données de Villejuif). Les résultats sont représentés dans la figure 4.6.

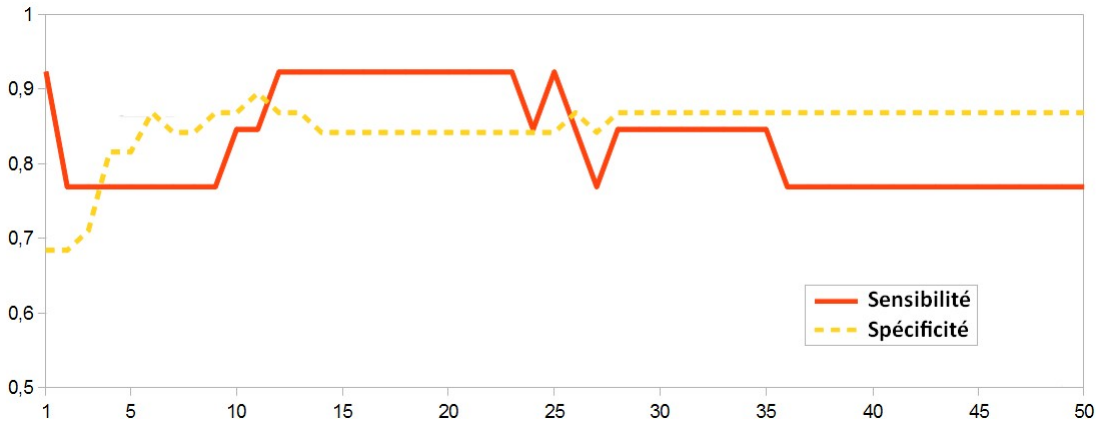


Figure 4.6 – Graphes des variations de la spécificité et de la sensibilité en fonction du nombre de gènes sélectionnés. Ensemble d'apprentissage : Houston. Ensemble de test : Villejuif.

On remarque que les courbes des deux figures 4.5 et 4.6 sont sujettes aux mêmes variations. On retrouve les quatre phases énoncées précédemment, avec de très légères variations sur les intervalles du nombre de gènes sélectionnés. Ces résultats sont particulièrement intéressants, car ils montrent qu'il est possible de créer une procédure sélectionnant automatiquement un bon nombre de gènes, en se basant uniquement sur les données d'apprentissage. Ce nombre de gènes devra être sélectionné afin de maximiser la précision et la sensibilité, comme c'est le cas dans la troisième phase que nous avons repérée.

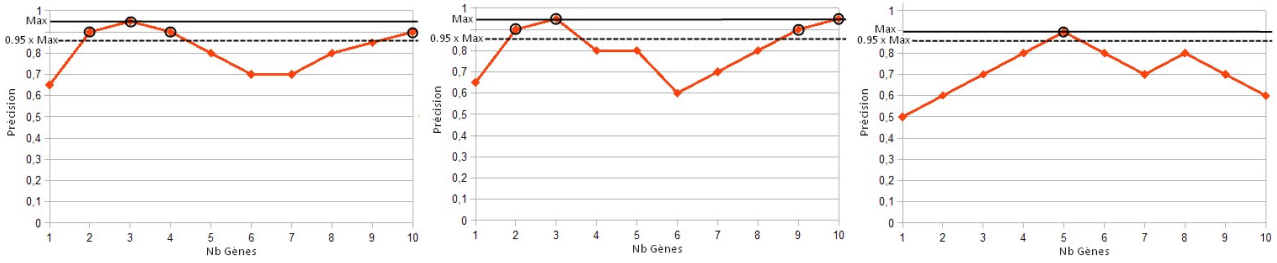
4.4.2 Principe de la méthode

Notre méthode δ-test retourne une liste de gènes triés par valeur δ décroissante. Nous désirons calculer le nombre optimal de gènes k^* à sélectionner. Nous allons alors observer les variations des performances de prédiction du classifieur en fonction du nombre de variables, sur l'**ensemble d'apprentissage**. L'étude de ces variations nous permettra de sélectionner le point qui nous semble le plus intéressant, c'est-à-dire qui corresponde à une valeur élevée de

précision de la prédiction. Nous essayerons également d'éviter de choisir des points isolés (c'est-à-dire pour lesquels les performances du classifieur augmentent soudainement, mais chutent directement après), car nous les jugeons non stables. Dans la mesure du possible, nous nous attacherons à repérer des zones où les performances du classifieur restent stables (on appellera ces zones des plateaux).

À cet effet, nous construisons successivement les modèles de prédiction du classifieur en utilisant les $k \in [1,k^+]$ premiers gènes de la liste calculée à partir de l'ensemble d'apprentissage, et nous estimons les performances de ces classifieurs sur ce même ensemble d'apprentissage. On évitera de prendre k^+ trop grand, puisque l'on cherche à minimiser le nombre de variables. Dans notre étude, nous fixons $k^+ = 50$. Nous obtenons alors une liste de k^+ valeurs de précision calculées pour chacun des sous-ensembles de gènes : {{gène$_1$}, {gène$_1$, gène$_2$}, {gène$_1$, gène$_2$, gène$_3$}, ..., {gène$_1$, gène$_2$, ..., gène$_{k^+}$}}.
Pour sélectionner le point le plus intéressant parmi les k^+ valeurs de précision, nous cherchons la valeur maximale **non singulière**. Nous commençons donc par chercher un "plateau" de trois valeurs successives $[k-1,k,k+1]$ qui soit "proche" du maximum, c'est-à-dire dont les écarts sont inférieurs à 5%. Si un tel plateau existe, nous considérons alors la solution k comme performante et non singulière et prenons $k^* = k$ (le plus petit k si de multiples plateaux existent). Si aucun plateau n'existe parmi les k^+ valeurs, nous cherchons une paire $[k-1,k]$ dont les deux valeurs successives soient "proches" du maximum. Si une telle paire existe, nous prenons $k^* = k$ (le plus petit k, si de multiples paires existent).
Dans le cas où il n'existe pas de plateaux, ni de paires, nous sommes contraints à prendre k comme le plus petit indice ayant une précision maximale, même si cette solution est singulière.

Pour illustrer ce principe, on peut considérer que l'on "coupe" la courbe des variations des précisions à 95% de la valeur maximale, et que l'on étudie les valeurs restantes. La figure 4.7 représente les trois cas possibles pouvant se présenter.



(a) Cas d'un plateau (b) Cas d'une paire (c) Cas d'une valeur singulière

Figure 4.7 – Les différents cas possibles de la procédure *SelectKStar*.

Ces courbes ont été représentées pour $k^+ = 10$. Dans la figure 4.7(a), on voit le cas où l'on a un plateau de trois valeurs successives "proches" du maximum ; on a alors $k^* = 3$, car le point singulier ne nous intéresse pas. La figure 4.7(b) représente un cas où l'on ne trouve pas de plateau, mais deux paires ; la valeur optimale est alors prise à $k^* = 3$, la deuxième paire correspondant à des valeurs de k supérieures. Le dernier cas est représenté dans la figure 4.7(c), dans lequel on ne trouve ni plateau, ni paire, parmi les valeurs comprises dans l'intervalle $[0.95 \times max, max]$; on doit alors choisir la valeur singulière correspondant au plus petit k, et on obtient $k^* = 5$.

La procédure résultante est appelée *SelectKStar* [Gardeux 11c], son pseudo-code est décrit dans la figure 4.8.

Fonction SelectKStar
début
lancer k fois la méthode DLDA, en sélectionnant à chaque fois les k premiers gènes retournés par le Delta-test, et créer un vecteur *tab* contenant les précisions obtenues.
calculer la valeur max(précision), maximum de la précision dans *tab*.
générer *indexes*, un vecteur contenant tous les indices de *tab* tels que $tab(i)$(précision) $>= max$(précision) $* 0.95$.
si un ou plusieurs triplets $[k-1,k,k+1]$ existe dans *indexes* **alors**
retourner le plus petit k
sinon si une ou plusieurs paires $[k-1,k]$ existe **alors**
retourner le plus petit k
sinon (il n'y a que des valeurs singulières)
retourner l'indice de la valeur maximale
fin si
fin

Figure 4.8 – Pseudo-code de la méthode *SelectKStar*, utilisée pour sélectionner automatiquement le nombre optimal de gènes k^* à partir de l'ensemble d'apprentissage.

Nous avons ensuite intégré la procédure *SelectKStar* à la méthode par filtre δ-test développée précédemment. L'algorithme résultant a été appelé *Automated* δ-test.
Pour chaque couple (ensemble d'apprentissage, ensemble de test) de patientes, nous :
- classons les gènes par valeurs δ décroissantes, après normalisation de l'ensemble d'apprentissage ;
- choisissons k^* avec la méthode *SelectKStar* ; les variables sélectionnées sont alors les k^* premiers gènes renvoyés par la méthode δ-test ;
- estimons les performances du prédicteur, en construisant le modèle sur l'ensemble d'apprentissage (non normalisé), et en validant sur l'ensemble de test.

4.4.3 Résultats

Nous avons appliqué cette procédure aux 133 patientes des jeux de données de Houston et de Villejuif avec une validation croisée de type *repeated random subsampling*, en suivant le même protocole que dans la section 4.3.2.3. Les résultats sont consignés dans le tableau 4.9.

Ce tableau montre qu'en moyenne le nombre de gènes sélectionnés est assez faible, et cohérent avec les résultats obtenus précédemment (cf section 4.3.2.3). Les résultats sont du même ordre que ceux obtenus sans sélection automatique. L'algorithme *SelectKStar* s'adapte donc correctement à l'ensemble d'apprentissage pour fournir les gènes les plus pertinents.

4.4.4 Normalisation

Avant de terminer cette partie, il convient de clarifier le rôle de la normalisation dans notre méthode. En effet, la méthode de normalisation proposée est très importante lorsque l'on dispose de données brutes, dont les valeurs ne sont pas déjà normalisées sur une même échelle. C'est

Nb Gènes	12,65	± 7,422
Précision	0,754	± 0,007
Sensibilité	0,739	± 0,018
Spécificité	0,745	± 0,011
VPP	0,471	± 0,014
VPN	0,903	± 0,006

Tableau 4.9 – Validation croisée sur les 133 patientes, en utilisant la méthode *Automated* δ-test.

le cas pour les données de Houston et Villejuif, mais pas pour tous les autres jeux de données que nous traiterons par la suite. Pour ces jeux de données déjà normalisés, une normalisation supplémentaire n'améliore que de très peu les résultats (de l'ordre de 1% de précision supplémentaire en moyenne). Cette phase de normalisation peut donc être toujours appliquée sans risque de détérioration des résultats, mais elle n'est vraiment utile que lorsque les données ne sont pas déjà sur une même échelle.

D'ailleurs, il est important de noter que, sur certains jeux de données, si la normalisation a déjà été effectuée, elle peut biaiser les résultats (cf section 4.2.5.3), vu qu'elle se sert de l'ensemble du jeu de données pour effectuer cette normalisation.

Nous souhaitons donc proposer une méthode destinée aux jeux de données non normalisés, afin d'améliorer les résultats de prédiction, tout en conservant une rigueur permettant d'éviter tout biais de sélection. Le but est de normaliser les données de test au fur et à mesure qu'elles sont présentées, en suivant le même protocole que pour les données d'apprentissage. Or, dans le cas pratique, les données de test peuvent ne pas entrer dans les échelles trouvées sur les données d'apprentissage. Pour pallier au problème, nous proposons la méthode suivante :

- normaliser les données d'apprentissage (et conserver les valeurs minimales et maximales de chaque variable) ;
- faire la sélection de variables et la création du modèle de prédiction sur ces données ;
- lorsqu'un nouvel exemple doit être prédit, le normaliser en accord avec les échelles trouvées sur la population d'apprentissage. Si une variable d'un nouvel exemple n'entre pas dans une échelle (par exemple la $i^{\text{ème}}$ variable est supérieure à la valeur maximale trouvée dans la population d'apprentissage), nous avons deux possibilités pour pallier au problème :
  1. assimiler une valeur inférieure au minimum comme étant égale au minimum (et respectivement pour le maximum) ;
  2. modifier l'échelle en prenant cette nouvelle valeur comme référence minimale (respectivement maximale), et donc normaliser à nouveau l'ensemble d'apprentissage pour cette variable.

La deuxième méthode semblant plus rigoureuse, nous l'avons appliquée en conservant le même protocole que dans la section 4.4.3. Les résultats sont disponibles dans le tableau 4.10.

On remarque que ce principe de normalisation améliore significativement les résultats, notamment au niveau de la précision et du nombre moyen de gènes sélectionnés. Ce type de méthode peut s'avérer particulièrement efficace sur des applications réelles. En effet, nous disposons en principe d'un prédicteur généré à partir d'un ensemble d'apprentissage, mais les données des patientes à prédire arrivent au "compte-gouttes". Ainsi, si ces nouvelles données ne sont pas déjà normalisées, il faut pouvoir les mettre sur la même échelle que l'ensemble d'apprentissage ayant servi à construire le prédicteur, ce que permet notre méthode.

Nb Gènes	7,039	± 5,263
Précision	0,765	± 0,007
Sensibilité	0,707	± 0,015
Spécificité	0,784	± 0,008
VPP	0,517	± 0,016
VPN	0,891	± 0,008

Tableau 4.10 – Validation croisée sur les 133 patientes, avec la méthode *Automated* δ-test, utilisée conjointement avec la méthode de normalisation globale.

4.5 Applications sur différents jeux de données

4.5.1 Création d'un *benchmark*

Afin de comparer notre méthode de sélection de variables à d'autres méthodes de la littérature, nous avons sélectionné 6 jeux de données publiques, afin de créer un *benchmark*. Celui-ci est composé uniquement de données à deux classes. Les caractéristiques des jeux de données sont regroupées dans le tableau 4.11.

Données	Références	Ex.	Sondes	Lien de téléchargement
Colon	[Alon 99]	62	2000	`http://genomics-pubs.princeton.edu/oncology/`
Lymphome	[Shipp 02]	77	5469	`http://www.gems-system.org/`
Leucémie	[Golub 99]	72	7129	`http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi`
Prostate	[Singh 02]	102	10509	`http://www.gems-system.org/`
Cerveau	[Pomeroy 02]	60	7129	`http://www.broadinstitute.org/cgi-bin/cancer/datasets.cgi`
Ovaires	[Berchuck 05]	54	22283	`http://data.cgt.duke.edu/clinicalcancerresearch.php`

Tableau 4.11 – Description des différents jeux de données du *benchmark*.

Pour chaque jeu de données, le but est de prédire la classe de nouveaux exemples. Ces classes sont très diverses :

- **Colon** : Les 62 exemples concernent le cancer du colon, ils sont constitués de 40 tissus tumoraux et 22 tissus sains.
- **Lymphome** : Les 77 exemples représentent deux types de tumeurs lymphatiques : 58 sont des cas de lymphomes diffus à grandes cellules B (DLBCL) et 19 sont des lymphomes folliculaires (FL).
- **Leucémie** : Ces données proviennent de l'article de Golub qui fait référence dans le domaine, puisqu'il a été le premier à démontrer la possibilité d'un diagnostic à partir de données de puces à ADN. Les 72 exemples représentent des échantillons de tissus atteints de deux types de leucémie : 47 sont du type leucémie lymphoblastique aiguë (ALL) et les 25 autres sont du type leucémie myéloïde aiguë (AML).
- **Prostate** : Les 102 exemples proviennent d'une étude sur le cancer de la prostate : 50 sont des tissus tumoraux et 52 sont des tissus sains.
- **Cerveau** : Les 60 exemples proviennent d'échantillons obtenus sur des patients traités pour des médulloblastomes : pour 39 d'entre eux le traitement a été efficace et ils ont donc survécu, alors que pour les 21 autres le traitement n'a pas été efficace. Le but est donc le même que pour l'étude sur le cancer du sein que nous avions faite précédemment.

– **Ovaires** : Ces 54 exemples sont des cas de cancer ovarien en stage III/IV, parmi lesquels 30 patientes ont survécu pendant une durée inférieure à 3 ans (*short-term survivors*) tandis que 24 patientes ont survécu plus de 7 ans (*long-term survivors*).

Si réunir les données pour former le *benchmark* est une chose aisée, la comparaison avec des méthodes de la littérature n'est pas simple, car les protocoles expérimentaux utilisés sont tous différents. Le plus difficile étant d'obtenir des résultats non biaisés (cf section 4.2.5.3). Par la suite, nous avons donc choisi des références bibliographiques utilisant un protocole expérimental permettant d'éviter le biais de sélection, même si la méthode de validation croisée n'est pas toujours strictement identique.

4.5.2 Résultats de notre méthode *Automated* δ-test sur ce *benchmark*

Nous avons appliqué notre méthode *Automated* δ-test sur le *benchmark* des 6 jeux de données que nous venons de définir. Or, le protocole expérimental défini précédemment n'est pas le plus rigoureux pour comparer nos résultats. En effet, la composante aléatoire de la méthode *repeated random subsampling* peut donner des résultats assez variables d'un essai à un autre. Ainsi, si l'on ne répète pas l'opération un très grand nombre de fois (par exemple 50 fois), comme c'est le cas dans de nombreux papiers de la littérature, il est possible d'avoir des écarts assez importants entre différents tests.

Pour pallier au problème, nous avons choisi d'utiliser une autre méthode de validation croisée, non aléatoire, afin d'obtenir un résultat comparatif plus stable : la validation croisée *leave one out*, qui a été présentée dans la section 3.9.2. Cette méthode est d'ailleurs devenue une technique de validation de référence dans le domaine des puces à ADN [Inza 02] [Bø 02]. Pour que les résultats soient rigoureux, nous avons également appliqué le principe de la figure 4.3 visant à éliminer le biais de sélection. La méthode résultante est détaillée dans la figure 4.9.

(On a un jeu de données $E = E_1, ..., E_n$ contenant n échantillons)
pour i = 1 à n **faire**
    On retire E_i de l'ensemble E, et on pose E comme ensemble d'apprentissage et E_i comme donnée de test
    On effectue la sélection de variables sur E
    On construit le modèle du classifieur à partir de E
    On prédit le comportement de E_i avec notre classifieur
    On reforme l'ensemble E initial en lui ajoutant E_i
fin pour
(On obtient un "tableau de confusion", qui nous permet d'estimer les performances de notre méthode)

Figure 4.9 – Méthode non biaisée de validation croisée *leave one out*.

Nous avons ensuite appliqué notre méthode *Automated* δ-test à notre *benchmark*, en utilisant ce protocole. Nous obtenons les résultats du tableau 4.12.

Nous remarquons que les précisions sont assez variables d'un jeu de données à un autre. Il semblerait donc que certains comportements soient plus simples à prédire, ou bien que le lien

Données	Colon	Lymphome	Leucémie	Prostate	Cerveau	Ovaires
Nb gènes sélectionnés	9,048	4,156	2,972	4,000	8,267	15,981
Précision	0,855	0,883	0,986	0,941	0,683	0,759
Sensibilité	0,925	1,000	0,960	0,960	0,619	0,667
Spécificité	0,727	0,845	1,000	0,923	0,718	0,833
PPV	0,860	0,679	1,000	0,923	0,542	0,762
NPV	0,842	1,000	0,979	0,960	0,778	0,758

Tableau 4.12 – Validation croisée *leave one out*, en utilisant la méthode *automated Delta-test*.

à la génomique soit plus fort. Une autre possibilité, malheureusement difficilement visible, est la rigueur avec laquelle les tests biologiques ont été effectués, et qui ont abouti aux données fournies. Une composante importante, qui apparaît clairement dans nos résultats, est le très faible nombre de gènes sélectionnés, pour former le prédicteur.

4.5.3 Comparaison avec les résultats de la littérature

Nous avons ensuite comparé nos résultats à ceux disponibles dans la littérature sur les mêmes jeux de données. Comme nous l'avons déjà dit, il est très difficile de trouver des résultats utilisant les mêmes protocoles expérimentaux, ou bien évitant le biais de sélection. De plus, les résultats correspondent souvent à la seule valeur de précision du classifieur et manquent donc de détails sur d'autres données de comparaison pourtant très importantes, telles que le nombre de variables, la sensibilité ou bien la spécificité. Nous avons regroupé ces différents résultats dans le tableau 4.13.

Données	Colon		Leucémie		Prostate		Cerveau		Lymphome		Ovaires	
	Préc.	Nb	Préc.	Nb	Préc.	Nb	Préc.	Nb	Préc.	Nb	Préc.	Nb
[Singh 02]	-	-	-	-	86,00	29	-	-	-	-	-	-
[Ramaswamy 02]	-	-	-	-	-	-	60,00	21	-	-	-	-
[Weston 03]	85,83	20	-	-	-	-	-	-	-	-	-	-
[Deutsch 03]	-	-	-	-	-	-	-	-	83,33	6	-	-
[Rakotomamonjy 03]	82,33	20	-	-	-	-	-	-	-	-	-	-
[Pochet 04]	82,03	-	94,40	-	91,22	-	-	-	-	-	-	-
[Shah 07]	-	-	-	-	94,12	22	-	-	-	-	-	-
[Orsenigo 08]	85,71	30	-	-	94,11	20	-	-	-	-	-	-
[Ghattas 08]												
F-test	84,05	15,1	-	-	91,18	126,4	-	-	-	-	-	-
∂W	76,70	35,1	-	-	94,60	756,6	-	-	-	-	-	-
∂RW	78,60	43,3	-	-	94,70	573,3	-	-	-	-	-	-
∂Spb	80,30	31,8	-	-	94,80	95,5	-	-	-	-	-	-
SVM-RFE	85,48	26,4	-	-	94,18	43,2	-	-	-	-	-	-
GLMPath	81,91	1,3	-	-	94,09	1,6	-	-	-	-	-	-
Forêts Aléatoires	89,40	49,8	-	-	94,10	81	-	-	-	-	-	-
Automated δ-test	85,50	9,05	98,60	2,97	94,10	4,00	68,30	8,27	88,30	4,16	75,90	15,98

Tableau 4.13 – Performances (en %) de différentes méthodes de la littérature sur notre *benchmark*.

Les valeurs "Nb." correspondent au nombre moyen de gènes sélectionnés, tandis que les valeurs "Préc." représentent la précision moyenne obtenue par le classifieur. Nous convions le lecteur à

se référer aux articles cités, pour plus de précisions sur les différentes méthodes utilisées. Pour les données sur les tumeurs ovariennes, nous n'avons pas pu trouver de résultats non biaisés.
On remarque que *Automated Delta-test* obtient des résultats de qualité équivalente, ou meilleure, à celle de la plupart des autres méthodes. De plus, elle sélectionne automatiquement le nombre optimal de gènes sur chaque itération, ce qui n'est pas le cas de toutes les méthodes. Le nombre de variables sélectionnées est assez faible, tout en conservant une bonne précision. Les résultats de [Ghattas 08] montrent cependant que l'algorithme GLMPath obtient des résultats d'assez bonne qualité, avec un nombre extrêmement faible de gènes sélectionnés.
Les algorithmes de *Forêts Aléatoires* (FA) semblent également intéressants à explorer, puisqu'ils obtiennent un très bon score de précision sur le jeu de données du colon. Leur point faible étant le nombre de gènes qu'ils sélectionnent.

4.6 Sélection de variables par l'optimisation des performances d'un classifieur

Nous avons vu au début de ce chapitre que, parmi les méthodes d'optimisation dédiées aux problèmes de sélection de variables, deux types de fonctions objectifs étaient réalisables. Tout d'abord, nous avons eu pour objectif de maximiser la distance interclasse, qui est un critère ne dépendant pas du classifieur utilisé. Dans la dernière partie de cette thèse, nous nous sommes concentrés davantage sur les méthodes *wrappers*. Nous avons donc repris notre méthode BEUS, et nous l'avons appliquée à une nouvelle fonction objectif, créée afin de maximiser directement les performances du classifieur DLDA.

4.6.1 Méthode

Nous avons repris le principe général de la méthode BEUS, en modifiant la fonction objectif :

$$f_w(S) = w \times \text{précision}(S) + (1 - w) \times (1 - |S|) \tag{4.6.1}$$

En désignant par précision(S) le résultat de précision obtenu après création du modèle du DLDA et estimation de ses performances sur l'ensemble d'apprentissage (restreint aux gènes de la solution S). Le but est alors de sélectionner les gènes qui maximisent la précision (et donc minimisent le taux d'erreur) sur l'ensemble d'apprentissage, tout en minimisant le nombre de ces gènes.

Ce type d'optimisation est beaucoup plus long en général, car la fonction objectif coûte beaucoup plus cher en termes de temps de calcul. Cependant, un grand nombre d'articles de la littérature utilisent ce principe. Par exemple [Li 01] maximise les performances du classifieur k-NN (cf section 3.5.3.1) à l'aide des algorithmes génétiques (cf section 1.10.4). D'autres papiers plus récents utilisent des versions de PSO (cf section 1.10.5.2) pour variables binaires, appelées alors *Binary PSO*, pour optimiser les performances des classifieurs k-NN [Chuang 08] ou LDA [Shen 08].

Nous allons nous baser sur ce même principe pour valider les performances de notre méthode d'optimisation BEUS. Cependant, nous allons légèrement modifier son fonctionnement, afin de simplifier la fonction bi-objectif en une fonction à un seul objectif. Cette modification permettra

d'éviter d'optimiser une fonction bi-objectif agrégée, souvent difficile à paramétrer. À cet effet, nous commençons par simplifier la fonction objectif en :

$$f(S) = \text{précision}(S) \tag{4.6.2}$$

Notre méthode n'aura alors qu'à maximiser $f(S)$, ce qui ne permet pas d'obtenir des ensembles de petite taille. La minimisation de la taille des sous-ensembles sera intégrée dans l'algorithme de la procédure ABEUS, en le modifiant légèrement. La procédure résultante est alors nommée *Aggregated BEUS* (ABEUS) (cf figure 4.10).

Procédure ABEUS
début
  Initialiser aléatoirement S un vecteur binaire
  faire
    $fit = f(S)$
    pour $i = 1$ **à** n **faire**
      $S_i = 1 - S_i$
      $fit' = f(S)$
      si(fit' $>$ fit) **alors**
        (la nouvelle solution est moins bonne que l'ancienne)
        $S_i = 1 - S_i$
      sinon si(fit' $=$ fit)
        (la nouvelle solution a les mêmes performances que l'ancienne)
        si $S_i = 0$ **alors**
          (on vient de retirer une variable sans dégrader la précision, on conserve donc cette solution)
        sinon
          (on vient d'ajouter une variable sans augmenter la précision, on ignore donc cette solution)
          $S_i = 1 - S_i$
        fin si
      fin si
    fin pour
  tantque une meilleure solution est trouvée durant une itération complète
fin

Figure 4.10 – Une itération de l'algorithme ABEUS.

Dans cette méthode, lorsque l'on parcourt une dimension, on ne conserve la nouvelle solution que si elle améliore la valeur de la fonction objectif, ou bien si la valeur de la fonction objectif est la même et la nouvelle solution contient une variable de moins. Cette simple modification permet, en théorie, de minimiser le nombre de variables, en même temps que l'on maximise la fonction objectif.

4.6.2 Protocole expérimental

Dans un premier temps, nous avons évalué les performances de notre nouvelle méthode d'optimisation. À cet effet, nous avons repris le *benchmark* présenté précédemment et nous avons effectué une validation croisée, de type *leave one out*, volontairement biaisée (définie dans la section 4.2.5.3). L'intérêt est double, puisque nous pourrons montrer les performances de notre méthode BEUS, mais nous pourrons également évaluer la qualité des solutions, qui peut résulter d'une telle optimisation biaisée. Le protocole utilisé sélectionne donc un sous-ensemble optimal de gènes, en même temps qu'il optimise la fonction objectif (la précision d'une validation croisée *leave one out*). C'est "malheureusement" une technique utilisée dans de nombreux papiers de la littérature, avec lesquels nous pourrons donc comparer nos résultats. On évalue ici une méthode d'optimisation, plutôt qu'un prédicteur efficace.

4.6.3 Résultats

Lors de l'exécution de notre méthode, on remarque tout d'abord une convergence extrêmement rapide, qui aboutit à un optimum local en quelques itérations seulement. Les sous-ensembles de gènes sélectionnés sont généralement de tailles très faibles, mais composés de gènes souvent très différents d'une itération à une autre. On retrouve encore une fois la convergence rapide de notre algorithme EUS présenté dans la section 2.4. Notre algorithme est donc apte à trouver rapidement un optimum local, mais, sans procédure de redémarrage, il ne peut plus en sortir. Afin d'obtenir les meilleurs résultats, nous avons donc utilisé une procédure de redémarrage aléatoire 10 fois sur chaque jeu de données, et nous avons gardé le meilleur résultat. Les résultats ont été consignés dans le tableau 4.14.

Données	Colon	Lymphome	Leucémie	Prostate	Cerveau	Ovaires
Nb gènes sélectionnés	11,000	6,000	6,000	9,000	10,000	12,000
Précision	0,952	1,000	1,000	0,990	0,900	1,000
Sensibilité	0,925	1,000	1,000	1,000	0,905	1,000
Spécificité	1,000	1,000	1,000	0,981	0,897	1,000
PPV	1,000	1,000	1,000	0,980	0,826	1,000
NPV	0,880	1,000	1,000	1,000	0,946	1,000

Tableau 4.14 – Validation croisée *leave one out*, optimisée par ABEUS.

La première constatation est que ce sont très souvent les mêmes exemples qui résistent à la prédiction. Par exemple, dans le cas de la Leucémie, nous obtenons assez rapidement une classification de tous les exemples, sauf le $84^{\text{ème}}$ (noté *T39_tumor*), qui n'est jamais prédit correctement. Nous remarquerons par la suite que les articles de la littérature rendent compte également de ce genre de résultats (comme [Martinez 10]), avec exactement les mêmes exemples. Différentes possibilités peuvent en être l'origine. Tout d'abord, ces erreurs peuvent provenir de données erronées, qui ne seraient donc pas cohérentes avec les autres. Une autre possibilité serait que ces données forment une autre classe d'exemples, qui ne serait pas suffisamment représentée dans l'ensemble d'apprentissage. Enfin, le problème pourrait être non linéairement séparable, et donc ne pourrait pas être correctement résolu par des classifieurs linéaires (LDA, DLDA, ...).

On remarque tout de même que, sur la moitié des jeux de données, notre méthode obtient un sous-ensemble de gènes de très petite taille, avec une précision de 100% par validation croisée *leave one out*. Ces résultats sont déjà bons, mais nous désirions vérifier s'il était possible de les améliorer encore davantage. Nous avons alors opéré une intensification sur chacun des meilleurs résultats trouvés, par une méthode que nous avons appelée *BestSubset*.

4.6.4 Méthode ABEUS + *BestSubset* sur protocole biaisé

Nous avons remarqué que sur l'ensemble des expériences, nous avons obtenu des sous-ensembles de gènes de tailles très faibles. Il est donc possible, en un temps très court, de vérifier les performances de tous les sous-ensembles de chacun de ces sous-ensembles de gènes. En effet, dans le pire des cas nous obtenons un sous-ensemble de gènes de taille 12. Si nous désirons améliorer ce résultat, il nous suffit de considérer $2^{12} = 4096$ sous-ensembles de gènes, ce qui est tout à fait dénombrable.

Nous avons alors appliqué cette procédure sur les 6 sous-ensembles optimaux trouvés dans le tableau 4.14, et nous avons obtenu les résultats consignés dans le tableau 4.15.

Données	Colon	Lymphome	Leucémie	Prostate	Cerveau	Ovaires
Nb sélectionnés	8,000	6,000	5,000	7,000	10,000	9,000
Précision	0,952	1,000	1,000	0,990	0,900	1,000
Sensibilité	0,925	1,000	1,000	1,000	0,905	1,000
Spécificité	1,000	1,000	1,000	0,981	0,897	1,000
PPV	1,000	1,000	1,000	0,980	0,826	1,000
NPV	0,880	1,000	1,000	1,000	0,946	1,000

Tableau 4.15 – Validation croisée *leave one out*, optimisée par ABEUS + BestSubset.

Nous pouvons constater que les sous-ensembles de gènes ont été améliorés, non en termes de précision, mais en termes de taille. Ces résultats sont tout à fait intéressants, puisqu'ils permettent d'affiner une solution d'une manière très simple, et en un temps très court.

4.6.5 Comparaison avec d'autres méthodes de la littérature

À titre de comparaison, nous avons repris quelques méthodes de la littérature qui étaient appliquées sur un processus de sélection biaisé. Les performances ont été consignées dans le tableau 4.16.

On remarque que GLMPath obtient, encore une fois, de bons résultats, avec très peu de gènes. À part cela, notre méthode obtient les meilleurs résultats sur l'ensemble des jeux de données. La méthode décrite dans [Martinez 10], utilisant un algorithme d'optimisation PSO binaire, obtient des résultats du même ordre que les nôtres, avec légèrement moins de gènes. D'ailleurs, [Martinez 10] ne parviennent pas à prédire le même exemple que nous.

Ces résultats sont bien évidemment optimistes, puisqu'ils sont sujets au biais de sélection, mais ils nous permettent tout de même de confirmer les performances de notre méthode d'optimisation.

Données	Colon		Leucémie		Prostate		Cerveau		Lymphome		Ovaires	
	Préc.	Nb	Préc.	Nb	Préc.	Nb	Préc.	Nb	Préc.	Nb	Préc.	Nb
[Shipp 02]	-	-	-	-	-	-	-	-	91,00	30	-	-
[Berchuck 05]	-	-	-	-	-	-	-	-	-	-	85,20	186
[Chuang 08]	-	-	-	-	92,16	1294	-	-	100,00	1042	-	-
[Ghattas 08]												
F-test	87,81	3	-	-	96,29	315	-	-	-	-	-	-
∂W	99,91	31	-	-	97,31	83	-	-	-	-	-	-
∂RW	99,71	33	-	-	97,31	902	-	-	-	-	-	-
∂Spb	99,71	34	-	-	98,91	45	-	-	-	-	-	-
SVM-RFE	99,43	32	-	-	100,00	64	-	-	-	-	-	-
GLMPath	93,60	2	-	-	100,00	3	-	-	-	-	-	-
Forêts Aléatoires	90,38	55	-	-	94,46	7	-	-	-	-	-	-
[Martinez 10]	-	-	-	-	99,02	4	-	-	100,00	3	-	-
ABEUS + BestSubset	95,20	8	100,00	5	99,02	7	90,00	10	100,00	6	100,00	9

Tableau 4.16 – Performances (en %) de différentes méthodes de la littérature sur notre *benchmark*.

4.6.6 Méthode ABEUS + BestSubset sur protocole non biaisé

Pour terminer notre étude, nous avons appliqué notre procédure de sélection de variables (ABEUS + BestSubset) sur les 6 jeux de données, mais en appliquant cette fois un protocole de test non biaisé, tel qu'il est décrit dans la figure 4.9. Les résultats sont consignés dans le tableau 4.17.

Données	Colon	Lymphome	Leucémie	Prostate	Cerveau	Ovaires
Nb gènes sélectionnés	7,000	9,312	6,736	7,294	14,033	8,759
Précision	0,839	0,909	0,972	0,873	0,700	0,685
Sensibilité	0,875	0,895	0,920	0,840	0,714	0,708
Spécificité	0,772	0,914	1,000	0,904	0,692	0,666
PPV	0,875	0,773	0,960	0,894	0,555	0,629
NPV	0,773	0,964	0,979	0,854	0,818	0,741

Tableau 4.17 – Validation croisée *leave one out*, optimisée par ABEUS + BestSubset.

On remarque que les résultats sont cohérents avec ceux obtenus précédemment par la méthode *automated* δ-test. Le nombre de gènes sélectionnés est plus important, sauf sur les données concernant les tumeurs ovariennes et le cancer du colon.
Le résultat le plus intéressant concerne les données sur les tumeurs ovariennes, qui est nettement de moins bonne qualité qu'en utilisant un protocole biaisé. Ce constat est d'autant plus important qu'à chaque itération on obtient pourtant une précision proche de 100% sur l'ensemble d'apprentissage.

4.6.7 Discussion

Ces dernières données confirment que les résultats obtenus par protocole biaisé sont très optimistes, et s'effondrent lorsque l'on utilise un protocole rigoureux.
On peut également remarquer que les méthodes *wrapper* permettent d'obtenir des résultats équivalents à une méthode par filtre, mais avec un coût calculatoire beaucoup plus important.

Dans le cadre de l'expérience avec protocole biaisé, nous avons consigné les sous-ensembles de sondes sélectionnées pour chaque jeu de données (cf tableau 4.18). Les sondes ont été appelées "sondes_i" lorsque leurs références n'étaient pas connues, le i correspondant à leur indice.

Colon	Lymphome	Leucémie	Prostate	Cerveau	Ovaires
sonde_0994	sonde_452	M31303_rna1_at	41504_s_at	HG2059-HT2114_at	200035_at
sonde_1221	sonde_1479	X80907_at	41671_at	J00073_at	202908_at
sonde_1635	sonde_1655	Y07604_at	37599_at	M77349_at	208357_x_at
sonde_1771	sonde_3146	Z35227_at	37639_at	S76475_at	209572_s_at
sonde_1884	sonde_3987	M84371_rna1_s_at	41755_at	U05291_at	212128_s_at
	sonde_4018		34876_at	U95740_rna2_at	212585_at
			37405_at	X57766_at	215568_x_at
				X74295_at	219531_at
				D43682_s_at	222225_at
				X00540_at	

Tableau 4.18 – Validation croisée *leave one out*, optimisée par ABEUS + BestSubset.

Nous avons pu remarquer que, sur plusieurs itérations, des sous-ensembles de sondes différents obtenaient exactement les mêmes performances que ces sous-ensembles optimaux. Nous avons renseigné quelques résultats explicites dans la figure 4.11. Pour plus de clarté, les intersections entre les sous-ensembles ont été grisées.

Leucémie				Prostate		
Sous-Ensemble 1	**Sous-Ensemble 2**	**Sous-Ensemble 3**	**Sous-Ensemble 4**	**Sous-Ensemble 1**	**Sous-Ensemble 2**	**Sous-Ensemble 3**
M31303_rna1_at		D88270_at	D86967_at	41504_s_at		
X80907_at	U41635_at	HG1612-HT1612_at	M27891_at	41671_at		31444_s_at
Y07604_at	U77604_at	M96803_at	M95678_at	37599_at		34078_s_at
Z35227_at	X67951_at	U50136_rna1_at	S72008_at	37639_at		39315_at
M84371_rna1_s_at	Y08612_at	X62654_rna1_at	X69111_at	41755_at		33767_at
		X68560_at	Z15115_at	34876_at	41468_at	36928_at
			U49020_cds2_s_at	37405_at	33915_at	37367_at
					32598_at	1980_s_at

Figure 4.11 – Différents sous-ensembles optimaux obtenus sur plusieurs itérations d'ABEUS + BestSubset.

Nous pouvons remarquer qu'il est possible d'obtenir des sous-ensembles optimaux complètement différents, dont plusieurs ont une intersection vide. Ce résultat aurait tendance à montrer que la méthode de sélection de variables par l'optimisation des performances d'un classifieur est instable et conduit à un sur-apprentissage. Ce constat est d'ailleurs renforcé par le fait que, lorsqu'un tel algorithme est utilisé dans un protocole non biaisé, il n'obtient pas de résultats exceptionnels, en comparaison d'une simple sélection par filtre (comme le δ-test).
Cependant, une grande partie des variables sélectionnées par cette méthode ont une valeur de *p-value* du t-test assez faible (et inférieure à 5%), ce qui prouverait que ces gènes sont tout de même assez pertinents.

Une méthode permettant de pallier ce problème serait d'affiner les différents optimums locaux obtenus, en les combinant, ou bien en ne conservant que les sondes les plus pertinentes, par exemple avec un tri effectué par *p-value* du t-test. Nous avons également examiné la possibilité d'améliorer nos résultats en utilisant une procédure permettant de combiner les différents optimums locaux trouvés par ABEUS. Nous avons alors utilisé des algorithmes génétiques élitistes,

en prenant comme population initiale ces optimums locaux, et en les faisant évoluer. Cependant, les calculs sont très longs et les résultats ne sont pas, ou très peu, améliorés, ce qui nous a conduits à mettre ces résultats de côté.

4.7 Conclusion

Dans ce chapitre, nous avons présenté différentes méthodes de sélection de variables pour des problèmes d'apprentissage automatique supervisés. Nous les avons ensuite appliquées à différents jeux de données et avons comparé leurs résultats à différentes méthodes de la littérature.

En premier lieu, nous avons présenté la méthode d'optimisation BEUS, dérivée de la méthode EUS présentée dans le chapitre 2. Cette dernière nous a permis d'optimiser une fonction bi-objectif, afin de maximiser la distance interclasse, tout en minimisant le nombre de variables à sélectionner. Les résultats obtenus étaient de bonne qualité et très stables. Ayant constaté que la méthode convergeait systématiquement vers les mêmes solutions, nous avons analysé précisément son fonctionnement et en avons déduit une méthode exacte, nommée δ-test, permettant de faire cette optimisation en temps linéaire. La méthode a été appliquée avec succès sur deux jeux de données issus d'études sur la prédiction de traitement chimiothérapique dans les cas du cancer du sein.

Le critère δ-test nous a tout de suite paru intéressant à exploiter, car il obtenait de bons résultats, comparé au critère du t-test de Welsh, pourtant très utilisé. Cependant, ce type de sélection de variables par filtre ne permet pas de trouver directement le nombre de gènes optimaux à utiliser dans notre sous-ensemble. Nous avons donc conçu une méthode nommée *SelectKStar* permettant de choisir un nombre de gènes optimal uniquement à partir de l'étude des données d'apprentissage.
Les tests ont ensuite été effectués sur un *benchmark* de six jeux de données. Celui-ci nous a permis de comparer les résultats de notre nouvelle méthode, appelée *Automated δ-test* à d'autres méthodes de la littérature, et a montré son efficacité, notamment au niveau du nombre très faible de gènes sélectionnés.

À la fin de ce chapitre, nous avons présenté d'autres types d'algorithmes d'optimisation permettant de faire de la sélection de variables. Le but est alors d'optimiser directement la sortie d'un classifieur (méthode *wrapper*). À ce moment nous avons clairement distingué deux types de résultats présents dans la littérature, afin d'éviter de les confondre : ceux plutôt optimistes, basés sur une sélection biaisée, et les autres, plus cohérents avec la réalité.

Nous avons alors modifié notre algorithme BEUS, afin de transformer un problème bi-objectif en un problème mono-objectif plus simple à optimiser. L'algorithme résultant, ABEUS, a ensuite été comparé à d'autres méthodes de la littérature et s'est avéré très efficace, tant pour les protocoles biaisés que non biaisés. Afin d'affiner ces résultats, nous avons introduit une méthode très simple, nommée *BestSubset*, permettant d'améliorer les performances d'un sous-ensemble de sondes existant en considérant tous les sous-ensembles de ce sous-ensemble.

BEUS et ABEUS sont des algorithmes d'optimisation à haute dimension. Leur utilisation permet de ne pas avoir à faire de présélection parmi les sondes initiales du jeu de données. En effet, ils peuvent fonctionner directement avec l'ensemble de toutes les sondes de la population, ce qui est novateur dans notre domaine.

Conclusion générale

Depuis quelques années, une classe de problèmes d'optimisation s'est énormément développée : les problèmes de grande dimension. Leur engouement est principalement motivé par l'apparition d'applications dans les domaines du *data mining* et du *web mining*. Ces domaines sont appliqués par exemple dans l'étude du génome, avec l'apparition de la technologie des puces à ADN, qui permet de créer des modèles de prédiction et de classification pour des problèmes biologiques réels, notamment en oncologie.

Les travaux présentés dans cette thèse apportent des contributions dans deux différents domaines de recherche. En premier lieu, dans le domaine de la recherche opérationnelle par des algorithmes d'optimisation adaptés pour les problèmes de grande dimension. Puis, dans le domaine de la prédiction en apprentissage automatique, où différentes méthodes de sélection de variables sont présentées, utilisant des algorithmes d'optimisation adaptés pour les problèmes de grande taille.

Dans la première partie de cette thèse, nous avions deux objectifs principaux : développer des méthodes simples et robustes pour la résolution de problèmes de grande dimension. Puis, dans un deuxième temps, saisir les caractéristiques leur permettant d'obtenir cette robustesse, tout en les affinant, afin de résoudre des fonctions plus complexes. Nous pouvons dire que ces objectifs ont été atteints. En effet, nous avons tout d'abord présenté deux méthodes adaptées pour les problèmes de grande dimension : CUS (*Classic Unidimensional Search*) et EUS (*Enhanced Unidimensional Search*). Ces méthodes de recherche locale sont basées sur le principe de *line search* (recherche linéaire) auquel nous avons ajouté un principe de redémarrage "guidé", afin d'assurer l'exploration de l'espace de recherche. Ces deux méthodes ont prouvé leur efficacité sur différents jeux de données, et ont montré une grande robustesse lors de l'augmentation de la dimension. La comparaison de ces méthodes a montré qu'EUS ne nécessitait pas de paramétrage (au contraire de CUS), et permettait de résoudre des problèmes plus complexes. Sa comparaison avec d'autres méthodes de la littérature, et notamment avec les méthodes présentées lors de la conférence CEC'08, a montré d'excellents résultats, prouvant l'efficacité et la robustesse de cette méthode.

Dans un deuxième temps, nous avons remarqué que le point faible d'EUS (et des méthodes de *line search* en général) était les fonctions non séparables. Nous avons donc amélioré EUS en l'hybridant avec une méthode de recherche unidimensionnelle développée par Fred Glover : la méthode 3-2-3. Cette méthode permet de faire une optimisation plus précise de chaque fonction unidimensionnelle, lors du parcours des dimensions. L'algorithme résultant, nommé EM323 (*Enhanced Multidimensional 3-2-3*), a par conséquent une vitesse de convergence plus lente que l'algorithme EUS. Nous avons mis en place une stratégie d'oscillation afin de rendre le parcours des dimensions "intelligent", ce qui permet d'accélérer cette convergence. Les résultats ont montré qu'EM323 conserve la robustesse d'EUS, mais permet également de résoudre des fonctions plus complexes, sur lesquelles EUS ne converge pas.

Dans une deuxième partie, nous nous sommes intéressés aux problèmes de fouille de données, et plus particulièrement à l'analyse de données d'expression de puces à ADN. Ces problèmes peuvent en effet être assimilés à des problèmes d'optimisation en grande dimension, puisque les puces à ADN contiennent des milliers, voire des dizaines de milliers de sondes. Dans un premier temps, nous avons travaillé sur la classification supervisée de cas de patientes atteintes d'un cancer du sein, afin de prédire leur réponse à un traitement de chimiothérapie. Les données de puces à ADN provenaient d'un essai clinique conduit conjointement à l'*Institut Gustave Roussy* à Villejuif, France, et au *MD Anderson Cancer Center* à Houston, USA. Dans notre étude, nous avons mis l'accent sur le prétraitement des données, avant la construction du modèle de prédiction, et plus particulièrement sur la phase de sélection de variables.

Nous avons développé une méthode heuristique issue de l'algorithme EUS, visant à maximiser la distance interclasse δ du sous-ensemble de sondes sélectionnées. La méthode résultante, nommée BEUS (*Binary* EUS), s'est avérée rapide et particulièrement robuste, les sous-ensembles de sondes sélectionnés étant parfaitement stables. Après analyse rigoureuse de la fonction objectif et de notre méthode d'optimisation, nous avons pu en déduire une méthode exacte, nommée δ-test, obtenant les mêmes résultats en temps linéaire. La comparaison de ces résultats avec ceux de la littérature a montré que notre méthode obtenait les meilleurs résultats, notamment au niveau du nombre très faible de sondes sélectionnées.
Cette méthode est caractérisée comme de type *filter*, et a donc été comparée à d'autres méthodes du même type, telles que le t-test de Welsh. Les résultats ont montré que notre méthode obtenait de meilleurs résultats, probablement du fait qu'elle ne pose aucune hypothèse concernant la distribution des valeurs, et est donc paramétrique.
Le point faible de ces méthodes par filtre est qu'elles classent les variables, mais ne nous donnent pas d'information sur le nombre de variables à sélectionner. Pour pallier ce problème, nous avons développé une méthode de sélection automatique du nombre optimal de sondes, nommée *SelectKStar*, basée sur l'étude des variations des performances du classifieur sur l'ensemble d'apprentissage seul. La procédure résultante, nommée *automated* δ-test, s'est avérée particulièrement robuste lors de tests de validation croisée (pas de sur-apprentissage des données) et, d'autre part, elle obtient des performances supérieures ou égales à celles des prédicteurs déjà publiés pour ces mêmes données.

Dans un deuxième temps, nous avons voulu comparer l'efficacité des méthodes par filtre, optimisant un critère statistique, aux méthodes dites *wrapper*, optimisant les performances d'un prédicteur particulier. Nous avons alors développé une méthode heuristique de sélection de variables, méthode ABEUS (*Aggregated BEUS*), visant à optimiser les performances d'une analyse discriminante linéaire portant sur l'ensemble des variables (gènes) sélectionnées. Cette méthode permet notamment d'optimiser la fonction, initialement bi-objectif, comme une fonction mono-objectif, en agrégeant le deuxième objectif dans le corps de son algorithme de recherche.
Les résultats que nous avons obtenus sur différents jeux de données publiques montrent que nos méthodes permettent de sélectionner des sous-ensembles de variables de tailles très faibles (de l'ordre d'une dizaine, voire moins), ce qui est important pour la robustesse des prédicteurs, et pour leur possible utilisation en routine clinique.

A l'issue de ce travail, de nombreuses pistes de recherche sont ouvertes.

Dans le contexte de l'optimisation de problèmes de grande dimension, nous pensons tout d'abord qu'il serait possible de paralléliser les algorithmes de *line search*, afin d'accroître leurs vitesses de convergence. Ce principe semble envisageable, puisque nous traitons le problème dimension par dimension.
Un des points forts des méthodes que nous avons développées, est d'obtenir très rapidement des

optimums locaux, même en haute dimension. Il serait alors intéressant de relancer plusieurs fois l'algorithme, de mémoriser les différents optimums locaux trouvés, et de combiner ces résultats, afin de les améliorer. Les pistes envisageables concerneraient, par exemple, les méthodes de *path relinking*, ou les algorithmes évolutionnaires.
Nous avons pu constater que la méthode d'évolution différentielle obtenait de très bons résultats sur les problèmes de grande dimension. Une autre piste envisageable serait de créer une méthode hybride à partir d'un de nos algorithmes, qui serait alors utilisé comme méthode d'optimisation locale, afin d'affiner les solutions trouvées par l'algorithme d'évolution différentielle.

Dans le contexte de l'apprentissage automatique, nous avons montré que les méthodes d'optimisation de grande dimension permettaient de rechercher une solution optimale parmi l'ensemble de l'espace de recherche. Or, de nombreux types de méthodes existent, et elles peuvent toutes potentiellement retourner des sous-ensembles de variables intéressants, mais dont les intersections peuvent être vides. Il pourrait être intéressant de combiner ces différentes approches, de la même façon que nous le suggérons pour les méthodes d'optimisation. Par exemple, nous pouvons calculer plusieurs sous-ensembles de variables à partir de méthodes de sélection par filtre (δ-test, t-test, ...), et combiner leurs résultats avec d'autres prédicteurs trouvés dans la littérature. Ce principe pourrait être mis en œuvre par des méthodes *wrapper* ou plus généralement par des méthodes évolutionnaires.
Une dernière piste envisagée, et actuellement en cours d'expérimentation, est d'affiner les résultats obtenus par notre méthode δ-test. En effet, le point faible des méthodes par filtre est de traiter les variables indépendamment ; il pourrait donc être intéressant d'affiner les résultats obtenus par ces méthodes, par exemple en cherchant un sous-ensemble de taille réduite, minimisant les corrélations entre les variables.

Annexe A

Annexes

A.1 Fonctions objectifs

Dans la suite, nous désignons par $x = \{x_1,...,x_N\}$ une solution, N la dimension du problème et f la fonction objectif considérée.
Une fonction objectif peut être :
- Séparable : les variables de décision sont indépendantes. Trouver l'optimum de la fonction objectif revient alors à trouver l'optimum de chaque fonction unidimensionnelle.
- Unimodale (à opposer à multimodale) : l'espace de recherche considéré ne contient qu'un seul optimum local, qui est donc également l'optimum global.
- *Shifted* : l'optimum global a été translaté (cf section A.1.3).

A.1.1 Définition des fonctions

Toutes les fonctions utilisées dans ce manuscrit sont décrites dans le tableau A.1.

A.1.2 Graphes des fonctions

Afin d'avoir une vision plus claire de chacune des fonctions présentées, nous les avons représentées pour $N = 2$, en utilisant le logiciel Matlab (cf figure A.1).

A.1.3 Fonctions *shifted*

De nombreuses fonctions objectifs sont minimales en $0_{\mathbb{R}}$. Ce comportement risque d'introduire un biais important lors de la résolution de ces fonctions par certains algorithmes d'optimisation. En effet, ceux-ci peuvent définir leur solution initiale égale à $0_{\mathbb{R}}$, et ainsi trouver immédiatement l'optimum. De même, le fait que toutes les composantes soient égales peut être une caractéristique simplificatrice de certains algorithmes qui trouveraient alors l'optimum de manière précoce.

Pour pallier ce problème, il est possible de translater (ou décaler ou encore *shift*) les fonctions objectifs, afin de modifier leur optimum. On remplace alors la fonction objectif $f(x)$ par $f(z)$ avec $z = x - o + x^*$, z et o étant des vecteurs de dimension N.
Cette méthode permet également de fixer une solution $o = \{o_1,...,o_N\}$ comme l'optimum global de la nouvelle fonction objectif translatée, la valeur optimale de la fonction objectif n'est, quant à elle, pas modifiée : $f(x^*) = f(o)$.

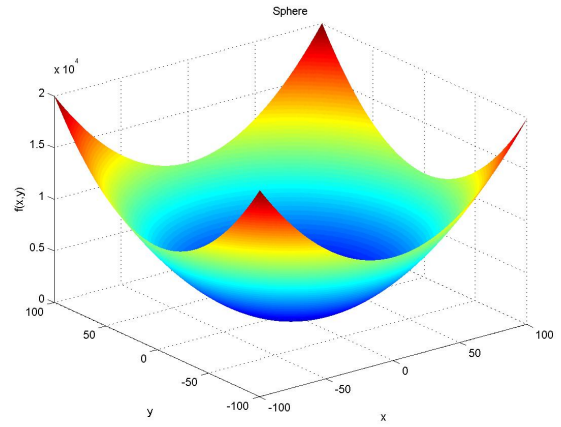
Fonction	Définition	U/M	S/NS
Sphere	$f(x) = \sum_{i=1}^{N} x_i^2$ $x^* = (0, .., 0), f(x^*) = 0$	U	S
Schwefel's Problem 2.21	$f(x) = \max_i\{\|x_i\|, 1 \leq i \leq N\}$ $x^* = (0, .., 0), f(x^*) = 0$	U	NS
Rosenbrock	$f(x) = \sum_{i=1}^{N-1} (100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2)$ $x^* = (1, .., 1), f(x^*) = 0$	M	NS
Rastrigin	$f(x) = \sum_{i=1}^{N} (x_i^2 - 10\cos(2\pi x_i) + 10)$ $x^* = (0, .., 0), f(x^*) = 0$	M	S
Griewank	$f(x) = \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} \cos(\frac{x_i}{\sqrt{i}}) + 1$ $x^* = (0, .., 0), f(x^*) = 0$	M	NS
Ackley	$f(x) = -20\exp(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}) - \exp(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)) + 20 + e$ $x^* = (0, .., 0), f(x^*) = 0$	M	S
Schwefel's Problem 2.22	$f(x) = \sum_{i=1}^{N} \|x_i\| + \prod_{i=1}^{N} \|x_i\|$ $x^* = (0, .., 0), f(x^*) = 0$	U	S
Schwefel's Problem 1.2	$f(x) = \sum_{i=1}^{N} (\sum_{j=1}^{i} x_j)^2$ $x^* = (0, .., 0), f(x^*) = 0$	U	NS
Extended f_{10}	$f(x) = f_{10}(x_N, x_1) + \sum_{i=1}^{N-1} f_{10}(x_i, x_{i+1})$ $f_{10}(x, y) = (x^2 + y^2)^{0.25}(\sin^2(50(x^2 + y^2)^{0.1}) + 1)$ $x^* = (0, .., 0), f(x^*) = 0$	U	NS
Bohachevsky	$f(x) = \sum_{i=1}^{N} (x_i^2 + 2x_{i+1}^2 - 0.3\cos(3\pi x_i) - 0.4\cos(4\pi x_{i+1}) + 0.7)$ $x^* = (0, .., 0), f(x^*) = 0$	U	NS
Schaffer	$f(x) = \sum_{i=1}^{N-1} (x_i^2 + x_{i+1}^2)^{0.25}(\sin^2(50(x_i^2 + x_{i+1}^2)^{0.1}) + 1)$ $x^* = (0, .., 0), f(x^*) = 0$	U	NS

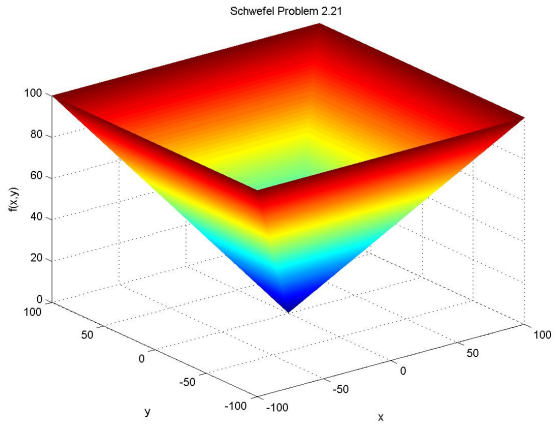
Tableau A.1 – Définition des fonctions objectifs utilisées dans ce mémoire (U = Unimodale, M = Multimodale, S = Séparable et NS = Non séparable).

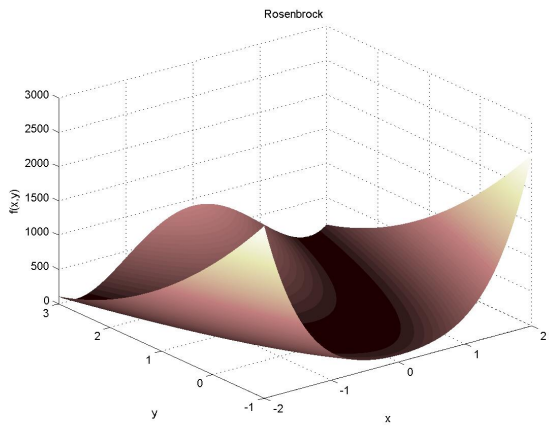
A.1.4 Ajout de biais

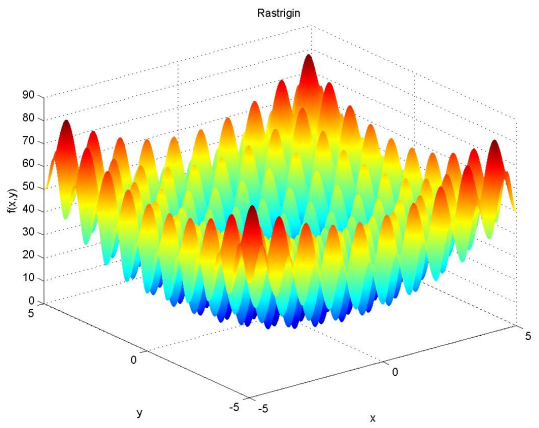
Très souvent, la valeur optimale prise par la fonction objectif est : $f(x^*) = 0$. Ce comportement peut biaiser les résultats, puisque cette information nous permettrait de calculer l'erreur entre cette "valeur optimale supposée" et la solution que nous avons calculée.

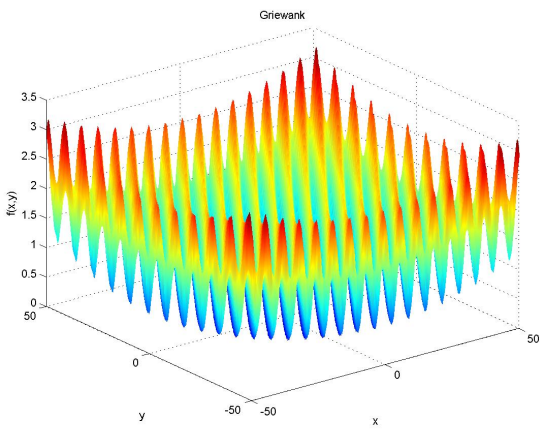
Pour éviter cela, il est préférable de modifier la valeur prise par l'optimum. On ajoute alors un biais à la fonction objectif, afin de modifier la valeur de l'image par la solution optimale. L'espace image étant défini dans $\mathbb{R}$, il suffit simplement d'ajouter une valeur scalaire au résultat de la fonction objectif. La nouvelle fonction objectif, notée F, s'écrit alors : $F(x) = f(x) + biais$.

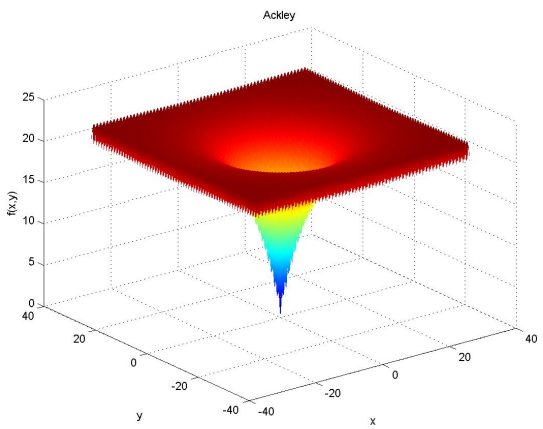

Sphere
x 10^4
f(x,y)
x
y


Schwefel Problem 2.21
f(x,y)
x
y


Rosenbrock
f(x,y)
x
y


Rastrigin
f(x,y)
x
y


Griewank
f(x,y)
x
y


Ackley
f(x,y)
x
y

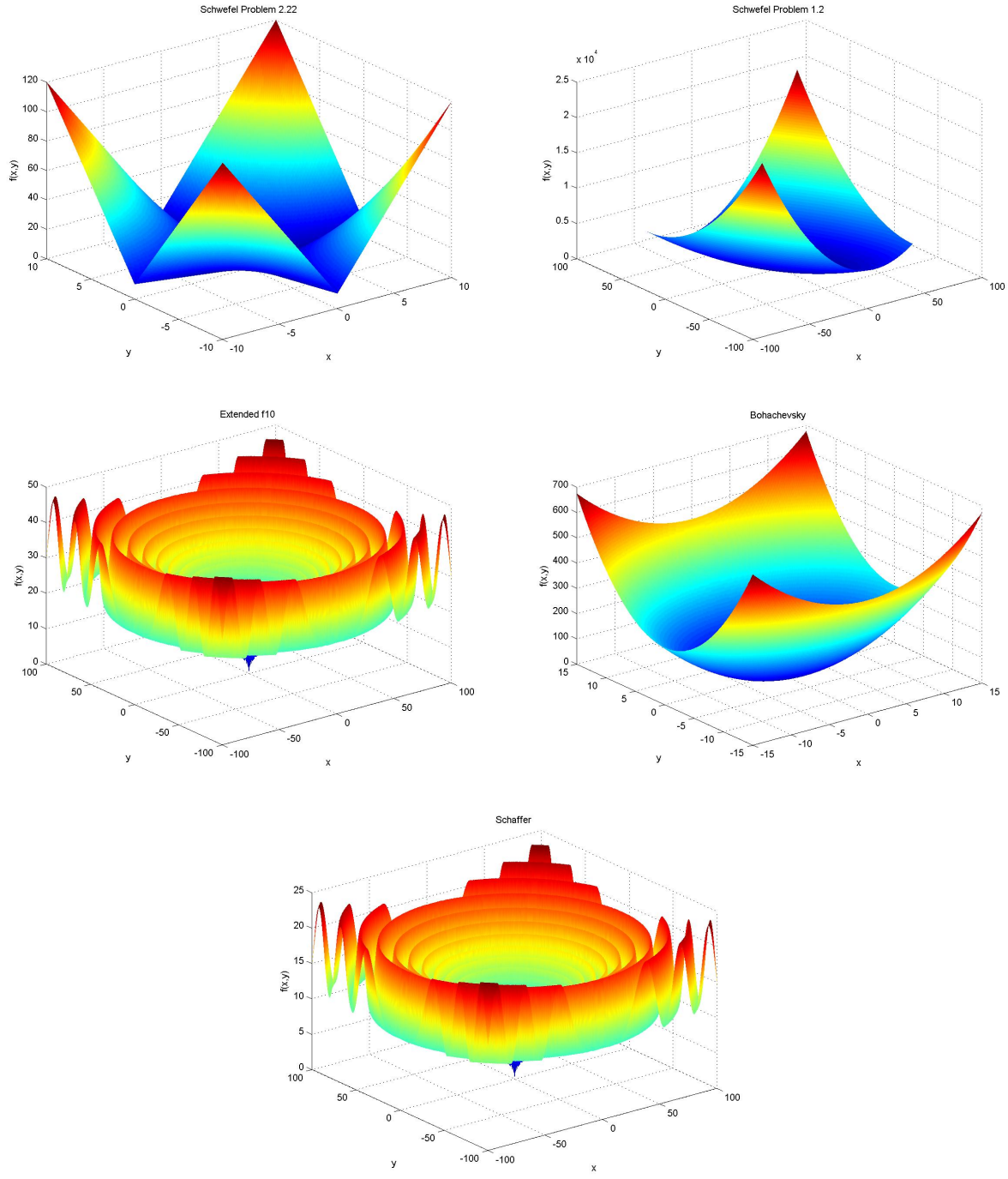


Figure A.1 – Graphes des 11 fonctions objectifs décrites précédemment, pour $N = 2$.

Références bibliographiques

[Aarts 97] E. H. L. Aarts & J. K. Lenstra. Local search in combinatorial optimization. John Wiley & Sons, 1997.

[Alon 99] U. Alon, N. Barkai, D. A. Notterman, K. Gish, S. Ybarra, D. Mack & A. J. Levine. *Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays.* Proceedings of the National Academy of Sciences (PNAS), vol. 96, no. 12, pages 6745–6750, 1999.

[Ambroise 02] C. Ambroise & G. J. McLachlan. *Selection bias in gene extraction on the basis of microarray gene-expression data.* Proceedings of the National Academy of Sciences (PNAS), vol. 99, no. 10, pages 6562–6566, 2002.

[April 03] J. April, F. Glover, J. Kelly & M. Laguna. *Practical introduction to simulation optimization.* In Proceedings of the Winter Simulation Conference, pages 71–78, Piscataway, New Jersey : Institute of Electrical and Electronics Engineers, December 7-10, 2003.

[Arora 09] S. Arora & B. Barak. Computational complexity : a modern approach. Cambridge University Press, 2009.

[Auger 05] A. Auger & N. Hansen. *A restart CMA evolution strategy with increasing population size.* In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, pages 1769–1776, Edinburgh, UK, September 2-5, 2005.

[Bäck 95] T. Bäck. Evolutionary algorithms in theory and practice. Oxford University Press, 1995.

[Bäck 02] T. Bäck. *Adaptive business intelligence based on evolution strategies : some application examples of self-adaptive software.* Information Sciences, vol. 148, issues 1-4, pages 113–121, 2002.

[Baldi 01] P. Baldi & S. Brunak. Bioinformatics : the machine learning approach. The MIT Press, 2001.

[Barlow 89] H. B. Barlow. *Unsupervised Learning.* Neural Computation, vol. 1, no. 3, pages 295–311, 1989.

[Basili 75] V. R. Basili & A. J. Turner. *Iterative enhancement : A practical technique for software development.* IEEE Transactions on Software Engineering, vol. 1, no. 4, pages 390–396, 1975.

[Bellman 57] R. E. Bellman. Dynamic programming. Princeton University Press, 1957.

[Bellman 61] R. E. Bellman. Adaptive control - a guided tour. Princeton University Press, 1961.

[Bentley 00] D. R. Bentley. *The human genome project - an overview.* Medicinal research reviews, vol. 20, issue 3, pages 189–196, 2000.

[Berchuck 05] A. Berchuck, E. S. Iversen, J. M. Lancaster, J. Pittman, J. Luo, P. Lee, S. Murphy, H. K. Dressman, P. G. Febbo, M. West*et al.* *Patterns of gene expression that characterize long-term survival in advanced stage serous ovarian cancers.* Clinical cancer research, vol. 11, no. 10, pages 3686–3696, 2005.

[Berrar 03] D. P. Berrar, W. Dubitzky & M. Granzow. A practical approach to microarray data analysis. Kluwer Academic Publishers, 2003.

[Berry 97] M. J. Berry & G. Linoff. Data mining techniques : For marketing, sales, and customer support. John Wiley & Sons, 1997.

[Bishop 95] C. M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995.

[Bishop 06] C. M. Bishop. Pattern recognition and machine learning (information science and statistics). Springer-Verlag, 2006.

[Blum 97] A. L. Blum & P. Langley. *Selection of Relevant Features and Examples in Machine Learning.* Artificial Intelligence, vol. 97, no. 1-2, pages 245–271, 1997.

[Bø 02] T. Bø & I. Jonassen. *New feature subset selection procedures for classification of expression profiles.* Genome biology, vol. 3, no. 4, pages 1–11, 2002.

[Bonabeau 99] E. Bonabeau, M. Dorigo & G. Theraulaz. Swarm intelligence : from natural to artificial systems. Oxford University Press, 1999.

[Boser 92] B. E. Boser, I. Guyon & V. Vapnik. *A Training Algorithm for Optimal Margin Classifiers.* In Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory (COLT'92), pages 144–152, Pittsburgh, PA, USA, July 27-29, 1992.

[Brest 08] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec & V. Zumer. *High-dimensional real-parameter optimization using Self-Adaptive Differential Evolution algorithm with population size reduction.* In Proceedings of the IEEE Congress on Evolutionary Computation (CEC'08), pages 2032–2039, Hong Kong, China, June 1-6, 2008.

[Brest 10] J. Brest & M. Maucec. *Self-adaptive differential evolution algorithm using population size reduction and three strategies.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0644-5.

[Brown 00] M. P. S. Brown, W. N. Grundy, D. Lin, N. Cristianini, C. W. Sugnet, T. S. Furey, M. Ares & D. Haussler. *Knowledge-based Analysis of Microarray Gene Expression Data By Using Support Vector Machines.* Proceedings of The National Academy of Sciences (PNAS), vol. 97, no. 1, pages 262–267, 2000.

[Cai 08] D. Cai, X. He & J. Han. *Training linear discriminant analysis in linear time.* In Proceedings of the 24th International Conference on Data Engineering Workshops (ICDE 2008), pages 209–217, Cancún, México, April 7-12, 2008.

[Chapelle 06] O. Chapelle, B. Schölkopf & A. Zien. Semi-supervised learning. Adaptive computation and machine learning. MIT Press, 2006.

[Charnes 57] A. Charnes & W. W. Cooper. *Management models and industrial applications of linear programming.* Management Science, vol. 4, no. 1, pages 38–91, 1957.

[Chelouah 97] R. Chelouah & P. Siarry. *Enhanced Continuous Tabu Search : a New Algorithm for the Global Optimization of Multiminima Functions.* In Proceedings of the Second International Conference on Metaheuristics (MIC'97), Sophia-Antipolis, France, July 21-24, 1997.

[Chen 00] L. Chen. *A new LDA-based face recognition system which can solve the small sample size problem.* Pattern Recognition, vol. 33, no. 10, pages 1713–1726, 2000.

[Chuang 08] L. Y. Chuang, H. W. Chang, C. J. Tu & C. H. Yang. *Improved binary PSO for feature selection using gene expression data.* Computational Biology and Chemistry, vol. 32, no. 1, pages 29–38, 2008.

[Clerc 03] M. Clerc. *TRIBES - Un exemple d'optimisation par essaim particulaire sans paramètres de contrôle.* In Proceedings of the Optimisation par Essaim Particulaire 2003, Carré des Sciences, Paris, October 2, 2003.

[Cochocki 93] A. Cochocki & R. Unbehauen. Neural networks for optimization and signal processing. John Wiley & Sons, 1993.

[Coello 95] C. A. C. Coello, A. D. Christiansen & A. H. Aguirre. *Multiobjective design optimization of counterweight balancing of a robot arm using genetic algorithms.* In Proceedings of the Seventh International Conference on Tools with Artificial Intelligence (ICTAI'95), pages 20–23, Herndon, VA, USA, November 1995.

[Collette 02] Y. Collette & P. Siarry. Optimisation multiobjectif. Eyrolles, 2002.

[Cook 71] S. A. Cook. *The complexity of theorem-proving procedures.* In Proceedings of the third annual ACM Symposium on Theory Of Computing (STOC'71), pages 151–158, New York, NY, USA, 1971.

[Cormen 90] T. H. Cormen, C. E. Leiserson & R. L. Rivest. Introduction to algorithms, chapitre 16 : Greedy Algorithms. MIT Press and McGraw-Hill, 1st edition, 1990.

[Croes 58] G. A. Croes. *A method for solving traveling salesman problems.* Operations Research, vol. 6, no. 6, pages 791–812, 1958.

[Dantzig 63] G. Dantzig. Linear programming and extensions. Princeton University Press, 1963.

[Darwin 59] C. Darwin. On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. John Murray, 1859.

[Davies 94] S. Davies & S. Russell. *NP-Completeness of Searches for Smallest Possible Feature Sets.* In Proceedings of The AAAI Symposium on Intelligent Relevance, pages 37–39, New Orleans, USA, November 1994. AAAI Press.

[De Jong 75] K. D. De Jong. *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, 1975.

[Deb 00] K. Deb, S. Agrawal, A. Pratap & T. Meyarivan. *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization : NSGA-II.* In Parallel Problem Solving from Nature PPSN VI, pages 849–858, Paris, France, September 16-20, 2000.

[Deneubourg 90] J.-L. Deneubourg, S. Aron, S. Goss & J. M. Pasteels. *The self-organizing exploratory pattern of the argentine ant.* Journal of Insect Behavior, vol. 3, no. 2, pages 159–168, 1990.

[Deutsch 03] J. M. Deutsch. *Evolutionary algorithms for finding optimal gene sets in microarray prediction.* Bioinformatics, vol. 19, no. 1, pages 45–52, 2003.

[DeVore 96] R. A. DeVore & V. N. Temlyakov. *Some remarks on greedy algorithms.* Advances in Computational Mathematics, vol. 5, no. 1, pages 173–187, 1996.

[Dorigo 92] M. Dorigo. *Optimization, Learning and Natural Algorithms (in Italian).* PhD thesis, Politecnico di Milano, Italy, 1992.

[Dréo 06] J. Dréo, A. Petrowski, E. Taillard & P Siarry. Metaheuristics for hard optimization methods and case studies. Springer, 2006.

[Duarte 09] A. Duarte & R. Marti. *An Adaptive Memory Procedure for Continuous Optimization.* In ISDA '09 : Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, pages 1085–1089, Pisa, Italy, November 30 - December 2, 2009. IEEE Computer Society.

[Duarte 10] A. Duarte, R. Martí & F. Gortazar. *Path relinking for large-scale global optimization.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0650-7.

[Dudoit 02] S. Dudoit, J. Fridlyand & T. P. Speed. *Comparison of discrimination methods for the classification of tumors using gene expression data.* Journal of the American Statistical Association, vol. 97, no. 457, pages 77–87, 2002.

[Dudzinski 87] K. Dudzinski & S. Walukiewicz. *Exact methods for the knapsack problem and its generalizations.* European Journal of Operational Research, vol. 28, no. 1, pages 3–21, 1987.

[Duncan 55] D. B. Duncan. *Multiple range and multiple F tests.* Biometrics, vol. 11, pages 1–42, 1955.

[Dupuy 07] A. Dupuy & R. M. Simon. *Critical review of published microarray studies for cancer outcome and guidelines on statistical analysis and reporting.* Journal of the National Cancer Institute, vol. 99, no. 2, pages 147–157, 2007.

[Edgeworth 85] F. Y. Edgeworth. *Methods of statistics.* Journal of the Statistical Society of London, pages 181–217, 1885. Jubilee Volume.

[Efron 79] B. Efron. *Bootstrap Methods : Another Look at the Jackknife.* The Annals of Statistics, vol. 7, no. 1, pages 1–26, 1979.

[Efron 83] B. Efron & G. Gong. *A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation.* The American Statistician, vol. 37, no. 1, pages 36–48, 1983.

[Eshelman 93] L. J. Eshelman & J. D. Schaffer. *Real-coded genetic algorithms and interval-schemata.* Foundation of Genetic Algorithms, vol. 2, pages 187–202, 1993.

[Faigle 92] U. Faigle & W. Kern. *Some convergence results for probabilistic Tabu Search.* ORSA Journal on Computing, vol. 4, no. 1, pages 32–37, 1992.

[Fayyad 96] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth & R. Uthurusamy. Advances in knowledge discovery and data mining. AAAI/MIT Press, 1996.

[Ferber 99] J. Ferber. Multi-agent systems - an introduction to distributed artificial intelligence. Addison Wesley, 1999.

[Fletcher 64] R. Fletcher & C. M. Reeves. *Function minimization by conjugate gradients.* The Computer Journal, vol. 7, no. 2, pages 149–154, 1964.

[Fogel 66] L. J. Fogel, A. J. Owens & M. J. Walsh. Artificial intelligence through simulated evolution. John Wiley, 1966.

[Fonseca 93] C. M. Fonseca & P. J. Fleming. *Genetic algorithms for multiobjective optimization : Formulation, discussion and generalization.* In Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA'93), pages 416–423, Urbana-Champaign, IL, USA, June 1993. Morgan Kauffman.

[Fortnow 09] L. Fortnow. *The status of the P versus NP problem.* Communications of the ACM, vol. 52, no. 9, pages 78–86, 2009.

[Fukunaga 90] K. Fukunaga. Introduction to statistical pattern recognition. Academic Press, 1990.

[Gao 07] Yu Gao & Yong-Jun Wang. *A Memetic Differential Evolutionary Algorithm for High Dimensional Functions Optimization.* In Third International Conference on Natural Computation (ICNC 2007), volume 4, pages 188–192, Haikou, Hainan, China, August 24-27, 2007.

[García-Martínez 09] C. García-Martínez & M. Lozano. *Continuous Variable Neighbourhood Search Algorithm Based on Evolutionary Metaheuristic Components : A Scalability Test.* In ISDA '09 : Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, pages 1074–1079, Pisa, Italy, November 30 - December 2, 2009. IEEE Computer Society.

[García-Martínez 10] C. García-Martínez, F. Rodríguez & M. Lozano. *Role differentiation and malleable mating for differential evolution : an analysis on large-scale optimisation.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0641-8.

[García-Nieto 10] J. García-Nieto & E. Alba. *Restart particle swarm optimization with velocity modulation : a scalability test.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0648-1.

[García 09] S. García, D. Molina, M. Lozano & F. Herrera. *A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour : a case study on the* CEC *'2005 Special Session on Real Parameter Optimization.* Journal of Heuristics, vol. 15, no. 6, pages 617–644, 2009.

[Gardeux 09a] V. Gardeux. *HeurisTest - A modular testbed for metaheuristics*, 2009. [En ligne]. `http://gardeux-vincent.eu/HeurisTest.php`.

[Gardeux 09b] V. Gardeux, R. Chelouah & P. Siarry. *HeurisTest - A modular testbed for metaheuristics.* In Proceedings of Innovation Technologique et systèmes de Transport, [Poster], ENSTA ParisTech, Paris 15ème, France, October 26-29, 2009.

[Gardeux 09c] V. Gardeux, R. Chelouah, P. Siarry & F. Glover. *Unidimensional Search for Solving Continuous High-Dimensional Optimization Problems.* In ISDA '09 : Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, pages 1096–1101, Pisa, Italy, November 30 - December 2, 2009. IEEE Computer Society.

[Gardeux 10] V. Gardeux, R. Chelouah & P. Siarry. *La recherche unidimensionnelle pour la résolution de problèmes d'optimisation à grande dimensionnalité.* In Proceedings of Recherche Opérationnelle et Aide à la Décision, Toulouse, France, February 24-26, 2010.

[Gardeux 11a] V. Gardeux, R. Chelouah, P. Siarry & F. Glover. *EM323 : a line search based algorithm for solving high-dimensional continuous non-linear optimization problems.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, vol. 15, no. 11, pages 2275–2285, 2011.

[Gardeux 11b] V. Gardeux, R. Natowicz, R. Chelouah, R. Rouzier, A. Padua Braga & P. Siarry. *Un algorithme d'optimisation à haute dimension pour la fouille de données : méthode et application en onco-pharmacogénomique.* In Proceedings of Recherche Opérationnelle et Aide à la Décision, Saint-Étienne, France, March 2-4, 2011.

[Gardeux 11c] V. Gardeux, R. Natowicz, R. Chelouah, P. Siarry, A. Padua Braga, L. Pusztai, F. Reyal & R. Rouzier. *Computing Molecular Signatures as Optima of a Bi-Objective Function : Method and Application to Two-Class Prediction Problems in High-Throughput Transcriptomic Studies.* BMC Bioinformatics, 2011. [Submitted].

[Ghattas 08] B. Ghattas & A. Ben Ishak. *Sélection de variables pour la classification binaire en grande dimension : comparaisons et application aux données de biopuces.* Journal de la société française de statistique, vol. 149, no. 3, pages 43–66, 2008.

[Glover 77] F. Glover. *Heuristics for integer programming using surrogate constraints.* Decision Sciences, vol. 8, no. 1, pages 156–166, 1977.

[Glover 86] F. Glover. *Future Paths for Integer Programming and Links to Artificial Intelligence.* Computers and Operations Research, vol. 13, no. 5, pages 533–549, 1986.

[Glover 89a] F. Glover. *Tabu search - part I.* ORSA Journal on Computing, vol. 1, no. 3, pages 190–206, 1989.

[Glover 89b] F. Glover. *Tabu search - part II.* ORSA Journal on Computing, vol. 2, no. 1, pages 4–32, 1989.

[Glover 95] F. Glover. *Tabu Thresholding : Improved Search by Nonmonotonic Trajectories.* ORSA Journal on Computing, vol. 7, no. 4, pages 426–442, 1995.

[Glover 02] F. Glover & S. Hanafi. *Tabu search and finite convergence.* Discrete Applied Mathematics, vol. 119, no. 1, pages 3–36, 2002.

[Glover 03] F. Glover & G. A. Kochenberger. Handbook of metaheuristics. International series in operations research and management science. Kluwer Academic Publishers, 2003.

[Glover 10] F Glover. The 3-2-3, stratified split and nested interval line search algorithms. Research Report, OptTek Systems, Boulder, CO, 2010.

[Glymour 97] C. Glymour, D. Madigan, D. Pregibon & P. Smyth. *Statistical Themes and Lessons for Data Mining.* Data Mining and Knowledge Discovery, vol. 1, no. 1, pages 11–28, 1997.

[Goldberg 89] D. E. Goldberg. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Professional, 1st edition, 1989.

[Golub 99] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri & C. D. Bloomfield. *Molecular classification of cancer : class discovery and class prediction by gene expression monitoring.* Science, vol. 286, no. 5439, pages 531–537, 1999.

[Grippo 86] L. Grippo, F. Lampariello & S. Lucidi. *A nonmonotone line search technique for Newton's method.* SIAM Journal on Numerical Analysis, vol. 23, no. 4, pages 707–716, 1986.

[Grosan 07] C. Grosan & A. Abraham. *Modified Line Search Method for Global Optimization.* In Proceedings of the First Asia International Conference on Modelling & Simulation (AMS'07), pages 415–420, Phuket, Thailand, March 27-30, 2007. IEEE Computer Society.

[Gunn 98] S. R. Gunn. *Support Vector Machines for Classification and Regression.* Rapport technique, University of Southampton, Image Speech and Intelligent Systems Research Group, 1998.

[Guo 07] Y. Guo, T. Hastie & R. Tibshirani. *Regularized linear discriminant analysis and its application in microarrays.* Biostatistics, vol. 8, no. 1, page 86, 2007.

[Guyon 03] I. Guyon & A. Elisseeff. *An introduction to variable and feature selection.* J. Mach. Learn. Res., vol. 3, pages 1157–1182, 2003.

[Hajek 88] B. Hajek. *Cooling schedules for optimal annealing.* Mathematics of Operations Research, vol. 13, no. 2, pages 311–329, 1988.

[Hand 01] D. J. Hand, P. Smyth & H. Mannila. Principles of data mining. MIT Press, Cambridge, 2001.

[Hansen 09] N. Hansen, A. Auger, S. Finck & R. Ros. *Real-parameter black-box optimization benchmarking 2009 : Experimental Setup.* Rapport technique RR-6828, INRIA, 2009.

[Hart 68] P. E. Hart, N. J. Nilsson & B. Raphael. *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pages 100–107, 1968.

[Hastie 01] T. Hastie, R. Tibshirani & J. H. Friedman. The elements of statistical learning : data mining, inference, and prediction. Springer-Verlag, 2001.

[Heller 02] M. J. Heller. *DNA microarray technology : Devices, Systems, and Applications.* Annual Review of Biomedical Engineering, vol. 4, no. 1, pages 129–153, 2002.

[Heppner 90] F. Heppner & U. Grenander. *A stochastic nonlinear model for coordinated bird flocks.* The ubiquity of chaos, pages 233–238, 1990.

[Hernandez 08] J. C. H. Hernandez, B. Duval & J.-K. Hao. *SVM-based local search for gene selection and classification of microarray data.* In Bioinformatics Research and Development (BIRD), pages 499–508, Technical University of Vienna, Vienna, Austria, July 7-9, 2008. Springer.

[Herrera 10a] F. Herrera, M. Lozano & D. Molina. *Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems.* Rapport technique, University of Granada, Granada, Spain, 2010. [En ligne]. `http://sci2s.ugr.es/eamhco/\#LSOP`.

[Herrera 10b] F. Herrera, M. Lozano & D. Molina. Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. Available : http ://sci2s.ugr.es/eamhco/testfunctions-SOCO.pdf, 2010.

[Hess 06] K. R. Hess, K. Anderson, W. F. Symmans, V. Valero, N. Ibrahim, J. A. Mejia, D. Booser, R. L. Theriault, A. U. Buzdar, P. J. Dempsey*et al.* *Pharmacogenomic predictor of sensitivity to preoperative chemotherapy with paclitaxel and fluorouracil, doxorubicin, and cyclophosphamide in breast cancer.* Journal of clinical oncology, vol. 24, no. 26, pages 4236–4244, 2006.

[Hestenes 52] M. R. Hestenes & E. Stiefel. *Methods of Conjugate Gradients for Solving Linear Systems.* Journal of Research of the National Bureau of Standards, vol. 49, no. 6, pages 409–436, December 1952.

[Holland 75] J. H. Holland. Adaptation in natural and artificial systems. The University of Michigan Press, 1975.

[Hopfield 85] J. J. Hopfield & D. W. Tank. *"Neural" computation of decisions in optimization problems.* Biological Cybernetics, vol. 52, no. 3, pages 141–152, 1985.

[Horn 94] J. Horn, N. Nafploitis & D. E. Goldberg. *A niched Pareto genetic algorithm for multiobjective optimization.* In Proceedings of the First IEEE Conference on Evolutionary Computation (ICEC'94), pages 82–87, Orlando, Florida, USA, June 27-29, 1994. IEEE Press.

[Howard 60] R.A. Howard. Dynamic programming and markov process. The MIT Press, 1960.

[Hsieh 08] S.-T. Hsieh, T.-Y. Sun, C.-C. Liu & S.-J. Tsai. *Solving large scale global optimization using improved Particle Swarm Optimizer.* In Proceedings of the IEEE Congress on Evolutionary Computation (CEC'08), pages 1777–1784, Hong Kong, China, June 1-6, 2008.

[Huang 02] R. Huang, Q. Liu, H. Lu & S. Ma. *Solving the Small Sample Size Problem of LDA.* In Proceedings of the 16th International Conference on Pattern Recognition (ICPR'02), volume 3, pages 29–32, Quebec, Canada, August 11-15, 2002. IEEE Computer Society.

[Hvattum 09] L. M. Hvattum & F. Glover. *Finding local optima of high-dimensional functions using direct search methods.* European Journal of Operational Research, vol. 195, issue 1, pages 31–45, 2009.

[Inza 02] I. Inza, B. Sierra, R. Blanco & P. Larrañaga. *Gene selection by sequential search wrapper approaches in microarray cancer class prediction.* Journal of Intelligent & Fuzzy Systems, vol. 12, no. 1, pages 25–33, 2002.

[Inza 04] I. Inza, P. Larranaga, R. Blanco & A. J. Cerrolaza. *Filter versus wrapper gene selection approaches in DNA microarray domains.* Artificial Intelligence in Medicine, vol. 31, no. 2, pages 91–103, 2004.

[Jain 97] A. Jain & D. Zongker. *Feature selection : Evaluation, application, and small sample performance.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 2, pages 153–158, 1997.

[Jarvis 05] R. M. Jarvis & R. Goodacre. *Genetic algorithm optimization for pre-processing and variable selection of spectroscopic data.* Bioinformatics, vol. 21, no. 7, pages 860–868, 2005.

[Johnson 85] D. S. Johnson. *The NP-completeness column : an ongoing guide.* Journal of Algorithms, vol. 6, no. 3, pages 434–451, 1985.

[Kaelbling 96] L. P. Kaelbling, M. Littman & A. Moore. *Reinforcement Learning : A Survey.* Journal of Artificial Intelligence Research, vol. 4, issue 1, pages 237–285, 1996.

[Kantorovich 60] L. V. Kantorovich. *Mathematical Methods of Organizing and Planning Production.* Management Science, vol. 6, no. 4, pages 366–422, 1960.

[Karp 72] R. M. Karp. *Reducibility among combinatorial problems.* Complexity of Computer computations, pages 85–104, 1972.

[Kennedy 95] J. Kennedy & R. Eberhart. *Particle swarm optimization.* In Proceedings of the IEEE International Conference on Neural Networks, volume 4, pages 1942–1948, November 27 - December 1, 1995.

[Kiefer 53] J. Kiefer. *Sequential minimax search for a maximum.* Proceedings of the American Mathematical Society (AMS), vol. 4, no. 3, pages 502–506, 1953.

[Kirkpatrick 83] S. Kirkpatrick, C. D. Gelatt & M. P. Vecchi. *Optimization by simulated annealing.* Science, vol. 220, no. 4598, pages 671–680, 1983.

[Knuth 73] D. E. Knuth. The art of computer programming. Addison-Wesley, Reading, MA, 1st edition, 1973.

[Kohavi 95] R. Kohavi. *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.* In Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95), pages 1137–1143, Montreal, Quebec, Canada, August 20-25, 1995. Morgan Kaufmann.

[Kohavi 97] R. Kohavi & G. H. John. *Wrappers for feature subset selection.* Artificial Intelligence, vol. 97, no. 1, pages 273–324, 1997.

[Kohl 95] N. Kohl. *Exact methods for time constrained routing and related scheduling problems.* PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 1995.

[Koza 90] J. R. Koza. *Genetic programming : a paradigm for genetically breeding populations of computer programs to solve problems.* Rapport technique STAN-CS-90-1314, Stanford University, Stanford, CA, USA, 1990.

[Land 60] A. H. Land & A. G. Doig. *An automatic method of solving discrete programming problems.* Econometrica, vol. 28, no. 3, pages 497–520, 1960.

[LaTorre 10] A. LaTorre, S. Muelas & J.-M. Peña. *A MOS-based dynamic memetic differential evolution algorithm for continuous optimization : a scalability test.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0646-3.

[Lee 98] W. Lee & S. J. Stolfo. *Data mining approaches for intrusion detection.* In Proceedings of the 7th conference on USENIX Security Symposium, volume 7, Marriott Hotel, San Antonio, Texas, January 26-29, 1998.

[Lettvin 59] J. Lettvin, H. Maturana, W. McCulloch & W. Pitts. *What the Frog's Eye Tells the Frog's Brain.* In Proceedings of the Institute of Radio Engineers, volume 47, issue 11, pages 1940–1951, November 1959.

[Li 01] L. Li, C. R. Weinberg, T. A. Darden & L. G. Pedersen. *Gene selection for sample classification based on gene expression data : study of sensitivity to choice of parameters of the GA/KNN method.* Bioinformatics, vol. 17, no. 12, pages 1131–1142, 2001.

[Lin 73] S. Lin & B. W. Kernighan. *An Effective Heuristic Algorithm for the Traveling-Salesman Problem.* Operations Research, vol. 21, no. 2, pages 498–516, 1973.

[Liu 95] H. Liu & R. Setiono. *Chi2 : Feature Selection and Discretization of Numeric Attributes.* In Proceedings of the Seventh International Conference on Tools with Artificial Intelligence, pages 388–391, Herndon, VA, USA, November 5-8, 1995.

[March 91] J. G. March. *Exploration and Exploitation in Organizational Learning.* Organization Science, vol. 2, no. 1, pages 71–87, 1991.

[Marti 06] R. Marti, M. Laguna & F. Glover. *Principles of scatter search.* European Journal of Operational Research, vol. 169, issue 2, pages 359–372, 2006.

[Martinez 10] E. Martinez, M. M. Alvarez & V. Trevino. *Compact cancer biomarkers discovery using a swarm intelligence feature selection algorithm.* Computational biology and chemistry, vol. 34, no. 4, pages 244–250, 2010.

[Meiri 06] R. Meiri & J. Zahavi. *Using simulated annealing to optimize the feature selection problem in marketing applications.* European Journal of Operational Research, vol. 171, no. 3, pages 842–858, 2006.

[Metropolis 49] N. Metropolis & S. Ulam. *The Monte Carlo Method.* Journal of the American Statistical Association, vol. 44, no. 247, pages 335–341, 1949.

[Metropolis 53] N. Metropolis, A. R. Rosenbluth, M. N. Rosenbluth, A. Teller & E. Teller. *Equation of state calculations by fast computing machines.* The Journal of Chemical Physics, vol. 21, no. 6, pages 1087–1092, 1953.

[Miller 05] L. D. Miller, J. Smeds, J. George, V. B. Vega, L. Vergara, A. Ploner, Y. Pawitan, P. Hall, S. Klaar, E. T. Liu & J. Bergh. *An expression signature for p53 status in human breast cancer predicts mutation status, transcriptional effects, and patient survival.* Proceedings of the National Academy of Sciences of the United States of America, vol. 102, no. 38, pages 13550–13555, 2005.

[Mitchell 97] T. M. Mitchell. Machine learning. McGraw Hill, 1997.

[Molina 09] D. Molina, M. Lozano & F. Herrera. *Memetic Algorithm with Local Search Chaining for Continuous Optimization Problems : A Scalability Test.* In ISDA '09 : Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, pages 1068–1073, Pisa, Italy, November 30 - December 2, 2009. IEEE Computer Society.

[Molina 10] D. Molina, M. Lozano, A. Sánchez & F. Herrera. *Memetic algorithms based on local search chains for large scale continuous optimisation problems : MA-SSW-Chains.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0647-2.

[Montes de Oca 10] M. Montes de Oca, D. Aydin & T. Stützle. *An incremental particle swarm for large-scale continuous optimization problems : an example of tuning-in-the-loop (re)design of optimization algorithms.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0649-0.

[Moré 94] J. J. Moré & D. J. Thuente. *Line search algorithms with guaranteed sufficient decrease.* ACM Transactions on Mathematical Software (TOMS), vol. 20, no. 3, pages 286–307, 1994.

[Muelas 09] S. Muelas, A. LaTorre & J. M. Peña. *A Memetic Differential Evolution Algorithm for Continuous Optimization.* In ISDA '09 : Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, pages 1080–1084, Pisa, Italy, November 30 - December 2, 2009. IEEE Computer Society.

[Mühlenbein 96] H. Mühlenbein & G. Paaß. *From recombination of genes to the estimation of distributions.* Parallel Problem Solving from Nature - PPSN IV, vol. 1141, pages 178–187, 1996.

[Natowicz 08] R. Natowicz, R. Incitti, B. C. Euler Horta, P. Guinot, K. Yan, C. Coutant, F. Andre, L. Pusztai & R. Rouzier. *Prediction of the outcome of preoperative chemotherapy in breast cancer using DNA probes that provide information on both complete and incomplete responses.* BMC bioinformatics, vol. 9, no. 149, 2008.

[Natowicz 10] R. Natowicz, C. Moraes Pataro, R. Incitti, M. Costa, A. Cela, T. Souza, A. Padua Braga & R. Rouzier. *Prediction of Chemotherapy Outcomes by Optimal Gene Subsets Selected by Dynamic Programming.* In Proceedings of the Cancer Bioinformatics Workshop, Cambridge Research Institute, Cambridge, UK, September 2-4, 2010.

[Nelder 65] J. A. Nelder & R. Mead. *A simplex method for function minimization.* Computer Journal, vol. 7, no. 4, pages 308–313, 1965.

[Neumaier 10] A. Neumaier, H. Fendl, H. Schilly & T. Leitner. *VXQR : derivative-free unconstrained optimization based on QR factorizations.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0652-5.

[Newton 11] I. Newton. *De analysi per aequationes numero terminorum infinitas (1669).* London, UK : W. Jones, 1711.

[Omran 10] M. G. H. Omran, V. Gardeux, C. Chelouah, P. Siarry & F. Glover. Using interval scanning to improve the performance of the enhanced unidimensional search method. Research Report, OptTek Systems, Boulder, CO, [En Ligne] Available on : `http://gardeux-vincent.eu/Research/SEUS.pdf`, 2010.

[Orsenigo 08] C. Orsenigo. *Gene selection and cancer microarray data classification via mixed-integer optimization.* Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, vol. 4973, pages 141–152, 2008.

[Papadimitriou 03] C. H. Papadimitriou. Computational complexity. John Wiley and Sons, 2003.

[Pareto 96] V. Pareto. Cours d'économie politique. F. Rouge, 1896.

[Pham 06] D. T. Pham, E. Kog, A. Ghanbarzadeh, S. Otri, S. Rahim & M. Zaidi. *The Bees Algorithm - A Novel Tool for Complex Optimisation Problems.* In Proceedings of the 2nd International Virtual Conference on Intelligent PROduction Machines and Systems (I*PROMS 2006), pages 454–461, Internet Conference, July 3-14, 2006. Elsevier.

[Pochet 04] N. Pochet, F. De Smet, J. A. K. Suykens & B. L. R. De Moor. *Systematic benchmarking of microarray data classification : assessing the role of non-linearity and dimensionality reduction.* Bioinformatics, vol. 20, no. 17, pages 3185–3195, 2004.

[Polak 69] E. Polak & G. Ribiere. *Note sur la convergence de méthodes de directions conjuguées.* Revue Française d'Informatique et de Recherche Opérationnelle, vol. 3, no. 1, pages 35–43, 1969.

[Pomeroy 02] S. L. Pomeroy, P. Tamayo, M. Gaasenbeek, L. M. Sturla, M. Angelo, M. E. McLaughlin, J. Y. H. Kim, L. C. Goumnerova, P. M. Black, C. Lau*et al. Prediction of central nervous system embryonal tumour outcome based on gene expression.* Nature, vol. 415, no. 6870, pages 436–442, 2002.

[Press 07] W. H. Press, S. A. Teukolsky, W. T. Vetterling & B. P. Flannery. Numerical recipes : The art of scientific computing, chapitre 10.2. Golden section search in one dimension, pages 390–395. New York : Cambridge University Press, 3rd edition, 2007.

[Quinlan 86] J.R. Quinlan. *Induction of decision trees.* Machine learning, vol. 1, no. 1, pages 81–106, 1986.

[Quinlan 93] J.R. Quinlan. C4.5 : programs for machine learning. Morgan Kaufmann, 1993.

[Rakotomamonjy 03] A. Rakotomamonjy. *Variable selection using SVM based criteria.* The Journal of Machine Learning Research, vol. 3, pages 1357–1370, 2003.

[Ramaswamy 02] S. Ramaswamy, K. N. Ross, E. S. Lander & T. R. Golub. *A molecular signature of metastasis in primary solid tumors.* Nature genetics, vol. 33, no. 1, pages 49–54, 2002.

[Rechenberg 65] I. Rechenberg. *Cybernetic Solution Path of an Experimental Problem.* Ministry of Aviation, Royal Aircraft Establishment, Library Translation 1122, Farnborough, UK, 1965.

[Rechenberg 73] I. Rechenberg. *Evolutionsstrategie-Optimierum technischer Systeme nach Prinzipien der biologischen Evolution.* PhD thesis, Stuttgart-Bad Cannstatt : Frommann-Holzboog, 1973.

[Reeves 93] C. R Reeves. Modern heuristic techniques for combinatorial problems. Blackwell Scientific Publications, Oxford, 1993.

[Renders 96] J. M. Renders & S. P. Flasse. *Hybrid methods using genetic algorithms for global optimization.* IEEE Transactions on Systems, Man, and Cybernetics, vol. 26, no. 2, pages 243–258, 1996.

[Reynolds 87] C. W Reynolds. *Flocks, herds and schools : a distributed behavioral model.* Computer Graphics, vol. 21, no. 4, pages 25–34, 1987.

[Ritzel 94] B. J. Ritzel, J. W. Eheart & S. Ranjithan. *Using genetic algorithms to solve a multiple objective groundwater pollution containment problem.* Water Resources Research, vol. 30, no. 5, pages 1589–1603, 1994.

[Roberts 91] J. E. Roberts & J. M. Thomas. *Mixed and hybrid methods.* Handbook of numerical analysis, vol. 2, pages 523–639, 1991.

[Ros 08] R. Ros & N. Hansen. *A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity.* Parallel Problem Solving from Nature - PPSN X, vol. 5199, pages 296–305, 2008.

[Rosenblatt 58] F. Rosenblatt. *The perceptron : a probabilistic model for information storage and organization in the brain.* Psychological Review, vol. 65, no. 6, pages 386–408, 1958.

[Rosenbrock 60] H. H. Rosenbrock. *An automatic method for finding the greatest or least value of a function.* The Computer Journal, vol. 3, no. 3, pages 175–184, 1960.

[Rumelhart 86] D. E. Rumelhart, G. E. Hinton & R. J. Williams. Parallel distributed processing : Explorations in the microstructure of cognition, volume 1 : Foundations, chapitre 8 : Learning internal representations by error propagation, pages 318–362. MIT Press, 1986.

[Schaffer 85] J. D. Schaffer. *Multiple Objective Optimisation with Vector Evaluated Genetic Algorithm.* In Proceedings of the First International Conference on Genetic Algorithm and their Applications, pages 93–100, Carnegie-Mellon University, Pittsburgh, PA, USA, July 24-26, 1985.

[Schena 95] M. Schena, D. Shalon, R. W. Davis & P. O. Brown. *Quantitative monitoring of gene expression patterns with a complementary DNA microarray.* Science, vol. 270, no. 5235, pages 467–470, 1995.

[Shah 07] S. Shah & A. Kusiak. *Cancer gene search with data-mining and genetic algorithms.* Computers in Biology and Medicine, vol. 37, no. 2, pages 251–261, 2007.

[Shang 06] Y.-W. Shang & Y.-H. Qiu. *A Note on the Extended Rosenbrock Function.* Evolutionary Computation, vol. 14, no. 1, pages 119–126, 2006.

[Shannon 48] C. E. Shannon. *A mathematical theory of communication.* Bell System Technical Journal, vol. 27, pages 379–423 and 623–656, 1948.

[Shen 04] Q. Shen, J. H. Jiang, C. X. Jiao, G. Shen & R. Q. Yu. *Modified particle swarm optimization algorithm for variable selection in MLR and PLS modeling : QSAR studies of antagonism of angiotensin II antagonists.* European Journal of Pharmaceutical Sciences, vol. 22, no. 2-3, pages 145–152, 2004.

[Shen 08] Q. Shen, W. M. Shi & W. Kong. *Hybrid particle swarm optimization and tabu search approach for selecting genes for tumor classification using gene expression data.* Computational Biology and Chemistry, vol. 32, no. 1, pages 53–60, 2008.

[Shipp 02] M. A. Shipp, K. N. Ross, P. Tamayo, A. P. Weng, J. L. Kutok, R. C. T. Aguiar, M. Gaasenbeek, M. Angelo, M. Reich, G. S. Pinkus*et al. Diffuse large B-cell lymphoma outcome prediction by gene-expression profiling and supervised machine learning.* Nature medicine, vol. 8, no. 1, pages 68–74, 2002.

[Simon 03] R. Simon, M. D. Radmacher, K. Dobbin & L. M. McShane. *Pitfalls in the use of DNA microarray data for diagnostic and prognostic classification.* Journal of the National Cancer Institute, vol. 95, no. 1, pages 14–18, 2003.

[Singh 02] D. Singh, P. G. Febbo, K. Ross, D. G. Jackson, J. Manola, C. Ladd, P. Tamayo, A. A. Renshaw, A. V. D'Amico, J. P. Richie*et al. Gene expression correlates of clinical prostate cancer behavior.* Cancer cell, vol. 1, no. 2, pages 203–209, 2002.

[Sipser 92] M. Sipser. *The history and status of the P versus NP question.* In Proceedings of the 24th annual ACM Symposium on Theory Of Computing (STOC'92), pages 603–618, Victoria, BC, Canada, May 4-6, 1992. ACM.

[Sivagaminathan 07] R. K. Sivagaminathan & S. Ramakrishnan. *A hybrid approach for feature subset selection using neural networks and ant colony optimization.* Expert Systems with Applications, vol. 33, no. 1, pages 49–60, 2007.

[Socha 04] K. Socha. *ACO for continuous and mixed-variable optimization.* Ant colony optimization and swarm intelligence, vol. 3172, pages 53–61, 2004.

[Solis 81] F. J. Solis & R. J.-B. Wets. *Minimization by Random Search Techniques.* Mathematics of Operations Research, vol. 6, no. 1, pages 19–30, 1981.

[Spielberg 79] K. Spielberg. *Enumerative methods in integer programming.* Annals of Discrete Mathematics, vol. 5, pages 139–183, 1979.

[Sprecher 94] A. Sprecher & C. L. Hwang. Resource-constrained project scheduling :(exact methods for the multi-mode case). Springer, 1994.

[Srinivas 95] N. Srinivas & K. Deb. *Multiobjective function optimization using nondominated sorting genetic algorithms.* Evolutionary Computation, vol. 2, no. 3, pages 221–248, 1995.

[Stekel 03] D. Stekel. Microarray bioinformatics. Cambridge University Press, 2003.

[Stone 77] M. Stone. *An Asymptotic Equivalence of Choice of Model by Cross-Validation and Akaike's Criterion.* Journal of the Royal Statistical Society B, vol. 39, no. 1, pages 44–47, 1977.

[Storn 97] R. Storn & K. Price. *Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces.* Journal of Global Optimization, vol. 11, no. 4, pages 341–359, 1997.

[Suganthan 05] P. N. Suganthan, N. Hansen, Liang J. J., K. Deb, Y. P. Chen, A. Auger & S. Tiwari. *Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization.* Rapport technique May-30-05, Nanyang Technological University (NTU), Singapore, 2005. [En Ligne] Available on : `http://web.mysites.ntu.edu.sg/epnsugan/PublicSite/Shared%20Documents/CEC2005/Tech-Report-May-30-05.pdf`.

[Taillard 98] É. D. Taillard, L.-M. Gambardella, M. Gendreau & J.-Y. Potvin. *Programmation à mémoire adaptative.* Calculateurs Parallèles, Réseaux et Systèmes répartis, vol. 10, pages 117–140, 1998.

[Tan 05] P.-N. Tan, M. Steinbach & V. Kumar. Introduction to data mining. Addison Wesley, 2005.

[Tang 07] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen & Z. Yang. *Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization.* Rapport technique, University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL), Héféi, Ānhuī, China, 2007. [En Ligne] Available on : `http://nical.ustc.edu.cn/cec08ss.php`.

[Tang 08] K. Tang. *Summary of Results on CEC'08 Competition on Large Scale Global Optimization.* [Slides] Nature Inspired Computation and Application Lab (NICAL), Department Of Computer Science and Technology, University of Science and Technology of China, 2008. [En Ligne] Available on : `http://nical.ustc.edu.cn/papers/CEC2008_SUMMARY.pdf`.

[Tang 09] K. Tang, L. Xiaodong, P. N. Suganthan, Z. Yang & T. Weise. *Benchmark Functions for the CEC'2010 Special Session and Competition on Large Scale Global Optimization.* Rapport technique, University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL), Héféi, Ānhuī, China, 2009. [En Ligne] Available on : `http://nical.ustc.edu.cn/cec10ss.php`.

[Tebbens 07] J. D. Tebbens & P. Schlesinger. *Improving implementation of linear discriminant analysis for the high dimension/small sample size problem.* Computational Statistics & Data Analysis, vol. 52, no. 1, pages 423–437, 2007.

[Tertois 03] S. Tertois. *Réduction des effets des non-linéarités dans une modulation multiporteuse à l'aide de réseaux de neurones.* PhD thesis, ETSN department, Supélec, Rennes, France, December 2003.

[Thomaz 05] C. E. Thomaz & D. F. Gillies. *A Maximum Uncertainty LDA-Based Approach for Limited Sample Size Problems - With Application to Face Recognition.* In Proceedings of the 18th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2005), pages 89–96, Natal, RN, Brazil, October 9-12, 2005. IEEE Computer Society.

[Tibshirani 96] R. Tibshirani. *Regression shrinkage and selection via the lasso.* Journal of the Royal Statistical Society. Series B (Methodological), vol. 58, no. 1, pages 267–288, 1996.

[Tizhoosh 05] H. R. Tizhoosh. *Opposition-based learning : A new scheme for machine intelligence.* In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation (CICMA'05), pages 695–701, Vienna, Austria, November 28-30, 2005. IEEE.

[Torczon 89] V. J. Torczon. *Multi-Directional Search : A Direct Search Algorithm for Parallel Machines.* PhD thesis, Rice University, Houston, Texas, May 1989.

[Tseng 08] L.-Y. Tseng & C. Chen. *Multiple trajectory search for Large Scale Global Optimization.* In Proceedings of the IEEE Congress on Evolutionary Computation (CEC'08), pages 3052–3059, Hong Kong, China, June 1-6, 2008.

[Turban 10] E. Turban, R. Sharda & D. Delen. Decision support and business intelligence systems. Prentice Hall Press, 9th edition, 2010.

[Vapnik 95] V. N. Vapnik. The nature of statistical learning theory. Springer-Verlag, New York, USA, 1995.

[Venter 01] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt & et al. *The sequence of the human genome.* Science, vol. 291, no. 5507, pages 1304–1351, 2001.

[Wächter 06] A. Wächter & L. T. Biegler. *On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming.* Mathematical Programming, vol. 106, no. 1, pages 25–57, 2006.

[Wang 07] X. Wang, J. Yang, X. Teng, W. Xia & R. Jensen. *Feature selection based on rough sets and particle swarm optimization.* Pattern Recogn. Lett., vol. 28, no. 4, pages 459–471, 2007.

[Wang 08] Y. Wang & B. Li. *A restart univariate estimation of distribution algorithm : sampling under mixed Gaussian and Lévy probability distribution.* In Proceedings of the IEEE Congress on Evolutionary Computation (CEC'08), pages 3917–3924, Hong Kong, China, June 1-6, 2008.

[Wang 09] H. Wang, Z. Wu, S. Rahnamayan & L. Kang. *A Scalability Test for Accelerated DE Using Generalized Opposition-Based Learning.* In ISDA '09 : Proceedings of the 2009 Ninth International Conference on Intelligent Systems Design and Applications, pages 1090–1095, Pisa, Italy, November 30 - December 2, 2009. IEEE Computer Society.

[Wang 10] H. Wang, Z. Wu & S. Rahnamayan. *Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0642-7.

[Weber 10] M. Weber, F. Neri & V. Tirronen. *Shuffle Or Update Parallel Differential Evolution for Large Scale Optimization.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0640-9.

[Weiss 98] S. M. Weiss & N. Indurkhya. Predictive data mining : a practical guide. Morgan Kaufmann, 1998.

[Weiss 99] G. Weiss. Multiagent systems : A modern approach to distributed artificial intelligence. MIT Press, 1999.

[Weston 03] J. Weston, A. Elisseeff, B. Schölkopf & M. Tipping. *Use of the zero norm with linear models and kernel methods.* The Journal of Machine Learning Research, vol. 3, no. 7-8, pages 1439–1461, 2003.

[Whitley 95] D. Whitley, R. Beveridge, C. Graves & K. Mathias. *Test Driving Three 1995 Genetic Algorithms : New Test Functions and Geometric Matching.* Journal of Heuristics, vol. 1, no. 1, pages 77–104, 1995.

[Witten 05] I. H. Witten & E. Frank. Data mining : Practical machine learning tools and techniques. Morgan Kaufmann series in data management systems. Elsevier, 2nd edition, 2005.

[Wolf 99] D. Wolf, P. Keblinski, S. R. Phillpot & J. Eggebrecht. *Exact method for the simulation of Coulombic systems by spherically truncated, pairwise r summation.* The Journal of chemical physics, vol. 110, no. 17, pages 8254–8283, 1999.

[Wolfe 59] P. Wolfe. *The secant method for simultaneous nonlinear equations.* Communications of the ACM, vol. 2, no. 12, pages 12–13, 1959.

[Wolpert 96] D. H. Wolpert, W. G. Macready, H. David & G. William. *No free-lunch theorems for search.* Rapport technique SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM, 1996.

[Xing 01] E. P. Xing, M. I. Jordan & R. M. Karp. *Feature Selection for High-Dimensional Genomic Microarray Data.* In Proceedings of 18th International Conference on Machine Learning (ICML 2001), pages 601–608, Williams College, Williamstown, MA, USA, June 28 - July 1, 2001. Morgan Kaufmann.

[Xu 09] P. Xu, G. N. Brock & R. S. Parrish. *Modified linear discriminant analysis approaches for classification of high-dimensional microarray data.* Computational Statistics & Data Analysis, vol. 53, no. 5, pages 1674–1687, 2009.

[Yang 97] Y. Yang & J. O. Pedersen. *A Comparative Study on Feature Selection in Text Categorization.* In Proceedings of 14th International Conference on Machine Learning (ICML 1997), pages 412–420, Nashville, TN, USA, July 8-12, 1997. Morgan Kaufmann.

[Yang 98] J. Yang & V. G. Honavar. *Feature Subset Selection Using a Genetic Algorithm.* IEEE Intelligent Systems, vol. 13, no. 2, pages 44–49, 1998.

[Yang 10] Z. Yang, K. Tang & X. Yao. *Scalability of generalized adaptive differential evolution for large-scale continuous optimization.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0643-6.

[Ye 05] J. Ye & B. Yu. *Characterization of a family of algorithms for generalized discriminant analysis on undersampled problems.* Journal of Machine Learning Research, vol. 6, pages 483–502, 2005.

[Zhao 10] S.-Z. Zhao, P. Suganthan & S. Das. *Self-adaptive differential evolution with multi-trajectory search for large-scale optimization.* Soft Computing - A Fusion of Foundations, Methodologies and Applications, 2010. [To Appear] DOI : 10.1007/s00500-010-0645-4.

[Zitzler 00] E. Zitzler, K. Deb & L. Thiele. *Comparison of multiobjective evolutionary algorithms : Empirical results.* Evolutionary Computation, vol. 8, no. 2, pages 173–195, 2000.