Isomorphisme Inexact de Graphes par Optimisation Évolutionnaire

Thomas Bärecke

École Doctorale d'Informatique, Télécommunication et Électronique de Paris

Isomorphisme Inexact de Graphes par Optimisation Évolutionnaire

Thèse de Doctorat de l'Université Paris VI

présentée pour obtenir le grade de

Docteur de l'Université Pierre et Marie Curie - Paris VI
(spécialité informatique)

par

Thomas Bärecke

Soutenue publiquement le 22 octobre 2009
devant le jury composé de

Rapporteurs :

Evelyne Lutton	Directeur de recherche, INRIA
Michèle Sebag	Directeur de recherche, CNRS

Examinateurs :

Bernadette Bouchon-Meunier	Directeur de recherche, CNRS
Marcin Detyniecki	Chargé de recherche, CNRS
Patrick Gallinari	Professeur, Université Paris VI
El-Ghazali Talbi	Professeur, Université Lille I

Bärecke, Thomas :
Isomorphisme inexact de graphes par optimisation évolutionnaire
Thèse de doctorat, Université Pierre et Marie Curie - Paris VI, 2009.

Résumé

L'isomorphisme inexact de graphes est un problème crucial pour la définition d'une distance entre graphes, préalable nécessaire à une multitude d'applications allant de l'analyse d'images à des applications biomédicales en passant par la reconnaissance optique de caractères. Ce problème est encore plus complexe que celui de l'isomorphisme exact. Alors que ce dernier est un problème de décision de complexité au moins de classe P et qui ne s'applique qu'à des graphes exactement identiques, l'isomorphisme inexact est un problème combinatoire de complexité de classe NP qui permet de prendre en compte des perturbations dues au bruit, qui apparaissent fréquemment dans les applications réelles.

Dans ce cadre, nous choisissons d'étudier une solution basée sur les algorithmes génétiques pouvant être appliquée à l'isomorphisme exact et inexact. Nous proposons des opérateurs de croisement généraux pour tout problème représenté par un codage de permutation, ainsi que des opérateurs spécifiques à l'isomorphisme de graphes qui exploitent une heuristique gloutonne. Nous réalisons une étude exhaustive pour comparer ces opérateurs avec les opérateurs existants, soulignant leurs propriétés, avantages et inconvénients respectifs.

Nous étudions par ailleurs plusieurs pistes d'amélioration de l'algorithme, en théorie ou en pratique, considérant successivement les objectifs d'accélération de l'exécution, d'augmentation de la précision et de garantie de résultat optimal. Nous proposons pour cela de combiner l'approche proposée avec d'autres techniques telles que des heuristiques générales comme la recherche locale, des heuristiques dédiées comme l'algorithme A*, et des outils pratiques comme la parallélisation.

Ces travaux conduisent à la définition d'une méthode générique pour la résolution de tous les problèmes d'isomorphismes de graphes, qu'il s'agisse d'isomorphismes exact ou inexact, d'isomorphismes de graphes de même taille ou d'isomorphismes de sous-graphes. Nous illustrons enfin la validité de cette solution générale par trois applications concrètes issues de domaines différents, la recherche d'images et la chimie, qui présentent chacune des caractéristiques spécifiques, utilisant des graphes attribués ou non, soumis aux perturbations plutôt structurelles ou au niveau d'attributs.

Mots-clés: isomorphisme inexact de graphes, distance de graphes, algorithme évolutionnaire, recherche locale, algorithme mémétique, optimisation combinatoire, opérateurs de croisement, approche hybride, parallélisation, méta-heuristique

Evolutionary Optimisation for Inexact Graph Isomorphism

Abstract

The solution to the inexact graph matching problem is the key for defining any type of graph distance. It is even more complex than the exact graph isomorphism problem. On the one hand, inexact graph matching is a combinatorial optimization problem in NP taking into account perturbations inherent in noisy real world environments. Exact graph matching, on the other hand, is a decision problem for which it has not yet been shown if its complexity class is P and which applies only to exactly identical graphs.

In this thesis, we study an approach based on genetic algorithms addressing both exact and inexact isomorphisms. We introduce several new crossover operators, some more general for use with any kind of permutation encoding, some specialized which include a greedy heuristic specific to graph matching. We conduct an exhaustive study in order to compare these operators with the existing ones, underlining their respective characteristics, advantages and disadvantages.

Furthermore, we examine several aspects for enhancing the algorithm, both theoretical and practical ones, leading to faster execution, better precision or even the assurance of finding the global optimum. We combine the genetic algorithm with generalized black-box heuristics, such as local search, specialized heuristics such as the A* algorithm or practical tools like parallelization techniques. Our final aim is to present a method addressing all different types of graph matching problems, i.e. exact and inexact, isomorphisms of graphs having the same size and sub-graph isomorphisms. We illustrate the generality of our approach with three applications with very distinct properties which cover the different problem types.

Keywords: inexact graph matching, graph distance, evolutionary algorithm, local search, memetic algorithm, combinatorial optimization, crossover operators, hybrid method, parallelization, metaheuristic

Remerciements

Cette thèse n'aurait pas vu le jour sans l'aide d'un grand nombre de personnes que je tiens à remercier ici.

J'aimerais exprimer toute ma gratitude à Marcin Detyniecki pour son indéfectible soutien tout au long de ma thèse. Il a activement contribué à ma formation scientifique en me permettant de participer à des écoles d'été et des congrès, aussi lointains qu'ils fussent, et en menant tambour battant de fructueuses discussions jusque sur les courts de tennis. Mais il a également su m'ouvrir d'autres portes, en résolvant tous les problèmes que j'ai pu rencontrer et en m'offrant une autre vision de la culture française qui m'a permis de m'adapter plus facilement. Je le remercie enfin tout particulièrement pour sa grande disponibilité et la talentueuse pondération de son regard sur mon travail entre critique exigeante et enthousiasme stimulant.

Je remercie très chaleureusement Bernadette Bouchon-Meunier pour m'avoir suivi tout au long de mon parcours, pour m'avoir fait bénéficier de ses conseils précieux et en particulier pour ses encouragements pendant la « phase terminale » de rédaction.

J'adresse également mes remerciements les plus sincères à Evelyne Lutton pour avoir accepté d'être rapporteur de ma thèse. Ses commentaires et notre discussion pendant la soutenance m'ont permis d'améliorer mes connaissances des algorithmes co-évolutionnaires.

Je suis très honoré que Michèle Sebag ait accepté d'être rapporteur de ma thèse malgré ses importantes contraintes, professionnelles et personnelles. Ses très nombreux commentaires m'ont permis non seulement d'améliorer mon travail mais aussi de découvrir de nouvelles idées.

J'apprécie tout particulièrement qu'El-Ghazali Talbi et Patrick Gallinari aient accepté d'être examinateurs de ma thèse et aient ainsi contribué à élargir mon horizon avec leurs commentaires lors de ma soutenance.

Je tiens à remercier sincèrement l'équipe LOFTI pour son accueil chaleureux, les discussions fructueuses lors et en dehors des réunions d'équipes, les pauses café, chocolat et plus récemment fruits. J'aimerais remercier en particulier Marie-Jeanne et Christophe pour leur disponibilité et leurs solutions à toutes mes questions que ce soit au sujet de LaTeX, Matlab, ou n'importe quel autre énigmatique outil.

J'ai eu la chance de travailler dans des conditions optimales et d'avoir eu beaucoup d'aide au niveau administratif. J'aimerais exprimer ma gratitude au personnel administratif du LIP6 et de l'Université ainsi qu'aux ingénieurs systèmes, qui sont d'ailleurs disponibles même le week-end. Merci à Ghislaine, Jacqueline, Thierry, Christophe, Jean-Pierre, Vincent... J'aimerais également remercier Ghislaine Hannot et Marilyn Galopin d'avoir fait tout ce qui était nécessaire pour que ma soutenance ait lieu malgré un dossier très en retard.

Je voudrais également remercier les personnes extérieures au LIP6 avec lesquelles j'ai eu l'occasion de collaborer pendant ma thèse. Grâce aux travaux de Stefano Berretti, je me suis lancé dans

le domaine de l'appariement de graphes. Je remercie également Christine Solnon, que j'ai pu rencontrer à l'occasion d'un séminaire en 2008, pour notre discussion très bénéfique.

Bien qu'un lien avec le présent manuscrit soit inexistant, je tiens à remercier l'équipe du CEA, François Werkoff, Christine Mansilla et Sophie Avril, avec qui j'ai eu l'occasion de travailler pendant la première année de ma thèse sur un sujet très éloigné mais pas pour autant moins intéressant.

J'adresse un grand merci à tous ceux qui ont, à plusieurs reprises, relu ce manuscrit pour éliminer les coquilles et améliorer le style : merci à Marie-Jeanne, Marie, Jean-Marc, et bien sûr, Bernadette et Marcin.

Je tiens à remercier tous ceux qui m'ont accompagné sur le chemin vers la thèse et qui m'ont donné envie de me lancer dans le domaine de l'intelligence artificielle, pendant mes études à Magdebourg : j'adresse ma gratitude à Christian Borgelt, Rudolf Kruse, Andreas Nürnberger qui m'a d'ailleurs beaucoup aidé lors de ma venue en tant qu'étudiant Erasmus, Tobias Scheffer,... . Je remercie également tous ceux qui m'ont enseigné l'outillage d'informaticien, à commencer par Ralf Feuerstein - toujours avec l'autocollant « Des ordinateurs nous aident à résoudre des problèmes que nous n'aurions jamais eus sans eux » - en passant par les équipes du Fraunhofer IFF et d'icubic - les lundis sans Burger King, c'est dur au début.

J'adresse ma gratitude à tous ceux qui devraient figurer ici et que j'ai oubliés. J'espère qu'ils ne m'en voudront pas trop.

Je remercie enfin mes proches et amis pour leur soutien sans faille, et bien sûr Noemí, pour être toujours à mes côtés.

A Noemí

Table des matières

Sigles

Nous utilisons des sigles qui proviennent des termes anglais (en italique). L'équivalent de ces termes en français est donné en dessous.

AES	*Average (fitness function) Evaluations to Success* Nombre moyen d'appels à la fonction d'évaluation nécessaires pour trouver la bonne solution
ARG	*Attributed Relational Graph* Graphe relationnel valué
COIL	Columbia Object Image Library
CX	*Cycle crossover* Croisement cyclique
DPX	*Distance Preserving crossover* Croisement préservant la distance
EA	*Evolutionary Algorithm* Algorithme évolutionnaire
ES	*Evolution Strategy* Stratégie d'évolution
GA	*Genetic Algorithm* Algorithme génétique
GGA	*Generational GA* GA générationnel
GP	*Genetic Programming* Programmation génétique

HST *Holland's Schema Theorem*
Théorème du schéma de Holland

HUX *Half-Uniform crossover*
Croisement semi-uniforme

k-NN *k-Nearest-Neighbors*
k plus proches voisins

NDPX *Node DPX*
DPX basé sur les nœuds

NFL *No-Free-Lunch theorem*
Théorème *no-free-lunch*

OX *Order(-based) crossover*
Croisement basé sur l'ordre

PBX *Strict Position-Based crossover*
Croisement basé strictement sur la position absolue

PMX *Partially Matched crossover*
Croisement de correspondance partielle

POS *Position based crossover*
Croisement basé sur la position

QAP *Quadratic Assignment Problem*
Problème de l'affectation quadratique

SDPX *Signature DPX*
DPX basé sur une signature

SGGA *Sorted GGA*
GGA trié

SR *Success Rate*
Taux de succès

SSGA *Steady-state GA*
GA steady-state

TSP *Traveling Salesman Problem*
Problème du voyageur de commerce

UGGA *Unique GGA*
GGA unique

UOX *Uniform Order-based crossover*
Croisement uniforme basé sur l'ordre

UPBX *Uniform strict Position-Based crossover*
Croisement uniforme basé strictement sur la position absolue

UPMX *Uniform Partially Matched crossover*
Croisement uniforme à correspondance partielle

USGGA *Unique sorted GGA*
GGA unique et trié

UX *Uniform crossover*
Croisement uniforme

Symboles

$*$	symbole *don't care*
E	espérance mathématique
G	graphe
H	schéma
I	intensité de sélection
Ω	espace phénotypique, ensemble de toutes les solutions possibles du problème
α	fonction qui affecte un attribut à un sommet
β	fonction qui affecte un attribut à une arête
δ_E	distance entre arêtes
δ_H	distance de Hamming
δ_V	distance entre sommets
δ	distance (en général ou entre graphes)
λ	nombre d'enfants dans les stratégies d'évolution
$\mathcal{A}_E$	ensemble des attributs d'arêtes
$\mathcal{A}_V$	ensemble des attributs de sommets
$\mathcal{A}$	alphabet
$\mathcal{E}$	ensemble des arêtes d'un graphe
$\mathcal{F}$	ensemble de valeurs de *fitness* d'une génération
$\mathcal{G}$	espace génotypique, ensemble de tous les chromosomes possibles
$\mathcal{O}$	complexité algorithmique
$\mathcal{V}$	ensemble des sommets d'un graphe
μ	nombre de parents dans les stratégies d'évolution
σ	écart type
τ	tolérance numérique, seuil de précision des calculs de nombres à virgule flottante
ε	longueur d'intervalle du bruit ajouté (loi uniforme)

b	opération élémentaire définissant le voisinage pour la recherche locale
c	enfant
e	une arête
f	fonction d'évaluation (*fitness*)
g	fonction générique
l_d	longueur définie d'un schéma
l	longueur d'un chromosome
m	appariement entre les sommets de deux graphes
n	taille d'un graphe
$o(H)$	ordre d'un schéma
p_c	probabilité de croisement
p_d	probabilité de perturbation d'un schéma
p_m	probabilité de mutation
p_{AES}	p-valeur du nombre d'appels à la fonction d'évaluation après Kruskal-Wallis
p_{ARS}	p-valeur du temps de calcul après Kruskal-Wallis
p_{RL}	probabilité de recherche locale
p_{SR}	p-valeur du taux de succès après Kruskal-Wallis
p	parent
s_p	taille de la population
sim	similarité
t	temps, habituellement le numéro de la génération
v	un sommet
w	pondération entre la distance de sommets et la distance d'arêtes dans la méthode de Berretti

Table des figures

Chapitre 1

Introduction

1.1 Motivation

Les graphes constituent un outil très général pour décrire des structures. Ils permettent notamment la modélisation des entités composées et structurées. Un graphe contient un ensemble de sommets qui sont reliés par des arêtes. Il peut modéliser par exemple le plan d'une ville et servir pour chercher un chemin d'un point A à un point B. Dans ce cas les sommets sont des endroits marquants ou simplement des carrefours. Les arêtes sont données par les rues qui les relient. On peut attribuer des étiquettes aux arêtes ou aux sommets, par exemple la distance ou le nom de l'endroit, respectivement. D'une manière très similaire, le réseau des transports publics peut être décrit par un graphe. Par ailleurs, la description d'une molécule est aussi possible : les sommets sont alors les atomes et les liaisons sont modélisées par les arêtes. Des relations plus complexes dans des macromolécules peuvent également être prises en compte (Artymiuk et al., 2005). Il y a de nombreuses autres applications comme par exemple des circuits électroniques, tous les types de schémas, etc. On peut même décrire des images à l'aide de leurs composants. Dans ce cas, on détermine des régions pertinentes (qui correspondent dans le cas idéal à des objets) qui deviennent des sommets et les arêtes sont les relations spatiales entre ces régions. Comme la définition d'un graphe se place à un niveau abstrait, les graphes peuvent modéliser n'importe quelle entité du monde réel sous condition qu'elle puisse être décomposée en sommets - connectés ou non par des relations quelconques, par exemple spatiales. L'avantage d'une telle description abstraite est son universalité. Des problèmes dont on cherche la solution dans les différentes applications font souvent partie de problèmes de graphes généraux. Si l'on trouve une solution ou un algorithme au niveau théorique, alors toutes les applications en profitent directement.

La richesse en termes d'expressivité a pour effet que les problèmes de graphes sont le plus souvent d'une complexité assez élevée. Dans ce qui suit, nous donnons d'abord quelques exemples de problèmes classiques que l'on peut rencontrer à l'intérieur d'un graphe : une famille regroupe les

différents problèmes de routage cherchant des chemins possédant des propriétés spécifiques, comme le chemin le plus court ou simplement un chemin possible. Comme fondation historique de la théorie des graphes, on cite souvent les sept ponts de Königsberg d'Euler qui remontent au dix-huitième siècle. Le problème consiste à trouver un chemin permettant de traverser tous les sept ponts de la ville exactement une fois. Dans le cadre abstrait, on cherche un parcours fermé dans un graphe qui contient toutes les arêtes exactement une fois. Ce parcours est dénommé chemin eulérien. Il existe si le degré de chaque sommet est pair (sous la condition que le graphe soit connexe). Le problème est donc assez facile. Un problème largement plus complexe mais en premier regard similaire est l'existence des chemins hamiltoniens. Au lieu de chercher un circuit contenant toutes les arêtes, on cherche un parcours qui contient tous les sommets exactement une fois. Ce problème est NP-complet. Il est lié au problème d'optimisation du problème du voyageur de commerce (TSP) qui cherche le chemin hamiltonien le plus court. D'autres problèmes classiques incluent la coloration de graphes, utilisée par exemple pour des cartes, la recherche de la clique maximale (le plus grand graphe complet qui soit un sous-graphe du graphe donné), l'optimisation des flux dans un réseau, et l'analyse par graphes de visibilité dont le problème des gardiens de musée.

Les problèmes précédents s'intéressent uniquement à un seul graphe. Il existe une deuxième famille de problèmes à plusieurs graphes. Ils s'intéressent avant tout à la question de l'égalité ou de similarité de graphes donnés. On se demande alors si les structures sont comparables et quelle est leur distance.

En premier lieu, il existe le problème de l'égalité exacte. Pour les données non-structurées ceci est très simple. Si l'on cherche à comparer des vecteurs, alors on compare leurs composants un par un. Dans ce cas, on sait quel élément il faut comparer à quel autre. Pour les graphes, on ne possède pas cette information. Chaque sommet d'un graphe peut être mis en relation avec n'importe quel nœud dans le deuxième. Parmi toutes les combinaisons possibles, la question de l'égalité structurelle exacte est formalisée par le problème d'isomorphisme de graphes.

On peut souligner la grande importance du problème d'isomorphisme particulièrement dans le domaine de la reconnaissance de formes. Parmi les nombreuses applications concrètes on trouve la classification d'images segmentées en général (Harchaoui & Bach, 2007) ou des applications plus spécifiques comme la détection d'humains (Thome et al., 2008) ou la classification d'images cérébrales (Boeres et al., 2004). De fait, ce problème a déjà une longue tradition datant de plus de trente ans. Conte et al. (2004) présentent un état de l'art couvrant cette période. En effet, on cherche à identifier une forme à partir de sa structure. Si l'on souhaite par exemple reconnaître une table, on peut utiliser l'information qu'elle est composée d'un plateau et de quatre pieds, qui se trouvent aux quatre coins. Cette approche structurée permet des comparaisons plus fines qu'une approche globale. Il est également possible d'apparier les objets, c'est-à-dire d'extraire l'information sur les sommets mis en relation, au lieu de simplement donner une seule valeur (oui/non dans le cas de l'existence d'isomorphisme ou bien une distance ou une mesure de similarité).

Une formulation courante est que l'on recherche un appariement entre les sommets qui préserve la structure (y compris les attributs). Il faut noter que ce problème est bien différent du problème de recherche d'appariements dans un graphe (cf. Sec. 1.3, p. 8). Pour les distinguer, on ajoute souvent le mot bipartite parce que la plupart des applications d'appariement des sommets d'un graphe se basent sur des graphes bipartites. Deuxièmement, la question de l'inclusion d'un graphe dans un autre correspond au problème d'isomorphisme de sous-graphes. Comme dernier exemple, on peut noter la recherche du sous-graphe maximal commun qui vise à identifier des similarités plus générales entre deux graphes.

Le calcul d'une distance dans des domaines vectoriels (voire des chiffres) ne pose pas de problème particulier à part le choix de la mesure tandis que dans le cas de graphes il y a deux difficultés principales. Le premier problème concerne la définition d'une distance entre structures qui soit assez générale et remplisse les conditions d'une métrique. Le deuxième est la complexité du calcul pour les distances proposées. De fait, une distance euclidienne sur un vecteur, même de très grande taille, se calcule très rapidement. La distance d'édition de graphes (cf. Sec. 1.4.4, p. 10) nécessite l'optimisation sur toutes les séquences d'édition possibles, ce qui est un problème difficile. Il n'est donc applicable que sur de très petits graphes. Ci-dessous, nous introduisons quelques problèmes liés.

Bien que notre travail aborde uniquement les différents problèmes d'isomorphisme de graphes, ces derniers ne sont pas les seuls problèmes dans la famille des problèmes à deux ou plusieurs graphes. Il existe également des problèmes d'identification de sous-structures, normalement dans un contexte de classification, ou encore l'inclusion d'une structure dans une autre. Dans l'exemple des villes on peut donner par exemple le plan d'un quartier uniquement et chercher la ville dans laquelle il se trouve. Il existe également des problèmes de recherche des sous-graphes communs entre deux graphes, ou la fouille des bases de graphes pour l'identification des sous-graphes fréquents (Inokuchi et al., 2005).

Dans le cadre des problèmes d'isomorphisme de graphes, on distingue les problèmes exacts des problèmes inexacts. L'isomorphisme exact nécessite que les deux graphes soient exactement isomorphes. Dans le cas inexact en revanche des variations des attributs ou même de la structure sont acceptables. Par conséquence, le cas inexact est souvent formulé comme problème de minimisation de distance tandis que l'isomorphisme exact est sous forme d'un problème de décision dont la solution est oui ou non. Ceci ne permet donc pas le calcul d'une distance. Les algorithmes nécessaires pour les deux versions sont très différents. Tandis que le cas exact permet de filtrer les sommets (en fonction des caractéristiques telle que l'étiquette, le degré, le degré des voisins, par exemple) au préalable afin de réduire le nombre d'affectations possibles, dans le cas inexact une telle approche pourrait exclure des solutions valables. En général, les algorithmes qui résolvent le problème d'isomorphisme exact ne sont pas applicables au problème inexact. Quant aux algorithmes qui résolvent le problème inexact, ils sont applicables au problème exact mais ils montrent des performances

médiocres sur le problème exact comparé aux méthodes spécialisées pour ce dernier.

Quand a-t-on besoin de traiter des problèmes inexacts ? En général, ceci est le cas quand le graphe est étiqueté avec des attributs numériques continus : de petites variations dans les attributs doivent alors être tolérées. En effet, il peut s'agir simplement de bruit. Par exemple dans le cadre du traitement d'images, deux images de la même scène ne sont jamais exactement (dans le sens numérique) égales. De fait, des variations de lumière, d'angle et la présence du bruit pendant la prise de la photo sont inévitables. De plus, les objets sur la photo peuvent évoluer, par exemple s'il s'agit de personnes. Non seulement les caractéristiques utilisées sont soumises à l'incertitude, mais encore la segmentation de l'image. Les algorithmes de segmentation automatique ne donnent pas des résultats parfaits. La plupart des approches sur-segmentent les images et les méthodes d'extraction de points d'intérêt sont aussi influencées par le bruit. Pour toutes ces raisons, il n'existe aucun modèle de graphe capable de capturer la totalité du contenu d'une image donnée. En outre, il ne suffit pas de décrire une image, il faut aussi avoir une mesure de similarité de haut niveau. En effet, on ne s'intéresse pas aux caractéristiques de bas niveau comme la couleur, la texture et la forme, mais plutôt aux similarités entre les objets présents dans les images. La mesure doit être robuste au bruit, ce qui nous ramène au problème de l'isomorphisme inexact de sous-graphes.

Nous abordons le problème d'isomorphisme de graphes, plus précisément le problème d'isomorphisme inexact de sous-graphes. L'isomorphisme inexact de graphes est un problème NP-difficile d'optimisation combinatoire (Tsai & Fu, 1979; Köbler et al., 1993). Le terme inexact signifie que, malgré de petites différences, on souhaite considérer deux graphes comme isomorphes s'il s'agit des graphes suffisamment proches, en particulier, si les différences sont dues au processus de l'extraction de caractéristiques ou au bruit. Des méthodes exhaustives peuvent seulement résoudre des problèmes de très petite taille. La nature imprécise du problème favorise l'application des algorithmes approximatifs. En effet, même une solution numériquement optimale n'est pas forcément la solution qui met le mieux en correspondance les deux objets d'origine. Elle peut être « meilleure » que cette dernière grâce au bruit. Le manque d'une garantie d'optimalité importe donc peu. Nous proposons l'utilisation des algorithmes génétiques. Ces sont des méthodes d'optimisation inspirées de la théorie de l'évolution darwinienne. Dans la version que nous avons réalisée, qui constitue un algorithme *mémétique*, les individus « apprennent » après leur création (Reynolds, 1994; Ong et al., 2006; Smith, 2007). Pour cela, une heuristique de recherche locale est appliquée. Nous avons choisi des algorithmes évolutionnaires parce que, d'une part il a été montré qu'ils peuvent efficacement résoudre des problèmes d'optimisation combinatoire, et d'autre part ils sont capable de livrer une solution candidate assez bonne à n'importe quel moment. En réalité, ceci est un grand avantage. Si l'on cherche par exemple à optimiser l'itinéraire pour des facteurs (techniciens à domicile etc.) pour le lendemain, alors que l'on connaît les adresses à livrer peu à l'avance, il faut quand même pouvoir donner un résultat avant qu'ils ne commencent leur travail. Si l'on a un grand nombre d'endroits, des contraintes temporelles, etc., il se peut qu'avec une méthode exacte on doive attendre plus long-

temps et les facteurs ne travaillent pas du tout. Dans ce cas, on peut se baser sur le meilleur itinéraire trouvé jusque-là, même si celui n'est pas toujours optimal.

Comment obtient-on un graphe à partir d'un objet réel et est-ce que ce modèle est pertinent ? Il faut identifier les composantes de l'objet, leurs relations, et définir des caractéristiques pertinentes. Dans de nombreux domaines, la modélisation est un problème difficile en soi. Considérons par exemple le domaine d'images. Une image n'est rien d'autre qu'un ensemble de pixels dont chacun possède une couleur. Il se pose donc naturellement la question de savoir pourquoi une représentation sous forme de graphe est pertinente. La raison est liée au fait que les valeurs des pixels ne disent pas grand chose sur la perception de l'image par un humain. En particulier, l'homme est capable d'identifier des objets et ses relations. Le but est donc de modéliser l'image en utilisant des caractéristiques locales de haut niveau et les rapports entre elles pour permettre une meilleure compréhension du contenu. Si l'on a par exemple une image d'un pré sous un ciel bleu, alors il contient un groupe de pixels qui correspond au pré et un autre groupe qui correspond au ciel. Le graphe a donc deux sommets et la relation entre les sommets est le ciel est au-dessus du pré.

Dans cette thèse, nous nous intéressons à l'application d'un algorithme évolutionnaire (EA) pour le problème général de l'isomorphisme de graphes. Nous étudions de façon très détaillée la manière de trouver une bonne implémentation d'un tel algorithme. Chaque choix d'un opérateur et d'un paramètre est validé par des expériences exhaustives sur une base de graphes synthétiques.

En ce qui concerne le cadre algorithmique, nos contributions principales sont, d'une part l'introduction de nouveaux opérateurs génétiques, et d'autre part la combinaison des approches inexactes (évolutionnaires) avec des approches exactes (A*). Notre but est de proposer un cadre général qui permette au praticien d'appliquer facilement nos méthodes. Pour cela, nous menons des études pour chaque étape importante de l'algorithme (hors sélection parce que de nombreux travaux existent). Nous effectuons une comparaison exhaustive de nos opérateurs et des opérateurs existants (cf. Ch. 3, p. 59). De fait, notre logiciel est très générique et permet d'intégrer de nouveaux opérateurs facilement. Dans ce contexte, nous étudions les effets de l'intégration d'une méthode de recherche locale en utilisant quatre stratégies différentes (cf. Ch. 4, p. 109). Nous incluons des commentaires sur les mesures de performances et la façon de comparer différentes approches (évolutionnaires ou non ; cf. Ch. 2, p. 23). Nous appliquons notre approche dans des contextes divers, dont notamment des graphes synthétiques, des molécules, des images en trois dimensions et des images de la base COIL.

Cette thèse est structurée comme suit : après un chapitre introductif, le deuxième chapitre s'intéresse aux EA pour l'isomorphisme de graphes. Nous définissons les principes de base nécessaires afin d'appliquer l'algorithme sur le problème. Dans le chapitre suivant, nous étudions les différents opérateurs et les paramètres de l'algorithme. Ceci a pour but l'optimisation des performances dans le cadre des graphes synthétiques. Nous attachons une attention particulière aux opérateurs de croisement. Le quatrième chapitre est dédié à la combinaison des algorithmes génétiques avec des heuristiques de recherche locale. Nous évaluons en particulier les différentes stratégies d'intégration. Puis,

nous abordons la question de la parallélisation qui semble tout à fait opportune pour accélérer l'exécution sur des grands graphes surtout en vue de la diffusion croissante des ordinateurs parallèles. Le chapitre 6 aborde la possibilité de garantir le résultat optimal, ce qui n'est à priori pas possible avec un algorithme génétique. Nous proposons la combinaison avec un algorithme performant de recherche d'arbre. Ensuite nous illustrons l'application de notre algorithme sur des données réelles, dont notamment des modèles d'images en trois dimensions, des graphes issus d'une triangulation à partir de la base d'image COIL, et des molécules. Le travail se termine avec les conclusions et perspectives.

Dans ce chapitre, nous introduisons d'abord les notions élémentaires concernant les graphes. Puis, nous détaillons le problème d'isomorphisme et les problèmes liés dont particulièrement la notion d'une distance entre graphes. Nous nous intéressons par la suite à la complexité algorithmique ainsi qu'aux algorithmes existants. Finalement, nous introduisons le concept général des métaheuristiques comme heuristiques universelles et en particulier les approches inspirées par l'évolution.

1.2 Graphes

Dans cette partie, nous caractérisons d'abord les différentes classes de graphes afin d'établir la définition formelle du problème traité dans cette thèse. Les définitions qui suivent sont reproduites principalement de Diestel (2005).

Définition 1.2.1. On définit un graphe comme une paire $G = (\mathcal{V}, \mathcal{E})$ de deux ensembles avec $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Les éléments de $\mathcal{V}$ sont les sommets ou nœuds, tandis que $\mathcal{E}$ est l'ensemble des arêtes.

Des graphes permettent la modélisation des relations entre des objets. Formellement, il n'y a pas de contraintes sur l'ensemble des arêtes. Il est notamment possible qu'$\mathcal{E}$ contienne des arêtes multiples entre deux sommets (et donc identiques) ainsi que des boucles ($e = (v, v)$) qui relient un sommet v à lui-même. En général, on suppose implicitement que des graphes ne contiennent ni boucles ni arêtes multiples. Dans le cas contraire, on le mentionne explicitement. Des graphes contenant des arêtes multiples sont dénommés multigraphes.

La taille d'un graphe n est définie par le nombre de ses sommets. On écrit également $|G|$ et on note $\|G\|$ le nombre d'arêtes. Chaque arête peut être décrite par les sommets qu'il joint. On dit que les sommets v_1 et v_2 sont *incidents* avec l'arête $e = (v_1, v_2)$. Le nombre total d'arêtes *incidentes* à un sommet est dénommé son degré. Il existe des graphes orientés et non-orientés. Dans les graphes orientés, les arêtes ont une direction et sont dénommées arcs. À la différence des graphes non-orientés, l'arc (v_1, v_2) n'est pas équivalent à l'arc (v_2, v_1).

Il existe diverses classes de graphes ayant des caractéristiques spécifiques. Par exemple, un graphe est appelé *(k-)régulier* si le degré de tous ses sommets est une constante (k). Un graphe

complet est un graphe entièrement connecté. Deux autres classes assez importantes en pratique sont les graphes *planaires* et les graphes *bipartites*. Les graphes planaires sont intuitivement des graphes que l'on peut dessiner sans intersections des arêtes. Formellement, il s'agit des graphes qui peuvent être représentés sur une sphère sans intersection d'arêtes. Cette notion peut être généralisée pour des surfaces plus complexes, comme par exemple le tore :

Définition 1.2.2 (Genre d'une surface)**.** Le genre d'une surface est le nombre maximum de courbes simples fermées sans points communs que l'on peut tracer sur cette surface sans la morceler (Bouvier et al., 2009).

Définition 1.2.3 (Genre d'un graphe)**.** Le genre d'un graphe est le plus petit genre d'une surface orientable sur laquelle le graphe puisse être représenté sans intersection d'arêtes (Bouvier et al., 2009).

Un graphe est donc planaire si et seulement si son genre est égal à zéro.

Les graphes bipartites sont des graphes dont il existe une bipartition de sommets en deux sous-ensembles ($\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$ et $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$), telle qu'il n'y ait pas d'arêtes connectant deux sommets appartenant à un seul sous-ensemble. Formellement une condition suffisante est la suivante : $\forall (v_1, v_2) \in \mathcal{E} : v_1 \in \mathcal{V}_1 \Leftrightarrow v_2 \in \mathcal{V}_2$.

Définition 1.2.4 (Sous-graphe)**.** Soit $G = (\mathcal{V}, \mathcal{E})$ un graphe. $G' = (\mathcal{V}', \mathcal{E}')$ est dénommé sous-graphe de G ($G' \subseteq G$) si $\mathcal{V}' \subseteq \mathcal{V}$ et $\mathcal{E}' \subseteq \mathcal{E}$

Définition 1.2.5 (Sous-graphe induit par les sommets)**.** Soient $G = (\mathcal{V}, \mathcal{E})$ un graphe et $G' = (\mathcal{V}', \mathcal{E}')$ un sous-graphe de G. Si $\forall v_1, v_2 \in \mathcal{V}' : (v_1, v_2) \in \mathcal{E} \Rightarrow (v_1, v_2) \in \mathcal{E}'$ alors les sommets $\mathcal{V}'$ induisent le sous-graphe G' dans G.

Dans de nombreuses applications concrètes, les objets, ainsi que les relations, ont des caractéristiques importantes. Afin de les inclure dans le modèle, on affecte des attributs aux nœuds et/ou aux arêtes. Si les attributs sont exclusivement numériques, on obtient un graphe pondéré. Dans le cas plus général on parle d'un graphe étiqueté ou d'un graphe relationnel valué (ARG).

Définition 1.2.6 (Graphe étiqueté (ARG))**.** Un graphe relationnel valué est un quadruplet $G = (\mathcal{V}, \mathcal{E}, \alpha, \beta)$. Les applications $\alpha : \mathcal{V} \to \mathcal{A}_V$ et $\beta : \mathcal{V} \times \mathcal{V} \to \mathcal{A}_E$ associent des attributs aux nœuds et aux arêtes.

L'ensemble d'arêtes $\mathcal{E}$ peut être implicitement décrit par la fonction β. Il faudrait inclure l'information de l'existence d'une arête dans l'ensemble d'attributs $\mathcal{A}_E$. De fait, une attribut ne peut être attribué que quand une arête existe. Il suffit alors d'ajouter un symbole supplémentaire à $\mathcal{A}_E$ qui permet d'identifier des arêtes sans attributs. De cette manière, la notation peut être simplifiée à $G = (\mathcal{V}, \alpha, \beta)$ dans certains cas.

1.3 Appariements dans un graphe

La recherche d'appariements dans un graphe vise à trouver des couples de sommets compatibles dans des graphes généralement bipartites. Ainsi, étant donné un ensemble d'usines et un ensemble de produits, on peut construire un graphe de compatibilité en connectant chaque usine avec les produits qui peuvent y être fabriqués. A chaque arc on attribue le coût de production. On cherche alors à produire un maximum de produits à coût minimum. Un autre exemple classique est la formation des couples à partir d'ensembles de garçons et de filles dont les préférences sont modélisées par les poids des arcs.

Un appariement est un ensemble d'arêtes indépendantes $\mathcal{E}_i$ dans un graphe $G = (\mathcal{V}, \mathcal{E})$. L'indépendance d'arêtes est définie par l'absence de sommets incidents communs entre elles. Il s'agit d'un appariement de $\mathcal{V}' \subseteq \mathcal{V}$ si tous les sommets de $\mathcal{V}'$ sont couverts par $\mathcal{E}_i$, c'est-à-dire s'il existe une arête dans $\mathcal{E}_i$ qui relie chaque sommet dans $\mathcal{V}_i$ à un autre. Une propriété importante est la stabilité d'un appariement. Chaque sommet a une liste de préférences, c'est-à-dire les arêtes partant de lui sont pondérées et il existe un ordonnancement des poids. L'idée est que l'on ne souhaite pas qu'il y ait deux sommets qui préfèrent échanger leurs affectations actuelles par le lien direct entre eux (qui n'est pas dans $\mathcal{E}_i$). Un appariement stable existe pour tout ensemble de préférences d'après le théorème du mariage stable.

Un appariement maximal est l'appariement ayant le nombre maximal d'arêtes. Le plus souvent, on cherche des appariements maximaux. Le problème est beaucoup plus facile que le problème d'appariements entre deux graphes. L'algorithme de Hopcroft & Karp (1973), par exemple, a une complexité de $\mathcal{O}(n^{\frac{5}{2}})$. Dans la littérature, on utilise souvent le terme appariement de graphes bipartites (*bipartite graph matching*) afin de lever l'ambiguïté avec les appariements entre deux graphes (*graph matching*). Cependant, il existe des appariements dans un graphe non-bipartite bien que ceci ne soit pas très commun. On peut également imaginer des problèmes de recherche d'isomorphismes entre deux graphes bipartites. Dans cette thèse, nous allons utiliser de préférence le terme d'isomorphisme (voir les sections suivantes) quand cela est possible pour éviter des confusions. Quand nous utilisons le terme appariement, il est utilisé dans le sens du problème d'isomorphisme également.

1.4 Isomorphisme de graphes

Beaucoup de travaux ont été effectués pendant plusieurs décennies, visant les différents problèmes d'isomorphisme sous divers points de vue en appliquant une grande variété de méthodes (Read & Corneil, 1977; Fortin, 1996; Conte et al., 2004). La plupart des problèmes avec un impact important en pratique est NP-difficile (Garey & Johnson, 1979), tandis que pour l'isomorphisme des graphes non-étiquetés, une réponse à la question de savoir s'il est NP-complet ou se situe dans P n'a pas encore été trouvée (Köbler et al., 1993; Torán, 2004). Toutefois des algorithmes efficaces sont

connus pour certaines classes spécifiques de graphes, par exemple les graphes planaires (Hopcroft & Wong, 1974) ou plus généralement les graphes de genre [1] fini (Filotti & Mayer, 1980).

1.4.1 Isomorphisme *vs* homomorphisme

L'isomorphisme de graphes est une relation univoque des nœuds entre deux graphes qui respecte leurs attributs et ceux des arêtes. Dans le cas d'isomorphisme de sous-graphes, on considère deux graphes G et G' de taille différente, $|G'| = n' \leq |G| = n$. Afin de déterminer si G' est un sous-graphe de G une telle relation est recherchée entre les nœuds du graphe G' et tous les sous-graphes du graphe G induits par n' nœuds. De manière générale, on essaie de déterminer si deux graphes donnés sont égaux. Formellement, on définit les notions de l'homomorphisme et de l'isomorphisme.

Définition 1.4.1 (Homomorphisme de graphes)**.** Soient $G = (\mathcal{V}, \mathcal{E})$ et $G' = (\mathcal{V}', \mathcal{E}')$ deux graphes. S'il existe une relation univoque $m : \mathcal{V} \to \mathcal{V}'$ telle que $(v_1, v_2) \in \mathcal{E} \Rightarrow (m(v_1), m(v_2)) \in \mathcal{E}'$, alors G et G' sont homomorphes.

Définition 1.4.2 (Isomorphisme de graphes)**.** Soient $G = (\mathcal{V}, \mathcal{E})$ et $G' = (\mathcal{V}', \mathcal{E}')$ deux graphes. S'il existe une relation univoque $m : \mathcal{V} \to \mathcal{V}'$ telle que $(v_1, v_2) \in \mathcal{E} \Leftrightarrow (m(v_1), m(v_2)) \in \mathcal{E}'$, alors G et G' sont isomorphes.

Par conséquent, un homomorphisme est un isomorphisme lorsque la relation m est bijective. Les deux notions sont souvent confondues (Kropatsch et al., 2007). En effet, un homomorphisme de G vers G' permet qu'il y ait une arête entre deux sommets dans G' faisant partie de la relation, sans qu'il y en ait entre leurs antécédents dans G. Dans le cas de l'isomorphisme, une telle situation est impossible.

1.4.2 Isomorphisme de sous-graphes

Le problème de l'isomorphisme de sous-graphes consiste à trouver une application partielle entre les sommets de deux graphes, telle que les conditions définies dans la définition 1.4.1 soient satisfaites. Soient G et G' deux graphes avec $n' \leq n$, on cherche un sous-graphe de G induit par n' sommets. Si l'on considérait des sous-graphes en général au lieu de sous-graphes induits par des sommets, on ne traiterait pas le problème d'isomorphisme mais celui de l'homomorphisme.

Nous considérons ici des ARG, c'est-à-dire que des valeurs sont associées aux nœuds ainsi qu'aux arêtes. Le problème d'identification d'une correspondance entre deux ARG ainsi que la question de leur isomorphisme sont des tâches centrales de la reconnaissance de formes structurelles. Les ARG constituent une description abstraite des objets réels ayant une structure inhérente. En général, les composants de l'objet sont modélisés ainsi que leurs relations. Les définitions de

1. Cf. Déf. 1.2.3, p. 7.

l'isomorphisme peuvent être facilement étendues à des graphes étiquetés. Dans ce but, il suffit d'assurer que les attributs des nœuds et des arêtes soient respectés.

Définition 1.4.3. Soient $G = (\mathcal{V}, \mathcal{E}, \alpha, \beta)$ et $G' = (\mathcal{V}', \mathcal{E}', \alpha', \beta')$ deux graphes étiquetés. Si la relation $m : \mathcal{V} \to \mathcal{V}'$ décrit un isomorphisme entre $(\mathcal{V}, \mathcal{E})$ et $(\mathcal{V}', \mathcal{E}')$, et $\forall v \in \mathcal{V} \alpha(v) = \alpha'(m(v))$ ainsi que $\forall (v_1, v_2) \in \mathcal{E} \beta((v_1, v_2)) = \beta'((m(v_1), m(v_2)))$ alors G est *isomorphe* à G'.

1.4.3 Problèmes inexacts

En pratique, deux graphes sont rarement exactement égaux, même s'ils ont été extraits du même objet. Les variations structurelles sont causées par l'incertitude pendant le processus d'extraction de caractéristiques, par des environnements bruités, etc. Les algorithmes d'isomorphisme de graphes devraient donc être tolérants au bruit, c'est-à-dire être capables d'identifier la correspondance entre deux graphes, même s'il reste des variations mineures. Cela nous amène aux problèmes *inexacts* d'isomorphisme décrits dans la classification de Conte et al. (2004). La résolution de ce type de problème nécessite une notion de distance entre graphes.

1.4.4 Distances de graphes

Le problème de l'isomorphisme inexact de graphes, étant donné deux graphes de la même taille $G = (\mathcal{V}, \alpha, \beta)$ et $G' = (\mathcal{V}', \alpha', \beta')$, consiste à trouver la fonction $m : \mathcal{V} \to \mathcal{V}'$ entre leurs nœuds telle qu'une métrique de dissimilarité soit minimisée. Une telle métrique est par exemple la distance d'édition de graphe (Sanfeliu & Fu, 1983). L'idée fondamentale est de chercher une séquence d'opérateurs qui transforme le premier graphe dans le deuxième. Les opérateurs sont l'insertion, la suppression et la substitution de nœuds et d'arêtes. Un coût est associé à chacun de ces opérateurs et les coûts sont additionnés sur toute la séquence. La distance est ensuite déterminée par la séquence qui nécessite le moindre coût. Le problème de trouver cette séquence est NP-difficile. La recherche des bonnes valeurs pour les différents coûts est non-triviale et dépend de l'application. La distance d'édition est donc très générale mais la difficulté reste dans son application aux problèmes concrets. En particulier, le choix d'une fonction de coût est parfois très difficile mais tout de même très important (Bunke, 1999). Il existe toutefois des méthodes d'apprentissage des coûts, comme celle proposée par Neuhaus & Bunke (2007). Bien que les articles cités s'appliquent uniquement à la distance d'édition, on peut noter que toutes les autres distances de graphes nécessitent aussi une fonction qui mette en relation les différents types de différences élémentaires et subissent donc la même dépendance.

Au lieu de transformer un graphe en l'autre, on peut se baser aussi sur le plus grand sous-graphe commun. Bunke (1997, cf. aussi Bunke & Shearer, 1998) montre qu'il existe une forte relation entre la distance d'édition de graphe et la solution du problème du plus grand sous-graphe commun. Cependant, les deux problèmes restent NP-difficiles à résoudre. Hidovic & Pelillo (2004) proposent

deux mesures de distances qui généralisent les mesures basées sur le plus grand sous-graphe commun. Elles ne nécessitent plus la définition des opérations d'édition mais restent d'une complexité NP-complet. Marini et al. (2005) montrent le passage au problème des plus grands sous-graphes inexacts.

Nous utilisons la distance de graphes définie par Berretti et al. (2001), qui se concentre sur les différences d'attributs. Berretti et al. supposent des graphes complets, ce qui correspond à leur modélisation d'images et d'images 3D (cf. Sec. 7.1.2, p. 149). Par rapport à la distance d'édition leur distance est plus restreinte, notamment sur les perturbations structurelles. De fait, elle ne cherche pas à identifier les sommets supprimés dans le graphe plus petit. Elle suppose en particulier que, du point de vue structurel en ignorant les attributs, un graphe est le sous-graphe de l'autre. Elle est tout de même symétrique, car c'est toujours le graphe le plus petit qui est considéré comme sous-graphe du graphe le plus grand.

Cette distance est une combinaison convexe de la distance d'attributs de nœuds et de la distance d'attributs d'arêtes. Pour simplifier la notation, nous écrivons désormais pour la distance entre deux nœuds quelconques v and v' faisant partie de deux graphes $G = (\mathcal{V}, \alpha, \beta)$ et $G' = (\mathcal{V}', \alpha', \beta')$ respectivement, $\delta_V(v, v')$ au lieu de $\delta_V(\alpha(v), \alpha'(v'))$ et de même pour les distances d'arêtes. La distance entre deux graphes sous l'affectation m est alors définie comme :

$$\delta_m\left(G, G'\right) = w \sum_{v \in \mathcal{V}} \delta_V\left(v, m(v)\right) + (1 - w) \sum_{v_1, v_2 \in V} \delta_E\left((v_1, v_2), (m(v_1), m(v_2))\right) \tag{1.1}$$

Le paramètre $w \in [0, 1]$ permet de contrôler l'importance relative de la distance de nœuds et de la distance d'arêtes. Son choix dépend fortement de l'application et de la modélisation. Pour l'instant, nous restons encore dans le cas le plus général indépendant de l'application. Les évaluations sont donc limitées aux données artificielles. On suppose que des effets des deux distances sont égaux, et on fixe $w = 0,5$. Pour simplifier, on modifie la définition originale en normalisant les coefficients à 1. La distance absolue entre deux graphes est ensuite définie sur l'ensemble de toutes les fonctions valides par :

$$\delta\left(G, G'\right) = \min_m \delta_m\left(G, G'\right) \tag{1.2}$$

Les attributs varient selon l'application et les caractéristiques utilisées pour la description du contenu. Dans le but général, on suppose que les mesures de distances associées δ_V and δ_E sont définies comme suit :

$$\delta_V : \mathcal{A}_V \times \mathcal{A}_V \to \mathbb{R}^+ \tag{1.3}$$

$$\delta_E : \mathcal{A}_E \times \mathcal{A}_E \to \mathbb{R}^+ \tag{1.4}$$

On suppose de plus qu'elles respectent les conditions d'une métrique notamment la positivité, la réflexivité, la symétrie et l'inégalité triangulaire.

Comme pour le problème d'isomorphisme, il existe des travaux visant à faciliter le calcul en limitant la classe des graphes. Par exemple, Dickinson et al. (2004) se basent sur des graphes ayant des sommets avec un étiquetage unique, c'est-à-dire chaque sommet possède une étiquette différente.

Le clustering de graphes, qui pose la question du regroupement des graphes similaires, est étroitement lié à des questions de distances. Bien qu'il ne nécessite pas forcément une distance explicite, une distance existe au moins implicitement. Deux exemples du clustering de graphes sont l'utilisation du supergraphe commun minimal (Bunke et al., 2003) et des cartes auto-organisatrices (Günter & Bunke, 2002, 2003).

Les noyaux de graphes représentent également une façon de quantifier la (dis)similarité entre graphes. L'idée fondamentale des noyaux est une transformation de l'espace des données, $\mathcal{X}$, dans un espace nommé des caractéristiques $\mathcal{F}$ (*feature space*) :

$$\phi : \mathcal{X} \to \mathcal{F} \tag{1.5}$$

L'espace $\mathcal{F}$ représente mieux les données mais est généralement d'une dimension élevée voire infinie. Afin d'éviter le fléau de la dimension, on ne travaille qu'implicitement dans cet espace. Les algorithmes de noyaux se basent uniquement sur le calcul des produits scalaires entre les éléments de $\mathcal{X}$. La fonction noyau k permet de calculer le produit scalaire dans l'espace $\mathcal{F}$ en utilisant la représentation des données dans l'espace $\mathcal{X}$, afin de ne jamais utiliser les transformées $\phi(x)$ explicitement :

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \tag{1.6}$$

$$\forall x, y \, \langle \phi(x), \phi y \rangle = k(x, y) \tag{1.7}$$

La difficulté principale réside dans la définition d'une fonction noyau k appropriée au données qui d'une part, représente bien les données et, d'autre part, permet le calcul efficace des produits scalaires. Des noyaux de chaînes ont été proposé par Lodhi et al. (2002). Ils forment la base de plusieurs noyaux de graphes. Il y a notamment des noyaux de diffusion sur des graphes (Kondor & Lafferty, 2002), des noyaux basés sur la distance d'édition, des noyaux basés sur des opérateurs de régularisation (Smola & Kondor, 2003) et autres. Gärtner et al. (2003) montrent que le calcul d'un noyau de graphe qui utilise tout l'information disponible est NP-difficile. Ceci implique des limitations importantes des approches par noyaux. Si l'on souhaite conserver toute la généralité du modèle de graphe, alors les noyaux ne sont pratiquement pas calculables. Afin de réduire le temps de calcul, il faut accepter la perte d'information en projetant les graphes dans un espace moins complexe et souvent moins discriminant.

1.4.5 Appariements multivoques

En particulier dans le domaine des images, on trouve parfois des graphes de différentes granularités, c'est-à-dire qu'un sommet décrit un objet à une échelle plus ou moins grande. Par exemple, si l'on considère un graphe qui modélise des régions d'une image, on est confronté aux problèmes de sous- et sur-segmentation. Selon l'algorithme (et ses paramètres) de segmentation choisi, on obtient des graphes différents pour la même image. Dans le but d'établir un appariement entre de tels graphes, il s'avère utile de permettre un appariement multivoque, c'est-à-dire une relation où chaque sommet peut être en relation avec plusieurs sommets de l'autre graphe. A ce jour, il existe trois approches pour définir une telle distance.

Boeres et al. (2004) et Deruyver et al. (2005) utilisent une relation non-bijective. Plus précisément, dans leur application plusieurs sommets d'un graphe peuvent être affectés à un seul sommet dans le deuxième graphe. Leur application est l'appariement des images à des modèles prédéfinis. Les images sont (automatiquement) sur-segmentées tandis que les modèles sont créés manuellement.

Ambauen et al. (2003) proposent une extension de la distance d'édition de graphes. Ils ajoutent la subdivision d'un sommet et la fusion de sommets à l'ensemble des opérations. La subdivision est définie de telle sorte qu'elle remplace un sommet par un ensemble de nouveaux sommets ; la fusion représente l'opération inverse. Bien qu'il soit désormais possible de traiter certains cas d'appariements multivoques, l'approche est soumise à des contraintes. De fait, si l'on veut mettre en correspondance les sommets v_1 et v_2 du premier graphe avec le sommet v'_1 du deuxième graphe, il faut fusionner les deux. Il devient alors impossible de mettre en correspondance le sommet v_1 avec un autre sommet v'_2 sans aussi mettre en correspondance v_2 avec v'_2.

Champin & Solnon (2003) suivent une approche plus générique appliquée aux graphes multi-étiquetés. Leur mesure met en rapport le nombre de caractéristiques communes (dénommées $\sqcap$) entre deux graphes, le nombre de divisions de sommets et le nombre total de caractéristiques.

$$sim_m(G, G') = \overset{max}{m} \frac{g_1(G \sqcap_m G') - g_2\left(splits(m)\right)}{g_1(\alpha \cup \alpha' \cup \beta \cup \beta')} \tag{1.8}$$

Les fonctions génériques g_1 et g_2 permettent la pondération entre les étiquettes correspondantes et le nombre de divisions de sommets. Il est important de pénaliser un nombre de divisions trop grand. En effet, quand on divise un sommet, alors il y aura plus d'éléments dans l'ensemble de correspondances d'étiquettes. Tandis que cette mesure de similarité ne traite pas le problème d'isomorphisme inexact, une distance générique est proposée par Sorlin et al. (2007). Les paramètres génériques sont, non seulement les distances élémentaires (c'est-à-dire entre les sommets et entre les arêtes), mais aussi la fonction d'agrégation qui reste à définir. Ceci rend l'approche très générique, mais la nécessité d'implémenter tous ces éléments rend l'utilisation moins confortable.

1.4.6 Appariements pour la recalage

Une application différente pour les appariements est la recalage de graphes, qui est généralement effectué dans le contexte visuel. On essaye d'aligner des graphes afin d'identifier les sommets correspondants localement. On s'intéresse alors aux graphes d'une manière plus profonde. Ceci est particulièrement nécessaire pour retrouver un endroit sur une photo satellite, si l'on ne possède qu'un plan (Li et al., 2005).

1.5 Algorithmes

Il existe une grande variété de méthodes qui abordent les problèmes d'appariements de graphes. Conte et al. (2004) donnent une vue globale complète dont nous reprenons ici quelques éléments. Leur classification différencie d'abord le problème exact et le problème inexact.

Exact - recherche d'arbre Pour le problème exact, de nombreux algorithmes sont basés sur la recherche d'arbre. On traverse un arbre d'appariements partiels, dont la racine est l'appariement vide et les feuilles sont des appariements complets. À chaque niveau de l'arbre, une paire de sommets est ajoutée à l'appariement. Les algorithmes diffèrent alors dans la stratégie utilisée à l'intérieur de cet arbre, notamment sur le choix de la prochaine paire de sommets à ajouter. On utilise une stratégie de chaînage arrière (*backtracking*), afin de garantir l'optimalité globale de la solution.

Un algorithme très célèbre de cette catégorie est l'algorithme A* d'Ullmann (1976). L'arbre est parcouru d'abord en profondeur ce qui nécessite moins de mémoire qu'un parcours en largeur. L'algorithme contient une procédure de raffinement qui tient compte des appariements futurs. De fait, les appariements futurs sont filtrés afin d'exclure ceux qui ne sont pas cohérents avec l'appariement partiel en cours de développement. Cette stratégie qui consiste à essayer d'élaguer un maximum de branches basée sur une heuristique a par la suite été adoptée par de nombreux autres auteurs et représente l'élément clef pour distinguer les approches et en particulier leurs performances. Dans la même classe et aussi basé sur un parcours en profondeur, on trouve par exemple l'algorithme VF et son successeur VF2 (Cordella et al., 2004). Foggia et al. (2001) montrent l'efficacité de ces algorithmes.

Exact - Nauty Parmi les approches qui ne sont pas basées sur un parcours d'arbre, il faut citer Nauty (McKay, 1981), qui cherche un étiquetage canonique des sommets permettant leur ordonnancement. Les règles utilisées dans le système sont basées sur la théorie des groupes. Une fois qu'il a trouvé un tel étiquetage (très difficile à trouver si les graphes sont trop réguliers), il suffit d'aligner les sommets des deux graphes (en utilisant cet ordre bien évidemment) et de comparer les étiquettes des sommets correspondants. Nauty permet, comme par ailleurs aussi les algorithmes VF et VF2, d'apparier des graphes d'un millier de sommets en moins d'une seconde. Il n'y a pas d'algorithme

meilleur que les autres pour tous les graphes. Si l'on compare Nauty et VF2, Nauty est plus rapide sur des graphes denses, larges, et avec des connexions aléatoires tandis que VF2 a des avantages sur des graphes plus réguliers et creux. Finalement on peut noter que pour plusieurs classes de graphes, il existe des algorithmes d'une complexité plus faible qui exploitent les restrictions. Les exemples classiques sont premièrement les graphes planaires pour lesquels un algorithme linéaire existe (Hopcroft & Wong, 1974). Miller (1980) propose une extension qui s'applique aux graphes de genre fini (*bounded genus*). Deuxièmement, un algorithme d'un complexité $\mathcal{O}(n^{\frac{5}{2}})$ a été proposé pour des graphes bipartites. Troisièmement, pour la classe plus large des graphes pour lesquels le degré est borné, un algorithme polynomial existe (Luks, 1982). Enfin, Babai et al. (1982) proposent deux algorithmes polynomiaux pour des graphes dont la multiplicité des valeurs propres est bornée.

Un graphe *vs* une base On souhaite souvent comparer un graphe non seulement à un autre mais à une base de graphes connus, en particulier dans les applications qui s'intéressent à l'identification. Un algorithme particulièrement efficace pour cette tâche est Nauty. En effet, l'étiquetage d'un graphe ne dépend pas du deuxième graphe, c'est-à-dire que l'on peut calculer l'étiquetage pour tous les graphes dans la base en amont et au moment de la comparaison avec un nouveau graphe, on calcule uniquement l'étiquetage de ce dernier. L'étape de comparaison qui suit est très légère. Messmer & Bunke (2000) proposent de décomposer les graphes et de construire un index basé sur cette décomposition. La décomposition est faite de manière récursive. Par conséquent l'index contient des sous-graphes de différentes tailles dont la taille minimale est 1. L'idée fondamentale est d'exploiter les sous-graphes qui sont communs à plusieurs graphes dans la base. Dans ce cas, l'appariement n'est fait qu'une seule fois. La complexité est sous-linéaire avec le nombre de graphes dans la base à condition qu'il existe des sous-structures communes. La complexité diminue quand les similarités dans la base augmentent. Dans le cas extrême de graphes très similaires, la complexité devient indépendante de la taille de la base. Dans le but de réduire le nombre de graphes à comparer, Irniger & Bunke (2005, cf. aussi Irniger, 2005) utilisent un filtrage basé sur les arbres de décision.

Inexact - recherche d'arbre Le problème inexact n'est pas un problème de décision mais un problème d'optimisation combinatoire. Il est donc NP-difficile. On peut classer les algorithmes pour le problème inexact en deux groupes : d'une part les algorithmes optimaux qui garantissent l'optimum local, d'autre part les algorithmes approchés ou sous-optimaux dont le but est de réduire le temps de calcul à un niveau acceptable.

On trouve en premier lieu des approches basées sur la recherche d'arbre comme dans le cas exact. On effectue une recherche exhaustive de l'espace contenant la totalité des affectations possibles entre les nœuds des graphes (Tsai & Fu, 1979). Proposé initialement pour des problèmes venant du domaine visuel, le problème est aussi dénommé *error-correcting* (Tsai & Fu, 1979; Sanfeliu & Fu, 1983; Eshera & Fu, 1984). On commence par une affectation vide. Ensuite on ajoute des

paires de nœuds une par une. L'espace des solutions prend la forme d'un arbre. Le niveau k est composé de toutes les affectations partielles contenant exactement k paires de nœuds (Eshera & Fu, 1984; Tsai & Fu, 1979). Le choix de la paire à ajouter est effectué par un algorithme glouton. Une affectation partielle est évaluée par la somme de la distance qu'elle impose sur les graphes, prenant en compte uniquement les nœuds déjà fixés, et une estimation du coût minimal nécessaire pour la compléter. Si l'on regarde une branche quelconque, la fonction d'évaluation est croissante avec le niveau de l'arbre. Comme l'estimation du coût d'extension est une borne inférieure de la distance réelle, chaque sous-arbre dont la racine a une valeur supérieure à la distance maximale acceptable peut être ignoré sans risque d'omission d'une solution valable. L'algorithme A^* (Ullmann, 1976) effectue cette recherche vers l'optimum local. Au cas où l'extension n'est plus possible, une stratégie de chaînage arrière est appliquée. Quand la recherche arrive dans une feuille, une solution candidate a été trouvée. Ici l'optimum global n'est pas garanti non plus, mais la distance imposée par cette solution candidate est ensuite utilisée comme borne plus stricte et la recherche continue jusqu'à ce que toutes les branches aient été évaluées.

De fait, l'algorithme initial ne tient pas compte du coût de l'extension. Plusieurs approches proposent des heuristiques admissibles pour ajouter une borne inférieure au coût d'extension à la fonction d'évaluation. La méthode de Berretti et al. (2001) utilise un algorithme de relaxation discrète pour la résolution d'un problème d'isomorphisme de graphes bipartites dont le résultat détermine l'estimation du coût. En partant d'une affectation partielle de k sommets, ils cherchent à établir une estimation de la distance finale en prenant en compte les coûts d'affectation des k sommets déjà appariés mais aussi des $n-k$ sommets non encore appariés ainsi que leurs relations. La complexité du calcul de l'extension optimale vers un appariement complet équivaut celle du problème d'isomorphisme de graphes de $n-k$ sommets. Afin de rendre le calcul polynomial et d'obtenir une borne assez strict en même temps, Berretti et al. (2001) abandonnent la contrainte que les extensions élémentaires soient cohérentes entre eux. Pour chaque niveau entre k et n, ils choisissent l'appariement du coût minimum qui est cohérent avec l'appariement d'origine. Toutefois, les appariements utilisés pour l'estimation du coût de l'extension ne sont pas forcément cohérents entre eux. Cette astuce permet d'estimer ce coût en temps polynomial. Basée sur cette valeur, les affectations partielles, dont l'extension vers une solution complète avec une similarité acceptable est impossible, peuvent être ignorées plus tôt. Ceci accélère l'algorithme en conservant l'optimalité du résultat. La méthode de Berretti et al. (2001) est très performante et compte toujours parmi les algorithmes les plus efficaces disponibles. Par contre, les tailles des graphes que l'on peut traiter en un temps faisable sont plutôt de l'ordre de la dizaine que du millier (comme dans le problème exact).

Optimisation continue Les algorithmes d'optimisation continue forment le second grand groupe. Le problème est transformé dans le domaine continu et devient un problème d'optimisation non-linéaire. Les méthodes dans cette catégorie sont en général approchées et elles nécessitent la trans-

formation de la solution dans le domaine d'origine. Il existe des méthodes basées sur la relaxation probabiliste, par exemple celle de Wilson & Hancock (1997), ou sur l'appariement pondéré de graphes (*weighted graph matching*). Ce dernier est lié au problème du plus grand sous-graphe commun. On utilise une matrice d'appariements avec des poids compris entre 0 et 1 et on optimise cette matrice dans le domaine continu. Les algorithmes d'Almohamad & Duffuaa (1993) et Gold & Rangarajan (1996) suivent cette approche. Une troisième transformation se base sur le théorème de Motzkin & Straus (1965) lié à la recherche de cliques (Pelillo, 1999). L'appariement de graphes est alors transformé en un problème d'optimisation quadratique.

Cesar et al. (2005) présentent une comparaison des différentes méthodes d'optimisation dans le domaine d'application de la reconnaissance. Ils considèrent notamment des approches de recherche d'arbre, des algorithmes génétiques et des algorithmes à estimation de distribution (Bengoetxea et al., 2002).

Méthodes spectrales Les méthodes basées sur la théorie spectrale des graphes (Chung, 1997; Godsil & Royle, 2001; Shokoufandeh et al., 2005) analysent les valeurs et les vecteurs propres de la matrice d'adjacence. Les valeurs propres sont indépendantes des permutations de sommets. De fait, si deux graphes sont isomorphes, alors leurs spectres sont égaux. L'inverse par contre n'est pas vrai : des graphes non-isomorphes peuvent avoir le même spectre. On les dénomme des graphes co-spectraux. Une des premières méthodes est proposée par Umeyama (1988). Il utilise la décomposition en valeurs et vecteurs propres des matrices d'adjacence pour en déduire la matrice orthogonale qui est optimisée par la suite. Pour cela il suppose *à priori* que les graphes sont isomorphes. S'ils le sont, alors la méthode trouve la permutation optimale. Si les graphes sont proches de l'isomorphisme, alors la solution est sous-optimale. Toutefois, pour des graphes non-isomorphes la qualité des résultats diminue. Une idée utilisée en combinaison avec une approche spéciale est le clustering. D'une part, un clustering des sommets avant l'appariement permet d'associer d'abord les clusters et ensuite les sommets (Carcassoni & Hancock, 2001). D'autre part, Caelli & Kosinov (2004) proposent de projeter les sommets dans l'espace propre des graphes et d'utiliser le clustering dans cet espace afin de trouver les sommets à mettre en correspondance. Les méthodes spectrales donnent lieu également aux méthodes exploitant les caractéristiques des parcours aléatoires (Gori et al., 2005).

A côté de ces trois principaux groupes d'approches il en existe encore de nombreuses autres. On peut citer notamment les méta-heuristiques suivantes : les réseaux de neurones (Jain & Wysotzki, 2005), l'optimisation par colonies de fourmis et le tabou réactif (Sammoud et al., 2006), ou encore les EA. Nous donnons plus de détails sur ces derniers dans la section 1.8. Parfois, on rencontre également des travaux plus généraux qui appliquent une méta-heuristique à une classe de problèmes. Ainsi, Campos et al. (2005) appliquent la *scatter search* à la classe des problèmes de permutation. Bien que l'isomorphisme de graphes ne soit pas parmi les problèmes choisis pour illustrer la méthode, il fait partie des problèmes de permutation.

1.6 Méta-heuristiques

Les problèmes dont la complexité est NP ne permettent pas leur résolution exacte en un temps raisonnable. Dans cette catégorie se retrouvent tous les problèmes d'optimisation combinatoire. Prenons par exemple le problème du voyageur de commerce. Étant donné un ensemble de points (ou bien villes) et les distances entre eux, on cherche le chemin le plus court qui les relie. Formellement, on cherche le cycle hamiltonien minimal (dans le sens de poids accumulés) dans un graphe complet pondéré. Si n est le nombre de sommets dans le graphe, il existe $\frac{(n-1)!}{2}$ cycles différents à évaluer. Supposons que l'on possède une machine pouvant évaluer trois milliards de cycles par seconde ce qui correspond à la fréquence de 3GHz des ordinateurs actuels [2]. Le temps de calcul pour un problème à 15 sommets est alors de 14.5 secondes. Si l'on passe à 20 sommets celui-ci s'élève à 234.7 jours. Le temps de résolution pour une instance à 22 sommets dépasse déjà l'espérance de vie humaine avec 270 ans. Or, les problèmes réels sont de taille bien plus importante. Il n'est donc pas envisageable de mettre en œuvre des approches exactes en réalité.

Quand on se trouve face à des problèmes d'une complexité élevée en pratique, on recourt souvent à des heuristiques. Tandis qu'un algorithme classique obtient toujours la solution optimale et permet de démontrer qu'une borne supérieure acceptable de son temps de calcul existe, une heuristique assouplit un de ces deux critères : la plupart des heuristiques ne garantissent pas la solution optimale mais recherchent uniquement une solution de bonne qualité. Cette simplification du problème mène à une réduction considérable du temps de calcul. D'un autre côté, il existe des heuristiques qui sont rapides en général, mais qui ne permettent pas de définir une borne supérieure du temps de calcul dans le pire cas. Les heuristiques sont dépendantes du problème, on ne peut donc pas appliquer une heuristique pour le problème X au problème Y.

Une méta-heuristique est une heuristique généralisée et très flexible. Elle peut notamment être appliquée à n'importe quel problème d'optimisation. Il existe deux types de méta-heuristiques : celles qui construisent une solution, parmi lesquelles on peut citer les algorithmes gloutons et les EA, et celles qui améliorent une solution existante. Ces dernières sont des algorithmes de recherche locale dont le représentant le plus connu est l'algorithme de la descente de gradient. Á la différence des heuristiques, les méta-heuristiques ne sont pas entièrement implémentées, mais elles définissent une stratégie générale indiquant comment trouver une solution. Certains éléments de cette stratégie doivent être implémentés en fonction du problème. En particulier elles nécessitent toujours une fonction objective permettant d'évaluer l'état actuel de la recherche. Par la suite nous nous focalisons sur les approches évolutionnaires et en particulier les GA. Nous intégrons aussi une méthode de recherche locale.

Même si les méta-heuristiques considèrent ces parties comme des « boîtes noires » , la qualité

2. Ceci est une idéalisation surestimant la puissance de la machine parce que l'évaluation n'est pas une opération qui se fait pendant un seul cycle du processeur.

d'une méta-heuristique sur un problème concret varie beaucoup selon le choix de ces paramètres. Il faut aussi noter que si une heuristique spécialisée pour un problème existe, ses performances sont souvent meilleures que celles d'une méta-heuristique. L'avantage des méta-heuristiques est leur variabilité et non pas forcément leur performance.

1.7 Algorithmes évolutionnaires

Les algorithmes évolutionnaires (EA) sont des méthodes d'optimisation inspirées de la théorie de l'évolution darwinienne (Goldberg, 1989; Davis, 1991; Michalewicz, 1996; Bäck et al., 1997). Le domaine se base historiquement sur trois approches toutes développées à partir des années 60.

Le point commun de ces approches et des nombreuses autres approches évolutionnaires est l'utilisation d'un processus inspiré par la théorie de l'évolution darwinienne. Elles sont basées sur une population d'individus dont chacun est une solution candidate du problème. Il existe un moyen d'établir la qualité d'un individu, par la fonction d'évaluation.

On applique de façon répétée des opérateurs génétiques, notamment la sélection, le croisement et la mutation, afin d'obtenir une qualité de solution croissante au fil des générations. D'une part, les algorithmes génétiques permettent de résoudre des problèmes difficiles dans un temps raisonnable, d'autre part une solution valide peut-être livrée à n'importe quel moment, la qualité augmentant en général avec le temps. L'utilisateur peut, dans certaines bornes, fixer le compromis entre précision et temps d'exécution directement à travers un critère d'arrêt adapté.

Stratégies évolutionnaires (ES) Les ES de Rechenberg (1973) et Schwefel (1981) abordent le problème de l'optimisation continue. Le domaine de définition est un sous-ensemble de $\mathbb{R}^d$. Un individu correspond donc à un vecteur de valeurs réelles. La mutation des paramètres est souvent le seul opérateur appliqué. Elle consiste à additionner une variable aléatoire gaussienne. Une stratégie est définie par le nombre des parents, μ, et celui des descendants, λ. On distingue la stratégie + et la stratégie ,. Dans la stratégie + les parents peuvent passer à la génération suivante. Dans ce but, on choisit parmi la population intermédiaire, composée de $\mu + \lambda$ individus, les μ meilleurs. Dans la stratégie , les parents sont toujours remplacés par leurs descendants. La nouvelle génération est créée en choisissant les μ meilleurs descendants. Ceci implique que le nombre d'enfants doit être supérieur [3] au nombre de parents ($\lambda > \mu$).

Algorithmes génétiques (GA) Les GA de Holland (1975) forment la base de ce que l'on appelle aujourd'hui les algorithmes génétiques classiques ou canoniques. Les individus sont représentés comme des chaînes de valeurs. L'algorithme applique des opérateurs de croisement afin de recombiner l'information de deux parents et de mutation. Le nombre des descendants est égal au nombre

3. Si $\mu = \lambda$ les enfants sont choisis indépendamment de leur qualité. Il n'y aura donc pas de pression de sélection.

des parents. A chaque génération les descendants remplacent la totalité des parents. La sélection se fait au niveau des parents qui auront plus ou moins de chances d'être sélectionnés pour le processus de reproduction.

Programmation génétique (GP) Au lieu de trouver une solution précise à un problème, la GP construit une description structurée de la solution dans un langage en général régulier. Les chromosomes modélisent des programmes qui définissent des stratégies permettant d'obtenir la solution : un individu est composé d'instructions et d'entrées organisées sous forme d'un arbre. Les entrées se trouvent uniquement dans les feuilles, tandis que les nœuds internes représentent des opérations sur les résultats de leurs sous-branches. L'algorithme ressemble à celui des GA à différence de la mutation et du croisement : le croisement échange des sous-arbres entre deux individus, la mutation remplace un sous-arbre par un nouveau sous-arbre aléatoire.

Les EA explorent des espaces de recherche immenses rapidement mais exploitent inefficacement au niveau local. Les heuristiques de recherche locale présentent des caractéristiques inverses. C'est pourquoi la combinaison des deux méthodes apporte une amélioration considérable à la performance totale (Wolpert & Macready, 1997; Renders & Flasse, 1996). On appelle ces approches hybrides des algorithmes *mémétiques* (Moscato, 1989; Krasnogor & Smith, 2005). L'analogie à la nature réside dans l'apprentissage pendant la vie. En effet, les algorithmes génétiques classiques limitent l'apprentissage à la création d'un individu, celui-ci ne subit ensuite aucune évolution pendant toute sa vie. Il est évident que pour les êtres naturels ceci n'est pas vrai : ainsi un enfant acquiert presque toutes ses connaissances et capacités après sa naissance y compris les capacités élémentaires de parler et de marcher. Les algorithmes *mémétiques* simulent cet apprentissage par des méthodes d'optimisation locale appliquées après le croisement. Un avantage par rapport à l'algorithme génétique de base est que des connaissances spécifiques au domaine d'application peuvent être prises en compte afin d'affiner les individus. Pour toutes ces raisons la qualité des solutions et la vitesse de convergence augmentent.

L'efficacité des EA dépend de manière critique de l'ajustement des hyper-paramètres (cf. Eiben et al., 2007). De fait, l'optimisation des paramètres constitue un problème d'optimisation combinatoire elle-même. Comme toutes les méta-heuristiques, les EA sont plus lents qu'une méthode spécifique pour l'application visée. Bien que des preuves de convergence existent, celles-ci supposent que le temps de calcul est infini. Par conséquent, en pratique la solution optimale globale n'est pas garantie.

En ce qui concerne les avantages des EA, on peut d'abord noter les nombreuses applications possibles. Deuxièmement, ils permettent de traiter des problèmes d'une grande complexité (de l'espace de recherche en particulier). Troisièmement, ils permettent une parallélisation facile comme nous le montrons dans le chapitre 5. Il s'agit par ailleurs des algorithmes *anytime* ce qui signifie qu'une solution candidate valide puisse être rendue à n'importe quel moment. Cette propriété est particuliè-

rement intéressante pour des applications très coûteuses qui nécessitent un compromis entre temps de calcul et précision (Horvitz & Zilberstein, 2001; Finkelstein & Markovitch, 2001; Hansen & Zilberstein, 2001). De plus, si une heuristique spécifique existe alors celle-ci peut-être intégrée dans les EA.

Dans le chapitre 2 nous présentons les paramètres principaux pour le problème d'isomorphisme inexact de graphes. Le but de cette étude n'est pas d'optimiser chacun d'eux mais d'illustrer l'efficacité d'un algorithme génétique sur ce problème.

1.8 Approches génétiques dans l'état de l'art

Il existe des approches génétiques pour la résolution du problème d'isomorphisme de graphes. Leurs différences techniques se trouvent principalement dans le choix des opérateurs. Ainsi, Singh et al. (1997) proposent deux types de codage différents, le codage binaire et la représentation de chaînes de nombres entiers (*integer string representation*) qui correspond au codage de permutations. Ils proposent le croisement et la mutation par couleur (*color crossover, color mutation*). L'idée fondamentale est d'employer une classification donnée des nœuds. Ensuite, on peut réduire l'espace de recherche en ne permettant que les alignements dont tous les nœuds correspondants possèdent la même classe. La classification doit être telle que la distance la plus grande à l'intérieur de n'importe quelle classe soit inférieure à la distance la moins élevée entre deux éléments de classes différentes. Néanmoins, le problème est d'obtenir une telle classification pour chaque application. Bien que ce soit facile dans certaines applications comme les molécules où la classification est donnée par les éléments chimiques, dans d'autres applications notamment lorsque les valeurs associées aux nœuds sont numériques cela devient plus difficile.

Singh et al. mènent leurs expérimentations sur des graphes comptant 10, 30 et 50 nœuds. Dix graphes sont construits aléatoirement pour chaque taille de telle sorte que le degré moyen de chaque sommet soit entre 3 et 4 pour les graphes de tailles 10 et 30, et autour de 8 pour les autres. Les algorithmes sont ensuite appliqués trois fois pour chaque instance du problème. Ils concluent qu'il n'y a pas de différence de performance entre le croisement basé sur l'ordre (OX), le croisement cyclique (CX) et le croisement à correspondance partielle (PMX) tandis que le croisement par couleur (*color crossover* ; en utilisant deux couleurs) améliore les résultats. Cependant ils ne spécifient pas quel opérateur (OX, PMX ou bien CX) est utilisé sur les deux parties des chromosomes. De plus, l'évaluation se limite sur l'intervalle du nombre de générations pour obtenir la bonne solution, par exemple 30-50 pour des graphes de taille 10 ce qui laisse une grande marge d'interprétation.

Wang et al. (1997) utilisent le PMX et la mutation d'échange. De plus, ils introduisent la mutation du premier (*first mutation*) qui consiste à effectuer une recherche locale du type *two-opt* uniquement autour du meilleur individu de chaque génération. Les expérimentations sont basées sur une base de graphes aléatoires dont les attributs sont compris entre 0 et 100. Ils considèrent en outre du

bruit en ajoutant une valeur entre $-\varepsilon$ et $+\varepsilon$ à tous les attributs. Les niveaux de bruit ε sont de 5, 10, 15 et 20. 50 paires de graphes sont créées pour chaque test. La taille maximale de graphes est de 20 et les résultats obtenus montrent que la qualité des solutions diminue beaucoup avec la taille et le niveau de bruit.

Torres-Velázquez & Estivill-Castro (2002) se basent sur la même stratégie que Wang et al. (1997) pour évaluer leur approche. Ils utilisent une recherche locale guidée par *quicksort*. Avec cela ils améliorent les résultats précédents.

Cross et al. (1997, cf. aussi Cross & Hancock, 1999) proposent le croisement géométrique (*geometric crossover*). Cet opérateur n'est applicable qu'à l'isomorphisme de graphes planaires. Plus précisément, il faut que les nœuds des graphes soient des points dans un espace de deux dimensions. On choisit une coupe aléatoire qui passe par le centre (*Centroid*) de l'un des graphes. Puis, on échange les appariements d'un sous-ensemble de nœuds par celles de l'autre parent. En effet, ceci peut-être considéré comme croisement brassé avec un brassage adapté aux problèmes de deux dimensions. Myers (1999, cf. aussi Myers & Hancock, 2001) montre quelques années plus tard que le croisement géométrique ne donne pas de résultats meilleurs que les croisements uniformes. En effet, les opérateurs considérés sont le croisement uniforme (UX), le croisement à deux points (*two-point-crossover*), le croisement semi-uniforme (HUX) [4] et le croisement géométrique. Les expérimentations sont menées sur une base de quatre paires de graphes de taille 30. Pour obtenir les graphes, on crée d'abord 30 points dans un espace 2D. Puis on connecte chaque nœud avec 6 de ses voisins les plus proches. Le deuxième graphe de chaque paire est obtenu par l'addition de 10% de bruit sur les attributs des nœuds et des arêtes. De même, Myers duplique 10% des nœuds afin de simuler un effet correspondant à la sur-segmentation des images.

Suganthan (2002) applique un algorithme génétique qui utilise le croisement à un point (*one-point-crossover*) et la mutation « standard », c'est-à-dire qu'un allèle est remplacé par une valeur aléatoire. Son travail se focalise sur la comparaison des formes géométriques. Plus tard, Khoo & Suganthan (2003) appliquent UX et introduisent un opérateur de croisement spécialisé. Il consiste à choisir de façon déterministe pour chaque gène de l'enfant le gène ayant la plus haute *fitness* parmi les deux de ses parents. La *fitness* d'un certain gène dépend des autres gènes. L'évaluation d'un gène seul n'est donc pas évidente. Khoo & Suganthan proposent une combinaison pondérée de la fonction d'évaluation unaire et de la fonction d'évaluation binaire. La première prend en compte seulement les attributs des deux nœuds associés directement par le gène tandis que la deuxième prend aussi en compte toutes les arêtes concernées. Le nom binaire vient du fait que chaque arête est définie par une paire de nœuds. Le poids de la *fitness* unaire devrait diminuer avec l'avancement de l'algorithme, car au début on considère que peu d'associations sont bonnes, tandis que vers la fin on peut supposer le contraire.

4. Le HUX fonctionne comme le croisement uniforme avec la condition supplémentaire d'échanger exactement 50% des gènes entre les parents.

Chapitre 2

Algorithmes génétiques pour l'isomorphisme de graphes

La performance d'une méta-heuristique en général et des algorithmes évolutionnaires (EA) en particulier sur un problème précis dépend fortement de l'implémentation des parties « boîtes noires » et des paramètres choisis. Par conséquent, il est important de l'adapter à chaque problème donné.

Dans ce chapitre, nous introduisons d'abord les méthodes pour étudier les EA théoriquement et empiriquement dans la section 2.1. Nous nous intéressons particulièrement aux conditions d'une bonne évaluation. Ceci concerne le choix des mesures de performance ainsi que le cadre de la comparaison avec d'autres approches.

Nous développons ensuite le cadre général de l'application de l'EA sur le problème d'isomorphisme de sous-graphe. Nous détaillons chaque étape dans la section 2.2.

2.1 Étude des algorithmes évolutionnaires

Le développement d'un cadre définissant un EA pour un problème donné nécessite la comparaison des différentes implémentations possibles. Pour cela, il faut évaluer chaque choix en ce qui concerne, en particulier, les opérateurs et les paramètres.

On peut étudier un EA donné principalement de deux manières. La première méthode est l'analyse théorique de ses caractéristiques. La deuxième approche est l'analyse empirique de l'algorithme. Les deux méthodes ont leurs avantages et inconvénients respectifs. En ce qui concerne l'analyse théorique, qui sera détaillée dans la section 2.1.1, elle fournit toujours des résultats généralisables, mais uniquement sur des EA simplifiés. Quant à l'analyse empirique (cf. Sec. 2.1.2, p. 27), elle peut être appliquée même sur les EA les plus complexes, mais on ne peut pas toujours généraliser les résultats obtenus. Ce dernier est le plus souvent utilisé pour la recherche appliquée.

2.1.1 Analyse théorique

Le développement de la théorie des EA est en retard par rapport à leur utilisation pratique. On se base souvent sur des études empiriques au lieu de résultats théoriques. Une compréhension théorique des EA pourrait expliquer par exemple sur quelles classes de problèmes ils fonctionnent bien. Une théorie soutiendrait le processus du choix du codage, des opérateurs et des autres paramètres. Une question particulièrement analysée est la question de convergence des EA (cf. par exemple Rudolph, 1994, 1996; Hansen et al., 2003; Auger, 2005).

Ces deux aspects sont aussi à la base de la recherche d'une modélisation théorique globale. De fait, les EA sont des systèmes complexes dynamiques avec une multitude de variables. Les résultats théoriques existants sont restreints à une certaine combinaison de paramètres fixés (codage, opérateurs, forme de la fonction d'évaluation,...). De plus, les problèmes sur lesquels les EA fonctionnent bien sont généralement des problèmes complexes et par conséquent difficiles à décrire. Une autre difficulté est qu'il s'agit de processus aléatoires. Une analyse stochastique est donc nécessaire. Cela implique que les résultats montrent globalement le comportement moyen et non chaque aspect séparément.

Par la suite nous allons présenter les principaux concepts théoriques existants.

Théorème du schéma La notion de schéma a pour but de grouper des chromosomes *similaires*. Un schéma H est une chaîne de caractères (comme les chromosomes eux-mêmes) qui contient des caractères de l'alphabet et des caractères génériques (*don't care symbol*). Pour ces derniers, on utilise souvent les symboles * ou #. Soit $\mathcal{A}$ l'alphabet des chromosomes, alors l'alphabet des schémas correspondants est $\mathcal{A} \cup (\{*\} \vee \{\#\})$. Un allèle est défini par un schéma, s'il contient un symbole de $\mathcal{A}$ et non le caractère générique. Un schéma décrit l'ensemble des chromosomes dont les gènes qui sont définis dans le schéma correspondent à ces derniers. En ce qui concerne les occurrences des caractères génériques les chromosomes peuvent contenir n'importe quelle valeur à cette position. Un schéma décrit ainsi un hyperplan dans l'espace de solutions. On peut caractériser un schéma en utilisant l'*ordre* correspondant au nombre des allèles définis et la *longueur définie* qui compte la longueur de la sous-chaîne entre le premier et le dernier allèle défini. Voici un exemple avec codage binaire : la chaîne *10*01* est un schéma. Son ordre est 4 et sa longueur définie est 5 (on prend en compte la seconde étoile, celle entre 0 et 0). Les chromosomes 0100010 et 1101011 correspondent à ce schéma tandis que le chromosome 0110010 ne lui correspond pas.

Le théorème du schéma de Holland (HST, 1975) est un résultat fondamental basé sur les schémas. Il montre que des schémas d'une *fitness* supérieure à la moyenne se reproduisent de manière exponentielle dans la population. De fait, il met en relation le nombre d'occurrences d'un schéma dans une génération en fonction de son nombre dans la génération précédente. Il s'applique aux GA qui utilisent la stratégie de sélection proportionnelle à la *fitness* (*fitness proportionate selection*). On

définit d'abord la *fitness* d'un schéma $f(H)$ comme moyenne des *fitness* des chromosomes qui lui correspondent. Holland (1975) utilise les occurrences dans la population au lieu de l'ensemble des chromosomes possibles. Soit $q(H,t)$ la quantité de chromosomes dans la population au moment t qui correspondent au schéma H, $\overline{\mathcal{F}}$ la valeur moyenne de la *fitness* sur la population et p_d la probabilité que le schéma soit perturbé (*disrupted*) par un opérateur, alors l'espérance du nombre de chromosomes du schéma à la génération $t+1$ vérifie l'inégalité

$$q(H,t+1) \geq q(H,t) \times \frac{f(H)}{\overline{\mathcal{F}}} \times (1-p_d) \tag{2.1}$$

La probabilité de perturbation du schéma p_d est très dépendante des opérateurs utilisés. Si l'on considère le croisement à un point et une mutation qui affecte un seul gène, alors elle dépend d'ordre $o(H)$ du schéma et de sa longueur définie $l_d(H)$ comme suit :

$$p_d = 1 - \left(1 - p_c \frac{l_d(H)-1}{l-1}\right) \times \left(1 - p_m \frac{o(H)}{l}\right) \tag{2.2}$$

Les variables p_c et p_m sont les probabilités de croisement et de mutation et l est la longueur totale des chromosomes. En effet, le croisement perturbe le schéma si le point choisi se situe entre des gènes définis du schéma. Pour la mutation il suffit qu'elle soit appliquée sur un gène défini. Pour d'autres opérateurs, le calcul de la probabilité de perturbation change évidemment. Les dépendances qu'il faut prendre en compte ne se limitent pas forcément à l'ordre et la longueur définie du schéma.

Le HST est une inégalité car il ne tient pas compte (dans sa version originale) de la possible création du schéma par les opérateurs. Il montre en bref que des schémas ayant une meilleure *fitness* que la moyenne se reproduisent exponentiellement dans les populations suivantes.

L'analyse des schémas est très liée à l'hypothèse que les EA fonctionnent à travers des *building blocks*. Un *building block* est un sous-ensemble du chromosome qui contient des gènes proches (de préférence des voisins directs sur le chromosome). De plus, sa *fitness* est plus haute que la moyenne. L'hypothèse est que l'algorithme trouve la solution à travers d'une recombinaison réitérée des *building blocks*. On suppose que, au fil du temps, les *building blocks* deviennent plus grands jusqu'au moment où ils couvrent la totalité du chromosome.

Le HST a de nombreuses limites. Tout d'abord il décrit uniquement le passage d'une génération à la suivante et ne permet pas d'analyses à long terme. Ceci vient du fait que la dynamique de certaines caractéristiques de la population n'est pas prise en compte. Par exemple, la *fitness* relative d'un schéma baisse au fil des générations car sa *fitness* absolue reste constante, mais la moyenne dans la population augmente. Il y a d'autres hypothèses qui limitent son application. D'abord, les schémas doivent être courts et de petit ordre, sinon la probabilité de perturbation domine le processus. De plus, il ne tient pas compte de tous les effets des opérateurs de croisement et de mutation. Enfin, le but final est la création des schémas longs, mais dans ce cas, le HST n'est plus applicable car la probabilité de perturbation est proche de un.

Malgré toutes les critiques (Vose, 1999; Reeves & Rowe, 2003) sur l'hypothèse des *building blocks*, des travaux sont toujours menés dans le domaine. Récemment, Stephens & Cervantes (2007) posent la question : *Just What are Building Blocks* ? Ils étudient notamment les conditions sous lesquelles la recombinaison est avantageuse.

Systèmes dynamiques Vose (1999) propose l'analyse du SGA (*Simple Genetic Algorithm*) avec des méthodes venant de la théorie des systèmes dynamiques (Katok et al., 1997). Un système dynamique contient trois éléments : premièrement, l'espace des états possibles du système, ce qui correspond à l'ensemble des populations ; deuxièmement, une notion du temps qui est généralement discrète pour les GA ; et enfin, une loi de transition de l'état actuel (ou d'une chaîne d'états précédents) vers le prochain état du système. Pour les EA le passage d'une génération vers la prochaine est un processus stochastique. Afin d'obtenir une fonction de transition déterministe, on utilise un modèle avec une population infinie. Dans ce cas, la population suivante correspond exactement à l'espérance sur celle-ci. L'application de la théorie devient donc possible. L'analyse se base sur des points fixes. De fait, le système converge vers l'un des points fixes. Il est possible de prédire la trajectoire complète du système mais ceci est évidemment très coûteux.

Pour des populations finies, les points fixes obtenus dans le cas des populations infinies peuvent servir d'approximation. Toutefois, un tel point fixe n'est pas forcément parmi les populations (finies) possibles. On peut modéliser la trajectoire à l'aide d'une chaîne markovienne. Ainsi, la preuve de convergence présentée par Rudolph (1994) utilise cette approche.

Tous les modèles précédents souffrent de leur complexité qui les rend pratiquement inapplicables aux problèmes réels. Une idée de simplification vient de la physique statistique (Shapiro, 2001; Burjorjee, 2007). Au lieu d'analyser les effets au niveau élémentaire (par exemple le mouvement de particules), on utilise des mesures macroscopiques (par exemple la température, des flux globaux). De cette manière, on réduit énormément le nombre de variables et ainsi la complexité algorithmique des simulations. Le modèle nécessite des hypothèses sur les caractéristiques macroscopiques qui limitent son application à certaines classes d'EA (McCall, 2005).

Il est important de noter que des approches théoriques visent à simplifier les EA pour rendre leur analyse mathématique faisable. Par exemple, Neubauer (2008) propose d'utiliser une sélection alternative (α-sélection) dans le cadre de l'algorithme génétique simple. Les algorithmes utilisés en pratique sont généralement beaucoup plus complexes que ceux qui sont analysables. Ceci montre que l'on est encore loin d'un modèle théorique complet qui réponde à la question principale dans la généralité : Comment et pourquoi les algorithmes génétiques fonctionnent-ils ? Des études empiriques sont alors nécessaires. Néanmoins plusieurs résultats théoriques facilitent déjà l'implémentation d'un EA. Par exemple un EA converge uniquement s'il mémorise la meilleure solution (Rudolph, 1994). Il semble aussi que des taux de mutation positifs, mais petits, et des taux de croisement élevés, mais inférieurs à 1, sont utiles.

2.1.2 Analyse empirique

Notre objectif de recherche est de trouver un EA performant pour le problème de l'isomorphisme de graphes. Nous nous basons sur plusieurs mesures pour évaluer la qualité d'un algorithme. En général, on analyse le temps de calcul ainsi que la précision, c'est-à-dire la qualité de la solution fournie. Alors que ces deux mesures font référence à la fin de l'algorithme, nous nous intéressons aussi à certaines mesures qui évoluent avec les générations. Ceci permet surtout pendant le paramétrage d'avoir une meilleure vue sur le processus évolutionnaire. Par exemple, l'observation de la diversité permet de déterminer si l'algorithme est en train de converger. Par la suite nous présentons d'abord les mesures de fin de l'exécution et ensuite les mesures évolutives.

Les EA sont des méthodes stochastiques. On n'obtient donc généralement pas la même solution à chaque exécution. Par conséquent, l'évaluation s'appuie sur une analyse statistique de plusieurs exécutions indépendantes. Nous proposons de toujours faire 30 itérations par paire de graphes dans la base, ce qui donne des résultats assez fiables. Cette approche génère un grand nombre d'observations qu'il est impossible d'évaluer manuellement. Nous fusionnons les résultats obtenus sur des paires de graphes différentes ayant les mêmes caractéristiques (par exemple la même taille et le même niveau de bruit). Avec une cinquantaine de paires de graphes, cela nous amène à 1500 observations à évaluer par test.

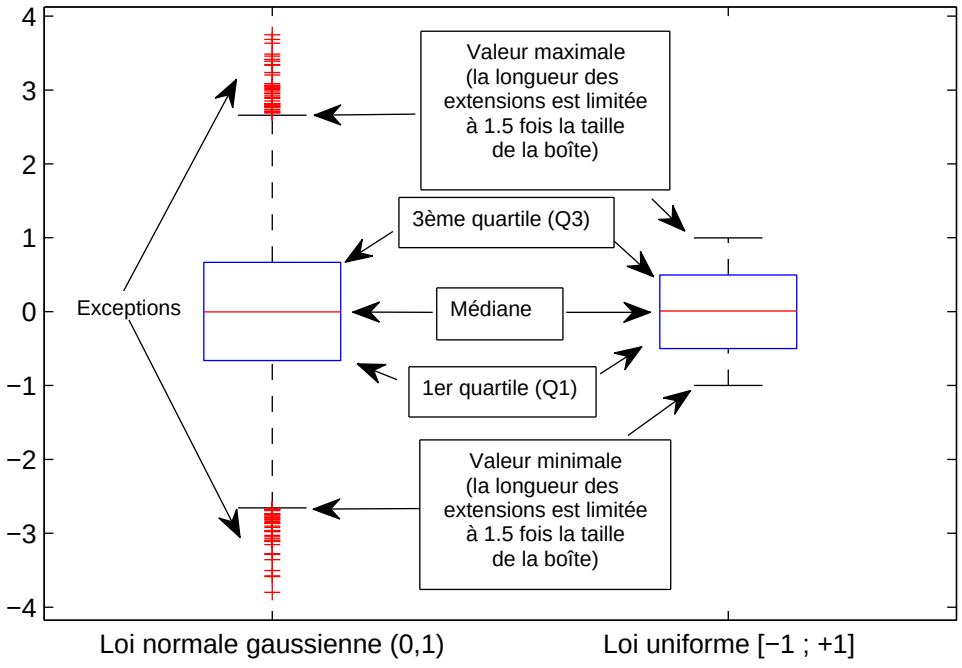

FIGURE 2.1 – Exemple de deux boîtes à moustaches représentant d'un échantillon de 10000 valeurs d'une distribution gaussienne et d'une distribution uniforme.

Nous avons constaté que l'utilisation exclusive de la valeur moyenne omet typiquement des informations importantes sur la variabilité des données. L'écart type est un moyen pour décrire la variabilité. Toutefois, la moyenne et l'écart-type décrivent un échantillon complètement seulement si les données suivent une distribution gaussienne. Les boîtes à moustaches (*boxplot* ou diagrammes *box-and-whisker*, Tukey, 1977; McGill et al., 1978; Velleman & Hoaglin, 1981) quant à elles ne nécessitent pas d'hypothèse sur la distribution des données. De plus, leur visualisation (cf. Fig. 2.1 pour deux exemples) est très intuitive. Une boîte à moustaches est basée sur les quartiles. Chaque quartile décrit exactement un quart de l'échantillon. Le premier quartile représente les 25% des données les plus faibles et sa valeur associée est sa plus grande valeur, le deuxième les 25% suivantes, etc.. La valeur du deuxième quartile est la médiane. Les boîtes à moustaches sont constituées d'un rectangle, la boîte, et ses extensions. La boîte contient la moitié des données et représente les observations moyennes, plus précisément l'écart interquartile, c'est-à-dire la distance entre le premier et le troisième quartile. On y ajoute une ligne à l'endroit où se trouve la médiane. Le minimum et le maximum de l'échantillon sont visualisés par une ligne d'extension à partir de la boîte. Afin de distinguer les exceptions on définit un plafond de la longueur totale des extensions, qui est normalement égal à 1.5 fois la taille de la boîte. Toutes les valeurs qui se trouvent à l'extérieur de cette borne sont considérées comme exceptions. Elles sont visualisées par des points.

Mesures à la fin de l'exécution

Dès que l'algorithme cesse l'exécution, on obtient la meilleure solution trouvée.

Précision En ce qui concerne la précision, celle-ci peut-être évaluée en utilisant la solution optimale si celle-ci est connue. Par exemple, on peut la déterminer avec une approche exhaustive. Toutefois, ceci n'est possible que pour de petites instances du problème en raison de la complexité exponentielle de ces approches. Quand on utilise des données artificielles, il est possible, sous certaines conditions, d'identifier la solution optimale au moment de la création du test : nous utilisons une méthode qui génère un graphe, et qui ensuite utilise une permutation aléatoire pour en extraire le graphe et finalement ajoute du bruit. On considère alors la permutation aléatoire comme la solution de référence. Nous pouvons alors facilement déduire la distance entre les deux graphes sous cette permutation. Nous supposons que celle-ci est bien la solution optimale, bien qu'il soit possible que l'introduction du bruit change le graphe de telle sorte qu'une autre solution ait une distance inférieure. Nous n'avons cependant pas constaté un tel phénomène jusqu'au niveau maximal du bruit choisi ($\pm 10\%$).

Par ailleurs, le même effet peut intervenir lorsque l'on traite des problèmes concrets, généralement en utilisant des données bruitées. Dans ce cas, on n'a aucune garantie que la meilleure solution numérique corresponde à la meilleure solution réelle. Contrairement aux données synthétiques, il n'y a aucun moyen d'éviter cela et non plus de l'identifier dans ces contextes.

Techniquement, il existe des faibles variations de la distance dues à la précision de calcul (cf. Ann. E, p. 257 pour plus de détails) dépendant de l'implémentation. Nous avons été donc contraints d'introduire une tolérance numérique pour la comparaison des résultats. On note que ceci n'est pas seulement nécessaire quand on passe à un autre algorithme ou environnement (par exemple, d'un EA vers un algorithme A* ou de MATLAB vers Java) mais aussi quand on calcule la valeur sur deux chemins différents, par exemple dans le cadre de la recherche locale dont la nouvelle *fitness* se détermine par la différence entre l'ancienne *fitness* et le gain. Avec une précision double une tolérance suffisante pour nos tests est $\tau = 10^{-15}$. Soient c un chromosome, d la distance de référence entre les deux graphes, et f la fonction d'évaluation (inverse), alors le fait que c est la bonne solution n'est pas équivalent à $f(c) = d$ mais à $f(c) < d + \tau$.

Étant donné une solution de référence, on est en mesure de calculer le taux d'exécutions de l'algorithme génétique pour lesquelles la solution optimale a été trouvée, ce que l'on appelle *taux de réussite*, ou SR pour *success rate* (Eiben & Smith, 2007). Nous ne nous intéressons pas seulement à ce pourcentage, mais aussi à la question de savoir si les solutions trouvées sont, pour la plupart, proches ou éloignées de la solution optimale, surtout dans le cas des solutions non-optimales. En fait, une solution qui est très proche de la solution optimale peut être satisfaisante dans de nombreuses applications tandis qu'une solution qui est loin ne l'est sûrement pas. Si l'on se place dans le contexte d'une application d'identification, il suffit que la distance trouvée avec le graphe correspondant soit inférieure à la distance des autres graphes de la base pour que le résultat final soit correct.

On introduit ainsi l'erreur par rapport à la solution de référence. La valeur absolue de l'erreur ($f(c) - d$) dépend non seulement de la qualité de la solution trouvée (déterminée par la fonction d'évaluation $f(c)$) mais aussi de la distance d entre les graphes choisis. Par conséquent, il y a aussi une dépendance à la taille de graphes et au niveau du bruit. Afin d'exclure toute influence de ce type, nous normalisons l'erreur par rapport à la solution de référence. La mesure normalisée se calcule alors par $\frac{f(c)-d}{d}$.

Dans le cas de graphes exactement isomorphes, ceci est impossible parce que la distance de référence d est nulle. Nous conservons donc les valeurs absolues non normalisées.

Si l'on souhaite comparer des résultats sur différents niveaux de bruit, nous constatons que les erreurs normalisées par la solution de référence sont croissantes avec le niveau de bruit. Nous cherchons donc à éliminer cette influence. Les données sont créées aléatoirement et du bruit a été ajouté indépendamment à chaque arête et à chaque sommet. Comme le nombre d'observations indépendamment du bruit est alors assez élevé, nous pouvons appliquer la loi des grands nombres et supposer que la distance moyenne entre des arêtes et des sommets correspondants des deux graphes est égale à l'espérance mathématique de la variable qui contrôle le bruit. Nous disposons alors d'un moyen de normalisation supplémentaire parce que nous pouvons facilement calculer l'espérance mathématique de l'erreur par rapport à la taille de graphes et du niveau du bruit.

Complexité de l'algorithme La mesure la plus directe de la complexité temporelle d'un algorithme est sa durée d'exécution. On constate cependant que cette mesure n'est pas optimale pour des raisons diverses : le temps de calcul est non seulement dépendant de l'ordinateur et de toutes ses caractéristiques, à la fois matérielles et logicielles (du système d'exploitation par exemple), mais aussi de l'implémentation. L'utilisation d'un langage plus ou moins efficace, du plus bas niveau comme de l'assembleur ou de plus haut niveau comme des langages orientées objets, peuvent considérablement changer le temps de calcul, alors que l'algorithme en lui-même reste identique. De même, certaines optimisations du code source peuvent apporter un grand avantage. Si l'on veut comparer des méthodes que l'on trouve dans la littérature, cette comparaison peut donc être fortement influencée par les connaissances en programmation des auteurs. Il est pratiquement impossible d'identifier si les éventuelles différences viennent de l'algorithme ou de son environnement. Même si l'on reimplémente tous les algorithmes soi-même, à part le fait que ceci prend beaucoup de temps, on n'est pas sûr d'avoir implémenté les deux algorithmes avec le même soin.

Bien que le temps de calcul soit parfois la seule valeur disponible pour comparer (notamment quand on s'intéresse à des méthodes très différentes), nous nous intéressons à des mesures qui visent plutôt l'évaluation de l'algorithmique. Dans le cadre des différentes approches génétiques, on peut utiliser d'autres mesures comme par exemple le nombre de générations. Il s'avère alors nécessaire d'examiner le critère d'arrêt. En effet, on cesse souvent l'exécution après qu'un nombre donné de générations n'améliore plus le résultat. Ce critère est normalement combiné avec un plafond en nombre maximal de générations, qui n'est en aucun cas franchi. De temps en temps, on peut aussi définir une qualité de solution suffisamment bonne pour s'arrêter. L'utilisation de ces trois critères pose problème dans leur comparaison. Par exemple, quand on arrive au plafond, alors qu'aucune bonne solution n'a été trouvée et que les dernières générations ont toujours apporté des améliorations. En ce qui concerne le critère adaptatif des générations sans changement, les dernières générations sont prises en compte bien qu'elles n'apportent pratiquement rien. Si l'algorithme cesse parce que la solution optimale est atteinte, l'arrêt est immédiat. Nous constatons cela notamment quand nous comparons des graphes exactement isomorphes, alors nous savons que la solution finale a été trouvée si l'on arrive à une distance de zéro. Afin de comparer le nombre de générations réellement, on ne s'intéresse qu'à *la génération dans laquelle la solution finale a été trouvée*. Néanmoins, cela ne nous permettra pas de comparer des algorithmes ayant des tailles de population différentes car l'effort de calcul par génération est dépendant du nombre de chromosomes à traiter.

Les GA sont en général appliqués sur des problèmes complexes qui sont NP. Le calcul de la fonction d'évaluation nécessite un temps polynomial par rapport à la longueur du chromosome ce qui est très important. Comme les opérations de l'algorithme lui-même ont généralement une très faible complexité, notamment par rapport à la fonction d'évaluation, cette dernière nécessite le plus de temps de calcul. On s'intéresse donc au nombre d'appels à la fonction d'évaluation. Cette mesure est invariante par rapport à la taille de la population (en général, elle est le produit de la taille de

population par le nombre de générations). Toutefois, lorsqu'un algorithme génétique contient des autres parties d'une complexité similaire à celle de la fonction d'évaluation, comme par exemple une heuristique de recherche locale ou une stratégie gloutonne, la comparaison basée sur le nombre d'appels à la fonction d'évaluation ne fournit pas de résultats fiables. C'est le cas, en particulier, si l'on souhaite comparer un algorithme qui contient une telle composante avec un autre qui n'en contient pas. Dans ce cas, on peut éventuellement mettre en relation la complexité de cette composante avec la complexité de la fonction d'évaluation (comme nous le faisons pour la recherche locale dans la section 4.1.1, p. 111) ou bien on utilise le temps de calcul comme mesure en acceptant les inconvénients mentionnés précédemment.

Mesures à observer pendant l'exécution

Une mesure particulièrement intéressante pendant le processus évolutionnaire est la diversité de la population qui décrit la variabilité de ces individus. Une population qui contient de nombreuses copies d'un seul individu est très peu diversifiée. Une faible diversité peut impliquer que l'algorithme est en train de converger pas nécessairement vers l'optimum global. Afin de bénéficier de l'exploration parallèle de l'algorithme génétique, il faut qu'il n'y ait pas trop d'individus identiques dans la population. Le problème principal est que les meilleurs chromosomes risquent d'absorber la population, c'est-à-dire que la population ne contienne que des copies d'eux. Si cela arrive trop tôt, on risque obtenir un optimum local et l'exploration cesse. On dénomme cet effet la convergence prématurée.

La diversité peut être mesurée par le taux de chromosomes non identiques dans la population, ce qui implique la comparaison de tous les chromosomes. Une simplification qui réduit le temps de calcul est de compter les valeurs uniques de la fonction d'évaluation. On peut, dans le cas de l'isomorphisme inexact, supposer que des individus ayant exactement la même *fitness* sont égaux.

Une mesure plus générique est basée sur la distance de Hamming entre les individus. Cette distance met en relation les gènes qui coïncident avec les gènes qui diffèrent sur une paire de chromosomes. Elle examine uniquement le génotype des chromosomes, contrairement à la première mesure. Afin d'obtenir une valeur sur toute la population, il faut d'abord calculer la distance de Hamming entre tous les chromosomes. Puis, on agrège les résultats en une seule valeur.

Il est particulièrement intéressant d'étudier s'il est possible d'atteindre n'importe quelle solution valable sans mutation à partir de la population actuelle. Nous avons donc appliqué une autre mesure visant à vérifier si, pour chaque position et chaque allèle, il y a au moins un individu avec cette combinaison. De fait, s'il existe une position sur laquelle il y a un allèle manquant, alors cette correspondance ne peut pas être établie avec des opérateurs de croisement basés sur la position absolue (non perturbateurs) ou du CX.

L'observation de la diversité de la population nous indique l'arrêt de l'exploration de l'EA, sans

que l'on ait une information de la qualité de la solution finale. En fait, on ne peut savoir s'il s'agit d'un optimum local ou global. L'étude de la qualité des solutions et en particulier du changement de celles-ci indique si l'algorithme continue à améliorer la solution. Pour cela nous utilisons la *fitness* des individus directement. On peut se focaliser sur la *fitness* du meilleur individu, sur la *fitness* moyenne, ou bien sur des boîtes à moustaches de toutes les valeurs dans la population. D'une manière générale, la *fitness* doit être croissante [5] avec le nombre de générations. Il se peut néanmoins que l'on observe des plateaux. En fait, le critère d'arrêt d'un certain nombre de génération sans changement coïncide avec un plateau. On peut donc estimer le paramètre en utilisant cette mesure, c'est-à-dire que, si après avoir fait plusieurs exécutions d'un certain EA, on constate qu'il n'y a que des plateaux (avant le plateau correspondant à la solution finale) d'une longueur inférieure à t générations, on peut définir le critère d'arrêt à t générations sans changement. Par ailleurs, une distance trop faible entre la *fitness* de la meilleure solution et la moyenne indique aussi que la diversité est faible. De toute façon ceci est normal dès que la solution finale a été trouvée. Ceci est encore plus visible si l'on utilise des boîtes à moustaches car leur longueur diminue.

2.2 Cadre général d'application

Afin d'appliquer l'algorithme génétique au problème de l'isomorphisme inexact de sous-graphes, nous devons modéliser des solutions candidates comme *chromosomes*, c'est-à-dire choisir un codage.

Dans l'implémentation classique, les solutions sont modélisées par des vecteurs binaires. Toutefois, de nos jours, une variété de codages adaptés à différents problèmes existe. Leur point commun est leur caractère vectoriel et leurs différences principales résident dans le domaine des valeurs ainsi que dans certaines contraintes. En analogie avec la nature, les vecteurs sont dénommés chromosomes. Le concept de transformer une solution candidate dans une représentation vectorielle est appelé codage. Puis, on dénomme l'ensemble des chromosomes à un moment précis la génération et indépendamment du temps la population.

Le codage ajoute des contraintes souples sur le choix des opérateurs. En fait, il est possible de faire un choix parmi tous les opérateurs existants pour un codage donné, mais certaines combinaisons ne sont pas conciliables, par exemple le croisement à un point et le codage de permutation. On souhaite en général qu'un opérateur ne puisse générer que des solutions valides par rapport au codage choisi. En fait, toutes les solutions générées doivent être évaluées ce qui est coûteux. Si l'on accepte la génération de solutions non valides, alors l'espace génotypique $\mathcal{G}$ est plus grand que l'espace phénotypique Ω ; le premier est l'espace de tous les chromosomes possibles tandis que le dernier correspond à l'espace de recherche du départ. En d'autres termes, le fait de n'accepter que des solutions valides exprime que l'on ne souhaite pas passer à un espace de recherche plus grand

5. Dans le cas d'une fonction d'évaluation inverse qui est à minimiser, la *fitness* peut être décroissante.

par le codage. Par exemple pour un problème de permutations, tous les descendants de deux permutations devront toujours rester des permutations, ce qui n'est pas le cas si l'on applique par exemple un croisement à deux points.

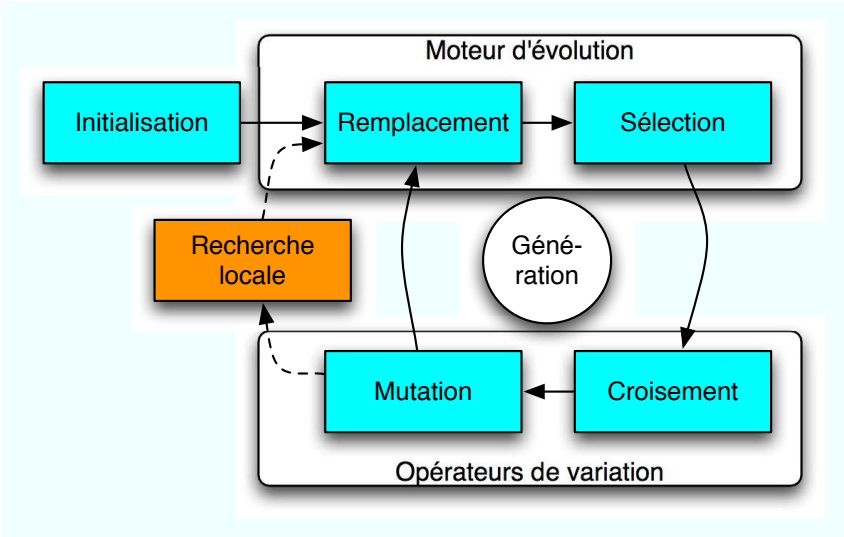

FIGURE 2.2 – Principe de base.

La population est soumise itérativement, comme dans la nature, à la sélection des individus les plus aptes, modélisée par un moteur d'évolution, et des opérateurs de variation nécessaires pour l'exploration de l'espace de recherche. Certains algorithmes incluent une stratégie de recherche locale. Ils sont dénommés algorithmes *mémétiques*. La figure 2.2 illustre le processus. Pendant la sélection, les chromosomes qui vont se reproduire sont choisis. Bien que l'on préfère les solutions qui sont les plus adaptées au problème (déterminées par une mesure de *fitness*), le processus n'est pas complètement déterministe afin de maintenir la diversité de la population. Ceci est particulièrement important pour pouvoir quitter les optimaux locaux visités. Le but du croisement est de recombiner des parties de deux chromosomes (parents) afin de créer un enfant. Enfin la mutation est appliquée pour manipuler une partie de l'enfant. Tous les changements des chromosomes sont aléatoires. On introduit avec la sélection un biais vers les solutions les plus adaptées au problème, ce qui permet d'obtenir une bonne solution après un certain nombre de générations.

Afin d'assurer la conformité des solutions créées, il existe quatre approches. La première consiste à refuser les solutions invalides. On applique les opérateurs génétiques normalement et on vérifie ensuite la validité des descendants. Tant qu'il y a des solutions non valides, on les supprime. La deuxième méthode est la réparation. On reconstitue une solution valide à partir d'un descendant non valide. Par exemple dans le cas des permutations, on remplace toutes les valeurs présentes deux fois par des valeurs absentes dans le chromosome. Troisièmement, on accepte le fait d'avoir des solutions invalides mais on les pénalise. C'est une méthode largement utilisée si l'espace des solutions

valides se constitue de plusieurs fragments qui peuvent être déconnectés en raison d'une multitude de contraintes. Une fonction de pénalité est utilisée pour diminuer la *fitness* des solutions non valides. La quatrième méthode, celle que nous adoptons ici, est le codage *intelligent*. On utilise un codage et des opérateurs compatibles à celui-ci, tels que les contraintes ne sont jamais violées.

2.3 Codage et fonction d'évaluation

Phénotype et génotype En biologie, on distingue le phénotype d'un individu et son génotype : le phénotype contient les caractéristiques réelles tandis que le génotype représente les informations génétiques qui les provoquent. En particulier il peut y avoir des gènes dont les effets sont supprimés par d'autres. Dans les algorithmes génétiques le phénotype est la solution candidate tandis que le génotype est son codage. Il est nécessaire de passer d'un espace de recherche (ou phénotypique) Ω à l'espace des chromosomes (ou génotypique) $\mathcal{G}$. Le processus du codage est formellement une application $\Omega \rightarrow \mathcal{G}$. Les opérateurs génétiques fonctionnent sur $\mathcal{G}$ tandis que l'évaluation des individus est faite dans Ω. Le processus de retour vers l'espace phénotypique $\mathcal{G} \rightarrow \Omega$ est dénommé morphogénèse. La morphogénèse permet en particulier de récupérer la solution finale à partir du chromosome associé. On souhaite que des phénotypes proches soient modélisés par des génotypes qui sont eux aussi proches.

La solution au problème de l'isomorphisme de graphes est une fonction $m : \mathcal{V}' \rightarrow \mathcal{V}$ qui associe les sommets d'un graphe aux sommets de l'autre. Toute fonction valide possède une représentation équivalente dans l'espace des permutations de n éléments. En fait, on suppose que les éléments de $\mathcal{V}$ ainsi que ceux de $\mathcal{V}'$ sont numérotés aléatoirement de 1 à n. Alors l'affectation m correspondante contient toutes les paires de nœuds (v', v) ayant le même numéro. En fait, une permutation de la numérotation de $\mathcal{V}$ implique une fonction unique, et vice versa.

Épistasie On dénomme *épistasie* l'interaction entre les gènes d'un chromosome, et plus précisément, la dépendance d'un gène à d'autres gènes. L'épistasie est une caractéristique qui peut être très défavorable pour le succès de l'algorithme en combinaison avec certains opérateurs. Dans l'hypothèse des *building blocks* (cf. Sec. 2.1.1, p. 24), on suppose que l'épistasie est plus forte entre les gènes proches qu'entre des gènes lointains. Dans notre codage, il y a une épistasie constante entre tous les gènes du chromosome ce qui conduit à utiliser des opérateurs uniformes.

2.3.1 Codage robuste pour l'isomorphisme de sous-graphes

Dans le cas de l'isomorphisme de sous-graphes, un codage standard basé sur les sommets du graphe à comparer impliquerait un chromosome de taille égale au nombre de sommets du graphe plus petit. Certes, ce codage représenterait une permutation partielle mais il faudrait mettre en œuvre des méthodes de réparation des solutions non valides, ainsi que l'incorporation et l'échange des

valeurs manquantes. Ceci augmenterait non seulement la complexité temporelle de l'algorithme, mais aussi son nombre de paramètres.

En réalité, la distinction entre phénotype et génotype permet l'utilisation du codage de permutation même dans le contexte d'isomorphisme de sous-graphes. En effet, considérons la fonction d'affectation des nœuds depuis le graphe le plus petit vers le graphe le plus grand. Le chromosome, correspondant au génotype de la solution, contient une permutation entière des nœuds du graphe le plus grand. Les caractéristiques externes, c'est-à-dire le phénotype, sont déterminées par un sous-ensemble (de la taille du graphe le plus petit) de ses allèles uniquement. L'évaluation d'un individu est donc restreinte aux nœuds qui font partie de ce sous-ensemble. Les autres nœuds restent présents dans le génotype et nous les considérons comme la mémoire de valeurs manquantes qui naturellement préservent les caractéristiques de permutation pendant l'application des opérateurs génétiques, y compris l'heuristique locale. Aussi, le résultat est un codage assez robuste pour le traitement des problèmes de l'isomorphisme de graphes et de sous-graphes.

2.3.2 Appariements multivoques

Les appariements multivoques présentent une limitation réelle du codage de permutation. Dans un appariement multivoque, la relation entre les sommets n'est pas injective. Selon le domaine d'application, il peut y avoir des sommets qui doivent être appariés avec plusieurs. Le traitement des appariements multivoques nécessite ou bien une fonction calculant la distance entre sous-graphes (voire entre un sommet d'un graphe et un sous-graphe de l'autre) ou bien une méthode fiable d'agrégation des attributs des sommets et arêtes. Dans les deux cas, ces méthodes sont très dépendantes de l'application.

Notre but est de développer une méthode indépendante de l'application et dans ce contexte nous nous limitons à des appariements univoques. Toutefois, il est possible d'ajouter un module de recherche de fusion/division de sommets à l'algorithme pour une application précise. On pourrait aussi ajouter au chromosome une partie qui encode les fusions/divisions/suppressions des sommets que l'on traiterait séparément. Sur cette partie du chromosome on appliquerait d'autres opérateurs et on ne permettrait pas d'interaction avec la partie du codage de permutations. Avec cette approche on sépare le problème de la recherche de l'alignement du problème de la recherche des éléments des différents niveaux de granularité, ce qui nous semble une bonne idée étant donné la complexité des deux tâches.

2.3.3 Fonction d'évaluation

Le problème de l'isomorphisme inexact de sous-graphes vise également à minimiser l'équation 1.1 (cf. p. 11) pour des graphes de tailles différentes.

$$\delta_m\left(G, G'\right) = w \sum_{v \in \mathcal{V}} \delta_V\left(v, m(v)\right) + (1-w) \sum_{v_1, v_2 \in V} \delta_E\left((v_1, v_2), (m(v_1), m(v_2))\right)$$

Si l'on suppose sans perte de généralité que $n' < n$, l'espace de solution peut aussi être projeté sur l'espace des permutations de n éléments. Certes ceci implique une sur-occupation de mémoire de l'ordre $\mathcal{O}(n - n')$, alors que l'occupation optimale est de $\mathcal{O}(n')$, et plusieurs permutations impliquent la même fonction, mais cela simplifie l'usage d'un algorithme génétique utilisant le codage de permutation.

Étant donné la formulation du problème comme problème d'optimisation combinatoire par Berretti, notamment par les équations 1.2 ($\delta\left(G, G'\right) = \min_m \delta_m\left(G, G'\right)$) et 1.1 dont l'appariement m recherché est un appariement bijectif (quand les deux graphes ont la même taille) entre deux ensembles, il semble naturel de coder les chromosomes à l'aide du codage de permutation. De plus, plusieurs opérateurs bien étudiés sont disponibles pour ce type de codage.

La qualité d'un individu par rapport au problème est dénommé sa *fitness*. Au sens strict, la *fitness* est croissante avec la qualité de l'individu. La fonction qui calcule la *fitness* de l'individu porte plusieurs noms dont les plus utilisés sont fonction d'évaluation, fonction *fitness* ou fonction-objectif. En pratique, la limitation des EA à des problèmes de maximisation vient d'une simple question de définition de cette fonction. Surtout, le terme fonction *fitness* est souvent associé au sens strict tandis que fonction d'évaluation souligne la généralité et peut aussi bien être une fonction à minimiser (une fonction *fitness* inverse).

Dans notre cas, la fonction d'évaluation est à minimiser. Il s'agit d'une mesure d'erreur, plus précisément une mesure de la distance qu'un individu implique sur les deux graphes considérés. Elle est donnée par l'équation 1.1. Un individu est mieux adapté qu'un autre si sa *fitness* est inférieure.

Inspiration de problèmes similaires Quand on développe un EA pour un nouveau problème, des problèmes similaires peuvent fournir des points de départ. En particulier, s'il s'agit de problèmes bien étudiés pour lesquels de nombreux EA ont été proposés. Dans notre cas, les problèmes similaires sont d'une part le problème du voyageur de commerce (TSP) et le problème quadratique d'affectation (QAP). Pour tous les deux, de nombreuses approches se basent sur le codage de permutation ce qui permet de s'inspirer des opérateurs utilisés. Le TSP est, en effet, un problème très bien et très longtemps étudié, bien qu'aujourd'hui les meilleurs résultats ne soient plus obtenus avec le codage de permutation. Trois états de l'art qui focalisent sur les opérateurs se trouvent dans Oliver et al. (1987); Starkweather et al. (1991) et Larranaga et al. (1999). Ce dernier donne également des détails sur d'autres représentations. Buriol et al. (2004) présentent un algorithme mémétique très optimisé pour le TSP asymétrique. Dans ce travail on trouve plusieurs opérateurs de croisement spécialisés pour le TSP asymétrique, notamment une implémentation du croisement préservant la

distance (DPX) pour le cas asymétrique ainsi qu'une nouvelle méthode de recherche locale (l'insertion récursive des arcs). Une approche plus récente utilise des populations multiples avec également des opérateurs de croisement, mutation et de recherche locale spécialisés (Nguyen et al., 2007). De plus, la population est organisée hiérarchiquement. En ce qui concerne le QAP, il est moins étudié que le TSP surtout en termes d'opérateurs mais constitue également un problème d'optimisation dans l'espace de permutations. On peut citer notamment les travaux de Merz & Freisleben (1997) qui proposent l'opérateur DPX.

2.4 Moteurs d'évolution

Après avoir défini le codage des solutions et leur *fitness*, il est nécessaire de modéliser le processus d'évolution : les moteurs d'évolution déterminent des stratégies de reproduction des individus et de remplacement entre la génération parentale et les enfants qui viennent d'être créés. Ils sont directement liés avec les stratégies de sélection qui déterminent comment les géniteurs, c'est-à-dire les chromosomes qui vont se reproduire, sont choisis. Nous introduisons d'abord les moteurs d'évolution les plus importants, puis décrivons les stratégies de sélection.

2.4.1 Moteur générationnel et *steady-state*

Deux principales stratégies de remplacement peuvent être définies : le moteur générationnel remplace toute la génération parentale par la génération des enfants. Dans ce but, on crée autant d'enfants qu'il y a de parents. Il s'agit du moteur classique, c'est pour cela que l'on dénomme les algorithmes génétiques qui implémentent ce moteur des algorithmes génétiques canoniques.

Contrairement au moteur générationnel, le moteur *steady-state* ne remplace qu'un seul individu à la fois. On choisit donc seulement un ou deux géniteurs (suivant que l'on applique le croisement ou non). Ensuite l'individu créé à partir de ces géniteurs remplace un individu de la population. Il y a plusieurs façons de choisir l'individu qui est éliminé : on peut éliminer par exemple le pire individu, l'individu le plus âgé ou simplement un individu aléatoire. Il est également possible d'inverser une stratégie de sélection, et par exemple de réaliser un « anti-tournoi » qui consiste à choisir aléatoirement n (souvent on prend $n = 2$) individus parmi la population et à éliminer celui ayant la *fitness* la plus faible.

On peut définir également des moteurs intermédiaires entre le moteur générationnel et le moteur *steady-state* : au lieu de remplacer toute la population ou seulement un individu, on peut librement définir le taux d'individus à remplacer. On dénomme ce taux le fossé générationnel (De Jong, 1975; De Jong & Sarma, 1993). Le fossé d'un algorithme générationnel est donc 1 et celui d'un algorithme *steady-state* est de $\frac{1}{s_p}$. Nous nous limitons dans cette thèse à l'étude des moteurs générationnel et *steady-state* qui sont les moteurs les plus souvent utilisés.

2.4.2 Sélection

La stratégie de sélection détermine deux caractéristiques-clés des algorithmes génétiques : il s'agit d'une part du compromis entre exploration et exploitation et d'autre part de la diversité.

Exploration et exploitation Alors que l'exploration fait référence à la recherche en largeur dans l'espace de recherche pour identifier des régions prometteuses, l'exploitation consiste en la recherche en profondeur dans ces régions. Une sélection forte favorise beaucoup plus la reproduction des individus plus adaptés qu'une sélection plus faible. En d'autres termes, si la sélection est forte et la pression sélective est donc élevée, alors la probabilité qu'un individu soit transmis à la génération suivante est fortement corrélée à sa *fitness*. Dans ce cas, l'algorithme exploite très efficacement une région restreinte, mais il risque de ne trouver qu'un optimum local. Si la corrélation entre la *fitness* d'un individu et sa probabilité de reproduction est moins forte, alors la composante aléatoire a une influence plus élevée. Dans ce cas, l'algorithme explore bien l'espace mais il ne focalise pas aussi bien la recherche sur une région précise. Une pression élevée mène à une convergence plus rapide mais au coût d'une faible exploration de l'espace de recherche. Par conséquent, on ne peut obtenir qu'un compromis entre exploration et exploitation.

Une méthode pour quantifier la pression sélective est la notion d'intensité proposée par Blickle & Thiele (1996). Soit $\mathcal{F}$ la distribution des *fitness* de la population avant la sélection, $\mathcal{F}^*$ la distribution après la sélection et σ l'écart-type. L'intensité I est définie comme le changement de la moyenne des *fitness* causé par la sélection, normalisé par l'écart-type :

$$I = \frac{\overline{\mathcal{F}^*} - \overline{\mathcal{F}}}{\sigma(\mathcal{F})} \tag{2.3}$$

L'évolution de la variance des *fitness* d'une génération à l'autre est quantifiée par Blickle & Thiele (1996) :

$$\Phi = \frac{\sigma^2(\mathcal{F}^*)}{\sigma^2(\mathcal{F})} \tag{2.4}$$

Le comportement souhaité de la sélection est que la pression de sélection augmente avec le temps ou bien qu'elle reste constante au cours de l'algorithme. D'une part, un EA qui sélectionne trop fortement au début risque de converger vers des optima locaux trop tôt sans explorer les autres régions prometteuses. Les résultats ne seront donc pas de bonne qualité. D'autre part, si l'EA ne sélectionne pas assez strictement vers la fin, il ne converge que très lentement.

Pour les stratégies proportionnelles à la *fitness*, l'effet d'échelle pose un problème. Si l'on considère deux fonctions d'évaluation $f(c)$ et $f'(c) = f(c) + a$, alors leur maximisation est équivalente (cf. Fig. 2.3, p. 39 par exemple), par contre, les probabilités de sélection changent : une augmentation de la constante mène à une simple translation de la fonction d'évaluation, mais uniformise les

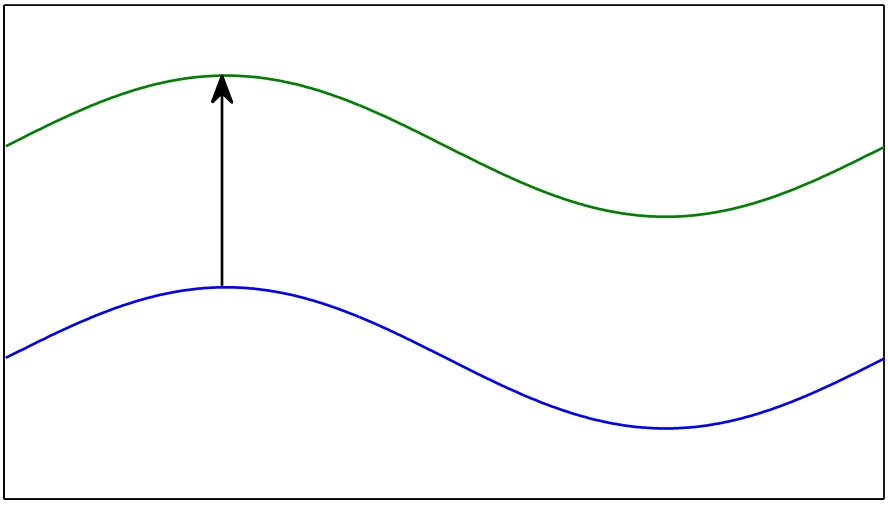

FIGURE 2.3 – Exemple de fonctions d'évaluation décalées.

probabilités. Pour $x \to \infty$ la probabilité pour chaque individu est inverse à la taille de la population (s_p) : $\frac{1}{s_p}$. On peut noter que ce problème est aussi causé par le moteur évolutionnaire lui-même. En fait, la *fitness* moyenne des individus augmente au fil des générations. Par conséquent, les probabilités s'uniformisent automatiquement au cours de l'algorithme. La sélection devient donc plus faible avec le temps.

Afin d'obtenir une stratégie invariante à des décalages, une normalisation de la *fitness* est nécessaire. Deux méthodes couramment utilisées dans ce but sont l'ajustement linéaire (*linear scaling*) et l'ajustement σ (σ*-scaling*). L'idée de l'ajustement linéaire est de ramener la *fitness* minimale de chaque génération à zéro. Simultanément un paramètre ϑ détermine la dispersion des valeurs. En général, ϑ est choisi tel que l'espérance du nombre de copies du meilleur individu dans la population soit égale à une constante (par exemple entre 1.2 et 2). La fonction d'évaluation normalisée f' se calcule par :

$$f'(c) = \vartheta \left(f(c) - \min(\mathcal{F}) \right) \tag{2.5}$$

Si l'on souhaite obtenir n_c copies du meilleur individu, ceci implique que la *fitness* maximale soit deux fois supérieure à la *fitness* moyenne de la génération. ϑ est donc donné par :

$$\vartheta = \frac{\overline{\mathcal{F}} - \min(\mathcal{F})}{\max(\mathcal{F}) - \min(\mathcal{F})} \times n_c \tag{2.6}$$

L'ajustement σ prend en compte la variance de la *fitness* de la génération. La fonction d'évaluation normalisée est donnée par :

$$f'(c) = f(c) - (\overline{\mathcal{F}} - \varrho\sigma(\mathcal{F})) \tag{2.7}$$

Le paramètre $\varrho \in [1, 2]$ contrôle l'influence de la variance.

Sélection par tournoi Contrairement à d'autres stratégies de sélection comme la sélection proportionnelle ou la sélection basée sur le rang, la sélection par tournoi ne nécessite ni la normalisation ni l'ordonnancement des valeurs sélectives de toute la génération. De plus son utilisation sur une fonction d'évaluation inverse par rapport à la *fitness*, comme nous la considérons, ne pose aucun problème. En outre, du point de vue du calcul, elle n'est pas coûteuse. Elle nécessite néanmoins un paramètre additionnel, la taille du tournoi. Puisque cette taille est un nombre entier, le contrôle de la pression de sélection se fait dans un domaine discret et ne peut jamais être effectuée en continu. Malgré cet inconvénient les avantages de la sélection par tournoi la font prévaloir.

Soit n_t la taille du tournoi, la sélection par tournoi se déroule comme suit : Un individu est sélectionné comme parent s'il gagne un tournoi impliquant n_t participants. On tire donc (avec remise) aléatoirement n_t individus de la population et on compare leur *fitness*. L'individu avec la meilleure *fitness* gagne et il est copié vers la population intermédiaire des parents. On répète cette procédure jusqu'à ce que un nombre suffisant de parents ait été sélectionné. La sélection par tournoi ne garantit pas que le meilleur individu soit préservé. De fait, s'il participait à un tournoi il le gagnerait, mais il se peut qu'il ne soit tiré comme participant à aucun tournoi. Par contre, le pire individu n'est jamais transmis.

2.5 Croisement

2.5.1 Caractéristiques générales

Dans l'analyse des opérateurs de croisement, il existe deux caractéristiques de grande importance : le biais de position (*positional bias*) et le biais de distribution (*distributional bias*). On tente généralement d'éviter tout type de biais. De fait, l'existence d'un biais indique une préférence pour certaines solutions. Par conséquent, il s'agit d'une forme de sélection à l'intérieur de l'opérateur. Cette sélection se base sur des critères autres que la *fitness* des individus. La direction de recherche d'un EA devrait cependant dépendre uniquement de la *fitness*, et le moteur évolutionnaire est constitué principalement de la stratégie de sélection (et celle de remplacement). Les tâches des autres opérateurs assurent la création des nouveaux échantillons et donc l'exploration.

Un opérateur a un biais de position, si la probabilité que deux gènes soient transmis ensemble dépend de leurs positions relatives. Par exemple, le croisement « un point » qui échange tous les gènes à partir d'une position fixe aléatoire présente un biais de position car deux gènes, dont l'un est proche de l'autre, ont une plus haute probabilité d'être transmis ensemble que deux gènes plus

éloignés. Comme la disposition des gènes dans le chromosome est primordiale pour le succès de l'algorithme génétique, cette propriété est - dans la plupart des applications - indésirable. Il existe des versions « uniformes » de la plupart des opérateurs concernés, qui éliminent ce biais.

Un biais de distribution se produit lorsque le nombre transmis d'un parent vers l'enfant ne suit pas une distribution uniforme. En conséquence, la probabilité de conserver des solutions partielles dans la génération suivante dépend de leur taille. Un exemple est le croisement uniforme où le nombre de gènes échangés suit une distribution binomiale.

Bien qu'il soit souhaitable de travailler avec des opérateurs sans biais de distribution ni de position, un tel opérateur n'est pas facile à trouver. Un exemple d'un opérateur d'un biais de distribution et de position très faible est le croisement brassé (cf. Sec. 2.5.2, p. 41).

Une autre propriété des opérateurs de croisement est leur force perturbatrice. On peut considérer la recombinaison des *building blocks* comme la tâche principale du croisement. Un *building block* est un sous-ensemble des gènes du chromosome fournissant une bonne solution partielle du problème. Un opérateur ne devrait pas perturber trop ces *building blocks*, c'est-à-dire les décomposer en deux parties qui seront transmises à deux descendants différents. Il serait souhaitable de pouvoir les identifier afin de les transmettre en entier et les recombiner avec un autre *building block* présent dans l'autre parent. Contrairement à un codage basé sur l'ordre des éléments où les *building blocks* sont des sous-chaînes connexes du chromosome, dans notre cas, un *building block* peut-être n'importe quel sous-ensemble des gènes. Ainsi, un opérateur uniforme n'est pas plus perturbateur que le croisement à un point par exemple.

Un opérateur plus perturbateur peut être identifié par le nombre de gènes qui ne sont hérités ni d'un parent ni de l'autre.

2.5.2 Opérateurs génériques

Les opérateurs de croisement pour le codage de permutation s'inspirent en partie des opérateurs classiques. Les quatre opérateurs qui seront présentés par la suite sont illustrés dans la figure 2.4 sur un exemple général. L'opérateur le plus simple est le croisement à un point, parfois appelé croisement standard. L'idée fondamentale est de couper les chromosomes des deux parents en la même position aléatoire et d'échanger la partie au-dessus (ou au-dessous) de cette coupure entre les chromosomes. Le croisement à deux points est défini de la même façon, sauf que deux positions sont choisies et la partie entre celles-ci est échangée. On constate que les deux opérateurs ont un biais sur la position car deux gènes voisins ont une plus grande probabilité d'être transmis ensemble que deux gènes lointains. Pour éviter cela, le croisement uniforme (UX) a été introduit : il consiste à choisir aléatoirement, pour chaque gène, s'il sera transmis ou non du parent à l'enfant lors du croisement. Bien que le croisement uniforme n'ait pas de biais sur la position, il présente un biais sur la distribution, c'est-à-dire qu'il est plus probable que chaque parent transmette la moitié des

gènes plutôt qu'un nombre plus ou moins grand. Le croisement brassé se déroule en trois phases : premièrement, les gènes sur les deux chromosomes sont brassés aléatoirement, mais en utilisant la même permutation pour les deux ; deuxièmement, on applique le croisement à un point ; enfin, on trie les enfants obtenus selon l'ordre d'origine. Il ne présente aucun des deux types de biais mentionnés auparavant.

2.5.3 Opérateurs spécifiques pour les permutations

Il existe trois familles d'opérateurs de croisement pour les problèmes de permutations : les opérateurs préservant la position relative comme le croisement basé sur l'ordre (OX), ceux préservant la position absolue des allèles comme le croisement cyclique (CX) et les opérateurs hybrides comme le croisement à correspondance partielle (PMX). Le choix d'un opérateur dépend de l'application et notamment du type d'information le plus important, autrement dit de l'information qui a le plus d'influence sur la *fitness* de l'individu. Les deux informations principales présentes sur le chromosome sont la position absolue des allèles et leur ordre. Dans le cas de l'isomorphisme de graphes, l'ordre des allèles n'a aucune importance, puisque, par définition, les numérotations dans les deux graphes sont aléatoires. L'influence d'un allèle spécifique sur la *fitness* d'un individu est alors soumise à ces relations avec chacun des autres allèles. En effet chaque allèle interagit avec tous les autres. Nous pouvons donc considérer le chromosome comme un ensemble non ordonné de paires de nœuds. Bien que l'opérateur le plus utilisé dans la littérature soit le PMX (Goldberg & Lingle, 1985), nous pouvons constater qu'étant donné que l'ordre des allèles n'a pas d'importance, des opérateurs basés sur l'ordre absolu sont à priori les plus adaptés au problème. Il existe bien évidemment une grande variété d'opérateurs spécialisés pour des problèmes spécifiques, comme par exemple le *Non-Wrapping Order Crossover, NWOX* (Cicirello, 2006) sur un problème de *scheduling* ou le *Edge-Assembly Crossover, EAX* (Nagata & Kobayashi, 1997) sur le TSP.

2.5.4 Opérateurs classiques

Les trois opérateurs suivants sont illustrés dans la figure 2.5.

Croisement cyclique Parmi les opérateurs de croisement visant la position absolue, l'exemple le plus classique est le CX (Oliver et al., 1987). Cet opérateur n'a pas de biais de position ni de distribution. Les chromosomes sont répartis en sous-ensembles de gènes contenant les mêmes allèles, ce qui signifie que l'ensemble des valeurs prises par les gènes inclus dans un sous-ensemble donné est identique dans les deux parents. Pour obtenir une telle partition, on utilise des cycles : en commençant par un gène aléatoirement choisi dans un parent, on l'ajoute au cycle actuel et on recherche son allèle dans l'autre parent. Puis on ajoute aussi ce gène et on répète itérativement cette stratégie jusqu'à retomber sur un gène déjà transmis. Le cycle est alors bouclé et on remplit les gènes manquants

Croisement à un point							
Parent A	1	2	3	4	5	6	7
Parent B	A	B	C	D	E	F	G
				⇓			
Enfant 1	1	2	3	D	E	F	G
Enfant 2	A	B	C	4	5	6	7
Croisement à deux points							
Parent A	1	2	3	4	5	6	7
Parent B	A	B	C	D	E	F	G
				⇓			
Enfant 1	1	2	C	D	E	6	7
Enfant 2	A	B	3	4	5	F	G
Croisement uniforme							
Parent A	1	2	3	4	5	6	7
Bitmap	1	0	1	0	1	0	1
Parent B	A	B	C	D	E	F	G
				⇓			
Enfant 1	1	B	3	D	5	F	7
Enfant 2	A	2	C	4	E	6	G
Croisement brassé							
Parent A	1	2	3	4	5	6	7
Parent B	A	B	C	D	E	F	G
Permutation	6	3	7	5	1	2	4
Brassage				⇓			
Parent A brassé	6	3	7	5	1	2	4
Parent B brassé	F	C	G	E	A	B	D
Croisement à un point				⇓			
Enfant 1 brassé	6	3	7	E	A	B	D
Enfant 2 brassé	F	C	G	5	1	2	4
Debrassage				⇓			
Enfant 1	A	B	3	D	E	6	7
Enfant 2	1	2	C	4	5	F	G

FIGURE 2.4 – Opérateurs basiques de croisement.

avec les allèles du deuxième parent.

On note qu'il est impossible qu'un gène puisse prendre une autre valeur que celles qu'il prenait chez les parents. Ceci peut mener à une exploration incomplète, notamment dans le cas où un certain allèle n'existe pas dans la population au début de l'algorithme ou pendant les premières générations. Cet allèle serait alors perdu à jamais, sauf s'il est réintroduit par un autre opérateur, notamment la mutation.

Croisement à correspondance partielle Le PMX (Goldberg & Lingle, 1985) vise à préserver partiellement la position absolue et l'ordre des gènes. Il est basé sur le croisement à deux points. La partie intermédiaire du chromosome est échangée. Toutefois, afin de préserver une permutation valide, les gènes ne sont pas copiés directement. En effet, après avoir déterminé quel gène doit être à une position spécifique, il y est transféré par une opération d'échange avec le gène actuellement sur ce locus. On répète cette procédure pour tous les gènes définis initialement. On remarque que cet opérateur représente aussi un biais sur la position. C'est pour cela qu'une version uniforme, l'croisement uniforme à correspondance partielle (UPMX), a été proposée par Cicirello & Smith (2000). Cet opérateur détermine pour chaque position (indépendamment) si le gène va être échangé et le cas échéant, effectue la même opération d'échange que le PMX classique.

Croisement basé sur l'ordre L'idée fondamentale d'OX (Davis, 1985) est de préserver l'ordre des gènes. On copie une partie du chromosome d'un parent. Puis on insère les gènes manquants dans leur ordre chez le second parent dans les positions manquantes. La version uniforme de l'opérateur, le croisement uniforme basé sur l'ordre (UOX, Syswerda, 1991), permet d'éviter le biais sur la position de la version originale. Tandis que OX coupe le chromosome à une ou deux positions pour obtenir le sous-ensemble des gènes à copier, UOX est basé sur UX soit un bitmap détermine les positions. Cela n'évite pas un biais sur la distribution.

2.5.5 Nouveaux opérateurs

Croisement strictement basé sur la position (PBX) PBX (Bärecke & Detyniecki, 2007b) a pour but de préserver au mieux la position absolue des allèles en ignorant complètement leur ordre. À la différence du croisement basé sur la position (POS) de Syswerda (1991), l'ordre des allèles ne joue aucun rôle. De plus, tous les allèles qui occupent la même position dans les deux parents sont toujours conservés. L'opérateur fonctionne de manière uniforme et n'a donc pas de biais de position. Quant au biais de distribution les solutions partielles de taille moyenne sont favorisées. En effet, ceci permet d'avoir une influence équilibrée des deux parents sur les descendants (*offspring*) - non seulement en moyenne sur tout l'algorithme mais à chaque reproduction individuelle. L'opérateur suit les étapes suivantes :

CX							
Parent A	1	2	3	4	5	6	7
Parent B	*3*	*4*	*7*	*6*	*1*	*2*	*5*
Cycles	0	X	0	X	0	X	0
				⇓			
Enfant 1	1	*4*	3	*6*	5	*2*	7
Enfant 2	*3*	2	*7*	4	*1*	6	*5*

PMX							
Parent A	1	2	3	4	5	6	7
Parent B	*3*	*4*	*7*	*6*	*1*	*2*	*5*
	Échange de la partie intermédiaire						
	(entre parenthèses la valeur antérieure)						
				⇓			
Enfant 1	1	2	*7*(3)	*6*(4)	*1*(5)	6	7
Enfant 2	*3*	*4*	3(*7*)	4(*6*)	5(*1*)	*2*	*5*
	Réparation de la permutation						
	Échange des mêmes valeurs à l'extérieur						
	(p.ex 7->3 de la zone choisie)						
				⇓			
Enfant 1	5	2	*7*	*6*	*1*	4	3
Enfant 2	*7*	*6*	3	4	5	*2*	*1*

OX							
Parent A	1	2	3	4	5	6	7
Parent B	*3*	*4*	*7*	*6*	*1*	*2*	*5*
Bitmap	0	0	X	X	X	X	0
				⇓			
Enfant 1	1	2	*3*	*4*	*6*	*5*	7
Enfant 2	*3*	*4*	1	2	6	7	*5*

FIGURE 2.5 – Opérateurs de croisement pour les permutations.

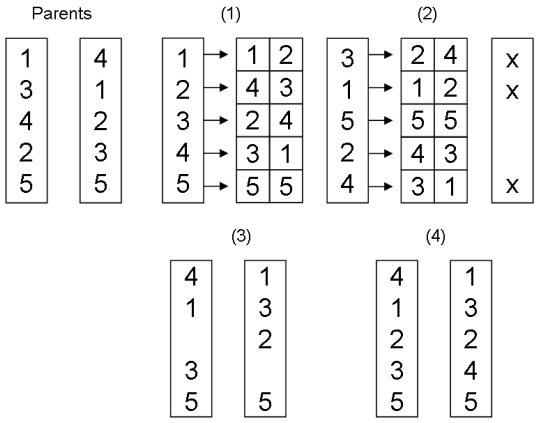

FIGURE 2.6 – Croisement uniforme basé strictement sur la position absolue (UPBX) - les étapes correspondent aux étapes mentionnées dans le texte - sur cet exemple l'étape 5 n'est pas nécessaire car les enfants sont complets après l'étape 4.

(1) Construire une liste l qui contient chaque allèle et sa position dans les deux parents.
(2) Réorganiser la liste selon un ordre aléatoire (UPBX uniquement) et choisir aléatoirement les allèles qui vont échanger leurs positions.
(3) Construire les enfants en parcourant la liste. Dans le cas d'une collision (un autre allèle occupe déjà la position désirée) dans un enfant, sauvegarder cet allèle et ses positions.
(4) Essayer de mettre les allèles non-utilisés à la position qu'ils ont occupée dans l'autre parent.
(5) Remplir les chromosomes avec les allèles manquants dans l'ordre de la liste.

L'algorithme est illustré sur un exemple sur la figure 2.6. L'idée de cet opérateur consiste à ne jamais utiliser l'ordre des allèles et à mettre le plus grand nombre d'allèles possible dans des positions qu'ils ont occupées dans un des parents. L'opérateur est uniforme par rapport au choix de gènes à échanger entre les parents. Finalement, le nombre d'allèles hérités des deux parents est égal et il est garanti que, si un allèle avait la même position dans ses parents, il gardera cette position dans les deux enfants. Un inconvénient du PBX est qu'il parcourt la liste des allèles toujours dans le même ordre. En cas de conflits sur un locus (c'est-à-dire s'il y a *a priori* deux allèles à placer sur cette position), ce serait donc toujours l'allèle le plus petit qui aurait la priorité. Comme les allèles viennent d'une numérotation aléatoire des sommets du deuxième graphe, l'ordre est aléatoire, mais il ne change pas pendant une exécution de l'algorithme génétique. La non-uniformité réside dans le fait que ce sont toujours les mêmes positions qui auront la priorité et l'opérateur est donc biaisé dans ce sens. Nous proposons alors l'opérateur de croisement uniforme basé strictement sur la position absolue (UPBX, Bärecke & Detyniecki, 2007b) comme variante uniforme du PBX. La seule différence par rapport au PBX est le brassage de la liste dans l'étape 2.

Croisement préservant la distance (DPX) Nous proposons trois nouveaux opérateurs qui s'inspirent de DPX proposé par Freisleben & Merz (1996) pour le problème du voyageur de commerce (TSP). L'idée fondamentale de DPX est de créer un enfant dont la distance à chacun des deux parents est égale à la distance entre les parents, on définit la distance d comme le nombre de gènes distincts entre les chromosomes ce qui correspond à la distance de Hamming dans l'espace génotypique. Soient p_1 et p_2 les parents, on cherche un enfant e tel que $d_H(e, p_1) = d_H(p_1, p_2) = d_H(e, p_2)$. Il n'est pas toujours possible d'obtenir des distances exactement égales, dans ce cas on se contente de distances proches (± 1). Les deux tours définis par les parents sont découpés en fragments communs, c'est-à-dire des segments qui contiennent les mêmes villes dans le même ordre. Afin de restituer un tour complet à partir de la liste des fragments, on applique itérativement une stratégie gloutonne sur l'un des fragments. Cette stratégie vise à connecter le bout du fragment à la ville la plus proche (qui représente le bout d'un autre fragment qui n'est pas encore connecté). Au final, tous les fragments sont connectés. Pendant la restitution du tour complet, on n'utilise en aucun cas (si possible) une connexion qui était présente dans un parent mais non dans les deux. Ceci est nécessaire afin que la distance du nouveau tour aux deux parents soit égale à la distance entre eux.

On cherche alors à reconnecter tous les fragments afin de restituer un tour complet. Ceux-ci sont ensuite reconnectés avec un algorithme glouton en évitant des combinaisons déjà trouvées dans un des parents. Plus précisément on parcourt les bouts des fragments et on connecte chaque bout à son voisin libre le plus proche - sans utiliser ses voisins dans les parents, si possible.

Pour transposer cette stratégie pour le problème d'isomorphisme de graphes, il nous faut faire l'abstraction de l'essentiel et adapter, en particulier, la stratégie gloutonne. Premièrement, nous copions les gènes identiques des deux parents (et non les séquences). Deuxièmement, nous appliquons une stratégie gloutonne pour la complétion du chromosome, tout en respectant les deux valeurs *taboues* données par les gènes des parents. Plus précisément, nous parcourons les positions libres dans un ordre aléatoire. Pour chaque position nous choisissons le *meilleur* gène non-tabou, basé sur une heuristique qui estime l'influence du gène donné sur la *fitness* de l'individu. Cette heuristique consiste à chercher la valeur imposant la plus faible augmentation de la distance à l'alignement partiel déjà établi. Nous avons alors besoin d'une estimation de cette augmentation. Il est possible de travailler avec des estimations plus ou moins exactes. En général, si une estimation offre une meilleure précision qu'une autre, alors elle prend en compte plus de variables. Par conséquent, sa complexité en temps de calcul est supérieure. Nous proposons trois estimateurs qui suivent deux stratégies différentes.

La première stratégie, que nous nommons DPX, prend en compte les correspondances (entre sommets) déjà établies dans l'alignement partiel. La distance partielle de départ se base uniquement sur les sommets et arêtes dont les correspondances sont (complètement) définies. Nous considérons une valeur de zéro pour les sommets et arêtes non-alignés. Considérons par exemple deux graphes de 5 sommets chacun et que l'alignement partiel au moment t contient les sommets un à trois de chaque

graphe, et supposons que l'on souhaite ajouter un paire de sommets contenant le sommet quatre du premier graphe. Alors, on pourrait aligner ce sommet aux sommets quatre ou cinq du deuxième graphe. On calcule alors pour chacune des possibilités la somme de toutes les distances des arêtes incidentes du sommet en question et d'un sommet déjà apparié (c'est-à-dire entre un et trois dans notre cas), majorées par la distance entre la paire de sommets à apparier. On choisit finalement le sommet ayant la valeur la plus petite avec cette estimation. Cette stratégie nécessite le calcul des distances atomiques à chaque itération et a donc une complexité assez élevée.

La deuxième stratégie se base uniquement sur des étiquettes des sommets calculés. Nous proposons deux opérateurs qui suivent cette stratégie. Le premier opérateur nommé DPX basé sur les nœuds (NDPX) utilise les étiquettes originales des sommets, tandis que le deuxième se base sur une signature (SDPX) où nous renforçons les étiquettes en utilisant les étiquettes des arêtes incidentes. Plus précisément, nous calculons la somme de l'étiquette du sommet original et des étiquettes des arêtes en amont. L'opérateur NDPX a une complexité très faible et il est applicable aux problèmes d'appariement de sous-graphes avec des étiquettes de type arbitraire. D'une part, le DPX basé sur une signature (SDPX) permet une estimation plus fine avec un temps de calcul supplémentaire négligeable car elle est calculée une seule fois en amont. D'autre part, il nécessite que l'agrégation (la somme) des étiquettes des sommets et des étiquettes des arêtes soit définie. Il ne fonctionne donc pas sans adaptation sur le problème d'appariement de sous-graphes. En effet, on peut renforcer la pertinence de l'étiquetage si l'on prend en compte un rayon plus élevé autour de chaque sommet et on agrège toutes les étiquettes que l'on trouve jusque là.

Les opérateurs d'agrégation réduisent en général la quantité d'information. On obtient par exemple une seule valeur qui agrège trois étiquettes d'origine. La perte d'information et donc de précision de l'heuristique gloutonne est compensée par un gain de vitesse à chaque itération.

Le problème principal qui mène à la perte de précision est que l'agrégation peut fournir la même valeur pour des ensembles de valeurs différents. Par exemple, la somme de $\{1, 2, 9\}$ est égale à la somme de $\{2, 3, 7\}$. Dans le cas exact, on peut remplacer des étiquettes ambiguës par des nouvelles étiquettes uniques. De fait, l'ensemble d'étiquettes possibles est fini et l'égalité est stricte ce qui permet une énumération exacte. Pour notre exemple on définirait (automatiquement au moment de créer les étiquettes agrégés pour les deux graphes) les valeurs d'agrégation 12_A et 12_B qui sont discernables.

Dans un environnement bruité, comme l'est l'alignement inexact, les problèmes deviennent plus complexes. Dans ce cas, la relation d'égalité n'étant pas stricte, deux sommets peuvent être alignés même si leurs attributs ne sont pas exactement égaux. L'origine de cette erreur, que l'on cherche à minimiser, peut être le bruit ou des différences réelles. Il n'est pas possible de déterminer l'origine exactement. Par conséquent les erreurs dues au bruit peuvent s'accumuler. Dans ce cas, en partant des erreurs tolérables au niveau élémentaire (c'est-à-dire pour chaque sommet/arête), il est possible que l'erreur totale des étiquettes après l'agrégation ne le soit plus. On conclurait à une non-

correspondance même si au niveau inférieur une correspondance est possible. Le bruit deviendrait donc un élément discriminant. De même les erreurs non dues au bruit peuvent se compenser jusqu'à ce qu'elles atteignent une valeur que l'on associe à du bruit.

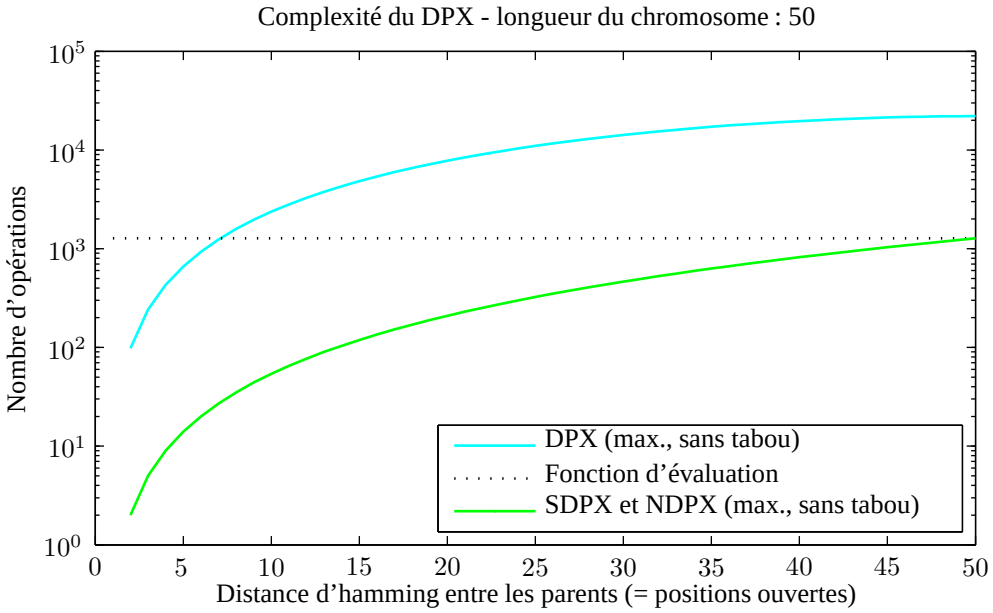

FIGURE 2.7 – Complexité du DPX par rapport à la distance d'Hamming des parents en échelle logarithmique.

On note que l'opérateur DPX utilise une fonction d'évaluation partielle afin d'assurer sa partie gloutonne. Soit l la longueur du chromosome, b le nombre de correspondances entre les parents, et $a = l - b$ le nombre de positions à remplir par l'approche gloutonne, alors l'opérateur requiert un nombre de comparaisons de sommets et d'arêtes d'ordre $\mathcal{O}(\frac{1}{2}na^2)$. La complexité exacte dépend donc, non seulement de la longueur du chromosome, mais aussi de la similarité entre les parents. Plus les parents sont similaires, moins il y a d'opérations à faire. Dans le pire cas, quand les parents diffèrent partout et que tout le chromosome est à remplir, l'opérateur nécessite $\mathcal{O}(\frac{1}{6}n^3 + \frac{1}{2}n^2 - \frac{2}{3}n)$ opérations (voir démonstration ci-dessous). En relation avec la fonction d'évaluation dont la complexité est d'environ $\mathcal{O}(\frac{n^2}{2})$, le DPX est dans le pire cas environ $3n$ fois plus coûteux que la fonction d'évaluation. La figure 2.7 montre le rapport entre le nombre de positions vides et la complexité pour un chromosome de longueur 50. En général, les parents se ressemblent plus vers la fin de l'algorithme qu'au début. Par conséquent, les chromosomes sont moins remplis au début et plus remplis vers la fin.

Les deux opérateurs SDPX et NDPX sont beaucoup plus rapides que DPX (cf. Fig. 2.7). Comparés entre eux, leur complexité est quasiment identique (env. $\mathcal{O}(\frac{1}{2}a^2)$). Il faut ajouter pour SDPX une étape préliminaire de calcul de signatures qui est, dans notre cas, d'une complexité $\mathcal{O}(2n^2)$. Cet étape n'est nécessaire qu'une seule fois au début de l'algorithme et non à chaque génération. Dans

le pire cas, leur complexité est égale à la complexité de la fonction d'évaluation.

Démonstration de la complexité de DPX, SPDX et NDPX. On détermine d'abord le nombre de correspondances b entre les deux chromosomes parents. L'enfant hérite ces b gènes directement. Pour le reste, c'est-à-dire pour $a = l - b$ positions, qui correspond à la distance de Hamming entre les parents $a = d_H(p_1, p_2)$, il faut calculer la distance partielle afin de déterminer le meilleur choix. Pour la première position libre, on calcule $a(1 + n - a)$ distances. De fait, pour chaque gène possible, il faut calculer une distance entre sommets et $n - a$ fois une distance entre arêtes incidentes. Après avoir fixé la première position, il reste un gène de moins mais il y a une distance d'arête de plus à calculer. Le résultat est alors : $(a - 1)(2 + n - a)$. Quand $a = 1$, on n'a pas besoin de calculer la distance, car il ne reste qu'un gène et une position libre. Le nombre de distances total à calculer est alors (sans exclusion des valeurs taboues) :

$$\begin{aligned}
\mathcal{O}(DPX) &= \sum_{i=2}^{a} i(1+n-i) \\
&= (n+1)\sum_{i=2}^{a} i - \sum_{i=2}^{a} i^2 \\
&= (n+1)\left(\frac{a(a+1)}{2} - 1\right) - \left(\frac{a(a+1)(2a+1)}{6} - 1\right) \\
&= \frac{-2a^3 + 3na^2 + (3n+n)a - 6n}{6} \\
&= -\frac{1}{3}a^3 + \frac{1}{2}na^2 + \left(\frac{1}{2}n + \frac{1}{3}\right)a - n \\
&\approx \mathcal{O}\left(\frac{1}{2}na^2\right) \qquad (2.8) \\
& \qquad (2.9)
\end{aligned}$$

Dans le pire cas, quand il n'y a pas de correspondances entre les parents et que leur distance de Hamming est n, on obtient :

$$\mathcal{O}(DPX) = \frac{1}{6}n^3 + \frac{1}{2}n^2 - \frac{2}{3}n \qquad (2.10)$$

Pour des grandes valeurs de n, ce terme est autour de $3n$ fois plus élevé qu'un appel à la fonction d'évaluation.

Pour les opérateurs SDPX et NDPX la complexité est beaucoup plus faible. Elle se détermine par :

$$\mathcal{O}(NDPX) = \mathcal{O}(SDPX) = \frac{a(a+1)}{2} - 1 \qquad (2.11)$$

Dans le pire cas, elle atteint le niveau de la fonction d'évaluation.

L'étape d'extraction des signatures du SDPX a un coût unique dépendant du type de signature utilisé. Si l'on utilise la somme des attributs des arêtes incidentes et de l'attribut du sommet, il

se calcule comme $\mathcal{O}(2n^2)$ (pour un seul graphe $\mathcal{O}(n^2)$). La signature n'est calculée qu'une seule fois au début de l'algorithme génétique, tandis que les opérateurs de croisement sont appliqués à plusieurs de chromosomes pendant toutes les générations. Ce coût initial est donc négligeable.

Il faut noter, que dans tous les cas, il s'agit également d'une légère surestimation. En fait, à chaque étape, il peut avoir entre zéro et deux gènes tabous. Les gènes tabous font partie de l'ensemble de gènes non identiques des parents $\mathcal{M}$ (les valeurs des positions ouvertes). Soit $\mathcal{N}(i) \subseteq \mathcal{M}$ l'ensemble des gènes qui ont déjà été utilisés après i itérations de l'approche gloutonne, alors les gènes possibles à l'étape suivante sont $\mathcal{M}\backslash\mathcal{N}(i)$. On s'intéresse au nombre de gènes tabous à chaque étape. Le nombre de gènes tabous est au maximum 2, un gène de chacun des deux parents. Il peut se réduire à 1 voire 0 si des gènes ont été déjà utilisés, ils ne sont plus disponibles dans la liste et non tabous. Leur nombre est alors déjà inclus. Un gène est tabou s'il se trouve sur un des deux parents à la même position et s'il est dans $\mathcal{M}\backslash\mathcal{N}(i)$.

Afin de calculer une approximation des probabilités, on peut se baser sur le tirage aléatoire de deux valeurs (les deux valeurs taboues qui sont en effet fixées dans les parents) parmi l'ensemble $\mathcal{M}$. Soit i le nombre de positions déjà fixées par l'algorithme glouton. La valeur de i va de 0 à $a-1$. Les probabilités p_0, p_1, et p_2 (zéro gènes tabous, un gène tabou, deux gènes tabous) sont les suivantes :

p_0 La probabilité d'avoir choisi une paire qui se trouve dans $\mathcal{N}(i)$.

p_1 La probabilité d'avoir choisi une valeur dans $\mathcal{N}(i)$ et l'autre dans $\mathcal{M}\backslash\mathcal{N}(i)$.

p_2 La probabilité d'avoir choisi une paire qui se trouve dans $\mathcal{M}\backslash\mathcal{N}(i)$.

Elles ne dépendent pas de la taille du chromosome mais de la distance de Hamming entre les parents ($a = d(p_1, p_2)$) et de la valeur de i. La figure 2.8 montre leurs valeurs en fonction de l'étape i pour un couple de chromosomes ayant une distance de Hamming de 25.

$$p_0(i) = \frac{C_i^2}{C_a^2} = \frac{i(i-1)}{a(a-1)} \tag{2.12}$$

$$p_1(i) = 1 - p_0(i) - p_2(i) \tag{2.13}$$

$$p_2(i) = \frac{C_{a-i}2}{C_a^2} = \frac{(a-i)(a-i-1)}{a(a-1)} \tag{2.14}$$

Cette approximation ne prend pas en compte l'historique des valeurs fixées précédemment. De plus, si l'on applique cette méthode, il faut noter qu'à l'avant-dernière étape, l'effet de deux valeurs taboues est égal à l'effet d'aucune (dans les deux cas on choisit parmi deux valeurs possibles). De toute façon, l'influence de ces probabilités sur la complexité totale n'est pas énorme. Si l'on a deux valeurs taboues, alors on n'a que deux gènes de moins à comparer de l'étape en cours. Les calculs précédents supposent qu'il n'y a pas de valeur taboue dans la liste, et constituent donc une borne supérieure de la complexité réelle. □

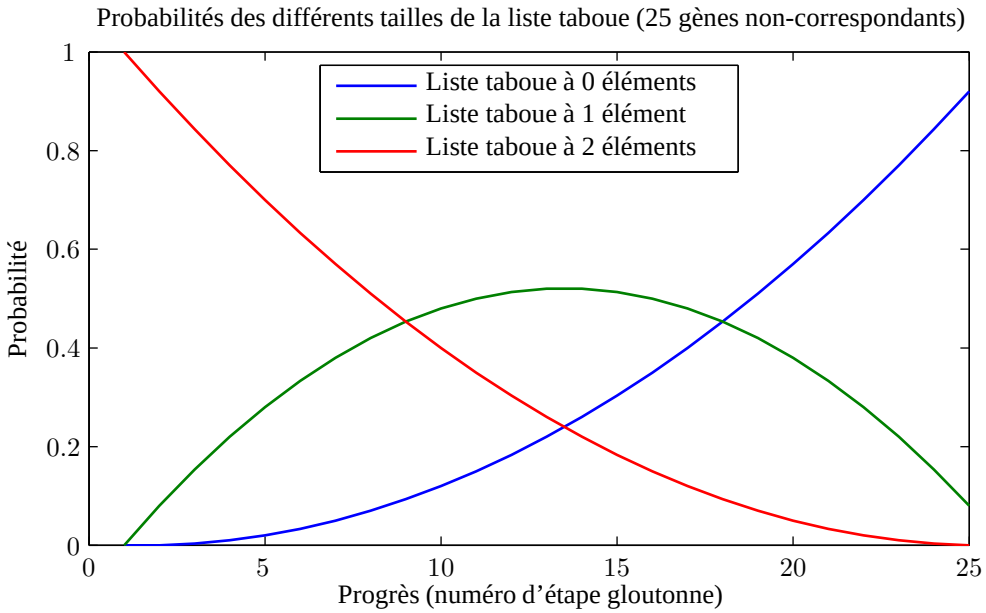

FIGURE 2.8 – Probabilités pour les différents longueurs de la liste taboue.

2.6 Mutation

2.6.1 Mutation d'échange

L'opérateur de mutation classique pour les permutations est un opérateur de transposition (mutation d'échange ou *swap mutation*) qui échange simplement deux gènes. Le but de la mutation est de préserver la diversité dans la population et donc d'éviter la convergence prématurée.

2.6.2 Mutation par brassage

La mutation par brassage (*scramble mutation*) est plus perturbatrice que la mutation d'échange. Au lieu d'échanger seulement deux gènes, elle brasse tous les gènes d'une sous-chaîne du chromosome. La séquence à muter est déterminée comme dans le croisement à deux points, c'est-à-dire que l'on définit deux positions et on applique la mutation à la séquence de gènes situés entre ces deux positions. Dans la mutation *scramble* uniforme les positions sont choisies uniformément parmi tous les gènes du chromosome. Elle nécessite un paramètre afin de contrôler la taille du sous-ensemble. Les changements du chromosome augmentent si l'on choisit une longueur plus élevée.

2.7 Initialisation gloutonne

Les algorithmes génétiques ne permettent pas seulement l'extraction d'une solution candidate à n'importe quel moment mais aussi l'insertion : si l'on dispose d'une bonne solution, il est possible de l'insérer dans la population afin de diriger la recherche aux alentours de cette solution. Pour

l'isomorphisme de graphes, il existe plusieurs méthodes gloutonnes de construction de candidats. On peut ainsi s'inspirer de l'approche de construction itérative de l'algorithme A* (Ullmann, 1976) : à partir d'une solution vide, on ajoute la plus proche paire de sommets jusqu'à ce que la solution couvre tous les sommets du graphe. En ne faisant pas de chaînage arrière, le temps de calcul est négligeable. La solution ainsi construite n'est pas nécessairement optimale mais normalement assez bonne. Une autre stratégie consiste à restreindre le nombre de variables en prenant en compte uniquement les attributs des sommets. Pour raffiner cette méthode, l'utilisation d'une valeur agrégée des attributs des sommets et des arêtes et éventuellement des sommets voisins est possible. L'agrégation peut se faire par exemple avec l'attribut du sommet et son degré.

Quelle que soit la méthode gloutonne choisie, l'individu que l'on introduit risque de prendre le pas sur le reste de la population. Comme, dans la plupart des cas, sa qualité est très bonne par rapport de celle des individus présents, il a plus de chances de se reproduire. Notamment, pendant les premières générations, les individus ne sont pas encore très spécialisés et c'est dans ces circonstances que l'on souhaite normalement intégrer l'individu. Or, il faut maintenir la diversité afin de pouvoir trouver l'optimum global si ce dernier se trouve ailleurs. Sinon l'algorithme va converger sur un optimum local autour de l'individu et l'utilisation de l'approche génétique n'aurait à la limite pas de justification.

Comment peut-on alors intégrer des stratégies gloutonnes afin qu'elles puissent indiquer de bonnes solutions locales sans nuire à la diversité et ainsi à la recherche globale ? La réduction de la pression sélective est une piste. L'inconvénient est que la convergence est alors ralentie. On peut aussi utiliser une population qui n'admet pas de copies mais elle risque alors d'être dominée par des individus proches du meilleur, par exemple issus d'une seule mutation, tandis que des solutions intermédiaires menant à un autre optimum ne seront pas admises.

Une autre option consiste à faire de l'introduction des individus gloutons un processus répétitif. Il faut, bien évidemment, s'assurer que les individus introduits à un instant donné ne soient pas (ou avec une faible probabilité) égaux à des individus introduits avant. Les opérateurs *DPX présentés dans la section 2.5.5 introduisent des individus gloutons à chaque génération. Ils assurent la diversité car, d'une part, l'ordre dans lequel les sommets sont parcourus est aléatoire et d'autre part une liste de valeurs taboues est utilisée. De plus l'amplitude des changements (le nombre de gènes altérés par rapport aux parents) est proportionnelle à la similarité des parents. Au début de l'algorithme, alors que les parents sont en général très différents, presque tous les chromosomes sont nouveaux. On pourrait supposer que ceci a des effets négatifs sur la diversité puisqu'une telle approche gloutonne construit toujours le même chromosome à partir d'un même point de départ (par exemple le chromosome vide si les parents n'ont rien en commun). Toutefois, en introduisant des contraintes, on peut augmenter la probabilité d'obtenir des chromosomes différents, par exemple en interdisant les gènes des parents. On peut donc maintenir la diversité.

En fait, la liste de gènes tabous que l'on obtient des parents est très différente pour chaque paire

de parents. Dans la deuxième génération, la population contient un certain nombre de solutions gloutonnes dont les contraintes se limitent aux listes taboues. Dans les générations suivantes, l'amplitude des changements baisse, ce qui aide à exploiter plus efficacement les bonnes solutions déjà trouvées (grâce à l'approche préservant la distance). On peut noter aussi que, si le nombre de solutions gloutonnes introduites dans la population devient trop grand, alors l'algorithme ne converge plus. Cet effet semble dépendant de la précision de la méthode gloutonne utilisée. Par exemple, l'opérateur DPX est plus précis que l'opérateur SDPX car il prend en compte plus d'informations. De même, SDPX est plus précis que NDPX. Nous avons évalué expérimentalement la probabilité de croisement optimale, qui signifie la fraction de solutions gloutonnes par générations dans ce cas, à 0,25 pour le DPX contre 0,35 pour le SDPX et 0,4 pour le NDPX (cf. Sec. 3.2.2, p. 62). Si l'on utilise une méthode plus stricte, alors il faut l'appliquer sur moins d'individus.

2.8 Opérateurs par couleur

L'idée générale des opérateurs par couleur (Singh et al., 1997) est de réduire l'espace de recherche en ne permettant que des appariements des sommets de la même classe. En fait, si l'on suppose deux graphes de 20 sommets dont la moitié appartient à une classe et l'autre moitié à une deuxième, alors l'espace de recherche se réduit d'environ $2,4 \times 10^{18}$ à $7,2 \times 10^{6}$.

Pour y parvenir, on traite les parties du chromosome correspondant aux différentes classes (couleurs) séparément. Il est évident que tous les opérateurs génétiques doivent respecter les classes par la suite, c'est-à-dire notamment la procédure d'initialisation, le croisement, la mutation et la recherche locale.

Quand on souhaite utiliser une stratégie par couleur, une classification de sommets adaptée au domaine d'application est indispensable. En particulier, cette classification doit assurer qu'il n'est pas possible d'obtenir un optimum en appariant deux sommets de classes différentes. Un tel optimum ne peut jamais être trouvé.

Nous étudions le croisement par couleur sur une base de molécules. Les résultats sont donnés dans le chapitre 7, plus précisément dans la section 7.2.

2.9 Réglage des paramètres

L'inconvénient majeur des EA est le grand nombre de paramètres. La recherche des paramètres optimaux est, en elle-même, un problème d'optimisation difficile (Eiben et al., 2007). En particulier, il n'existe aucune règle générale fournissant de bons paramètres pour tous les problèmes, même si l'effort a été considérable (voir par exemple Grefenstette, 1986; Goldberg, 1991b; Harik et al., 1999; Cicirello & Smith, 2000; Cervantes & Stephens, 2006; Cantú-Paz, 2007). Quant à la théorie des EA, elle ne s'applique pas aux algorithmes complexes. En 1999, Eiben et al. affirment qu'elle n'est pas

considérée comme base utile pour la pratique.

De Jong (1975) a effectué une recherche expérimentale des bons paramètres d'un GA classique pour différents problèmes numériques. En utilisant un codage binaire avec le croisement à un point et la mutation binaire (*bit mutation*) comme opérateurs, il a trouvé notamment qu'une taille de population de 50 avec une probabilité de croisement de 0,6 et une probabilité de mutation de 0,001 fournissent des résultats assez bons sur les fonctions testées. Par contre, on ne peut pas généraliser ces paramètres à d'autres problèmes, ou si l'on utilise un autre codage, des opérateurs différents, etc.

Grefenstette (1986) exploite les notions de la performance en ligne (*online*) et hors-ligne (*off-line*). La performance en ligne est une mesure tenant compte de la qualité de l'ensemble de la population tandis que la performance hors-ligne s'intéresse uniquement au meilleur individu de chaque génération. Il utilise un méta-GA pour optimiser les paramètres du GA considéré. Selon le type de performance qu'il cherche à optimiser, les paramètres optimaux trouvés diffèrent beaucoup. Par exemple la taille de la population est de 30 (en ligne) contre 80 (hors-ligne) et la probabilité de croisement est de 0,95 contre 0,45.

Il existe aussi des approches qui visent à construire un GA sans paramètres. Harik & Lobo (1999) argumentent que l'utilisateur ne s'intéresse pas aux paramètres, il cherche uniquement une méthode pour résoudre son problème. Dans leur tentative de supprimer les paramètres, ils fixent le taux de croisement à 0,5 et le taux de sélection qui contrôle la pression sélective à 4. De fait, l'algorithme n'est pas vraiment sans paramètres mais plutôt avec des paramètres fixés à l'avance.

Bien que les travaux sur les paramètres optimaux suggèrent des valeurs très différentes, on peut néanmoins tirer des conclusions sur leurs ordres de grandeur. La probabilité de mutation est généralement très faible, autour d'un pour cent. Une heuristique dépendante de la longueur l du chromosome (sous codage binaire) a été trouvée par Mühlenbein (1992) : un bon taux de mutation est alors $1/l$. La probabilité de croisement varie à partir de 0,6 jusqu'à légèrement en-dessous de 1. En général, elle ne devrait pas être trop petite. Les tailles de population documentées dans la littérature s'étendent principalement sur l'intervalle de 20 à 100 individus.

Parmi les premiers travaux sur des paramétrages adaptatifs, la *1/5 success rule* définie par Rechenberg (1973) pour les ES est très importante. D'abord, il introduit une notion de succès d'une mutation. De fait, on parle du succès si le descendant créé est meilleur que son parent. Dans ce cas la mutation a améliorée l'individu. La proportion des mutations avec succès parmi toutes les mutations devrait être $1/5$. L'adaptation est faite à l'aide de l'écart-type de la variable aléatoire utilisée pour la mutation d'un gène. Si le quota est trop faible, alors on diminue la dispersion, ainsi la mutation devient plus faible. Si le quota est trop fort, alors on utilise une mutation plus forte.

2.9.1 *Racing*

Une méthode classique de déterminer les paramètres d'un algorithme génétique consiste à évaluer plusieurs jeux de paramètres ou configurations afin de choisir le meilleur parmi eux. Dans ce but, l'algorithme est lancé plusieurs fois avec chaque jeu de paramètres afin de permettre une analyse statistique des résultats obtenu qui sert comme critère de sélection. En général, il est nécessaire d'effectuer un grand nombre d'itérations afin d'obtenir une estimation de la performance assez robuste ce qui représente un coût de calcul important. La précision de l'estimation de la performance dépend du nombre d'itérations.

L'idée du *racing* est de limiter le nombre d'itérations pour des jeux de paramètres peu prometteurs et de permettre ainsi de dédier plus de temps de calcul aux jeux de paramètres prometteurs. En particulier, les jeux de paramètres seront exclus dès que l'on est certain que leur performance réelle ne peut pas être supérieure à celle des autres. La certitude est exprimée par des intervalles de confiances. S'il existe deux jeux de paramètres dont les intervalles de confiance de leurs performances ne se recouvrent pas, alors le jeu de paramètres inférieur peut être exclu. Maron & Moore (1994) proposent l'utilisation de l'inégalité d'Hoeffding pour calculer la probabilité que la moyenne estimée d'un échantillon de n données ne diffère pas plus qu'un certain ϵ de la moyenne réelle. Soit E_{true} la moyenne réelle, E_{est} la moyenne estimée et B une borne de la variance des données, cette probabilité est donné par

$$Pr\left(|E_{true} - E_{est}| > \epsilon\right) < 2e^{\frac{-2n\epsilon^2}{B^2}} \tag{2.15}$$

La valeur d'ϵ qui correspond à un niveau de confiance $1 - \delta$ est donnée par (Maron & Moore, 1994)

$$\epsilon = \sqrt{\frac{B^2 \log(2/\delta)}{2n}} \tag{2.16}$$

L'inégalité de Bernstein permet de définir des bornes plus stricts, mais nécessitent l'écart-type de la distribution qui est en général inconnu (Heidrich-Meisner & Igel, 2009). En utilisant l'écart-type empirique $\widehat{\sigma}$ et pour le même niveau de confiance $1 - \delta$, ϵ est borné par (Heidrich-Meisner & Igel, 2009)

$$\epsilon \leq \widehat{\sigma}\sqrt{\frac{2\log\frac{3}{\delta}}{n} + \frac{3B\log\frac{3}{\delta}}{n}} \tag{2.17}$$

Birattari et al. (2002) proposent une méthode similaire, dénommé *F-Race*, basée sur le test de Friedman. Il s'agit d'un test non-paramétré basé sur l'ordonnancement des différents jeux de paramètres pour chaque instance du problème testé. L'hypothèse zéro est dans ce cas que tous les ordonnancements sont équiprobable. Si cette hypothèse est rejetée, alors les configurations sont comparés

par paires entre le meilleur est tous les autres. Les configurations qui montrent une différence signifiante au meilleur jeu de paramètres sont exclues. Si les données suivent approximativement une distribution gaussienne, l'analyse de variance (ANOVA) peut également être utilisée. Yuan & Gallagher (2004) montrent cependant que *F-Races* sont plus efficaces que les *A-Races* basés sur l'ANOVA.

Bien que les méthodes de *racing* permettent de réduire le temps de calcul nécessaire pour évaluer les configurations et de se focaliser d'avantage sur les jeux de paramètres prometteurs, ils ne permettent pas de surmonter le problème du nombre infini des configurations possibles du aux paramètres continues, par exemple la probabilité de mutation. Pour obtenir un nombre limité de configurations, on discrétise généralement l'espace des paramètres. Nannen & Eiben (2007) proposent REVAC. Il s'agit d'un algorithme d'estimation des distributions qui cherche des valeurs prometteuses pour chaque paramètre. Des configurations candidates sont ensuite tirées de ces distributions et évaluées.

Une idée complémentaire au *racing* est le *sharpening* (Smit & Eiben, 2009). Les méthodes de *racing* définissent un maximum du nombre d'itérations effectuées par configuration et réduisent ce nombre pour les configurations éliminées. *Sharpening* augmente ce nombre maximal pendant la recherche cherchant à augmenter la précision des estimations pour des bonnes configurations. La combinaison de *racing* et *sharpening* permet de concentrer le temps de calcul disponible sur des configurations prometteuses tout en limitant le temps dépensé pour l'évaluation des configurations peu prometteuses.

2.9.2 *No free lunch*

Le théorème *no free lunch* (NFL, Wolpert & Macready, 1996, 1997; Ho & Pepyne, 2002) montre du point de vue théorique que des paramètres optimaux généraux n'existent pas. En fait, Wolpert & Macready argumentent que toutes les méthodes d'optimisation du type boîte noire ont une performance identique moyennée sur l'ensemble des problèmes possibles. Ce résultat est indépendant de la mesure de performance utilisée. Ils se limitent aux algorithmes du type boîte noire généralistes (sans restrictions du domaine d'application), efficaces, c'est-à-dire qu'une solution déjà évaluée n'est jamais réévaluée, et aux domaines finis, ce qui est évidemment le cas pour les ordinateurs. La moyenne des performances est calculée sous une distribution uniforme sur l'ensemble de fonctions (d'évaluation), mais une généralisation sur des distributions arbitraires existe (Igel & Toussaint, 2004).

En bref, il n'existe pas de méthode d'optimisation qui fonctionne mieux sur tous les problèmes qu'une autre. Quand on obtient une meilleure performance sur une instance précise, on a forcément des résultats inférieurs sur une autre. Comme deux EA avec des paramètres différents sont deux méthodes différentes, le NFL implique qu'aucun jeu de paramètres n'est toujours meilleur qu'un autre.

Cependant, plusieurs travaux montrent que le théorème ne s'applique pas toujours, par exemple pour certaines classes de fonctions (Streeter, 2003), pour des algorithmes co-évolutionnaires (Wolpert & Macready, 2005), ou encore dans des domaines continus (Auger & Teytaud, 2008). Ces derniers n'ont pas été visés par le NFL classique. Rowe et al. (2008) donnent une réinterprétation récente du théorème basé sur la théorie des ensembles et non probabiliste comme le NFL classique. Au lieu d'essayer de quantifier les performances, ils s'intéressent aux symétries fondamentales et constatent qu'elles sont présentes. Ce n'est donc pas le NFL qui est la cause des échecs constatés mais plutôt son cadre probabiliste.

Chapitre 3

Choix des opérateurs et leurs paramètres

Dans ce chapitre, nous étudions le paramétrage de l'algorithme génétique en examinant successivement les différentes composantes nécessaires : nous comparons les différents opérateurs de croisement, ceux qui existent déjà (CX, PMX, UPMX et UOX) et ceux que nous proposons (PBX, UPBX, DPX, SDPX et NDPX). Pour cela il est d'abord nécessaire d'estimer les paramètres optimaux pour chacun d'entre eux car les performances d'un opérateur dépendent fortement des paramètres utilisés. Ces derniers varient beaucoup en fonction de l'opérateur, c'est-à-dire que le jeu de paramètres optimal en combinaison avec un opérateur spécifique n'est *a priori* pas optimal quand on utilise un autre opérateur. Les paramètres étudiés sont concrètement la taille de la population, la probabilité de croisement et la probabilité de mutation. Nous examinons ensuite le choix des stratégies de sélection (cf. Sec. 3.4, p. 101) ainsi que les stratégies de mutation (cf. Sec. 3.5, p. 102).

3.1 Constitution des données de test

Nous décrivons dans cette section les données utilisées pour réaliser les expériences : nous utilisons une base de graphes construits aléatoirement. La taille des graphes varie entre 5 et 100 par pas de 5. Les graphes sont complets, c'est-à-dire complètement connectés. Les attributs des sommets ainsi que ceux des arêtes sont des nombres aléatoires de l'intervalle $[0, 1]$, calculés avec l'algorithme *Mersenne Twister* de Matsumoto & Nishimura (1998). Toutes les valeurs de l'intervalle sont donc équiprobables.

Comme nous abordons le problème de l'isomorphisme inexact de graphes, nous étudions l'effet du bruit sur les résultats des différents algorithmes. Dans ce but, pour chaque graphe, on construit un deuxième graphe en ajoutant du bruit sur les attributs des sommets et des arêtes en considérant des distributions gaussiennes d'écart-type σ et de distributions uniformes sur l'intervalle $[-\sqrt{3}\sigma, +\sqrt{3}\sigma]$ (et par application d'une permutation aléatoire sur les sommets afin d'éviter des solutions triviales). Ces valeurs mènent à des espérances de distance égales entre chaque graphe et sa copie perturbée. Nous considérons, pour l'écart type σ, 6 valeurs de 0 à 0,1 ce qui correspond à 0

à 10% de la taille de l'intervalle des attributs générés précédemment. On peut donc aussi parler d'un niveau de bruit de 0, 2, 4, 6, 8, et 10%.

Pour les expériences d'optimisation de paramètres, nous utilisons surtout des graphes de taille 40 avec un niveau de bruit intermédiaire (de 6%). Des graphes plus grands et des niveaux de bruit plus élevés mènent à des temps de calcul plus longs. Comme les expérimentations supposent la prise en compte de nombreuses combinaisons de mauvais paramètres, nous estimons que 40 est un bon compromis entre temps de calcul et robustesse des résultats.

Les algorithmes sont exécutés 30 fois sur chaque paire de graphes de la base de graphes synthétiques. Nous considérons 50 paires de graphes pour chaque taille et niveau de bruit. Afin d'estimer la généralité de nos résultats, nous effectuons également certains tests en variant la taille de graphes ou le niveau de bruit.

3.2 Détermination des paramètres optimaux pour les opérateurs de croisement

Afin de permettre une comparaison juste entre les différents opérateurs génétiques, il ne suffit pas de les appliquer en utilisant les mêmes paramètres pour chacun. Bien que l'on puisse définir quelques règles généralement applicables pour les choisir, chaque opérateur a ses particularités. On estime souvent, pour le moteur générationnel, que le taux de croisement doit être assez élevé alors que la probabilité de mutation doit rester plutôt faible. Une importance particulière est souvent accordée à la taille de la population qui a effectivement montré dans nos expérimentations qu'elle joue un rôle important.

Une petite probabilité de croisement amène à un algorithme dont seule une partie des individus est soumise aux variations génétiques, ce qui a une certaine relation avec le concept du fossé générationnel (cf. Sec. 2.4, p. 37). En ce qui concerne la mutation, le raisonnement est inverse. Des probabilités de mutation élevées correspondent à une recherche de plus en plus aléatoire et peuvent détruire des solutions candidates déjà trouvées, ce qui conduit à des temps d'exécution plus grands.

3.2.1 Protocole expérimental

Comment peut-on trouver de bons paramètres ? L'optimisation combinatoire exacte dans un espace à trois dimensions continues n'est pas faisable, d'autant plus que nous utilisons un total de neuf opérateurs à évaluer. Il faut donc raisonnablement limiter la recherche du jeu de paramètres optimal. De plus, comme les algorithmes génétiques font partie des méthodes non déterministes, il faut effectuer un nombre représentatif de répétitions afin de s'assurer que les résultats obtenus sont statistiquement significatifs.

Nous étudions d'abord indépendamment les paramètres pour chaque opérateur de croisement,

puis nous comparons les opérateurs entre eux dans la section 3.3. Nos expérimentations dans cette section comportent deux étapes distinctes. Dans un premier temps, nous optimisons individuellement les paramètres des algorithmes génétiques pour chaque opérateur de croisement (séparément pour le moteur générationnel et *steady-state*). Pour cela nous appliquons une méthode itérative qui se termine quand elle trouve un optimum local stable. Dans un second temps, nous étudions l'influence du bruit sur les paramètres optimaux dans le voisinage des optima respectifs des opérateurs.

Méthode Le réglage de paramètres définit un problème d'optimisation non séparable. Une recherche exhaustive de toutes les combinaisons n'étant pas faisable, nous visons de trouver, au moins, un optimum local. Dans ce but, nous appliquons manuellement une stratégie correspondant à la descente de gradient : en commençant avec un jeu de paramètres fixe, nous varions les trois paramètres indépendamment. Nous déterminons ensuite le paramètre qui améliore le plus les résultats. Nous adaptons alors pour ce paramètre la meilleure valeur constatée pendant que les autres paramètres restent inchangés. Nous réitérons les tests pour les paramètres qui ne viennent pas d'être changés, à partir de ce nouveau jeu de paramètres modifié. Nous répétons cette procédure jusqu'à ce qu'aucun changement individuel d'un seul paramètre ne donne de meilleures performances. Dans les sections suivantes, nous présentons les variances des paramètres autour de l'optimum trouvé.

Nous utilisons initialement une population de 100 individus, une probabilité de croisement de 0,99 et une probabilité de mutation de 0,1. Nous définissons des intervalles non uniformes qui sont plus petits dans le domaine où nous espérons trouver l'optimum. Par exemple, pour la probabilité de croisement, on utilise généralement des probabilités assez élevées. On choisit donc des intervalles larges pour de petites probabilités (d'une largeur de 0,1 entre 0,1 et 0,8), et réduit successivement la taille de ces intervalles pour des probabilités plus élevées : nous utilisons des intervalles d'une largeur de 0,05 pour des probabilités entre 0,8 et 0,95, puis d'une largeur de 0,01 jusqu'à 0,99. Quant à la taille de la population, nous n'avons pas une idée très précise *a priori* de la taille optimale que l'on peut espérer. Nous couvrons donc un domaine très large entre 10 et 1000 individus.

Nous évaluons les différents choix des paramètres en regardant le taux de succès (SR) et la distance moyenne de la solution de référence ainsi que le temps de calcul également en termes de nombre d'appels à la fonction d'évaluation. Nous estimons l'importance relative de chaque paramètre en utilisant la variance de la série des SR obtenus. Bien que ces valeurs ne soient pas comparables directement pour les différents paramètres (du fait du choix « défini » des intervalles ainsi que de la nature différente des paramètres), on peut en tirer des conclusions du type : « un certain paramètre est plus important pour un opérateur que pour un autre. »

Présentation des résultats Les diagrammes (cf. par exemple Fig. 3.1, p. 63) représentent en même temps le taux de réussite et le nombre d'appels à la fonction d'évaluation (ou le temps de calcul pour les croisements intelligents). Les deux courbes sont tracées sur leur échelle respective. Les axes des

ordonnées correspondants sont indiqués à gauche, en rose pour le taux de succès, et à droite pour le nombre d'appels à la fonction d'évaluation ou le temps de calcul, en bleu. Afin de faciliter la comparaison entre les différents opérateurs, la plupart des figures utilise les mêmes échelles. Dans le cas contraire, nous changeons la couleur de l'échelle divergente en rouge. Cependant, nous utilisons toujours les valeurs exactes pour la détermination des paramètres, il se peut donc que certains effets décrits ne soient pas visibles sur les figures.

3.2.2 Moteur générationnel

Dans cette sous-section, nous examinons la taille de la population et les probabilités de croisement et mutation dans l'environnement d'un moteur générationnel (décrit dans le Ch. 2, Sec. 2.4, p. 37). Ensuite, nous présentons les résultats obtenus pour les différents opérateurs de croisement décrits dans la section 2.5 :

CX

CX (cf. Sec. 2.5.4, p. 42) est un opérateur qui ne perturbe que très peu les schémas. En particulier, il garantit que chaque gène du descendant vient de l'un de ses parents. Les trois diagrammes de la figure 3.1 montrent sa sensibilité aux trois paramètres, respectivement la taille de la population, et les probabilités de croisement et de mutation.

Nous nous intéressons d'abord à l'influence de la taille de la population. Nous constatons de bons taux de succès quand la taille de la population franchit une certaine borne inférieure qui dépend de la taille des graphes considérés : des populations très petites conduisent à des taux de succès nuls. A partir d'une certaine valeur, le taux de succès s'accroît brusquement jusqu'à 1 et reste à ce niveau. Le nombre d'appels à la fonction d'évaluation atteint sa valeur minimale pour des tailles de population supérieures au seuil observé pour le taux de succès. Les deux seuils sont croissants en fonction de la taille des graphes. Il n'est donc pas possible de définir une taille de population fixe menant à de bons résultats pour toute taille de graphe. Nous constatons que la taille de population minimale nécessaire pour obtenir un bon taux de succès correspond environ à deux fois la taille du graphe (par exemple 40 individus pour un graphe de taille 20). Afin d'optimiser le temps de calcul, une augmentation additionnelle est nécessaire. Cette augmentation n'a pas d'effet négatif sur le taux de succès. De manière générale, une taille assez élevée (à partir de 400) donne des performances acceptables, même si le temps de calcul peut être légèrement réduit en utilisant des valeurs plus petites pour des graphes moins grands. Par la suite nous utilisons une taille égale à 400.

En ce qui concerne la probabilité de croisement (cf. Fig. 3.1, p. 63, diagramme du milieu), on observe que le taux de réussite augmente en corrélation directe avec la probabilité de croisement. L'optimum se trouve alors au maximum, c'est-à-dire à 0,99. On peut toutefois noter qu'à partir d'environ 0,8 les améliorations sont assez faibles. Nous constatons également une corrélation in-

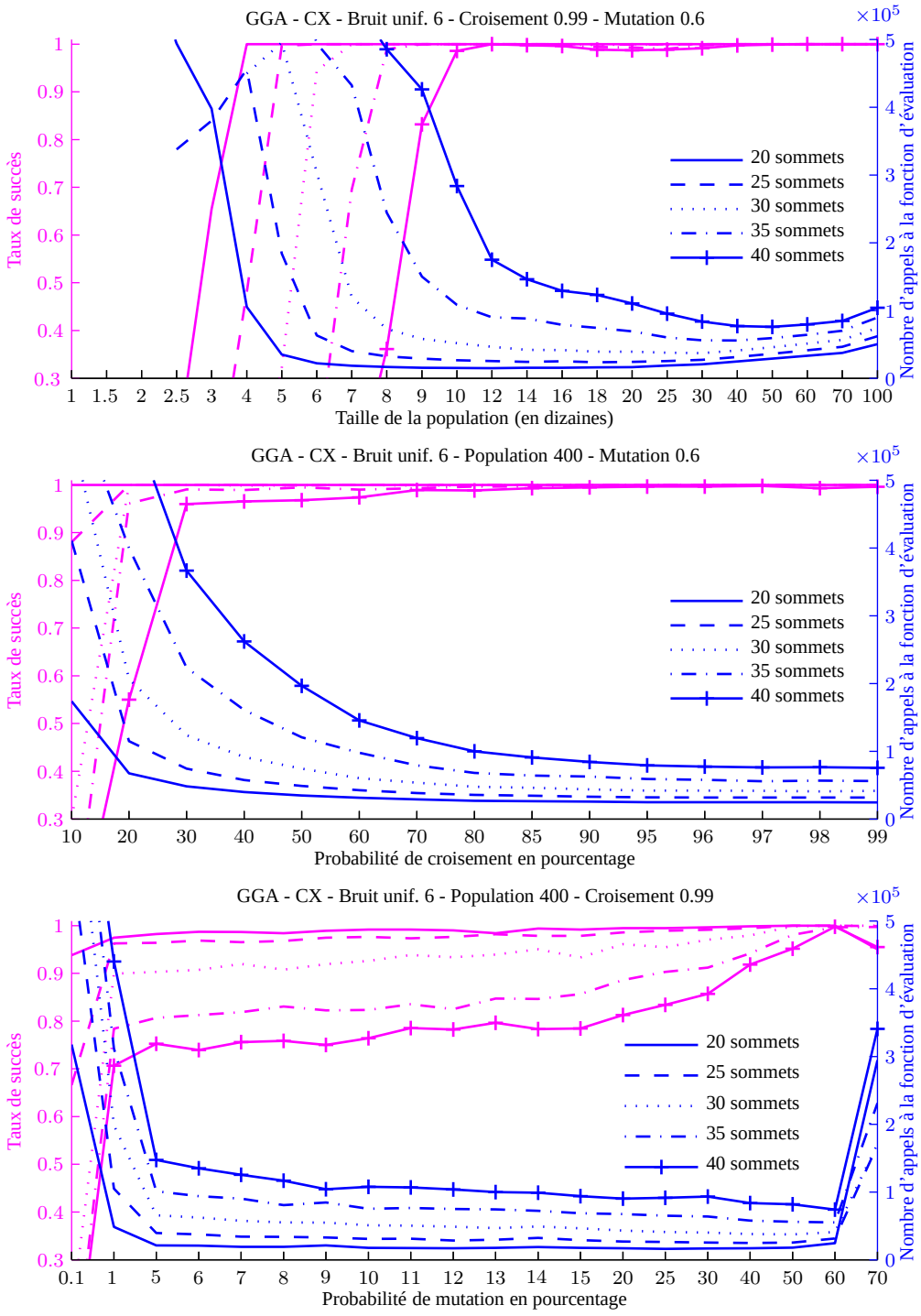

FIGURE 3.1 – Performance du croisement CX en fonction des trois paramètres.

verse faible entre le nombre d'appels à la fonction d'évaluation et la probabilité de croisement. Une probabilité de 0,99 est donc aussi préférable du point de vue du temps de calcul. On constate également que des probabilités de croisement inférieures à 0,3 mènent à des taux de succès insatisfaisants. Par la suite, nous utilisons donc une probabilité de croisement égale à 0,99.

En termes de probabilité de mutation (cf. Fig. 3.1, p. 63, diagramme du bas), nous observons pour toutes les tailles de graphes un taux de succès de près de 1, uniquement avec des probabilités de mutation élevées, autour de 0,6. Contrairement à la règle générale qui recommande d'utiliser des probabilités de mutation très faibles, celles-ci donnent des résultats médiocres, tandis que des probabilités moyennes conduisent à des valeurs de taux de succès autour de 0,75 pour la taille 40. Concernant le nombre d'appels à la fonction d'évaluation, nous observons un minimum à 0,6 qui coïncide avec le meilleur taux de succès. Les différences sont assez faibles sur une région étendue (de 0,1 à 0,6), seules des probabilités encore plus élevées que 0,6, notamment 0,7, ou inférieures à 0,05 subissent une forte croissance du temps de calcul.

PMX

Les résultats de l'opérateur PMX (cf. Sec. 2.5.4, p. 42), illustrés par la figure 3.2, ne montrent pas de relation monotone avec la taille de la population. Nous constatons deux optima locaux du taux de succès : celui-ci est maximal lorsque la population est assez petite (autour d'une vingtaine d'individus) mais aussi entre environ 120 et 200 individus (cf. diagramme du haut). Le nombre d'appels à la fonction d'évaluation est beaucoup plus important dans le premier cas. Toutefois, le minimum du nombre d'appels à la fonction d'évaluation ne coïncide pas non plus avec le deuxième maximum du taux de succès mais se trouve autour de 100, indépendamment de la taille de graphes. Pour des populations plus grandes, le nombre d'appels à la fonction d'évaluation augmente constamment. Nous constatons ainsi un conflit entre les deux buts de la recherche des paramètres optimaux, un taux de succès maximal et un temps de calcul minimal (ou du moins faible). Il faut donc définir un compromis entre ces deux objectifs. En général, nous cherchons un taux de succès élevé (de 0,99) et celui-ci n'est possible qu'à partir de 120 individus, valeur que nous considérons dans la suite. Bien que les valeurs du taux de succès et du nombre d'appels à la fonction d'évaluation varient selon la taille de graphes, la position des optima reste inchangée.

En regardant le rapport entre la probabilité de croisement et le taux de succès (cf. Fig. 3.2, p. 65, diagramme du milieu), nous constatons une corrélation directe : plus la probabilité est élevée, plus le taux de succès est élevé. En ce qui concerne le nombre d'appels à la fonction d'évaluation, il atteint son minimum pour des probabilités élevées ; pour des probabilités de croisement faibles, il est beaucoup plus important. A partir de 0,85, il est légèrement croissant. De manière générale, une probabilité très élevée est préférable bien que l'on n'observe plus de grandes différences à partir de 0,95. Nous utilisons 0,99 par la suite.

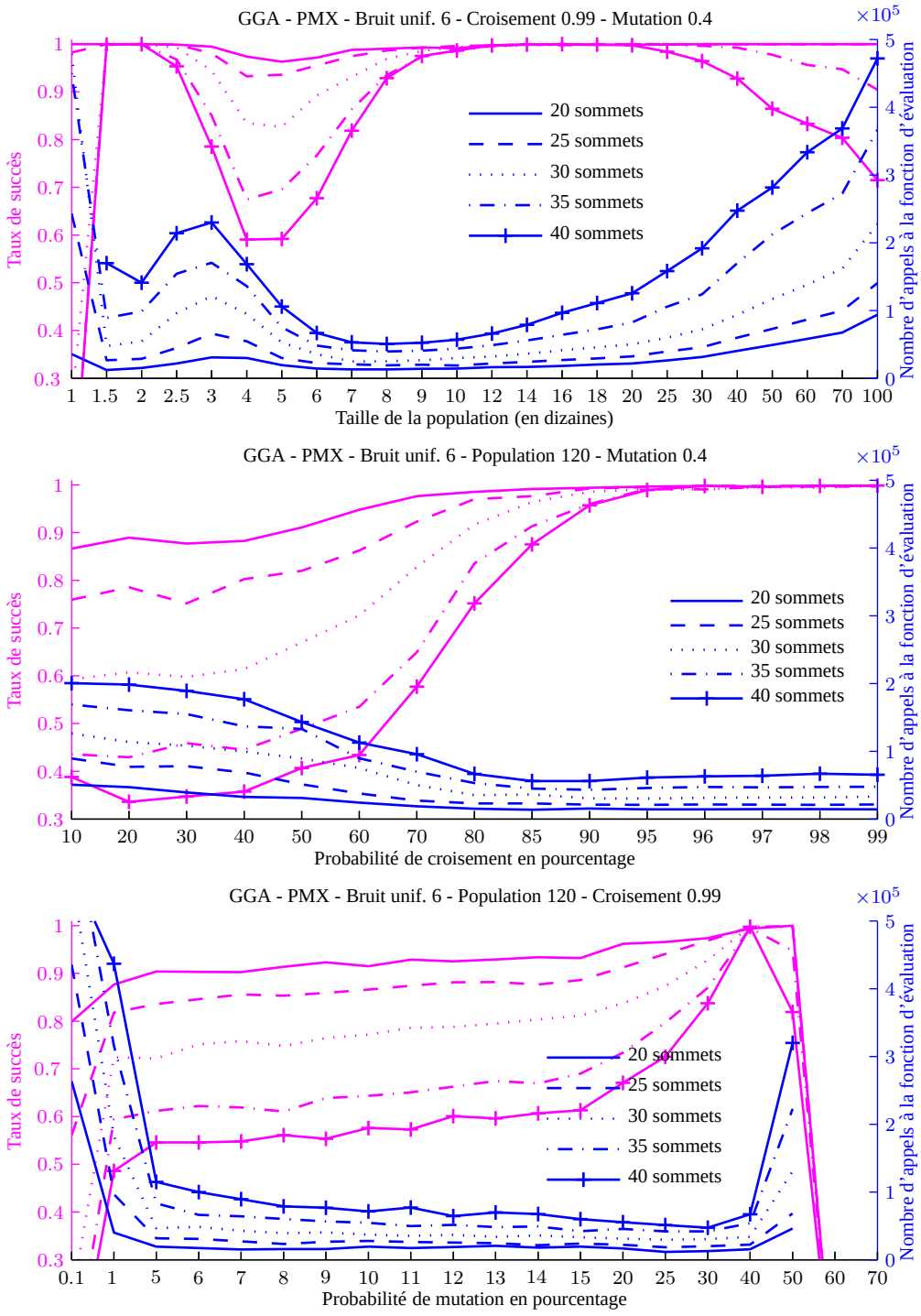

FIGURE 3.2 – Performance du croisement PMX en fonction des trois paramètres.

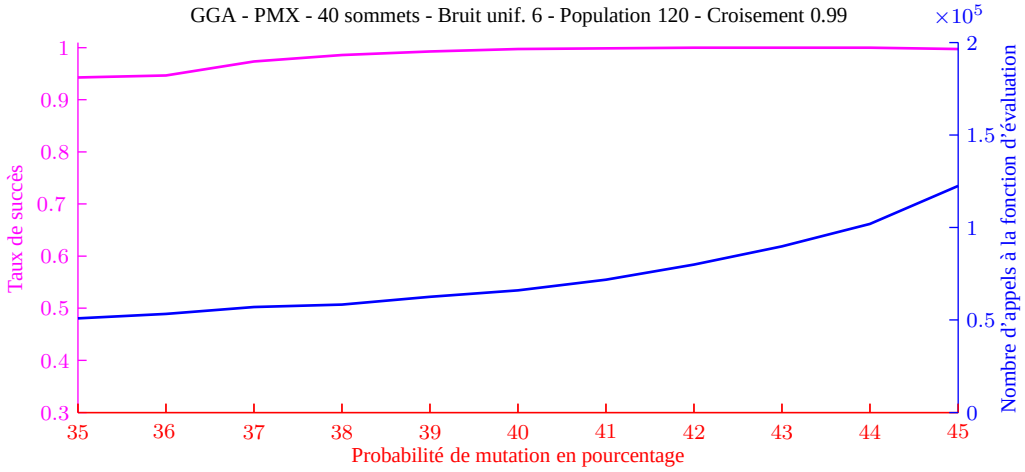

FIGURE 3.3 – Performance du croisement PMX en fonction de la probabilité de mutation.

Quand on s'intéresse à l'influence de la probabilité de mutation (cf. Fig. 3.2, p. 65, diagramme du bas), on constate qu'il existe une valeur optimale, proche de 0,4 pour le meilleur taux de succès, et légèrement inférieure pour le minimum du nombre d'appels à la fonction d'évaluation. De fait, des probabilités très élevées ou très faibles mènent à des résultats insatisfaisants. Le pic du meilleur taux de succès à 0,4 devient plus significatif avec des graphes plus grands, mais garde sa position. Ceci favorise le choix de cette valeur, même si le temps de calcul est plus élevé que pour des probabilités légèrement inférieures. La figure 3.3 montre le comportement autour du pic à une échelle beaucoup plus détaillée, pour des graphes de 40 sommets. Le nombre d'appels à la fonction d'évaluation montre une croissance importante avec les probabilités entre 0,35 et 0,45, passant de $5,1 \times 10^4$ à $12,2 \times 10^4$, tandis que le taux de succès monte d'environ 0,95 à plus de 0,99 entre 0,36 et 0,39 avant d'approcher 1. Les différences entre 0,4 et 0,42 de probabilité de mutation sont négligeables en termes de taux de succès mais importantes en termes de temps de calcul. C'est pourquoi nous favorisons la valeur de 0,4.

UPMX

Quand on utilise la version uniforme de l'opérateur PMX (UPMX, cf. Sec. 2.5.4, p. 42) dont les résultats sont représentés par la figure 3.4, on constate une forte sensibilité à la taille de la population et à la probabilité de croisement : en ce qui concerne la taille de la population, illustrée sur le diagramme du haut, nous constatons que l'optimum en termes de taux de succès se trouve autour de 90, ou, pour des graphes plus petits, sur un intervalle dont la borne inférieure est d'environ 90. Pour les grands graphes, il se forme un pic qui devient considérablement plus marquant avec la taille des graphes.

Contrairement au taux de succès, l'optimum en termes de temps de calcul se situe à des tailles plus faibles, autour de 70 pour des graphes de taille 40 et légèrement supérieures pour des graphes

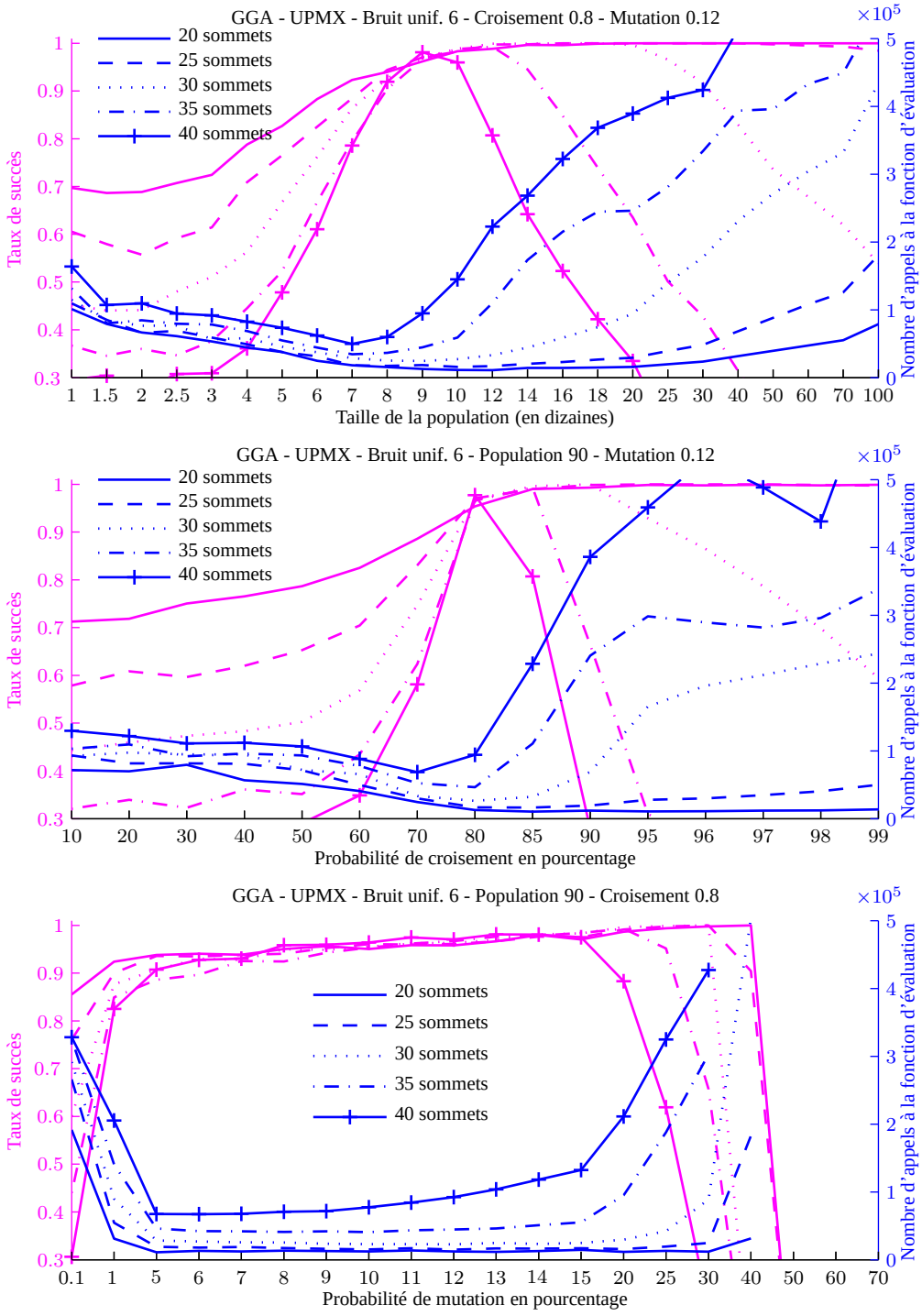

FIGURE 3.4 – Performance du croisement UPMX en fonction des trois paramètres.

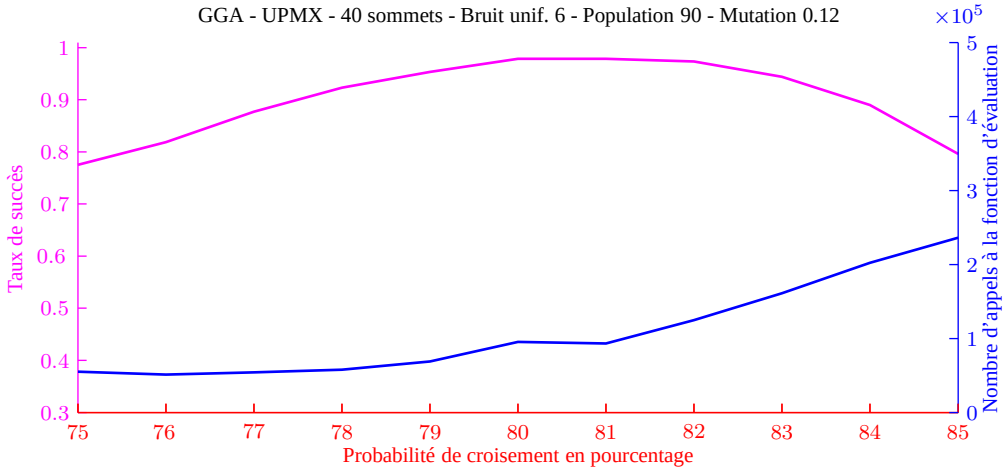

FIGURE 3.5 – Performance du croisement UPMX en fonction de la probabilité de croisement sur une échelle restreinte.

plus petits. La croissance du nombre d'appels à la fonction d'évaluation est très importante pour des probabilités plus élevées. Toutefois, le taux de succès correspondant au minimum du temps de calcul est seulement autour de 0,8 pour des graphes de 40 sommets, ce qui nous paraît insatisfaisant. Nous choisissons de définir un compromis qui privilégie le taux de succès plutôt que le nombre d'appels à la fonction d'évaluation et nous utilisons donc une population de 90 individus.

Quant à l'influence de la probabilité de croisement (cf. Fig. 3.4, p. 67, diagramme du milieu), on observe également un pic marquant sur la courbe du taux de succès, qui se situe autour de 0,8. Ce pic devient plus marquant quand la taille des graphes augmente. Les résultats autour de cette valeur pour des graphes de taille 40, en utilisant une échelle plus fine (restreinte sur l'intervalle $[0,75; 0,85]$), sont montrés sur la figure 3.5. Nous constatons que le taux de succès maximal est obtenu avec une probabilité de croisement de 0,81. Ce taux de succès n'est que marginalement supérieur à celui obtenu avec une probabilité de 0,8 mais le nombre d'appels à la fonction d'évaluation augmente considérablement. Il semble donc plus approprié d'utiliser la probabilité de 0,8. Il faut néanmoins noter que l'UPMX est extrêmement sensible à la probabilité de croisement.

La mutation joue un rôle moins important que les autres deux paramètres (cf. Fig. 3.4, p. 67, diagramme du bas). Pour des valeurs moyennes (entre 0,1 et 0,2), le taux de succès est autour de 0,9 et le nombre d'appels à la fonction d'évaluation reste assez faible. Pour des probabilités plus petites ou plus grandes, le taux de succès baisse et le temps de calcul augmente. Dans cette plage, les différences sont assez faibles, mais nous constatons que le taux de succès est croissant. Le minimum du temps de calcul est obtenu à 0,1. La probabilité *optimale* doit donc être choisie entre 0,1 et 0,2. Quand on augmente la probabilité à l'intérieur de cette plage, le taux de succès augmente légèrement ainsi que le nombre d'appels à la fonction d'évaluation. Nous choisissons une valeur de 0,12.

PBX

La figure 3.6 illustre la relation entre les valeurs des trois paramètres et les performances de l'algorithme pour le croisement PBX (cf. Sec. 2.5.5, p. 44). Le taux de succès montre deux optima locaux, le premier pour une population de taille 10 et le deuxième sur l'intervalle entre 120 et 250 individus. Entre les deux, nous constatons un minimum très marquant autour d'une trentaine d'individus. Les positions des trois extrema sont indépendantes de la taille de graphes. La perte de précision dans le minimum augmente cependant avec la taille de graphes. Le nombre d'appels à la fonction d'évaluation montre deux minima, le premier autour d'une dizaine d'individus et le deuxième sur l'intervalle entre environ 60 et 150 individus. Parmi ces deux minima, le meilleur est le deuxième dont l'intervalle recouvre en plus l'intervalle du second maximum pour le taux de succès. En étudiant le chevauchement de plus près, notamment sur l'intervalle entre 80 et 200 individus, on peut constater que le nombre d'appels à la fonction d'évaluation ainsi que le taux de succès sont croissants avec la taille de la population. Nous cherchons cependant à minimiser l'un et maximiser l'autre, ce qui mène à un conflit d'objectifs. Nous choisissons alors une population de 120 individus, ce qui donne des taux de succès de près de 1 avec un nombre d'appels à la fonction d'évaluation limité.

La probabilité de croisement doit être assez forte (cf. Fig. 3.6, p. 70, diagramme du milieu) : le taux de succès ainsi que le nombre d'appels à la fonction d'évaluation profitent en général de valeurs élevées. Cependant pour ce dernier, pour des probabilités supérieures à 0,9, le temps de calcul augmente légèrement. Nous avons choisi une valeur de 0,96 qui paraît être la meilleure, même si les différences entre 0,95 et 0,99 sont marginales.

En ce qui concerne la probabilité de mutation, nous observons un maximum de taux de succès autour de 0,3 à 0,4 indépendamment de la taille des graphes. Aussi le nombre d'appels à la fonction d'évaluation reste minimal sur une large plage de valeurs, entre 0,05 et 0,3. Même si les valeurs sont comparables sur ce plateau, on peut constater une légère tendance décroissante. En dehors de ces limitations, le temps de calcul devient beaucoup plus important.

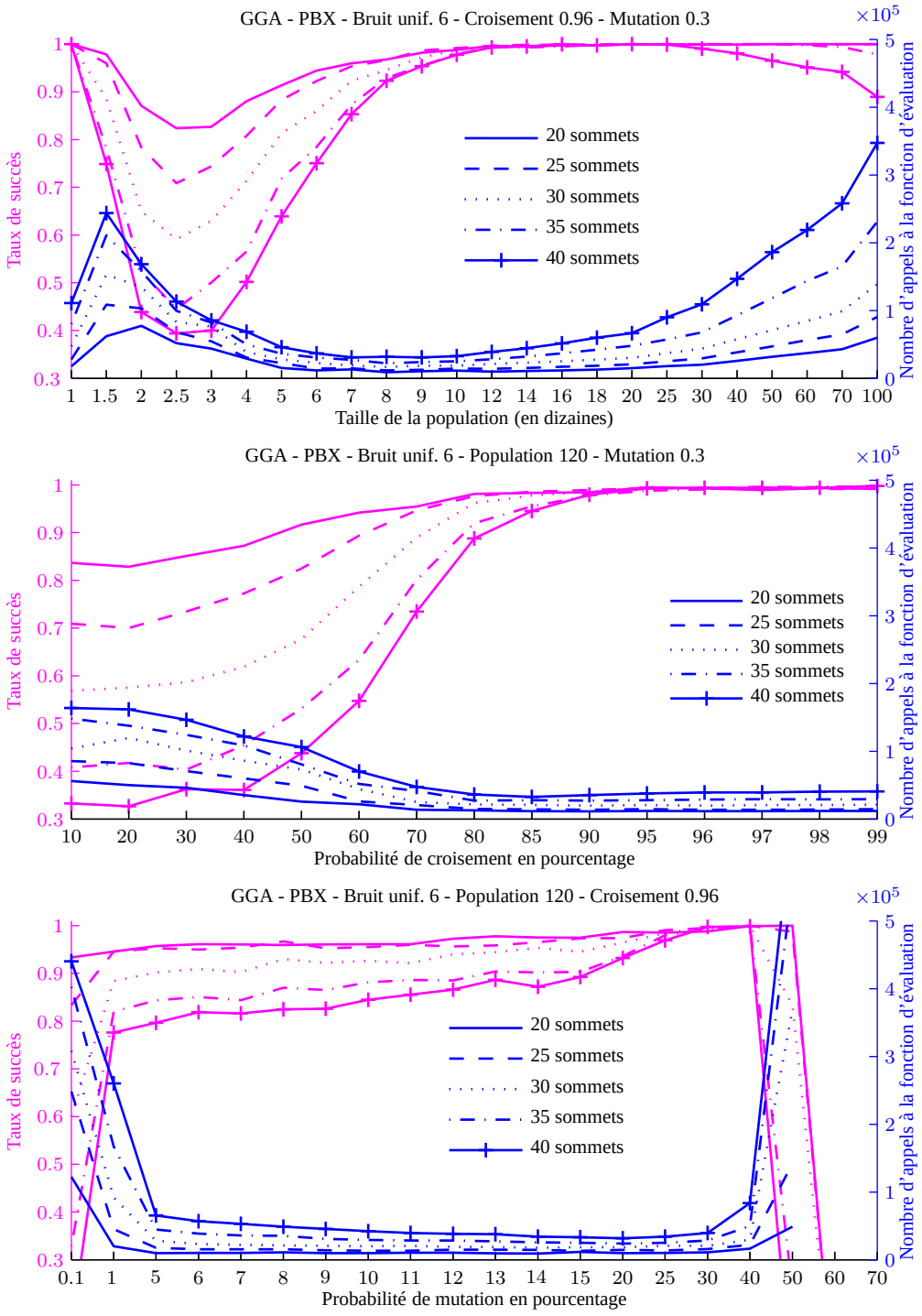

FIGURE 3.6 – Performance du croisement PBX en fonction des trois paramètres.

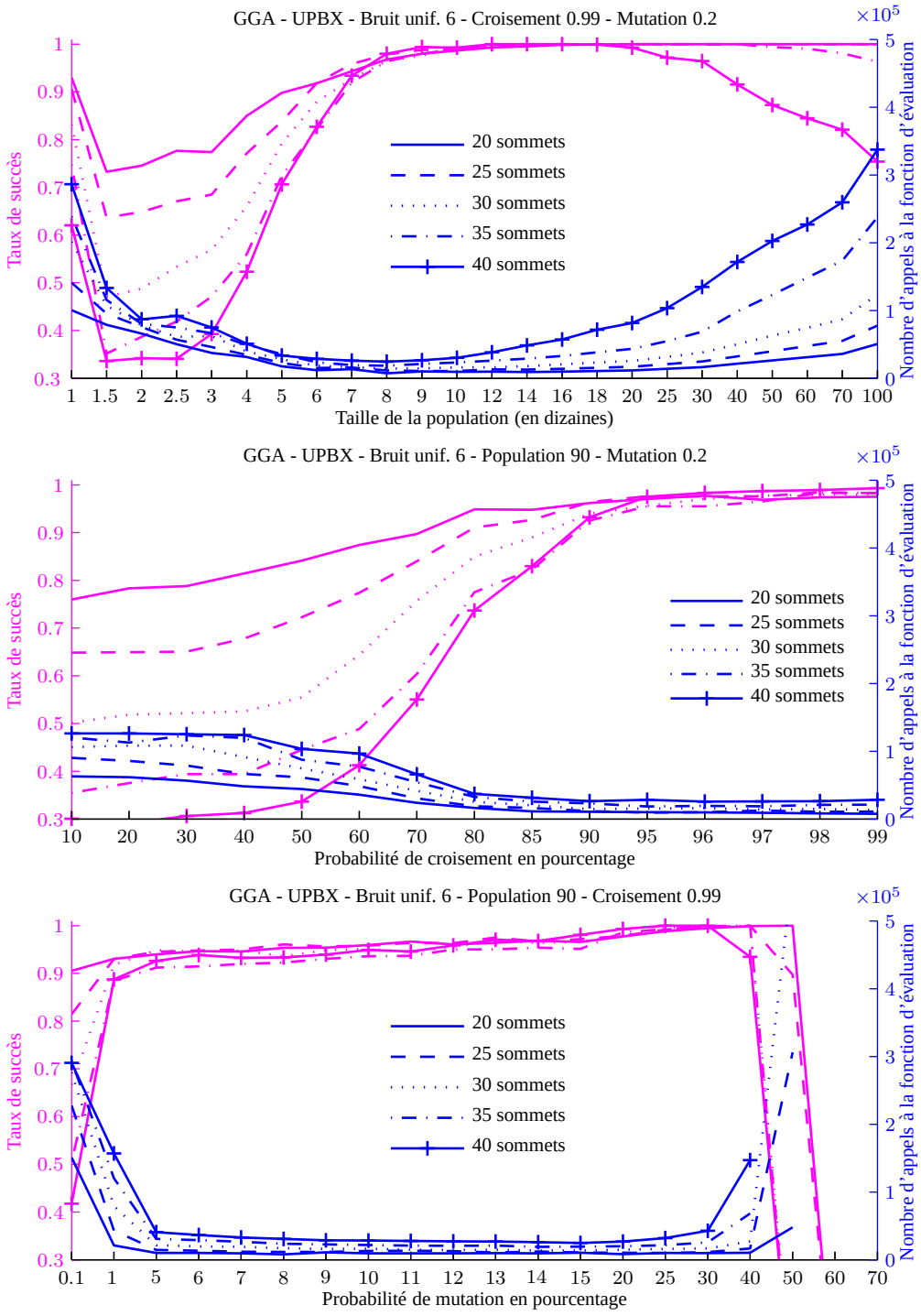

FIGURE 3.7 – Performance du croisement UPBX en fonction des trois paramètres.

UPBX

La figure 3.7 montre l'influence des trois paramètres sur l'opérateur UPBX (cf. Sec. 2.5.5, p. 44). Les effets de la taille de la population sur des graphes de différentes tailles sont illustrés sur le diagramme du haut. Les meilleurs taux de succès sont obtenus avec des tailles entre 80 et 160 individus indépendamment de la taille des graphes. Le minimum du nombre d'évaluations de la fonction d'évaluation se situe autour de 90. Le diagramme montre en particulier que, comme pour PMX, UPMX et PBX, la taille de graphes n'a pas d'influence sur la taille de population *optimale*.

De plus, UPBX profite d'une probabilité de croisement assez forte (cf. Fig. 3.7, p. 71, diagramme du milieu). Le taux de succès augmente et le nombre d'appels à la fonction d'évaluation diminue quand la probabilité augmente.

Une probabilité de mutation comprise entre 0,01 et 0,3 mène à des taux de succès de plus de 0,9 indépendamment de la taille de graphes. Pour des probabilités beaucoup plus élevées, le taux de succès baisse. La probabilité à partir de laquelle on constate cette baisse dépend de la taille de graphes : pour des graphes plus petits, elle est plus élevée. L'optimum du taux de succès se situe entre 0,2 et 0,3. Quant au nombre d'appels à la fonction d'évaluation, il atteint son minimum pour une probabilité de 0,2. Toutefois, nous constatons des valeurs similaires sur l'intervalle entre 0,05 et 0,3. Nous utilisons par la suite la probabilité de 0,2.

UOX

Le croisement basé sur l'ordre uniforme (UOX) (cf. Sec. 2.5.4, p. 42) est *a priori* peu adapté au problème car il est basé sur l'ordre des gènes uniquement et ne prend pas en compte leurs positions absolues. Comme illustré par la figure 3.8, il montre une forte sensibilité à tous les paramètres. Par conséquent, l'analyse globale avec les mêmes intervalles que pour les autres opérateurs donne uniquement une idée générale et une granularité plus fine s'avère nécessaire.

Nous constatons dans le diagramme du haut (cf. Fig. 3.8, p. 73) que la taille optimale de la population par rapport au taux de succès comme au nombre d'appels à la fonction d'évaluation est de 9 ou 10. On observe qu'il y a des chutes importantes du taux de succès entre 10 et 11 et entre 12 et 13. Le passage de 9 à 10 est également accompagné d'une forte augmentation du nombre d'appels à la fonction d'évaluation. Les différences observées semblent plus importantes qu'entre 11 et 12. Ceci s'explique par l'utilisation d'une probabilité de croisement fixée à 0,6 en combinaison avec une taille de la population très faible, le nombre d'individus créés avec le croisement étant un nombre entier. On obtient 6 individus pour une population de taille 10 et 11, et 7 individus quand la population est de 11 et 12. Il y a donc un individu de plus qui est transmis à la prochaine génération sans croisement, ce qui n'est pas dû à la taille de population directement mais à des effets d'arrondi. Nous estimons que ceci constitue une incohérence que l'on peut régler en fixant le nombre d'individus non soumis au croisement.

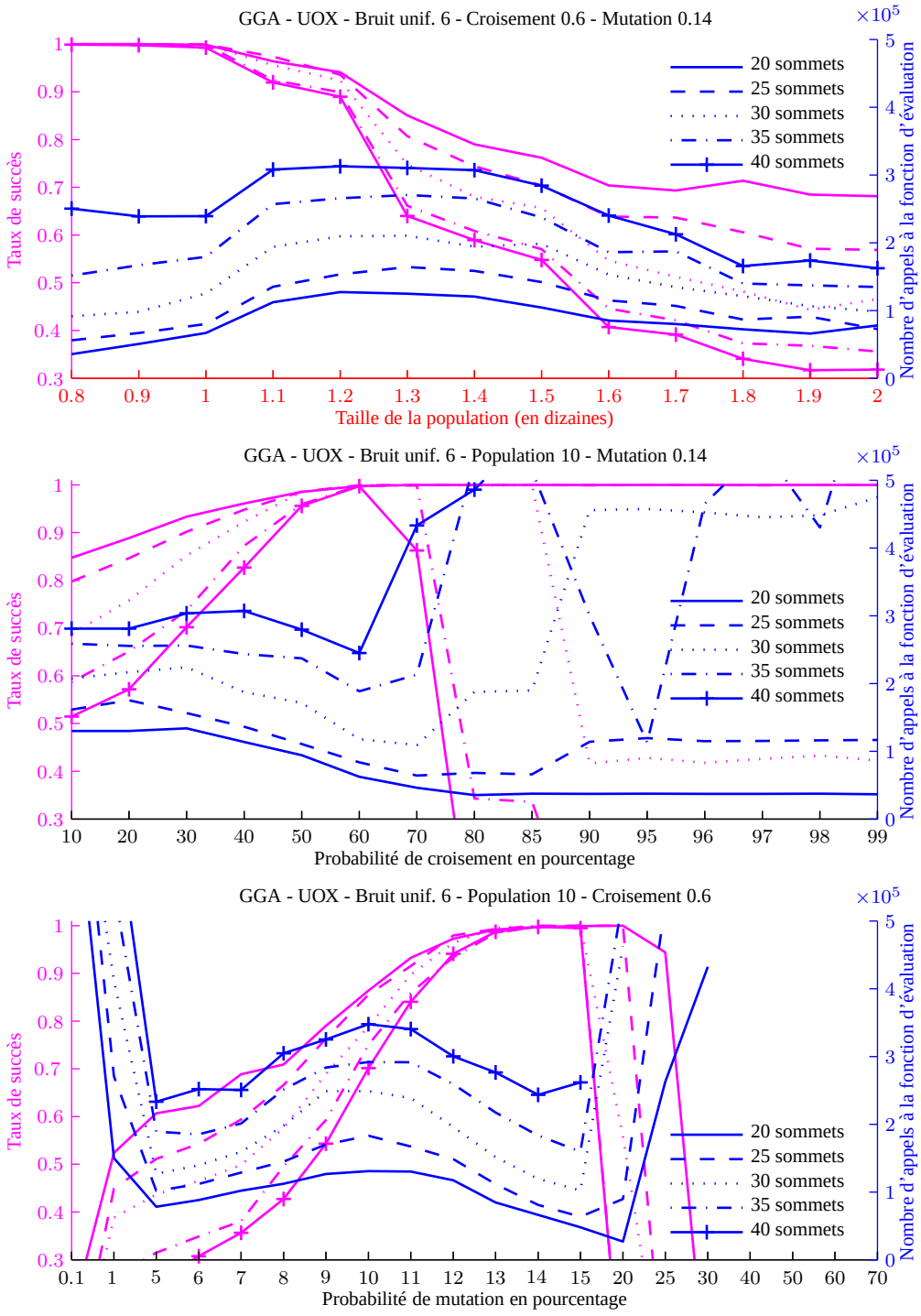

FIGURE 3.8 – Performance du croisement UOX en fonction des trois paramètres.

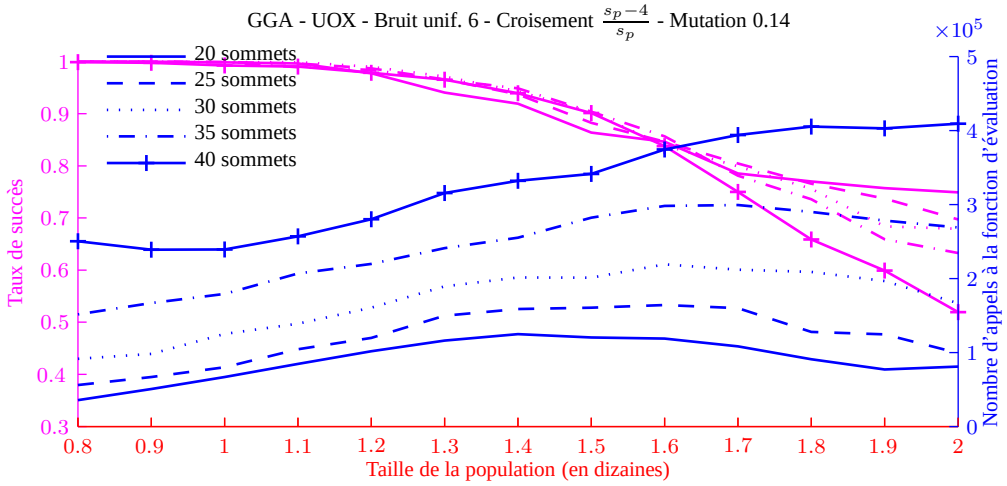

FIGURE 3.9 – Performance de l'opérateur UOX en fonction de la taille de la population s_p, pour des probabilités de croisement définies par $\frac{s_p-4}{s_p}$ telles que le nombre de chromosomes non soumis au croisement soit égal à 4.

Afin d'éliminer cet effet, nous utilisons une probabilité de croisement dépendante de la taille de population s_p et non constante : nous la définissons comme $p_c = \frac{s_p-4}{s_p}$ afin de garantir un nombre d'individus non soumis au croisement constant et égal à 4. La figure 3.9 montre les résultats. L'incohérence a été effectivement éliminée et nous constatons un léger changement de la taille de population optimale qui reste entre 9 et 10 individus. Dans les deux cas, en respectant une probabilité de croisement fixe ou un nombre d'individus non soumis au croisement fixe, la différence entre la taille 9 et la taille 10 n'est statistiquement pas significative. On obtient des p-valeurs autour de 0,7 (cf. Ann. D.12, p. 253).

En variant la probabilité de croisement, nous observons un seul pic du taux de succès qui atteint 1 pour une probabilité de 0,6 (cf. Fig. 3.8, p. 73, diagramme du milieu). Pour les probabilités plus faibles, le taux de succès diminue rapidement. Quant aux probabilités de croisement supérieures, le taux de succès atteint 0 à partir de 0,85.

Les résultats sont aussi très dépendants de la probabilité de mutation choisie. Nous observons sur le diagramme du bas que la valeur optimale est de 0,14. Les résultats avec 0,14 et 0,15 sont statistiquement indépendants ($p = 0,0001$), tandis qu'entre 0,13 et 0,14 une ambiguïté ne peut pas être rejetée ($p = 0,243$).

DPX

L'analyse des opérateurs intelligents (cf. Sec. 2.5.5, p. 44), DPX, SDPX et NDPX, nécessite le changement de la mesure du temps d'exécution. Pour les opérateurs précédents, nous avons utilisé le nombre d'appels à la fonction d'évaluation, car le coût total ne dépend quasiment que de celui-ci. Les opérateurs intelligents incluent le calcul d'une *fitness* partielle basée sur la fonction d'évaluation.

Le coût en termes de calcul varie beaucoup selon l'opérateur et selon la distance de Hamming entre les parents, comme décrit dans la section 2.5.5 (cf. Fig. 2.7, p. 49) et il peut atteindre des niveaux supérieurs à un appel à la fonction d'évaluation. Aussi, il est nécessaire de mesurer directement le temps. L'inconvénient de cette méthode est le manque de généralité. Il n'est en particulier plus possible de comparer les résultats obtenus avec ceux obtenus avec d'autres implémentations ou sur d'autres machines.

Le diagramme du haut de la figure 3.10 montre les résultats de DPX en fonction de la taille de la population. Nous constatons que le taux de succès est presque toujours d'1 indépendamment de la taille de la population et de la taille des graphes. La seule exception se situe autour d'une quinzaine d'individus où le taux baisse légèrement. La perte est croissante avec la taille des graphes. En ce qui concerne le temps de calcul, nous observons un plateau minimal entre environ 20 et 200 individus dont le minimum se trouve à 50.

La probabilité de croisement a une forte influence sur les résultats. Nous constatons que la probabilité optimale est beaucoup plus faible que pour les autres opérateurs ; avec une valeur de 0,25 en particulier, elle est même beaucoup plus faible que les probabilités normalement constatées dans la littérature (cf. Sec. 2.9, p. 54). En outre, si la probabilité dépasse un seuil de 0,5, alors le taux de succès tombe à 0, le temps de calcul dépasse une seconde et devient beaucoup trop important pour que la prise en compte de ces valeurs ait un intérêt. En ce qui concerne les probabilités faibles, nous observons une forte corrélation avec la taille de la population. Le diagramme du haut de la figure 3.11 illustre cette relation. En effet, pour chaque probabilité de croisement entre 0,15 et 0,45, il existe au moins un intervalle de la taille de populations qui mène à des taux de succès de 1 (comme l'intervalle $[50; 1000]$ sur le diagramme du haut de la figure 3.10). Parmi les valeurs de cet intervalle, nous choisissons celle qui minimise le temps de calcul. Cette taille de la population optimale est décroissante avec la probabilité de croisement (cf. la courbe en rouge sur la Fig. 3.11, p. 77). Le temps de calcul associé à la taille de la population optimale est montré en bleu ; dans l'intervalle illustré dans la figure, il reste quasiment inchangé, en dehors de cet intervalle (des deux côtés) il augmente énormément. Nous observons un optimum global pour le temps à 0,25 avec une taille associée de 50 individus. Par conséquent, la taille de la population et la probabilité de croisement ne peuvent pas être considérées comme indépendantes.

La probabilité de mutation, illustrée sur le diagramme du bas (cf. Fig. 3.10, p. 76), peut se situer dans un intervalle assez large entre 0,01 et 0,4 sans changer le taux de succès de 1. En dehors de ces bornes, le taux de succès chute fortement, ce qui est dû, pour les probabilités plus faibles, au manque de possibilité de quitter les optima locaux et donc à une convergence prématurée. Pour les probabilités très élevées, la recherche devient aléatoire et ressemble plus à une approche par force brute. Quand on étudie le temps de calcul, on observe un minimum autour de 0,25, ce qui est la valeur que nous avons choisie.

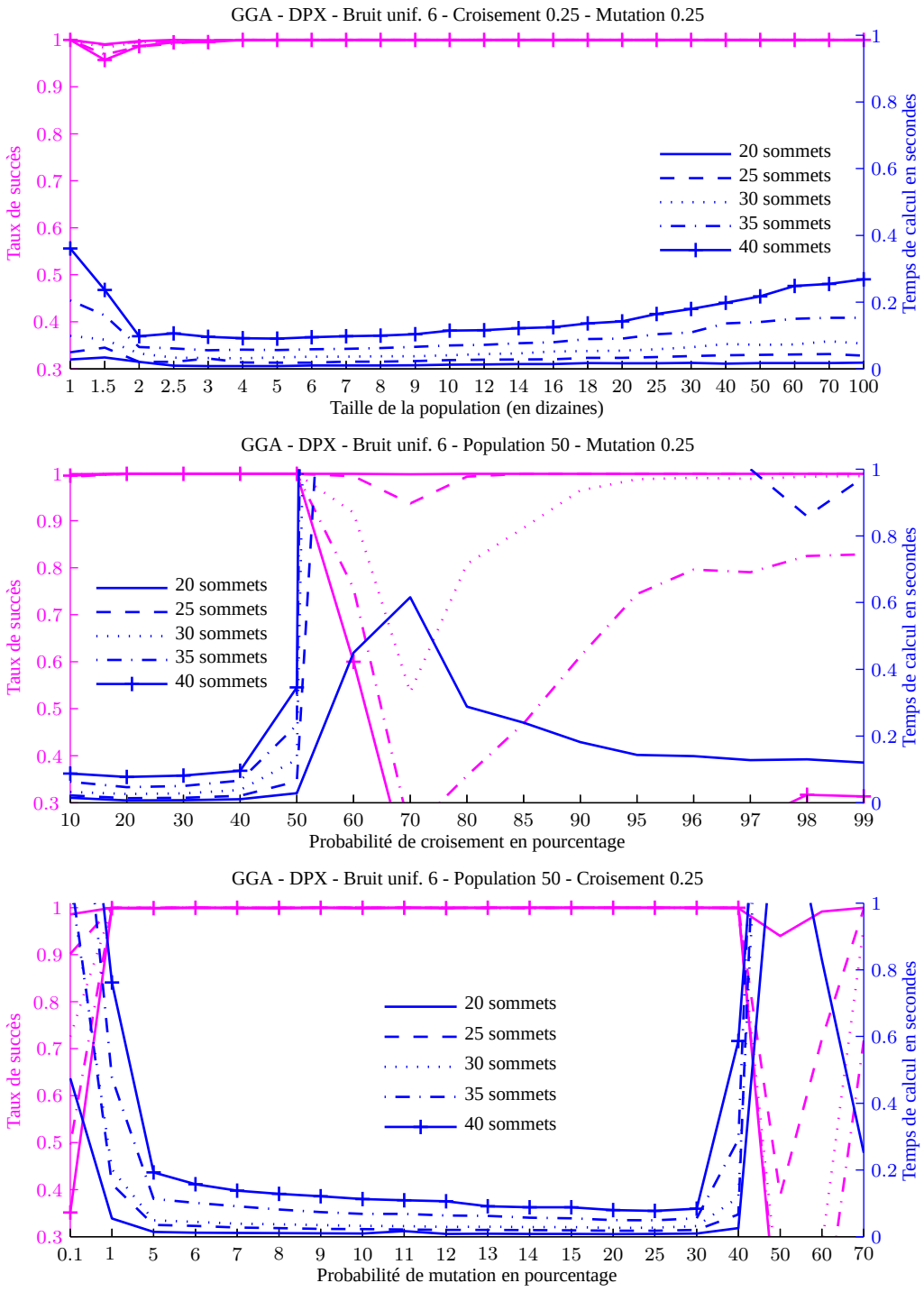

FIGURE 3.10 – Performance du croisement DPX en fonction des trois paramètres.

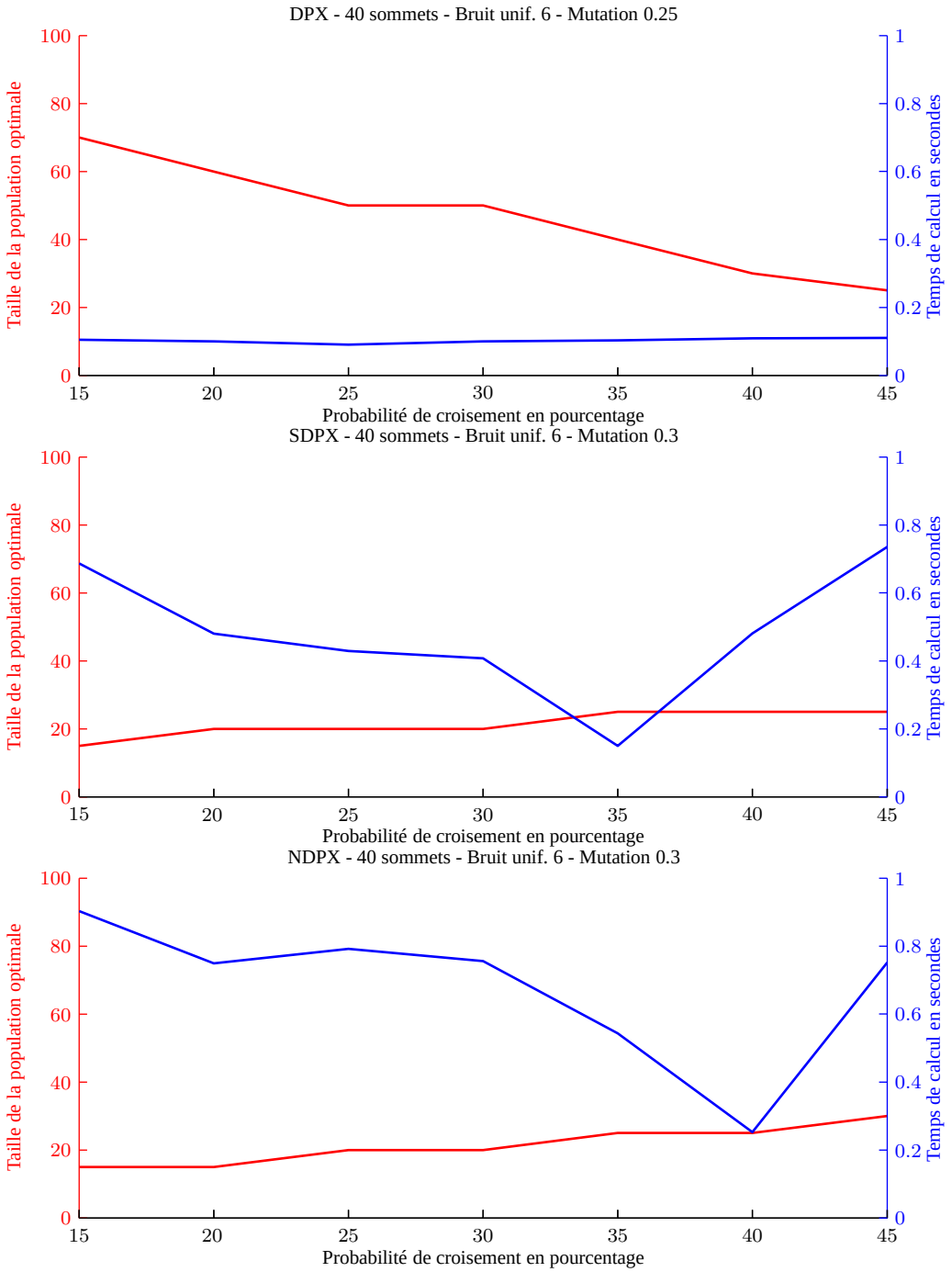

FIGURE 3.11 – Taille de population optimale en fonction de la probabilité de croisement pour DPX (en haut), SDPX (au milieu) et NDPX (en bas).

SDPX et NDPX

Les croisements SDPX et NDPX (cf. Sec. 2.5.5, p. 44) présentent des comportements tout à fait similaires (cf. Figs. 3.12 et 3.13, p. 79 et 80 respectivement). Nous les commentons donc simultanément. SDPX et NDPX sont plus sensibles à la taille de la population que DPX. Le diagramme du haut montre les résultats en fonction de la taille de population. Nous observons un intervalle entre 15 et 300 pour SDPX, et entre 15 et 100 respectivement pour NDPX, individus qui mène à un taux de succès de 1. Pour des petits graphes cet intervalle s'élargit. En ce qui concerne le temps de calcul, il existe un minimum autour de 25 individus, indépendamment de la taille des graphes. Pour des populations plus petites que cet optimum, le temps de calcul augmente beaucoup plus fortement que pour des populations plus grandes.

SDPX et NDPX montrent la même dépendance à la probabilité de croisement que DPX. Des bons taux de succès ne sont possible qu'avec des probabilités faibles (cf. diagrammes du milieu). Nous observons que l'intervalle est plus restreint pour NDPX que pour SDPX. Au-dessus de 0,4 le temps de calcul augmente énormément et à partir de 0,5 le taux de succès baisse sensiblement. Le diagramme du milieu pour SDPX et du bas pour NDPX de la figure 3.11 montrent la corrélation entre la probabilité de croisement et la taille de population optimale associée. Nous constatons que cette relation est beaucoup moins marquante que dans le cas de DPX (cf. Fig. 3.11, p. 77, diagramme du haut). Les populations varient entre 15 (avec une probabilité de croisement de 0,15) et 25 individus (avec une probabilité de croisement de 0,45) ou 30 individus respectivement pour NDPX. La taille de la population optimale croît donc légèrement avec la probabilité de croisement, ce qui est l'inverse de DPX. Les différences de temps de calcul sont, quant à elles, beaucoup plus importantes. Nous observons en particulier un minimum significatif à la probabilité de croisement de 0,35 (SDPX) et de 0,4 (NDPX).

SDPX est peu sensible à la probabilité de mutation mais plus sensible que DPX. Nous observons des taux de succès autour de 1 pour une probabilité d'environ 0,3 (cf. Fig. 3.12, p. 79, diagramme du bas). NDPX est encore plus sensible à la probabilité de mutation que SDPX. Nous observons un taux de succès supérieur à 0,9 entre 0,05 et 0,3. Si l'on souhaite un taux proche de 1, il faut une probabilité de mutation de 0,3 (cf. Fig. 3.13, p. 80 en bas). Le minimum du temps de calcul se situe, dans les deux cas, également autour de 0,3.

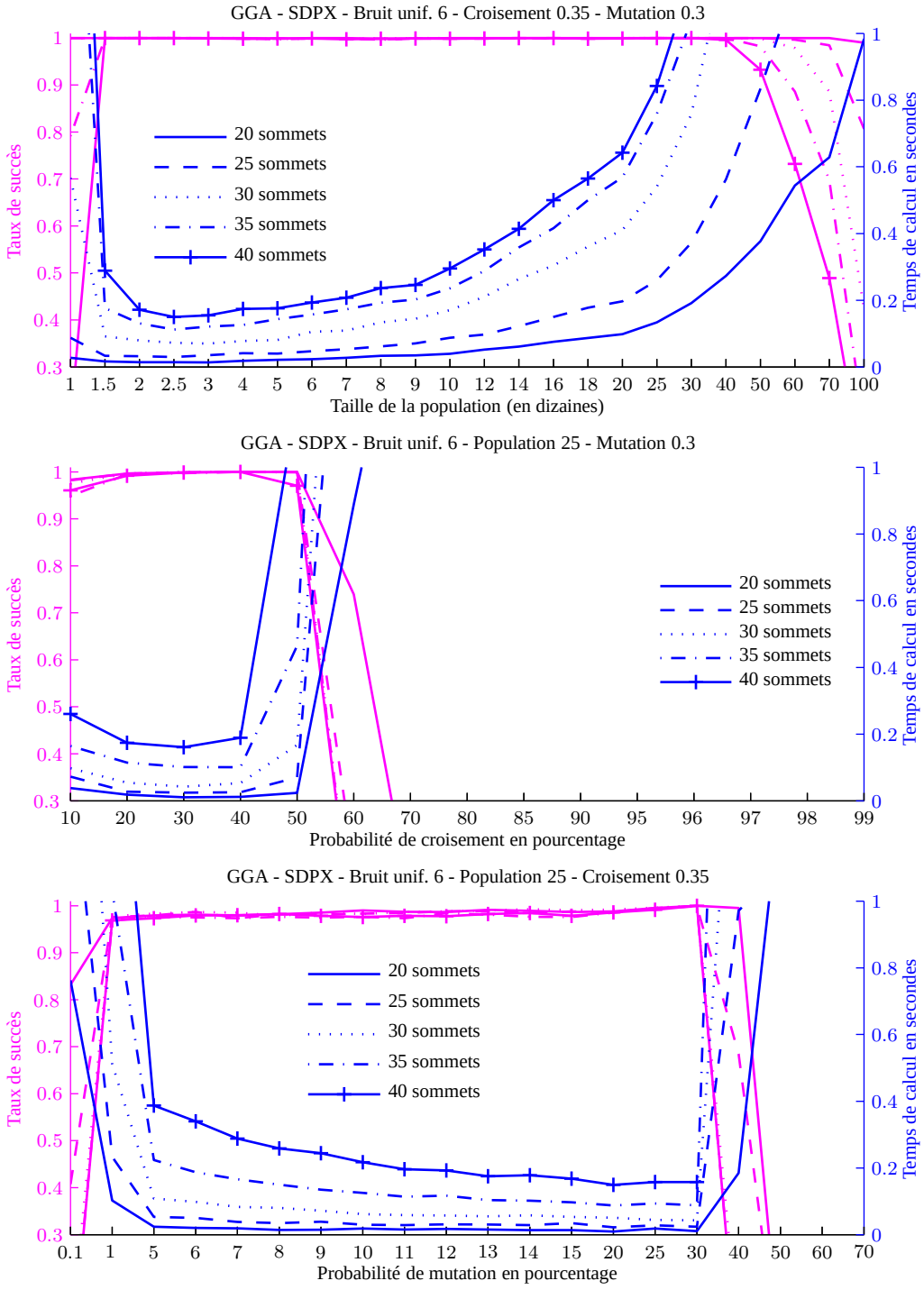

FIGURE 3.12 – Performance du croisement SDPX en fonction des trois paramètres.

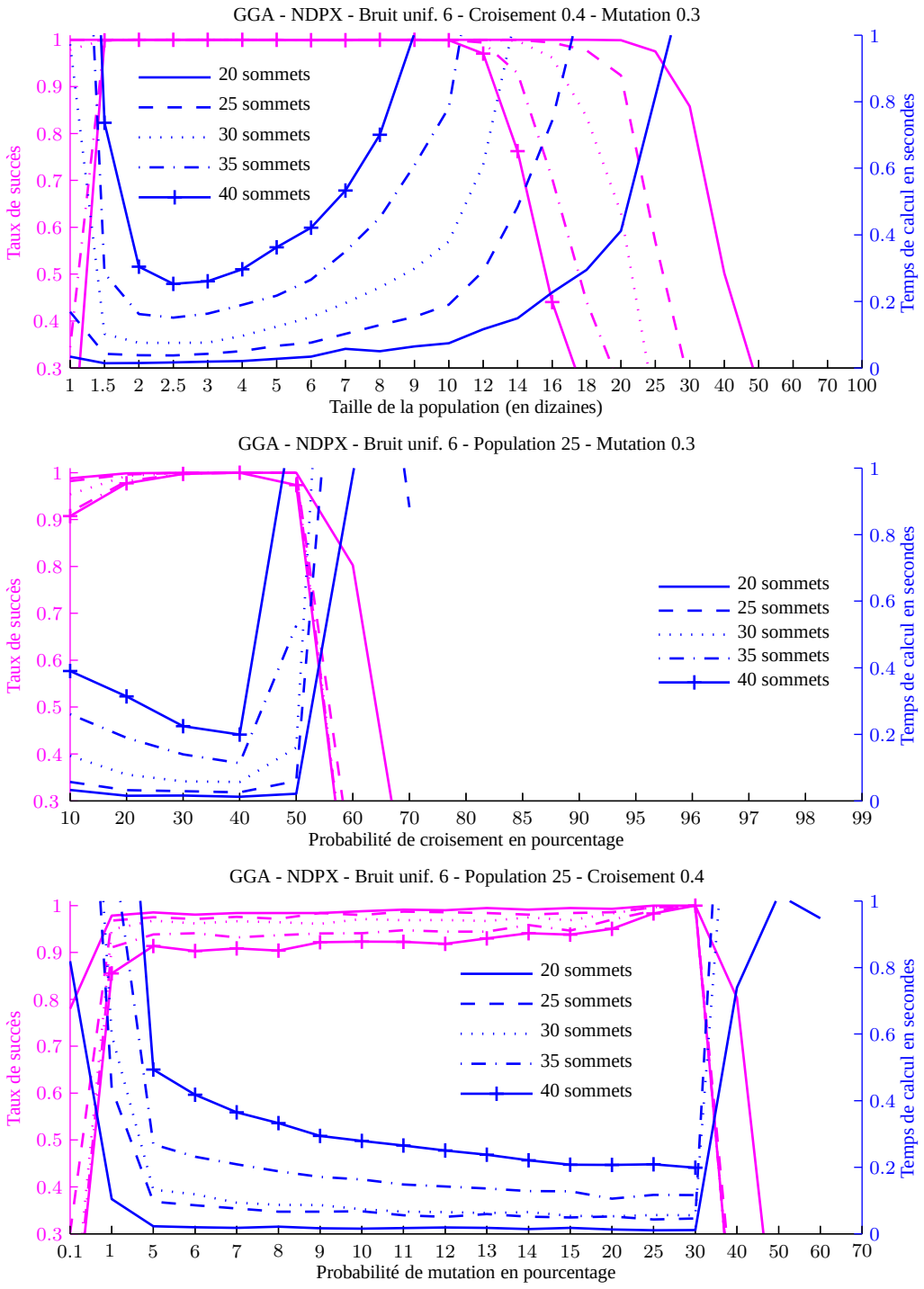

FIGURE 3.13 – Performance du croisement NDPX en fonction des trois paramètres.

3.2.3 Moteur *steady-state*

Nous examinons ensuite le comportement des opérateurs dans le cas des moteurs *steady-state* (décrits dans le Ch. 2, Sec. 2.4, p. 37). Nous utilisons le même protocole de test que précédemment avec un paramètre de moins : la probabilité de croisement. En effet, nous appliquons toujours le croisement à la création d'un enfant. Nous utilisons une population triée qui n'accepte pas de copies d'un même chromosome : si un enfant créé correspond exactement à un individu déjà dans la population, il est refusé et un autre est créé à sa place. S'il s'agit d'un chromosome qui n'est pas encore présent dans la population, il remplace le pire de la population. Le moteur *steady-state*, dans cet implémentation, effectue donc une recherche plus agressive qu'un moteur générationnel. Il risque donc de s'arrêter dans des optima locaux plus que dans l'optimum global.

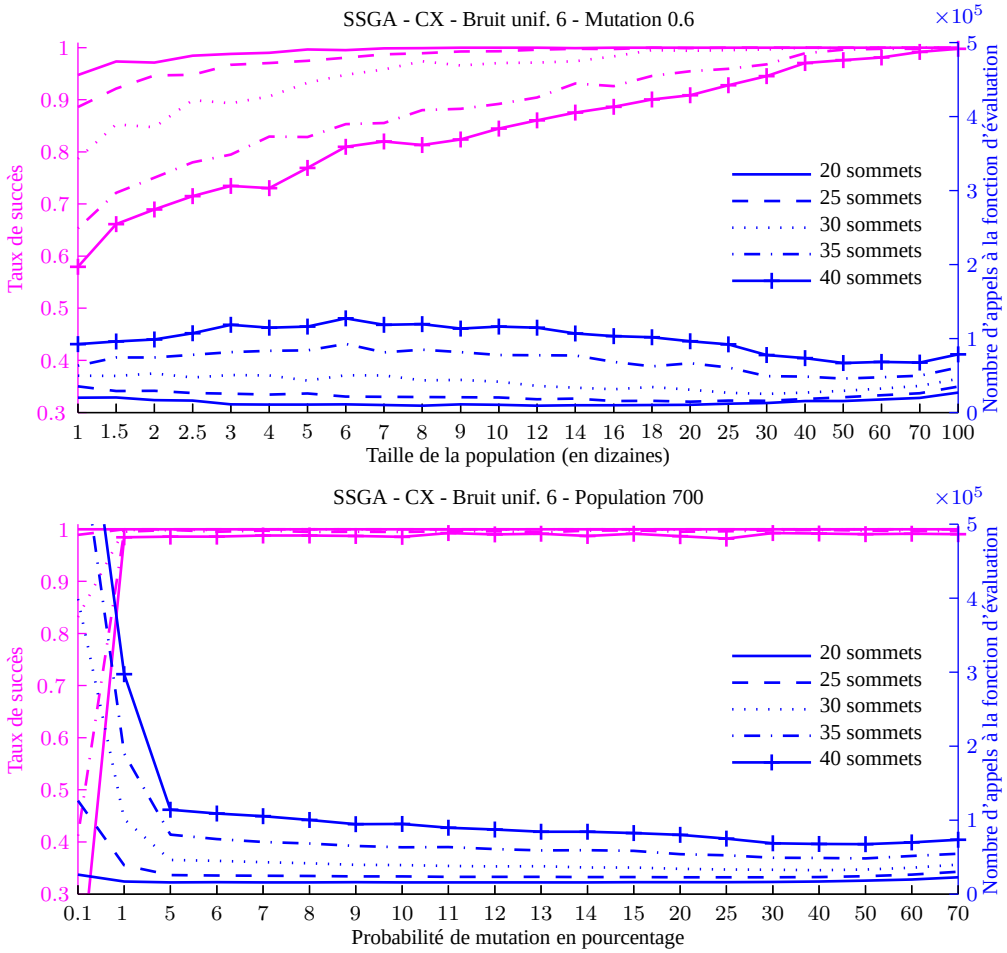

FIGURE 3.14 – Performance du croisement CX en fonction des deux paramètres.

CX

Le diagramme du haut de la figure 3.14 montre le taux de succès et le nombre d'appels à la fonction d'évaluation en fonction de la taille de la population. Nous observons que le taux de succès est croissant avec la taille de la population. Le nombre d'appels à la fonction d'évaluation présente un minimum dans l'intervalle entre 500 et 700 individus (sauf pour des graphes très petits pour lesquels il atteint son minimum avec des populations plus petites).

Le diagramme du bas de la figure 3.14 illustre le taux de succès et le nombre d'appels à la fonction d'évaluation en fonction de la probabilité de mutation. Nous observons que CX est très peu sensible à cette probabilité car le taux de succès atteint son maximum sur tout l'intervalle entre 0,01 et 0,7. Le nombre d'appels à la fonction d'évaluation diminue jusqu'à 0,6, il augmente ensuite très légèrement. L'optimum du temps de calcul se situe donc à 0,6. Il est donc réellement nécessaire d'utiliser une probabilité de mutation assez forte, même si l'on possède de la marge pour justifier la valeur exacte.

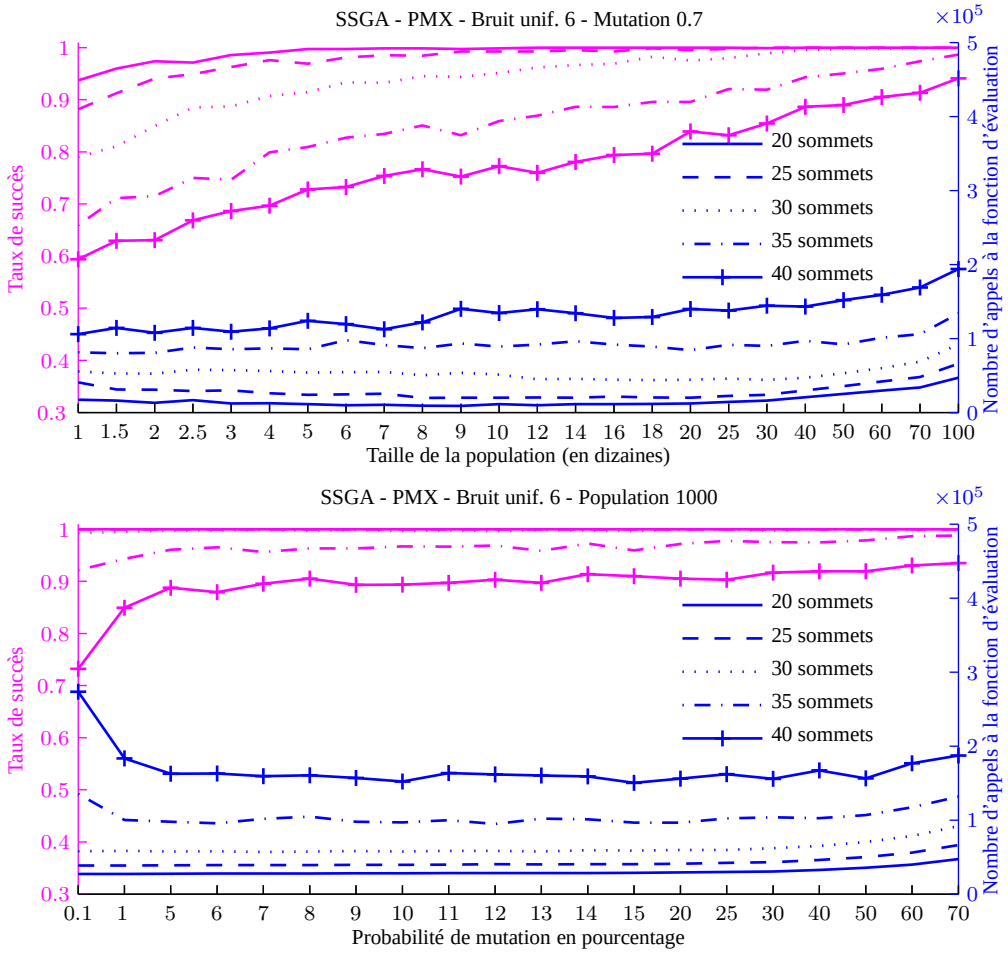

FIGURE 3.15 – Performance du croisement PMX en fonction des deux paramètres.

En comparaison avec les résultats obtenus avec un moteur générationnel, nous constatons que la taille de la population optimale est beaucoup plus élevée. Par contre, la probabilité de mutation optimale est identique.

PMX

Nous constatons que le taux de succès de PMX est croissant avec la taille de la population, de même que le nombre d'appels à la fonction d'évaluation (cf. Fig. 3.15, p. 83). Pour des graphes de moins de 35 sommets, un taux de succès de 1 est possible tandis que pour des graphes plus grands, le taux de succès maximal observé est inférieur. En particulier, pour des graphes ayant 40 sommets, nous observons un taux maximal d'environ 0,9 seulement.

En ce qui concerne la probabilité de mutation (cf. Fig. 3.15, p. 83, diagramme du bas), l'influence est moins marquée. Le taux de succès n'augmente que légèrement avec la probabilité de mutation. Les différences du nombre d'appels à la fonction d'évaluation sont faibles pour des probabilités comprises entre 0,01 et 0,7. Comme le taux de succès maximal est observé à 0,7, nous utilisons par

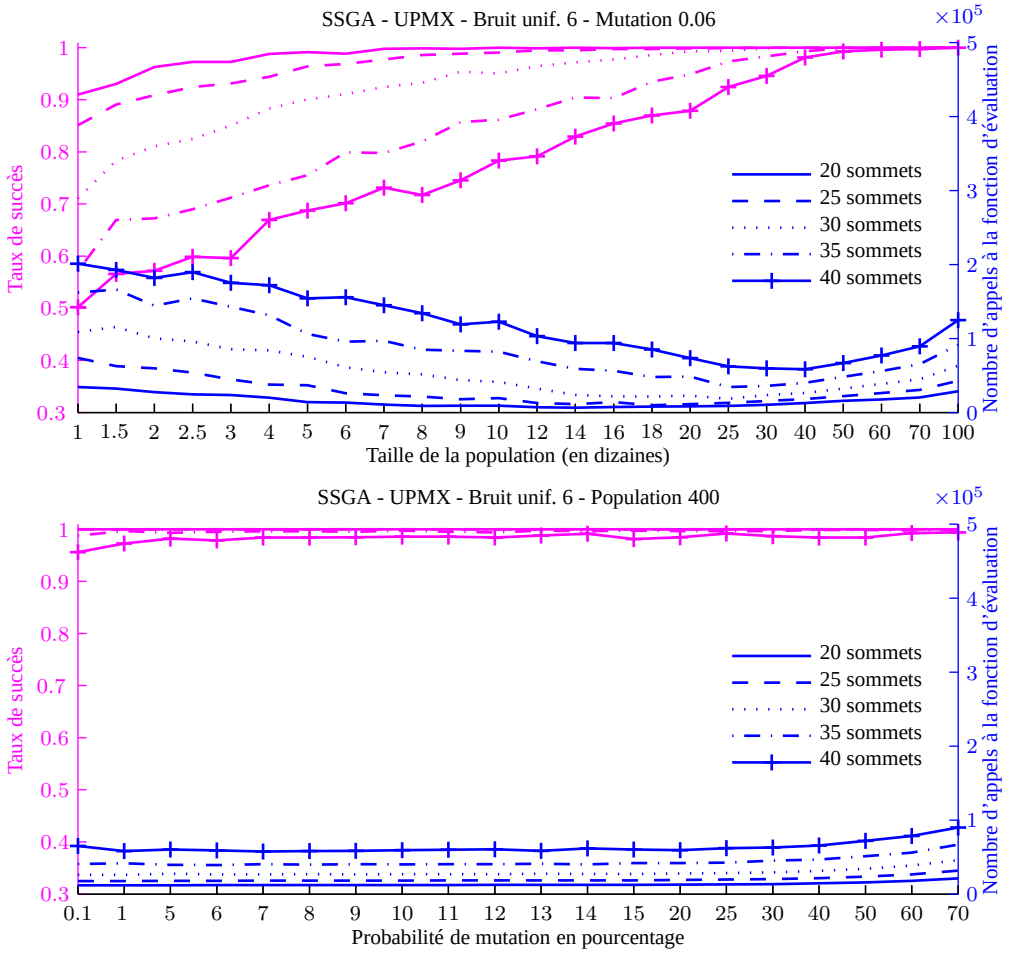

FIGURE 3.16 – Performance du croisement UPMX en fonction des deux paramètres.

la suite cette probabilité.

On peut noter que les deux paramètres choisis correspondent au maximum du domaine de l'étude que nous avons menée. Il est possible qu'en augmentant encore les paramètres, on puisse atteindre de meilleures précisions et un taux de succès de 1. Toutefois, une population de plus de 1000 individus est extrêmement grande et, en combinaison avec une probabilité de mutation de 0,7, elle revient à une recherche aléatoire. Il semble donc que le moteur *steady-state* ne soit pas compatible avec l'opérateur PMX en ce qui concerne notre problème d'optimisation.

UPMX, PBX et UPBX

Les opérateurs UPMX, PBX et UPBX montrent des comportements très similaires. Nous les commentons donc simultanément. Le diagramme du haut des figures 3.16, 3.17 et 3.18 respectives, montre que, pour les trois opérateurs, le taux de succès est croissant avec la taille de la population. A partir d'un seuil de 400 individus pour UPMX, et légèrement inférieur pour PBX et UPBX, le taux est proche de 1 pour toutes les tailles de graphes testées. Le nombre d'appels à la fonction

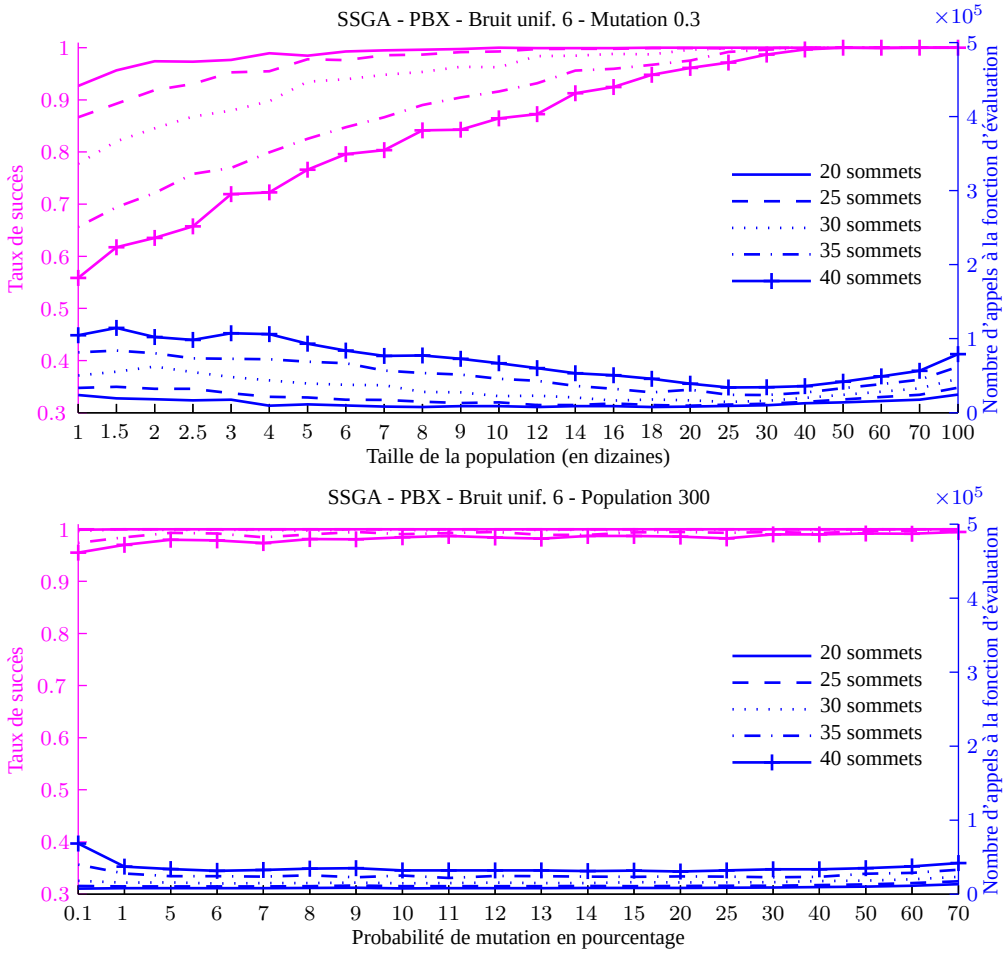

FIGURE 3.17 – Performance du croisement PBX en fonction des deux paramètres.

d'évaluation atteint un minimum autour entre 300 et 400 (UPMX), autour de 300 (PBX), entre 250 et 400 (UPBX) individus. Comme le taux de succès est croissant avec la taille de la population, nous utilisons les bornes supérieures respectives comme taille de population, soit 400 pour UPMX, 300 pour PBX et 400 pour UPBX.

Les résultats sont quasiment indépendants de la probabilité de mutation (cf. diagrammes du bas des Fig. 3.16, 3.17 et 3.18) à la fois en ce qui concerne le taux de succès et le nombre d'appels à la fonction d'évaluation. Nous identifions un minimum de ce dernier à 0,06 pour UPMX et UPBX et 0,3 pour PBX, mais ces trois minima ne montrent pas de différence significative aux valeurs voisines.

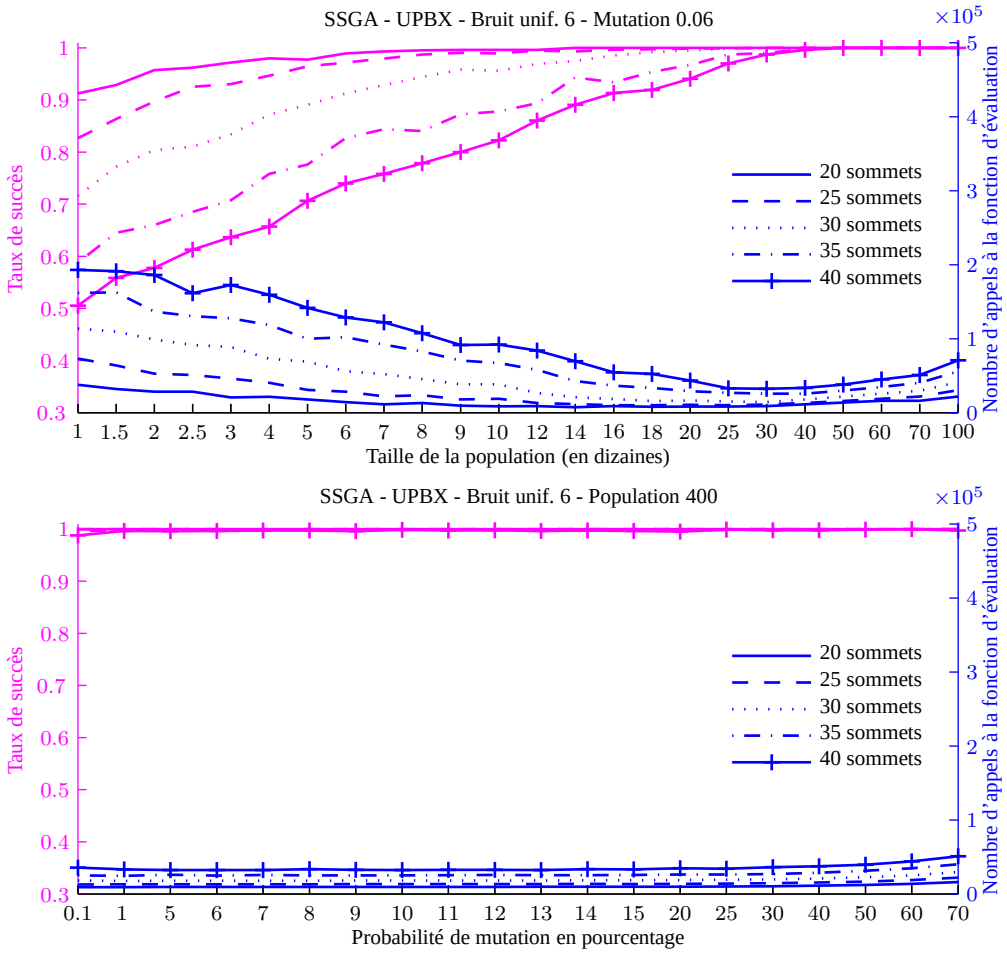

FIGURE 3.18 – Performance du croisement UPBX en fonction des deux paramètres.

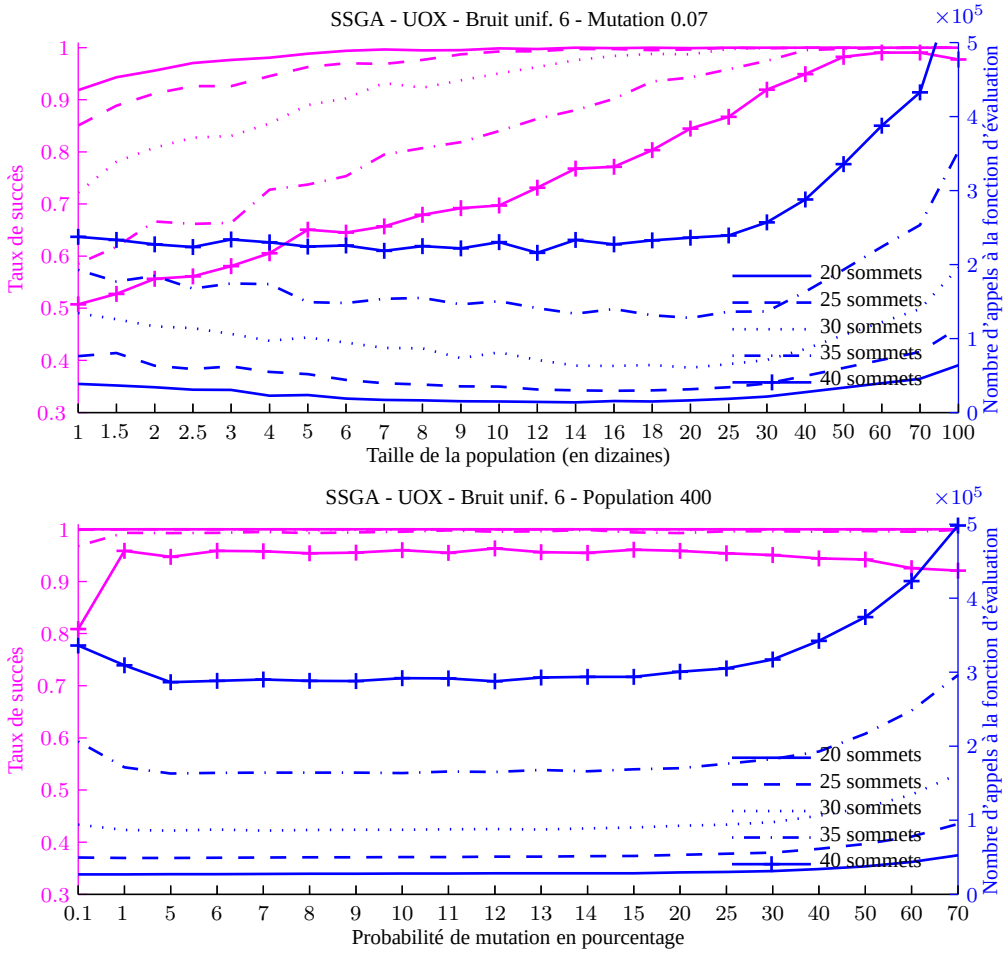

FIGURE 3.19 – Performance du croisement UOX en fonction des deux paramètres.

UOX

Le diagramme du haut de la figure 3.19 montre que le taux de succès est également croissant avec la taille de population pour UOX. Le nombre d'appels à la fonction d'évaluation est quasiment stable pour une taille de population comprise entre 10 et 300 (voire encore plus encore pour des petits graphes). Il augmente ensuite très fortement. On observe donc un conflit entre les deux objectifs de l'optimisation des paramètres. Nous choisissons un compromis entre les deux avec une taille de population de 400.

En ce qui concerne la probabilité de mutation (cf. Fig. 3.19, p. 87, diagramme du bas), nous observons sur tout l'intervalle de 0,05 à 0,2 des résultats quasiment identiques, bien qu'ils dépendent fortement de la taille des graphes en ce qui concerne le nombre d'appels à la fonction d'évaluation. Nous utilisons une valeur de 0,07.

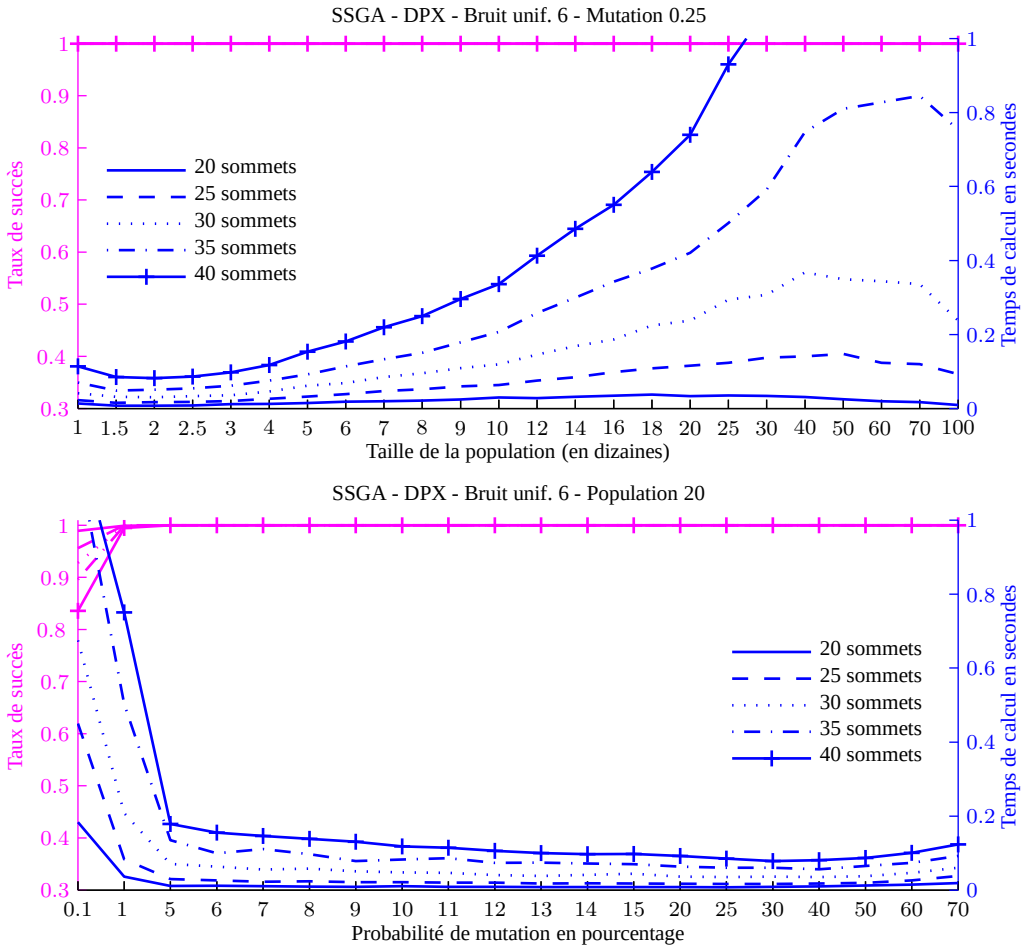

FIGURE 3.20 – Performance du croisement DPX en fonction des deux paramètres.

DPX

Comme dans le cas du moteur générationnel, l'étude des opérateurs intelligents DPX, SDPX et NDPX nécessite de changer le critère de mesure du temps d'exécution, car ils incluent une approche gloutonne assez coûteuse.

Le diagramme du haut de la figure 3.20 illustre les résultats en fonction de la taille de population. Le taux de succès est 1, indépendamment de la taille de population et de la taille des graphes pour DPX. Le temps de calcul atteint un minimum autour d'une vingtaine d'individus.

Le diagramme du bas de la figure 3.20 illustre les résultats en fonction de la probabilité de mutation. Hors des probabilités très faibles (inférieures à 0,05), on atteint toujours des taux de succès proche de 1. Le temps de calcul marque un minimum autour de 0,25 et reste quasiment constant jusque 0,5.

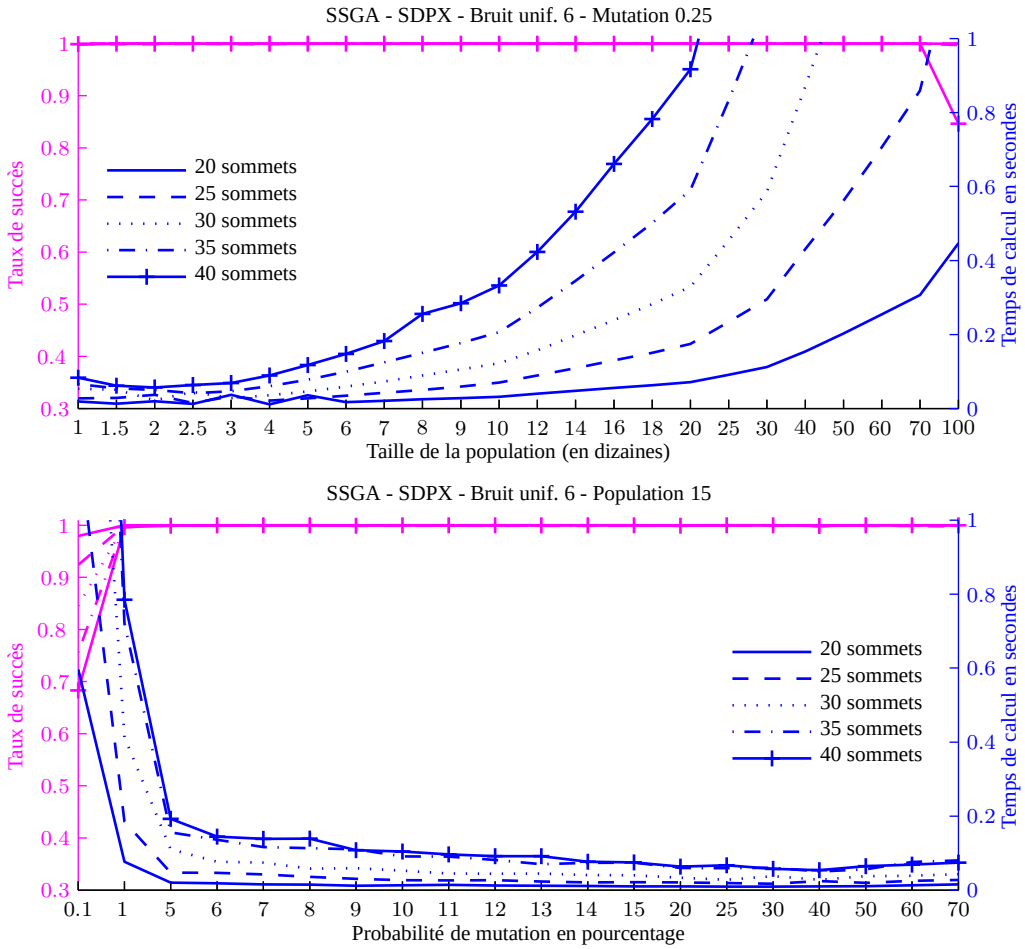

FIGURE 3.21 – Performance du croisement SDPX en fonction des deux paramètres.

SDPX et NDPX

Les diagrammes du haut des figures 3.21 et 3.22 montrent les résultats de SDPX et NDPX, en fonction de la taille de la population. Le taux de succès reste toujours à 1 (sauf pour 1000 individus en combinaison avec des graphes de 40 sommets pour SDPX, et plus de 600 individus en combinaison avec des graphes d'au moins 35 sommets pour NDPX). Nous constatons un minimum du temps de calcul entre 15 et 30 individus pour les deux opérateurs ; pour des populations plus grandes, le temps augmente fortement.

Le diagramme du bas montre les résultats en fonction de la probabilité de mutation. Le taux de succès atteint 1 à partir d'une probabilité égale à 0,05. A partir de cette probabilité, le temps de calcul devient aussi acceptable. Il marque son minimum entre 0,25 et 0,5 pour SDPX et entre 0,2 et 0,6 pour NDPX.

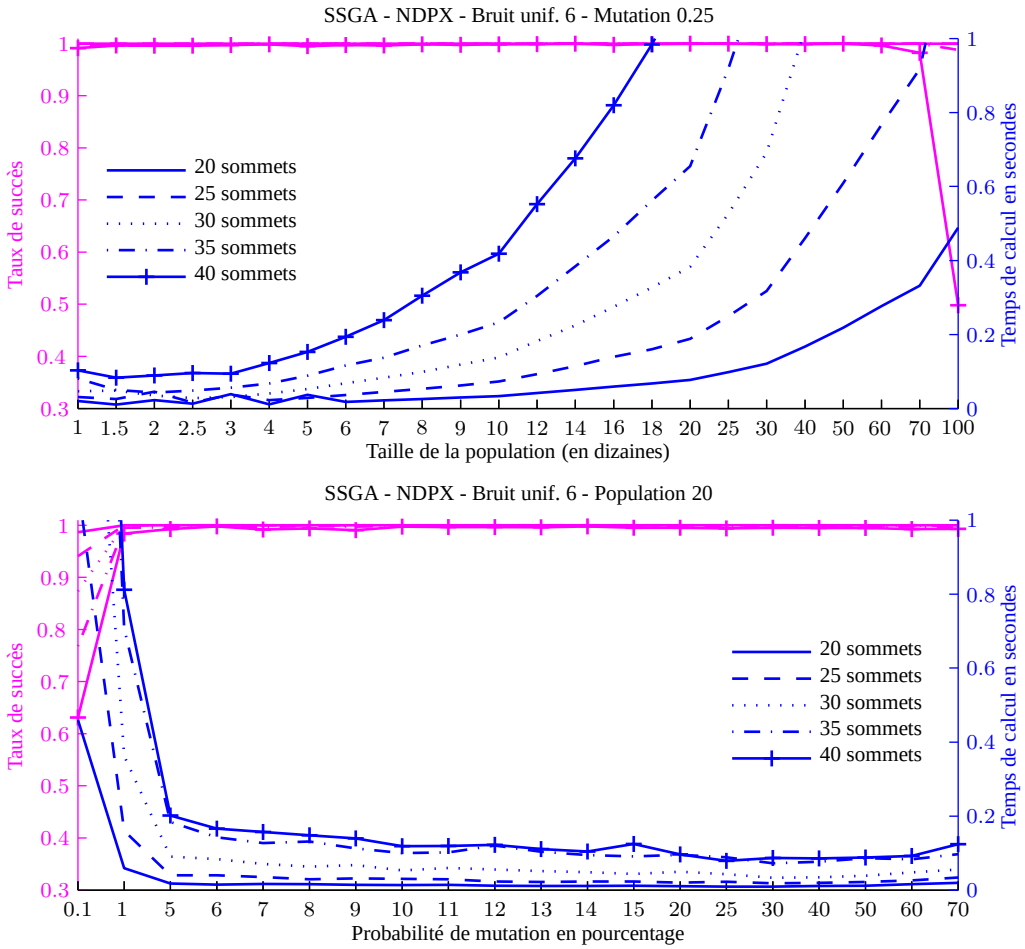

FIGURE 3.22 – Performance du croisement NDPX en fonction des deux paramètres.

3.2.4 Étude de la sensibilité au bruit

Les résultats précédents concernent un bruit uniforme de niveau 6%. Nous avons réalisé le même type d'étude en considérant les niveaux de bruit de 2, 4, 8 et 10% et le problème d'isomorphisme exact, qui correspond à un niveau de bruit de 0. Nous avons également étudié le cas du bruit gaussien. Les résultats complets sont présentés en annexe A, les figures 3.23 et 3.24 (cf. p. 92 et 93) les illustrent dans le cas de l'opérateur UPBX avec un moteur générationnel qui constitue un exemple représentatif : ils montrent que le niveau et le type de bruit n'influencent pas le choix des paramètres optimaux, et que les valeurs choisies lors des analyses présentées dans les sections 3.2.2 et 3.2.3 précédentes sont valables de façon générale.

Nous observons d'abord que le type de bruit (gaussien ou uniforme) n'a aucune influence sur les résultats. Le niveau de bruit, quant à lui, influence le temps de calcul et le taux de succès. Nous constatons que le temps de calcul est croissant avec ce paramètre tandis que le taux de succès montre une tendance décroissante. En ce qui concerne la taille de la population (diagrammes du haut), nous

observons que la forme de la courbe reste toujours similaire. En particulier, la position des minima et maxima reste inchangée. On peut néanmoins constater que l'intervalle de la population optimale se rétrécit quand le bruit augmente. Pour obtenir un taux de succès de 0,99 par exemple, on peut choisir une taille de population comprise entre 80 et 600 individus dans le cas exact. Quand on considère un niveau de bruit uniforme de 10, alors seule une taille entre 100 et 180 individus est possible. En ce qui concerne le nombre d'appels à la fonction d'évaluation, on constate que les différences s'accentuent. Si l'on passe, par exemple, d'une population de 90 individus, ce qui correspond à la taille optimale, à 100, le nombre d'appels à la fonction d'évaluation subit une augmentation de 5,6% pour un niveau de bruit de bruit uniforme de 2, alors que pour un niveau de 4, on observe déjà une augmentation de 9%.

En ce qui concerne la probabilité de croisement (diagrammes du milieu), nous observons des effets similaires pour le taux de succès. Quand le niveau de bruit augmente, le taux de succès baisse. La probabilité de croisement optimale est 0,99, mais dans les cas de bruit modéré, il est possible de la baisser sans perte de précision. Le nombre d'appels à la fonction d'évaluation augmente en général avec le niveau de bruit, mais les valeurs ne diffèrent pas beaucoup.

La valeur optimale pour la probabilité de mutation se trouve dans un intervalle où le taux de succès, ainsi que le nombre d'appels à la fonction d'évaluation, sont croissants. Elle constitue donc un compromis entre les deux objectifs de l'optimisation des paramètres. En ce qui concerne le taux de succès, nous observons que l'optimum se trouve à 0,25, alors que l'optimum du nombre d'appels de la fonction d'évaluation se situe près de 0,15. Nous nous intéressons donc aux différences dans l'intervalle entre ces deux valeurs. Le taux de succès pour une probabilité de mutation de 0,25 est toujours proche de 1 (± 0.005). Pour une probabilité de 0,15, elle décroît en fonction du niveau de bruit, de 1 (pour les niveaux de bruit 0 et 2) à 0,91 (pour un niveau de 10), en passant par 0,98 et 0,95. La perte devient donc plus importante avec le niveau de bruit. En ce qui concerne le nombre d'appels à la fonction d'évaluation, nous constatons que l'augmentation est entre 15% (niveaux de bruit 2 et 4) et s'accentue jusqu'à 45% (niveaux 8 et 10). On observe une augmentation plus forte pour le niveau 0 (22%) que pour le niveau 2. En effet, le niveau 0 est particulier car il correspond à l'isomorphisme exact. La contradiction entre les deux objectifs, précision et temps de calcul, et donc la nécessité de définir un compromis entre ces deux subsiste malgré l'analyse des différents niveaux de bruit.

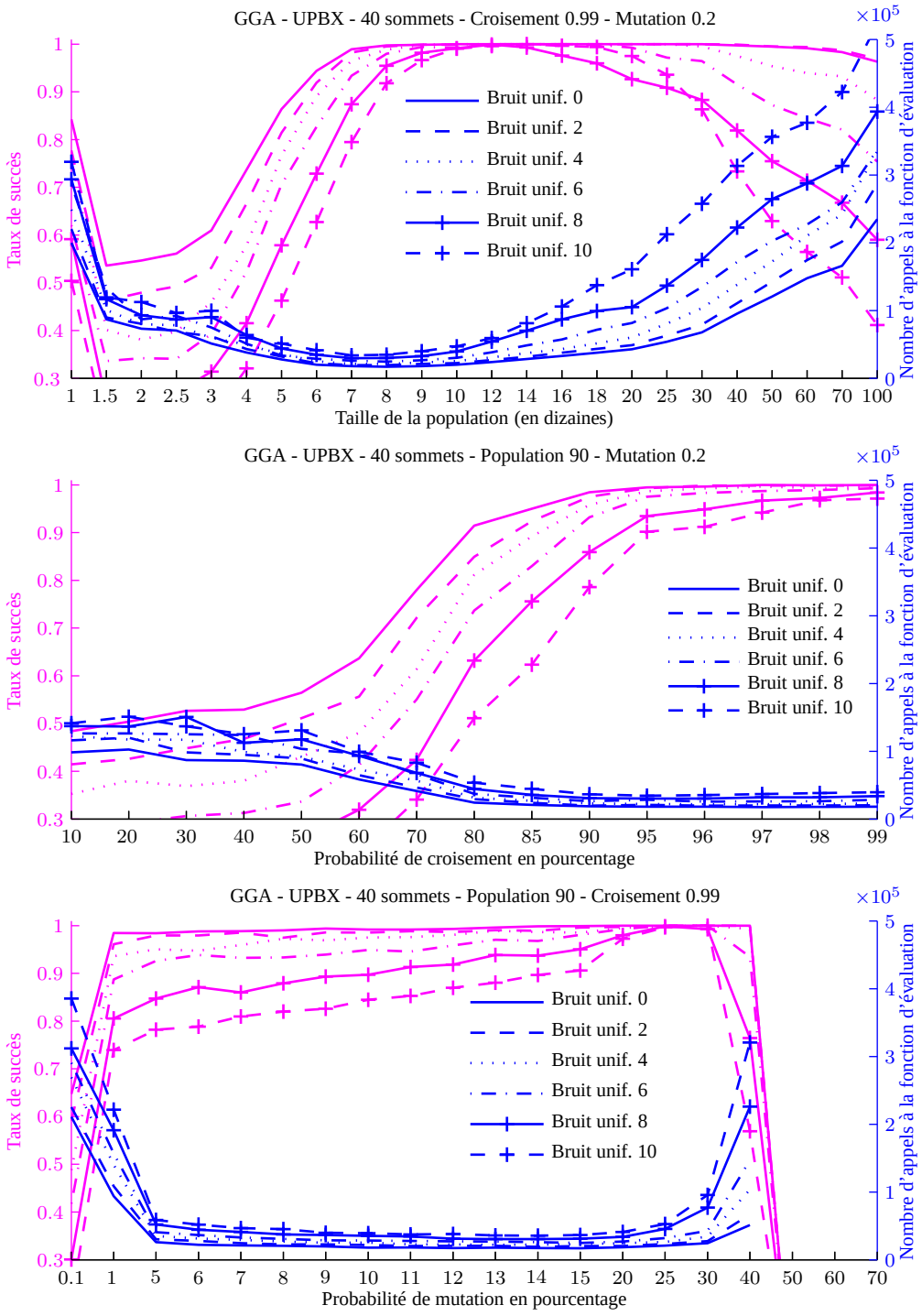

FIGURE 3.23 – Sensibilité au bruit uniforme de l'opérateur UPBX.

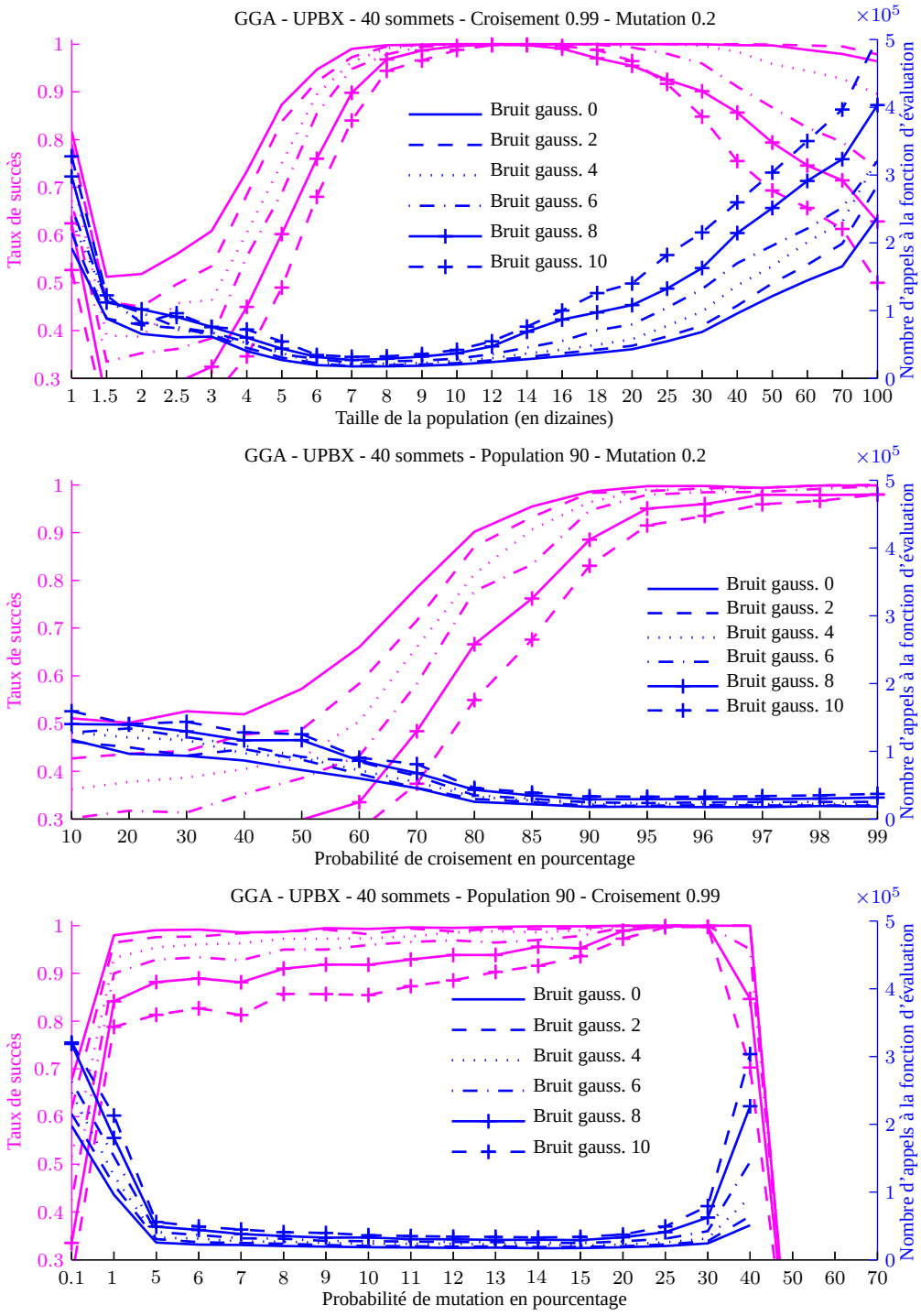

FIGURE 3.24 – Sensibilité au bruit gaussien de l'opérateur UPBX.

3.3 Comparaison des opérateurs de croisement

Dans la section précédente, nous avons déterminé individuellement les paramètres optimaux pour chaque opérateur. Nous comparons maintenant leurs performances, en analysant les valeurs optimales observées puis en examinant les performances que ces dernières permettent d'obtenir.

3.3.1 Comparaison des paramètres optimaux

Les meilleurs paramètres obtenus pour chaque opérateur sont récapitulés dans le tableau 3.1.

Taille de la population La taille de la population joue en général un rôle très important, qui conduit le plus souvent à des effets contradictoires sur le taux de succès et le temps de calcul. Une augmentation ralentit l'algorithme car le nombre d'individus à traiter à chaque génération augmente dans un moteur générationnel ; dans un moteur *steady-state* l'initialisation de la population prend plus de temps. Quand on analyse les diagrammes du taux de succès et du temps de calcul, nous trouvons un optimum pour le temps de calcul mais également un optimum pour le taux de succès qui est généralement un peu décalé vers des populations plus grandes par rapport à ce premier. Un compromis entre ces deux objectifs doit donc être fixé : nous avons généralement favorisé de meilleurs taux de succès, sauf s'ils sont déjà à un niveau très haut (supérieur à 0,99) ou si nous estimons que le ralentissement est trop important par rapport au gain de taux de succès.

Probabilité de croisement La probabilité de croisement, paramètre nécessaire uniquement dans le cas d'un moteur générationnel, est très proche de 1 pour les opérateurs classiques, excepté pour UPMX et UOX pour lesquels nous constatons de meilleures performances avec des probabilités de 0,8 ou 0,6. Quant aux opérateurs *DPX, une probabilité élevée mène à des taux de succès très faibles, voire de 0 à partir d'un certain seuil. De plus, dans ce cas, nous observons une corrélation de la probabilité de croisement avec la meilleure taille de la population correspondante. Elle est négative pour DPX et positive pour SDPX et NDPX.

Probabilité de mutation Nous constatons que la plupart des opérateurs nécessitent toujours des probabilités de mutation élevées, notamment CX, PMX, PBX et les trois opérateurs *DPX pour le moteurs générationnel comme le *steady-state*. D'une part, une telle probabilité élevée est nécessaire si l'opérateur est trop peu destructif, comme pour CX qui ne détruit rien au sens où chaque gène de l'enfant correspond au même gène dans un des parents, ou pour PMX et PBX qui sont limités en raison de leur non uniformité et dont certains gènes ont une probabilité d'être altérés supérieure à d'autres. D'autre part, en ce qui concerne les *DPX, la probabilité élevée réduit le risque d'une convergence prématurée provenant de la partie gloutonne de la construction des nouveaux individus. De plus, on constate que pour tous ces opérateurs qui nécessitent une probabilité de mutation élevée

Moteur / Opérateur	générationnel			*steady-state*	
	s_p	p_c	p_m	s_p	p_m
CX	400	99%	60%	700	60%
PMX	120	99%	40%	1000	70%
UPMX	90	80%	12%	400	6%
PBX	120	96%	30%	300	30%
UPBX	90	99%	20%	400	6%
UOX	10	60%	14%	400	7%
DPX	50	25%	25%	20	25%
SDPX	25	35%	30%	15	25%
NDPX	25	40%	30%	20	25%

Tableau 3.1 – Paramètres optimaux par opérateur.

dans le cadre d'un moteur générationnel, celle-ci est aussi importante dans le cadre d'un moteur *steady-state*.

Les autres opérateurs, UPMX, UPBX et UOX, nécessitent une probabilité plus modérée dans le cadre d'un moteur générationnel et ils ne sont pas sensibles à la probabilité de mutation dans le cas d'un moteur *steady-state*. Les effets de la mutation pour le moteur *steady-state* sont très faibles par rapport à l'influence de la taille de la population.

3.3.2 Comparaison des performances

Nous comparons ensuite les différents opérateurs avec leurs paramètres optimaux respectifs. Nous choisissons non seulement les paramètres optimaux pour chaque opérateur mais aussi le moteur d'évolution le mieux adapté (générationnel, GGA ou *steady-state*, SSGA). Afin de créer un classement, nous avons besoin de prendre en compte non seulement le taux de succès mais aussi le temps de calcul ou le nombre d'appels à la fonction d'évaluation.

Représentation graphique des performances Comme nous utilisons une base pour laquelle nous connaissons la solution optimale pour chaque paire de graphes, nous pouvons identifier pour chaque exécution le moment exact où cette solution a été trouvée. Nous limitons le nombre d'appels à la fonction d'évaluation à un million d'appels ce qui est largement supérieur au nombre normalement constaté. Nous interrompons également une exécution dès qu'elle trouve la bonne solution. De cette manière, nous assurons que quasiment toutes les exécutions ont terminé avant le nombre maximal fixé d'appels à la fonction d'évaluation. En revanche, si une exécution continue jusqu'à cette limite, nous pouvons conclure qu'elle est prisonnière d'un optimum local et qu'elle n'en sortira probablement pas en un temps raisonnable.

La courbe (cf. Fig. 3.25, p. 100, par exemple) montre à chaque instant le taux de succès sous condition que la solution a été trouvée au plus tard à ce moment. Nous comparons les opérateurs soit par rapport au nombre d'appels à la fonction d'évaluation (cf. diagramme du bas), ce qui est le plus approprié pour les opérateurs classiques, soit par rapport au temps écoulé (cf. le diagramme du haut) qui est nécessaire quand la comparaison inclut au moins un opérateur intelligent.

Nous pouvons dire avec certitude qu'un algorithme est supérieur à un autre si sa courbe est complètement au-dessus de la courbe de ce dernier. Si les courbes se croisent, alors aucun algorithme ne domine l'autre : l'un fournit plus rapidement quelques bonnes solutions, tandis que l'autre commence plus tard, mais rattrape le premier (et donne éventuellement un taux de succès final supérieur).

Étude des dominations Nous observons que l'on peut diviser les opérateurs en trois groupes en fonction de leurs performances (cf. Fig. 3.25, p. 100). Le meilleur groupe contient les opérateurs intelligents, DPX, SDPX, et NDPX. Les courbes de ces trois opérateurs sont très proches les unes des autres et la différence par rapport aux opérateurs non intelligents est importante. Parmi les opérateurs non intelligents, nous constatons que les opérateurs PBX et UPBX présentent des performances similaires supérieures à celles des autres. Nous observons que les courbes se croisent deux fois : UPBX fournit les premières solutions légèrement plus tôt que PBX, mais le taux de succès de ce dernier croît plus rapidement ensuite. Enfin, le taux de succès final d'UPBX est légèrement plus élevé que celui de PBX. La courbe d'UPBX paraît également plus régulière.

Le troisième groupe d'opérateurs contient les opérateurs UPMX, PMX et CX. En fait, ce groupe montre une variance interne plus élevée que les deux groupes précédents. Elle est visiblement séparée du deuxième groupe mais la distance est moins marquée qu'entre les opérateurs intelligents et les opérateurs PBX et UPBX surtout en termes de temps de calcul. Dans ce groupe, nous constatons que la courbe la plus régulière est celle de PMX qui croise chacune des deux autres courbes deux fois. Les deux autres montrent une croissance plus élevée à partir du moment où ils commencent à fournir les premières solutions. Nous observons également qu'UPMX est plus performant que CX, pour PMX l'ordre dépend des préférences : si l'on souhaite le meilleur taux de succès après une demi-seconde, alors UPMX, avec environ 0,9, est nettement meilleur que PMX, avec environ 0,75, qui est meilleur que CX, avec environ 0,7. Si l'on attend une seconde entière, alors les trois opérateurs ont trouvé la plupart des solutions correctes ; CX atteint presque 0,99, PMX 0,97 et UPMX 0,96. En général, nous constatons que CX est quasiment dominé par UPMX, tandis que PMX et UPMX sont plus proches l'un de l'autre.

Enfin, l'opérateur UOX montre des performances tellement faibles par rapport aux autres opérateurs que nous considérons qu'il ne constitue pas un choix raisonnable pour le problème d'isomorphisme de graphes. Ses performances faibles ne peuvent pas provenir de valeurs inadaptées des paramètres, puisque ces dernières ont fait l'objet d'une optimisation rigoureuse. De plus, des ex-

Mesure / Opérateur	AES		Temps de calcul	
	Moteur	Diff. de surface	Moteur	Diff. de surface
CX	SSGA	-5,28%	GGA	0,24%
PMX	GGA	286,73%	GGA	112,32%
UPMX	SSGA	-19,94%	SSGA	-13,08%
PBX	SSGA	-4,16%	SSGA	-4,93%
UPBX	GGA	2,97%	GGA	3,24%
UOX	GGA	762,32%	GGA	24826,46%
DPX	SSGA	-3,78%	SSGA	-0,28%
SDPX	SSGA	-11,07%	SSGA	-5,77%
NDPX	SSGA	-14,34%	SSGA	-7,36%

Tableau 3.2 – Meilleur moteur d'évolution par opérateur de croisement. Pour chaque opérateur, le gain relatif du moteur générationnel par rapport au moteur *steady-state* est indiqué (mesuré en utilisant l'intégrale de l'histogramme taux de succès/temps). Les résultats sont statistiquement significatifs. Les p-valeurs du test de Kruskal-Wallis sont inférieures à 0,01 pour toutes les paires.

périmentations complémentaires que nous ne détaillerons pas ici ont été menées afin de tester des paramètres de façon plus complète encore que pour les autres opérateurs.

Comparaison quantitative Dans le cas des courbes non-dominantes (c'est-à-dire des courbes qui se croisent) et si l'on souhaite établir un classement automatiquement, nous proposons l'utilisation de la surface sous la courbe comme quantification de la qualité d'un opérateur. Le segment utilisé pour calculer l'intégrale est de 0 à 2 secondes quand on mesure le temps et de 0 à 200000 quand on mesure le nombre d'appels. On peut noter qu'en général, des bornes supérieures de l'intervalle plus faibles favorisent les algorithmes qui commencent à converger plus rapidement tandis que des bornes plus longues favorisent des algorithmes qui fournissent un meilleur taux de succès à la fin.

Dans un premier temps, nous utilisons cette approche pour déterminer quel moteur d'évolution est le meilleur pour chaque opérateur. Le tableau 3.2 montre les résultats. On observe que les opérateurs intelligents fonctionnent mieux avec un moteur *steady-state*, les opérateurs classiques diffèrent dans leurs préférences. Nous constatons également que PMX et UOX montrent d'importantes différences entre les surfaces calculées. Pour UOX, ceci provient du fait qu'avec un moteur *steady-state* la courbe commence à augmenter uniquement avant que le temps n'atteigne la borne maximale. La surface est donc très petite et une légère amélioration des performances, avec un moteur générationnel, mène à une surface beaucoup plus grande.

Dans un second temps, nous constituons six classements entre les opérateurs. Les classements diffèrent selon le moteur d'évolution utilisé (générationnel, *steady-state* ou le meilleur pour chaque

Mesure / Rang	Temps de calcul			AES		
	SSGA	GGA	Meilleur moteur	SSGA	GGA	
1	SDPX	DPX	**SDPX**	DPX	DPX	DPX
2	DPX	SDPX	**DPX**	SDPX	SDPX	UPBX
3	NDPX	NDPX	**NDPX**	NDPX	NDPX	SDPX
4	PBX	UPBX	UPBX	**UPBX**	PBX	PBX
5	UPBX	PBX	PBX	**PBX**	UPBX	NDPX
6	UPMX	PMX	PMX	**UPMX**	UPMX	PMX
7	CX	CX	UPMX	**PMX**	CX	CX
8	PMX	UPMX	CX	**CX**	PMX	UPMX
9	UOX			UOX		

Tableau 3.3 – Opérateurs classés par l'intégrale des histogrammes de taux de succès, en fonction du temps de calcul, à gauche, ou du nombre d'appels à la fonction d'évaluation (AES), à droite ; en utilisant un moteur d'évolution fixe pour tous les opérateurs (colonnes 2,3,6 et 7) ou le meilleur moteur de chacun (colonnes 4 et 5).

opérateur) et la mesure de la durée qui peut être le temps de calcul ou le nombre d'appels à la fonction d'évaluation. Les résultats sont présentés dans le tableau 3.3. Nous ajoutons aussi les classifications qui utilisent pour chaque opérateur le meilleur moteur d'évolution, nous comparons ainsi par exemple CX - SSGA avec PMX - GGA (comme dans la Fig. 3.25). Dans les deux colonnes au centre, nous marquons en gras les préférences du critère de comparaison par opérateur : nous préférons comparer les opérateurs non intelligents en fonction du nombre d'appels à la fonction d'évaluation, parce que celui-ci est robuste aux différences d'implémentation et au choix de matériel. Pour les opérateurs intelligents, nous préférons la comparaison par temps de calcul, car la partie gloutonne peut augmenter le temps de calcul nécessaire pour un même nombre d'appels à la fonction d'évaluations.

Bien que les classements diffèrent sur certaines positions, nous observons que le classement entre groupes que nous avons constaté précédemment reste constant. Ces groupes sont séparés par des lignes horizontales. Des expérimentations complémentaires avec différentes bornes maximales pour le calcul d'intégrale ont montré que les classements obtenus sont assez robustes à des différentes valeurs pour la borne maximale de l'intégrale. Les courbes analogues à la figure 3.25 pour les colonnes du tableau qui classent les opérateurs en combinaison avec un moteur d'évolution fixe pour tous les opérateurs se trouvent, à des fins d'exhaustivité, dans l'annexe B.

La seule exception à cet ordre de groupes est le changement entre PBX et UPBX d'un côté et SDPX et NDPX de l'autre dans le cas du moteur générationnel en mesurant le nombre d'appels à la fonction d'évaluation : UPBX passe du second au premier groupe, en dépassant même SDPX, et

NDPX passe du premier au second groupe, obtenant un classement derrière PBX. De fait, NDPX et SDPX sont défavorisés parce qu'ils fonctionneraient mieux avec la stratégie *steady-state*. La perte de surface quand on passe du moteur *steady-state* au moteur générationnel, est de 11% pour SDPX et de 14% pour NDPX (cf. Tab. 3.2, p. 97). A celà, s'ajoute encore le gain léger d'UPBX de 3%. En ce qui concerne PBX dont les performances dépassent dans ce cas particulier celles de NDPX mais non celles de SDPX, nous constatons que le moteur générationnel le défavorise mais la perte est, avec 4% de la surface, beaucoup moins importante que pour SDPX et NDPX. Par ailleurs, la comparaison en termes de temps de calcul, qui inclut notamment le coût de tous les opérateurs génétiques, montre clairement que NDPX et SDPX sont plus performants que PBX et UPBX. En fait, les probabilités de croisement sont seulement de 35% et de 40% pour SDPX et NDPX, mais de 96% voire de 99% pour PBX et UPBX, et celles de mutation sont comparables. Par conséquent, PBX et UPBX nécessitent des appels beaucoup plus nombreux au croisement, ce qui est représenté uniquement si l'on effectue une comparaison en fonction du temps et non en fonction du nombre d'appels à la fonction d'évaluation. Enfin, nous constatons que bien que les complexités des opérateurs SDPX et NDPX soient très élevées dans le pire cas, et en pratique surtout au lancement de l'algorithme, elle diminue fortement pendant l'exécution. De plus, des expérimentations complémentaires indiquent qu'en moyenne, à l'unité, chaque opération de croisement réalisée par les opérateurs SDPX et NDPX est plus rapide que par les opérateurs PBX et UPBX, bien que l'on puisse s'attendre à ce que les premiers, plus intelligents, soient plus lents : mise à part la traversée du chromosome, ils sont déterministes, contrairement à PBX et UPBX et nécessitent donc au total moins d'appels aux générateurs de nombres aléatoires dont l'exécution est coûteuse.

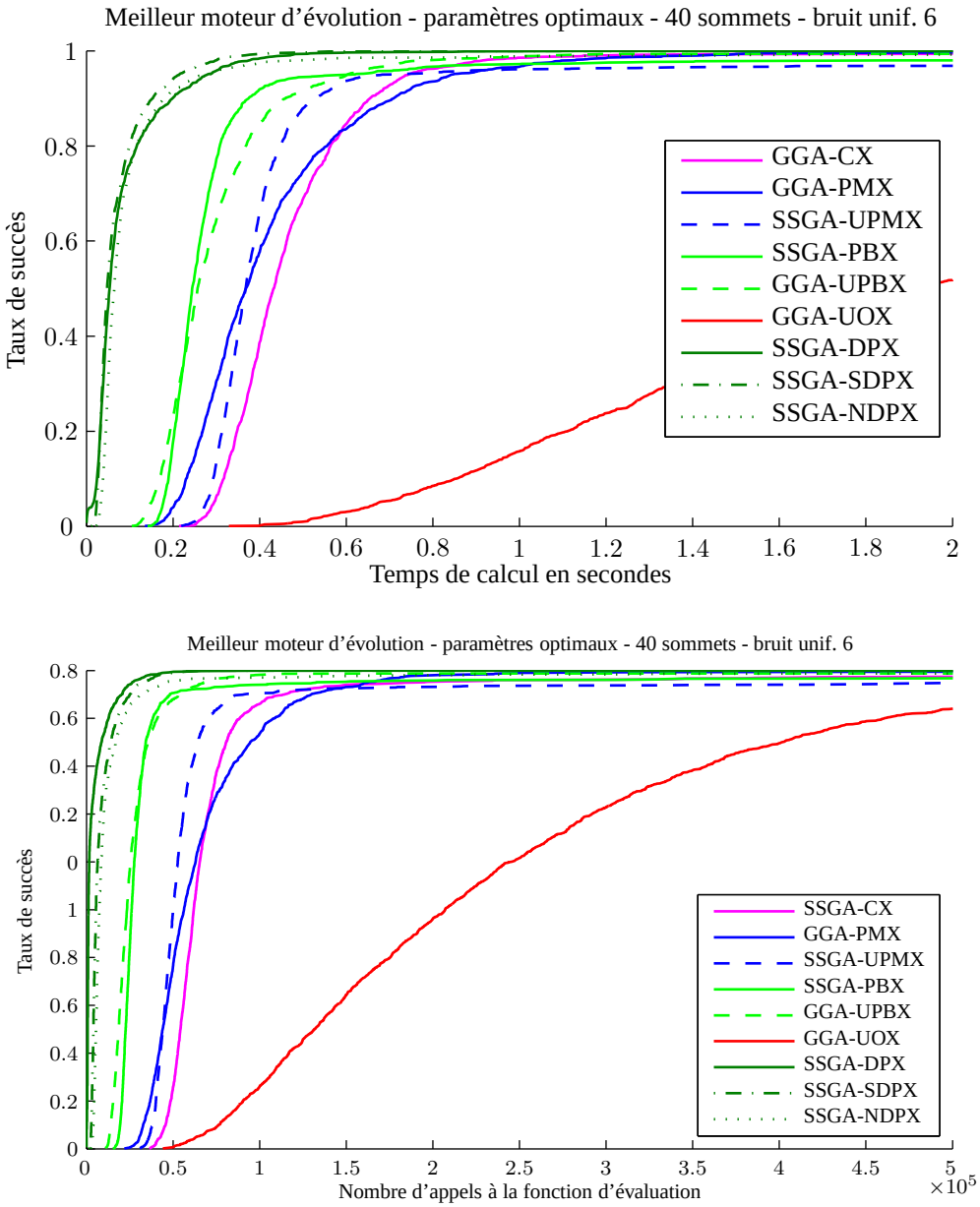

FIGURE 3.25 – Comparaison de tous les opérateurs avec le meilleur moteur. En haut en fonction du temps du calcul ; en bas en fonction du nombre d'appels à la fonction d'évaluation.

3.4 Étude des stratégies de sélection

Nous avons également étudié plusieurs stratégies de sélection, dans les expériences précédentes, nous avons choisi la sélection par tournoi. De fait, elle est parmi les stratégies le plus classiques comme le sont aussi la sélection proportionnelle à la *fitness* (*fitness proportionate selection* ou *roulette wheel*) et la sélection après classement (*ranking selection*). La figure 3.26 montre l'effet de plusieurs stratégies de sélection. La sélection stochastique universelle (*stochastic universal sampling*, SUS) est un développement de la sélection proportionnelle à la *fitness*. Nous constatons que le taux d'erreur augmente avec la taille des graphes. Une pression élevée favorise cet effet. Le choix de la sélection par tournoi est pertinent puisqu'elle montre les meilleurs résultats dans cette étude brève et en raison de ses avantages présentés par la suite.

Une comparaison plus approfondie des stratégies classiques est réalisée par Blickle & Thiele (1996). La sélection par tournoi présente plusieurs avantages : premièrement, elle est très facile et efficace à implémenter. Deuxièmement, elle n'est pas affectée par la dérive génétique qui décrit la diminution de la variance des valeurs de *fitness* pendant l'exécution d'un algorithme génétique (Rogers & Prügel-Bennett, 1999) : la population devient plus uniforme quand elle se déplace vers les meilleures solutions. Pour les stratégies utilisant la distribution des valeurs de *fitness*, on applique souvent du *fitness scaling*. Sinon, la pression de la sélection diminue avec le temps, ce qui est l'inverse du comportement souhaité. Troisièmement, elle ne nécessite pas la connaissance de la *fitness* de tous les individus mais seulement celles des participants du tournoi. En particulier, le tri parfois obligatoire pour d'autres stratégies n'est pas nécessaire. Ceci la rend plus facilement parallélisable. Enfin, la pression de la sélection est ajustable avec la taille du tournoi (une taille plus grande mène à une pression plus élevée).

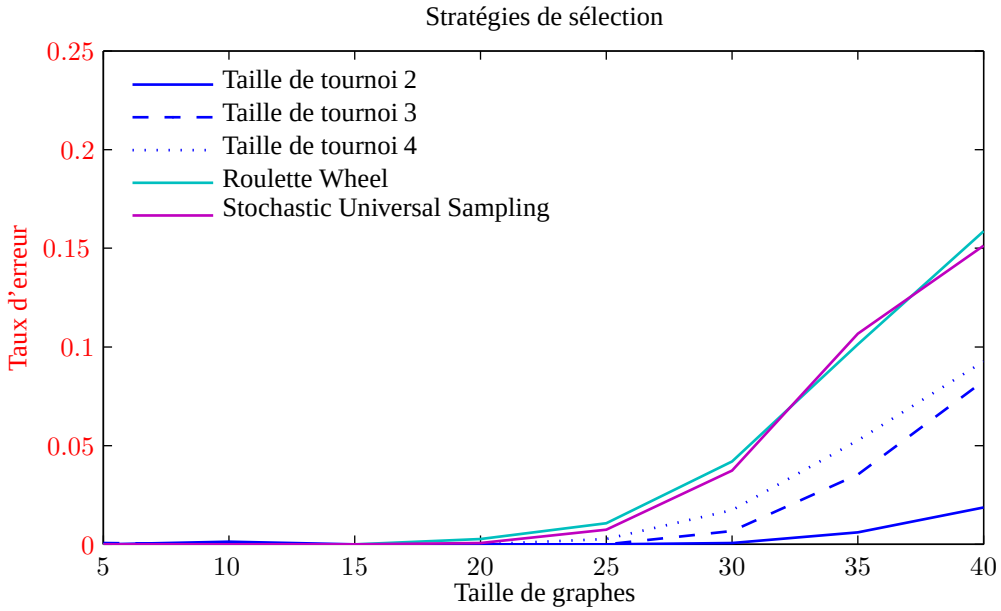

FIGURE 3.26 – Stratégies de sélection.

3.5 Étude des stratégies de mutation

Dans cette section, nous abordons la question de savoir si l'on peut obtenir de meilleures performances en remplaçant la mutation d'échange par un autre type de mutation. Comme nous avons pu constater que la plupart des opérateurs de croisement nécessitent une forte probabilité de mutation, nous cherchons en particulier une mutation plus forte.

3.5.1 Principe

La mutation *scramble* (cf. Sec. 2.6.2, p. 52) est une généralisation plus perturbatrice de la mutation d'échange : elle brasse un sous-ensemble de gènes du chromosome pouvant aller de 2 gènes (dans ce cas, on a la mutation d'échange) à la totalité (dans ce cas, un nouveau chromosome aléatoire est construit).

Cette approche est prometteuse surtout en combinaison avec des opérateurs de croisement dont la probabilité de mutation optimale est élevée. L'idée est que la nécessité d'une probabilité de mutation élevée signifie que l'opérateur en question risque de tomber dans des optima locaux sans s'en sortir par la suite (car il n'est pas assez destructif). Dans ce cas, une mutation plus destructive crée des individus plus éloignés des parents, et par conséquent de l'optimum local. L'inconvénient est qu'avec cette stratégie, la convergence peut être ralentie. Si par exemple nous trouvons dans la population un individu dont seulement deux gènes sont *faux*, alors avec la mutation d'échange, nous pouvons tomber directement sur la bonne solution si ces deux positions sont choisies. Avec la mutation *scramble*, il y a moins de chance que ces deux positions soient échangées.

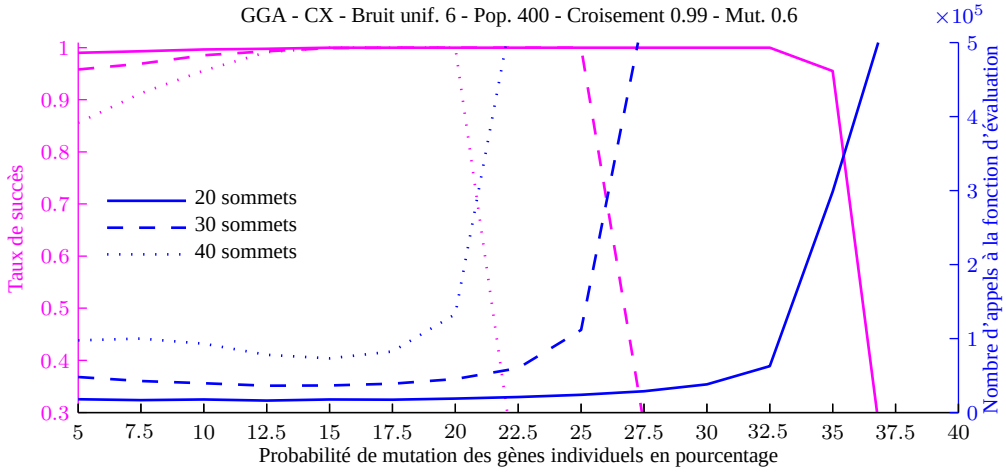

FIGURE 3.27 – Performance de la mutation *scramble* en fonction de son paramètre pour CX.

3.5.2 Expérimentation

Paramètre de la mutation *scramble* Dans un premier temps, nous étudions la valeur du paramètre de la mutation *scramble* qui détermine la taille de la sous-chaîne à brasser. Il y a deux façons de définir ce paramètre : d'une part comme une probabilité individuelle qui s'applique à chaque gène et détermine si celui-ci participe au brassage ; d'autre part, comme le nombre de gènes qui sont brassés. La différence entre ces deux définitions est que, dans la première, le nombre de gènes échangés est variable avec une espérance fixe, tandis que, dans la deuxième, il a chaque fois exactement la même valeur. Nous étudions empiriquement quelle définition est la plus appropriée dans notre cas.

En particulier, nous examinons le paramètre pour trois tailles de graphes différentes : 20, 30 et 40 ; pour chaque taille nous utilisons 50 paires de graphes et nous effectuons 30 itérations par chacune de celles-ci. La figure 3.27 montre le taux de succès et le nombre d'appels à la fonction d'évaluation pour le croisement CX en combinaison avec un moteur générationnel (les diagrammes associés aux autres opérateurs et au moteur *steady-state* se trouvent dans l'annexe C). Nous constatons que l'optimum se trouve autour de 15% de la longueur du gène pour les trois tailles de graphes.

La figure 3.28 montre le taux de succès par rapport au nombre d'appels à la fonction d'évaluations pour des valeurs proches de cet optimum, c'est-à-dire entre 12,5% et 20%, ainsi que les résultats obtenus avec la mutation d'échange, pour les graphes de 40 sommets. Nous utilisons la surface sous les courbes respectives pour chaque paramètre (en utilisant l'intégrale sur l'intervalle $[0; 200000]$) comme mesure de qualité à optimiser. Nous constatons que, d'après cette mesure, les valeurs optimales précises sont de 12,5% pour la taille 20 et de 15% pour les tailles 30 et 40. Ces valeurs correspondent à une espérance de la longueur échangée de 2,5, 4,5 et 6, respectivement.

Nous constatons une baisse sensible de la performance à partir d'un seuil diminuant avec la taille de graphes ($32,5\% \equiv 6,5$ pour 20, $25\% \equiv 7,5$ pour 30 et $20\% \equiv 8$ pour 40). En ce qui concerne ces seuils, ils semblent plus dépendants de la représentation comme longueur que celle comme

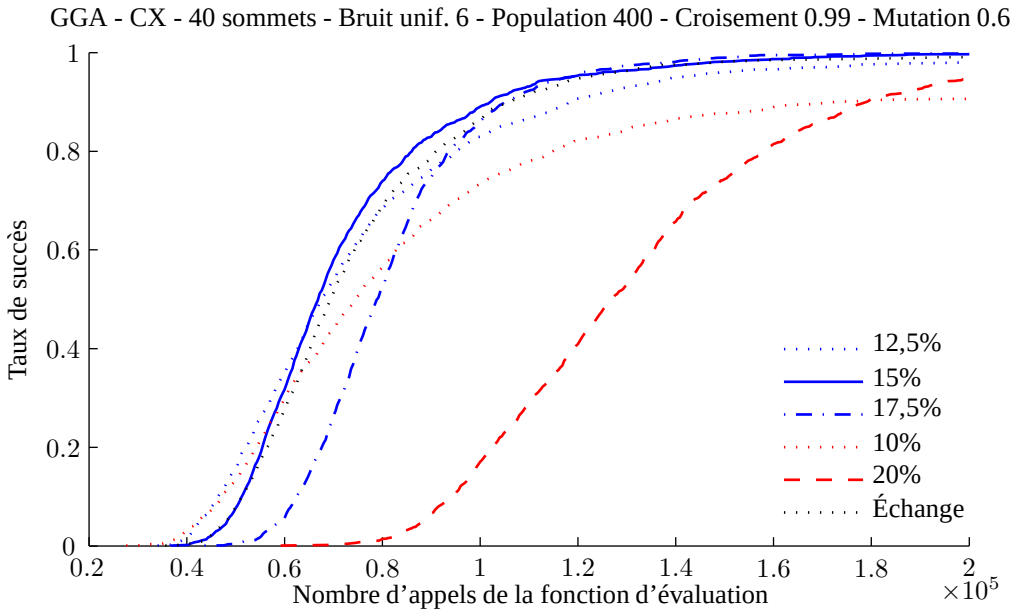

FIGURE 3.28 – Comparaison de la mutation *scramble* avec la mutation d'échange pour CX.

probabilité. On ne peut pas les exprimer avec une valeur fixe pour toutes les tailles indépendamment de la représentation du paramètre choisi. On peut conclure que les valeurs optimales sont mieux représentées en utilisant une probabilité plutôt qu'une longueur.

Comparaison des mutations *scramble* et d'échange Une fois que nous avons estimé la valeur du paramètre, nous estimons le gain maximal que l'utilisation de la mutation *scramble* peut obtenir par rapport à la mutation d'échange. Par exemple, la figure 3.28 montre les résultats de la mutation d'échange comme référence (la courbe noire pointillée) et ceux de la mutation *scramble* avec des différents paramètres proches de l'optimum (en bleu et rouge selon la distance). Nous observons une légère amélioration en utilisant la nouvelle mutation, mais uniquement avec le paramètre optimal.

Afin de vérifier si les résultats sont statistiquement significatifs, nous avons effectué le test de Kruskal-Wallis entre les résultats obtenus avec les deux types de mutation respectifs. Les p-valeurs sur la distribution du nombre d'appels à la fonction d'évaluation sont inférieures à 0,01 dans tous les cas. En ce qui concerne les p-valeurs calculées pour le taux de succès, nous constatons un seul cas où il dépasse 0,01 : pour un paramètre de la mutation *scramble* de 12,5% nous obtenons $p_{SR} = 0,18$. Pour une différence réelle de résultats, en général, il suffit qu'une des deux distributions, celle du taux de succès ou celle du nombre d'appels à la fonction d'évaluation, soit significativement différente.

L'avantage de la mutation *scramble* est donc très faible en combinaison avec CX (croisement qui nécessite la plus haute probabilité de mutation dans un moteur générationnel). Pour les autres opérateurs, nous constatons parfois que la mutation d'échange fonctionne mieux, par exemple dans

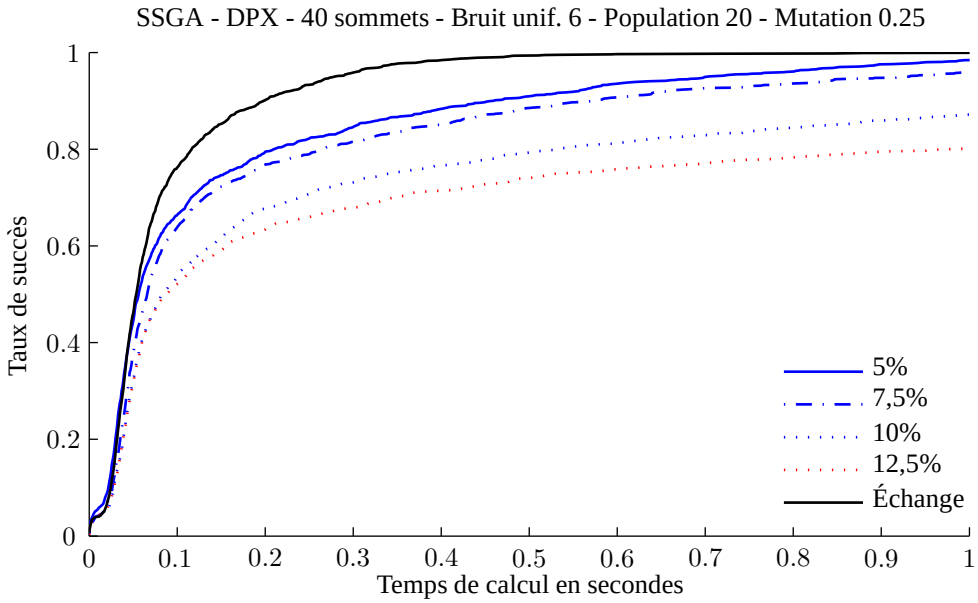

FIGURE 3.29 – Comparaison de la mutation *scramble* avec la mutation d'échange pour DPX.

le cas du croisement DPX (cf. Fig. 3.29, p. 105).

Dans ce cas nous avons également vérifié, avec le test de Kruskal-Wallis comme pour CX, si les résultats obtenus sont statistiquement significatifs. Nous constatons un cas où la p-valeur est supérieure à 0,01 : pour un paramètre de la mutation *scramble* de 5%, nous obtenons $p_{ARS} = 0,064$, la p-valeur basée sur le taux de succès étant non définie car on constate que la bonne solution a toujours été trouvée (la variance de la distribution est donc 0).

La figure 3.30 montre l'effet du paramètre pour les différentes tailles de graphes. Comme pour CX nous constatons que les probabilités optimales restent plus ou moins constantes tandis que les seuils des mauvaises performances sont en corrélation avec la taille. Cette corrélation se rapproche des espérances de longueur fixe, sans y arriver parfaitement.

Nous évaluons par la suite le gain maximal que l'on peut obtenir avec la mutation *scramble* pour chaque jeu de paramètres. Le tableau 3.4 montre les résultats détaillés. Nous constatons en général un gain très faible, voire une perte pour les opérateurs classiques, à l'exception de PMX dans la version SSGA et d'UOX. En fait, ces dernières ont des courbes décalées vers la droite par rapport aux autres opérateurs et donc des résultats médiocres en combinaison avec la mutation d'échange. En particulier, ils commencent à peine à fournir quelques résultats avant que l'on arrête le calcul de l'intégrale (200000 appels à la fonction d'évaluation, indépendamment de l'opérateur). Les hautes valeurs pour les gains respectifs de ces deux opérateurs sont donc dues à la limite définie pour le calcul de la surface.

Nous observons que les croisements *DPX subissent tous de la substitution de la mutation d'échange par la mutation *scramble* et leur paramètre optimal est de 7,5%, ce qui est équivalent

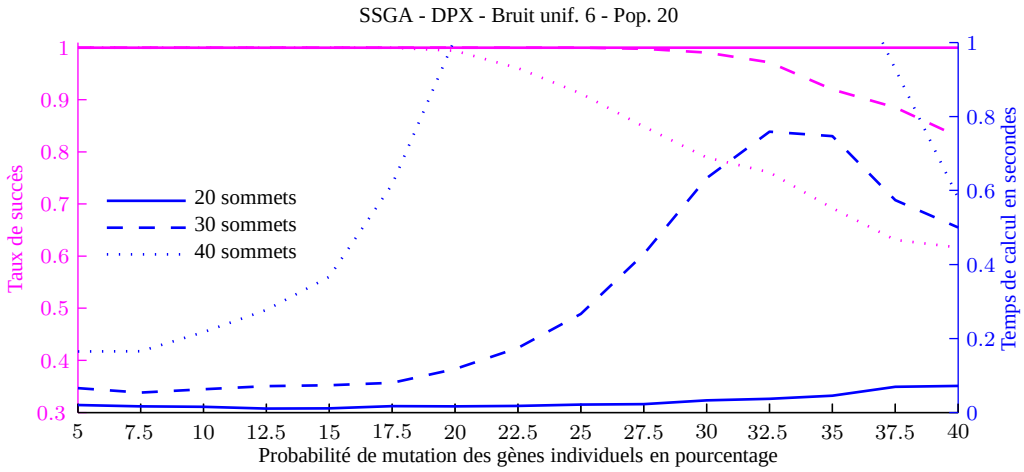

FIGURE 3.30 – Performance de la mutation *scramble* en fonction de son paramètre pour DPX.

à une longueur de trois gènes. Si l'on considère toutes les permutations de trois éléments d'après la loi uniforme, alors la probabilité que tous les trois changent de position n'est que de $2/6$. De plus, la probabilité que l'on échange exactement deux éléments (ce qui correspond à la mutation d'échange) est $1/2$. La mutation n'effectue aucun changement au chromosome avec une probabilité de $1/6$. Nous pouvons alors conclure que la mutation *scramble*, avec 7,5% comme paramètre dans ce cas, ressemble plus ou moins à la mutation d'échange. Les pertes constatées viennent en grand partie de la complexité de l'opérateur. Pour la mutation d'échange, on fait appel au générateur aléatoire deux fois par chromosome tandis que, pour la mutation *scramble*, on en a besoin pour chaque gène. Par conséquent les courbes se déplacent vers la droite et la surface devient plus petite. Les solutions optimales au problème sont trouvées plus tard. La mutation *scramble* détériore donc les résultats.

Opérateur	GGA		SSGA		SGGA	
	Gain	Paramètre	Gain	Paramètre	Gain	Paramètre
CX	2,1%	15%	2,0%	7,5%	0,4%	12,5%
PMX	1,5%	10%	98,1%	5%	11,0%	17,5%
UPMX	3,6%	5%	0,1%	12,5%	15,7%	35%
PBX	-0,8%	7,5%	0,1%	7,5%	-0,1%	10%
UPBX	-1,6%	7,5%	-0,3%	15%	0,4%	10%
UOX	-80,9%	7,5%	8,9%	5%	236,7%	40%
DPX	-3,6%	7,5%	-3,9%	7,5%	0,1%	7,5%
SDPX	-5,6%	7,5%	-3,3%	7,5%	7,6%	40%
NDPX	-7,2%	10%	-3,4%	5%	14,4%	30%

Tableau 3.4 – Gain maximal de la mutation *scramble* par rapport à la mutation d'échange, mesuré par la surface sous l'histogramme. Pour les premiers opérateurs en fonction du nombre d'appels à la fonction d'évaluation (maximum 200000), pour les *DPX en fonction du temps de calcul (maximum 2 secondes).

3.6 Bilan

Dans ce chapitre, nous avons étudié le paramétrage de l'algorithme génétique. Ceci inclut premièrement le choix d'un moteur d'évolution. Afin de limiter la recherche nous avons comparé uniquement les deux moteurs classiques : le moteur générationnel et *steady-state*. Deuxièmement, il faut choisir des opérateurs appropriés. Dans ce cas, nous nous sommes concentrés sur les croisements et les mutations en particulier. Pour ce qui concerne la sélection, nous avons choisi la sélection par tournoi. Elle est très performante et très facile à appliquer dans de nombreux contextes. De plus, elle fournit une pression constante. Troisièmement, on a besoin de choisir des opérateurs appropriés ainsi que leur jeu de paramètres correspondant. Ce dernier comprend en particulier la taille de la population, les probabilités de croisement et mutation ainsi que des paramètres propres aux opérateurs. Nous avons effectué une étude expérimentale riche en prenant en compte toutes les combinaisons de stratégies avec les opérateurs de croisement.

Nous constatons qu'avec un jeu de paramètres bien optimisé, on obtient de bons résultats avec la plupart des opérateurs. La question du meilleur moteur dépend aussi de l'opérateur. Les opérateurs proposés et en particulier les opérateurs spécialisés au problème d'isomorphisme de graphes fonctionnent mieux que les opérateurs classiques de permutation. Par contre, l'opérateur SDPX n'est applicable qu'au problème d'isomorphisme de graphe et non de sous-graphes, comme les autres. Quant à la mutation, la mutation d'échange nous semble favorable à la mutation *scramble*. Toutefois, il serait intéressant de voir si un algorithme peut profiter d'une combinaison des mutations plus ou moins perturbatrices. La mutation *scramble* pourrait être utilisée au début afin d'assurer

une bonne exploration, mais aussi vers la fin, dans un environnement adaptatif, si la diversité de la population a trop diminuée.

Chapitre 4

Recherche locale

La recherche locale (*local search*, LS) est, contrairement aux algorithmes évolutionnaires (EA), une méta-heuristique qui ne propose pas une nouvelle solution mais améliore une solution déjà existante. D'une part, elle est capable de trouver des optima locaux plus efficacement que les EA, d'autre part, elle explore moins bien l'espace de recherche que les EA et risque notamment de converger dans un optimum local qui n'est pas global. La combinaison de ces deux méthodes profite en même temps des qualités de l'exploration d'un EA et de l'exploitation de la recherche locale (Renders & Flasse, 1996). L'idée est qu'un individu, après avoir été créé par les opérateurs génétiques, est amélioré par la recherche locale. L'analogie avec la nature est simple : les nouveau-nés ne possèdent pas toutes les capacités nécessaires pour survivre ; une grande partie est acquise après la naissance, comme par exemple la marche. Sans cet apprentissage, même avec les meilleurs gènes possibles, un individu n'atteindra pas la même *fitness* qu'un individu qui l'a suivi. Les EA qui intègrent une stratégie de recherche locale ou plus généralement une stratégie d'apprentissage individuel sont dénommés algorithmes mémétiques (Moscato, 1989; Krasnogor & Smith, 2005; Ong et al., 2006). Il existe d'autres noms comme par exemple recherche locale génétique (*genetic local search*, *GLS*, cf. Ishibuchi & Murata, 1998), ou algorithmes culturels (*cultural algorithms*, cf. Reynolds, 1994).

La stratégie de recherche locale d'un algorithme mémétique consiste à remplacer chaque descendant par le chromosome dominant son voisinage [6]. Le voisinage d'un individu s est défini comme l'ensemble des individus atteignables à partir de s par l'application d'une opération élémentaire $b : s \to b(s)$. Il s'agit d'un sous-ensemble de l'espace génotypique $\mathcal{G}$ et il est noté $N(s) \subset \mathcal{G}$. Nous allons discuter le choix d'une opération b pour le problème d'isomorphisme de graphes ainsi que sa complexité dans la section 4.1.

Comme pour les EA en général, les résultats théoriques existants ne permettent pas de conclure sur la performance des algorithmes mémétiques sur des problèmes d'optimisation combinatoire. Merz (2000, 2004) analyse différents paysages de la fonction d'évaluation (*fitness landscape*). Il

6. Il existe également des approches qui mettent à jour uniquement la *fitness* et non le chromosome.

existe plusieurs caractéristiques décrivant la forme de la fonction d'évaluation qui influencent la performance de l'algorithme. En général, les caractéristiques locales influencent particulièrement la recherche locale tandis que les caractéristiques globales influencent plutôt l'exploration par l'EA. Les problèmes qui ont été étudiés incluent le problème d'affectation quadratique (*quadratic assignment problem*, QAP, Merz & Freisleben, 1997), la bipartition de graphes avec des opérateurs gloutons (Merz & Freisleben, 2000), ou le problème du voyageur de commerce (Merz & Freisleben, 2001).

Ishibuchi et al. (2003) soulèvent la question de l'équilibre entre la recherche génétique et la recherche locale. Ils se placent dans le cadre très spécifique de l'optimisation multi-objectif pour un problème d'ordonnancement. Ils s'intéressent à la répartition du temps de calcul entre les deux parties de l'algorithme. De fait, la recherche locale améliore la convergence de l'algorithme mais ceci a un coût en temps de calcul généralement élevé. Ce coût est dépendant de la méthode choisie.

Ils s'interrogent sur plusieurs questions, dont le nombre d'appels à la recherche locale pendant chaque génération ou le choix des individus auxquels elle est appliquée. En ce qui concerne leur problème (l'optimisation multi-objectif d'un ordonnancement), Ishibuchi et al. concluent par exemple que la recherche locale doit être appliquée aux meilleurs individus. Il est possible de s'interroger également sur la durée d'exécution de la recherche locale, c'est-à-dire le nombre de pas qui sont faits.

Ici, nous proposons de quantifier la complexité d'une stratégie de recherche locale adaptée au problème d'isomorphisme inexact de graphes (cf. Sec. 4.1, p. 110) et nous réalisons une étude sur le choix des individus (cf. Sec. 4.2, p. 113). Nous ne considérons pas le cas de plusieurs itérations de la recherche locale sur un seul individu (pendant une seule génération).

4.1 Complexité

Nous utilisons l'heuristique *two-opt* comme stratégie de recherche locale. L'opération b est l'échange de deux allèles. Le voisinage $N(s)$ d'un chromosome s contient donc tous les chromosomes qui peuvent être construits par l'échange de deux allèles. On cherche ensuite dans tout l'espace $N(s)$ l'individu optimal s' et on remplace s par s' [7]. L'évaluation d'un tel individu peut être restreinte au changement de distance que l'échange des allèles implique, au lieu de recalculer toute la fonction d'évaluation : soit v_1 et v_2 deux nœuds du graphe G et v'_1 et v'_2 les nœuds associés du graphe G', qui vérifient : $v'_1 = m(v_1)$ et $v'_2 = m(v_2)$. L'appariement $m' = b(m)$ est construit de telle façon que v'_1 et v'_2 soient échangés (suivant l'application d'un opérateur d'échange *two-opt* : $m'(v_1) = m(v_2)$ et $m'(v_2) = m(v_1)$) et que toutes les autres affectations élémentaires restent inchangées. Parmi tous les échanges possibles, on choisit celui qui mène à la plus forte réduction de distance. Soit $\delta_m(G, G')$ la distance entre les graphes G et G' sous l'appariement m et de même

7. Dans les cas qui mettent à jour la *fitness* uniquement, on associe la *fitness* de s' à s.

$\delta_{m'}(G, G')$ pour l'appariement m', on cherche l'opération b qui minimise la fonction suivante :

$$g = \delta_{m'=b(m)}(G, G') - \delta_m(G, G') \tag{4.1}$$

Soit δ_V la mesure de distance entre deux sommets et δ_E celle entre deux arêtes. Si v_1 et v_2 sont des éléments de $V* = V \backslash \{v_1, v_2\}$, alors :

$$\delta_V(v_1, m(v_1)) = \delta_V(v_1, m'(v_1)) \tag{4.2}$$

$$\delta_E\left((v_1, v_2), (m(v_1), m(v_2))\right) = \delta_E\left((v_1, v_2), (m'(v_1), m'(v_2))\right) \tag{4.3}$$

Cela signifie que l'évaluation d'un échange *two-opt* nécessite uniquement le calcul des distances de sommets et d'arêtes dont les affectations ont changé. Aussi la fonction g devient :

$$\begin{aligned} g = \quad & \delta_V(v_1, m'(v_1)) - \delta_V(v_1, m(v_1)) \\ + \quad & \delta_V(v_2, m'(v_2)) - \delta_V(v_2, m(v_2)) \\ + \quad & \sum_{v_1 \in V, v_2 \in \{v_1, v_2\}} \delta_E((v_1, v_2), (m(v_1), m'(v_2)) \\ - \quad & \delta_E((v_1, v_2), (m(v_1), m(v_2)) \end{aligned} \tag{4.4}$$

En conséquence, l'évaluation d'un tel échange se fait en un temps linéaire contrairement à l'évaluation du chromosome entier qui nécessite un temps quadratique en fonction du nombre de sommets (cf. Sec. 4.1.1, p. 111). Aussi, l'algorithme *two-opt* est un opérateur de mutation *intelligente* effectuant la meilleure transposition.

Adaptation à l'approche par couleur Dans le cas des opérateurs par couleur (cf. Sec. 2.8, p. 54), un sommet peut être remplacé uniquement par un sommet de la même classe, réduisant la taille de l'espace de recherche sur les solutions faisables. Ceci s'applique uniquement aux applications dans le cadre desquelles une classification des sommets est disponible. Nous avons également modifié notre algorithme de recherche locale afin qu'il tienne compte de telles restrictions dans ce cas. Ceci réduit la complexité de la recherche locale car le nombre d'échanges à comparer est réduit.

4.1.1 Complexité théorique

Soit n la taille de graphes (et donc aussi la longueur des chromosomes). Chaque appel à la fonction d'évaluation nécessite l'addition de toutes les distances de sommets (il y a donc n distances à calculer) ainsi que toutes celles d'arêtes (C_n^2). Par conséquent, la complexité d'un appel à la fonction d'évaluation est $\mathcal{O}\left(n + C_n^2\right) = n + \frac{n(n-1)}{2}$. A chaque génération, on calcule la fonction d'évaluation une fois par individu : le nombre d'appels à la fonction d'évaluation est égal à la taille de la

population (hors d'éventuels chromosomes élite que l'on garde en mémoire sans les soumettre aux opérateurs génétiques). Soit s_p la taille de la population hors individus élite, alors la complexité totale des appels de la fonction d'évaluation pendant une génération est $s_p \times (n + \frac{n(n-1)}{2})$.

Pour la phase d'échanges *two-opt*, on a besoin de vérifier toutes les possibilités de choisir deux sommets parmi tous les sommets du graphe, ce qui représente $C_n^2 = \frac{n(n-1)}{2}$ échanges possibles par chromosome. Chacun de ces échanges nécessite le calcul de $\mathcal{O}\left(4 + 4(n-2)\right)$ distances élémentaires. On calcule en effet les distances des deux sommets échangés sous l'appariement avant et après l'échange (4) puis on calcule les distances entre toutes les arêtes incidentes à chacun des sommets également avant et après l'échange. En effet, l'arête entre les sommets reste constante [8], ce qui mène à la valeur de $4(n-2)$. Si l'on applique la recherche locale à tous les chromosomes, alors la complexité est $\mathcal{O}\left(s_p \times \frac{n(n-1)}{2} \times (4 + 4(n-2))\right)$.

Le quotient entre la complexité de la recherche locale et le coût des appels à la fonction d'évaluation est de l'ordre de $\mathcal{O}(n)$, donc la recherche locale est de l'ordre $\mathcal{O}(n)$ fois plus coûteuse que les appels à la fonction d'évaluation (quand la recherche locale est effectuée pour chaque chromosome), comme démontré ci-dessous.

Démonstration. Soit s_p la taille de la population et n la longueur des chromosomes.

$$\begin{aligned}
\mathcal{O}\left(\frac{\text{recherche locale}}{\text{fitness}}\right) &= \frac{s_p \times C_n^2 \times (4 + 4 \times (n-2))}{s_p \times (n + C_n^2)} \\
&= \frac{\frac{n\times(n-1)}{2} \times (4 + 4 \times (n-2))}{n + \frac{n\times(n-1)}{2}} \\
&= \frac{4 \not{n}(n-1)(1+n-2)}{2 \not{n} + \not{n}(n-1)} \\
&= \frac{4(n-1)^2}{n+1} \\
&\approx \mathcal{O}(4n)
\end{aligned}$$

Toutefois, on peut encore réduire le complexité de deux façons : premièrement en précalculant toutes les distances entre sommets et arêtes affectés. Ceci réduira la complexité de la recherche locale pour un seul individu à

$$\mathcal{O}\left(C_n^2 \times (2 + 2 \times (n-2)) + \frac{1}{2}(n^2 + n)\right). \tag{4.5}$$

En effet, à l'intérieur de la recherche locale, on calcule désormais uniquement les affectations changées, tandis que les distances des affectations actuelles sont reprises du calcul précédent de la fonction d'évaluation. On obtient la moitié de la complexité précédente :

8. Dans un graphe orienté ce qui n'est pas le cas ici, elle change l'orientation.

$$\begin{aligned}
\mathcal{O}\left(\frac{\text{recherche locale}}{\text{fitness}}\right) &= \frac{s_p \times C_n^2 \times (2 + 2(n-2)) + \frac{1}{2}(n^2+n)}{s_p \times (n + C_n^2)} \\
&= \frac{2\ \not{n}(n-1)(n+1) + \not{(}n)(n+1)}{2\ \not{n} + \not{n}(n-1)} \\
&= 2(n-1) + 1 = 2n - 1 \\
&\approx \mathcal{O}(2n)
\end{aligned}$$

Deuxièmement, si l'on possède une classification des nœuds, il est possible de l'utiliser afin de réduire le nombre des échanges à vérifier. Étant donné une classification des nœuds en k classes dont chaque classe contient n_i nœuds ($\sum_{i=1}^{k} n_i = n$), on réduit le nombre d'échanges à évaluer : au lieu de C_n^2 on obtient $\sum_{i=1}^{k} C_{n_i}^2$. Ceci est beaucoup plus favorable pour la recherche locale. □

4.1.2 Complexité en pratique

Le quotient théorique entre la complexité de la recherche locale et la complexité d'un appel à la fonction d'évaluation et le quotient obtenu par expérimentations sont semblables, même s'il y a un léger avantage pour la recherche locale (LS) en pratique. Comment peut-on expliquer que la recherche locale soit plus rapide qu'elle ne devrait l'être en théorie ? Apparemment, la recherche locale exploite plus efficacement le processeur, la charge moyenne étant supérieure dans un environnement parallèle. Nous constatons une utilisation des processeurs à 100% avec la recherche locale et strictement inférieure à 100% sans. Il y a plusieurs pistes possibles, dont notamment une meilleure utilisation du cache, car un seul individu est traité répétitivement au lieu de changer la solution à chaque fois, ou encore le taux de remplissage de la pipeline, qui doit être supérieur grâce à des opérations ultérieures qui sont plus faciles à prédire. De toute façon, les différences sont dues aux effets techniques liés au matériel, ce qui n'est pas le sujet de cette thèse.

4.2 Stratégies de sélection pour la recherche locale

L'importance de l'équilibre entre la recherche locale et la fonction d'évaluation a déjà été soulignée dans l'introduction de ce chapitre. Afin d'obtenir une estimation du rapport du coût entre l'EA et la recherche locale, nous avons mis en rapport la complexité des appels à la fonction d'évaluation avec celle de la recherche locale. En fait, le calcul de la fonction d'évaluation est la partie la plus coûteuse de l'EA et elle est exécutée une fois pour chaque individu. Quant à la recherche locale (pour laquelle nous avons considéré jusqu'ici qu'elle est exécutée pour chaque individu), elle figure uniquement dans les algorithmes mémétiques et est encore plus coûteuse. Par conséquent, si l'on veut équilibrer le temps de calcul consacré aux deux tâches, il faut réduire le nombre d'appels à la

procédure de LS. Nous étudierons par la suite les réductions possibles avec différentes stratégies de sélection.

4.2.1 Choix des individus

Afin d'aborder la question du choix des individus, nous expérimentons plusieurs stratégies de sélection des individus pour la recherche locale. Toutes nécessitent un paramètre supplémentaire, la probabilité de recherche locale, ainsi que certaines adaptations à l'algorithme génétique classique. Certaines mènent à un besoin en mémoire plus élevé, d'autres ralentissent légèrement l'exécution en raison d'étapes de calcul supplémentaires. Les stratégies sont les suivantes :

GGA Le GA générationnel (GA générationnel (GGA)) est la variante la plus simple : après avoir été créé, un individu est soumis à la recherche locale avec une probabilité fixe. La sélection des individus est alors uniforme et indépendante de leur qualité. S'il existe des copies dans la population, la procédure de recherche locale est appliquée séparément à chaque copie.

UGGA L'idée de l'GGA unique (UGGA) est d'éviter une exécution répétée de la recherche locale sur des individus identiques. Nous utilisons alors une table de hachage permettant de vérifier efficacement la présence d'individus identiques. Quand on identifie un duplicata, il y a deux possibilités : d'une part, on peut copier le résultat de la recherche locale déjà effectuée sur son jumeau. De cette façon, on influence la sélection en donnant à la population un biais fort vers ce chromosome. La pression sélective peut donc devenir énorme surtout en combinaison avec une probabilité élevée. D'autre part, on peut seulement ignorer toute copie en appliquant la recherche locale uniquement aux individus uniques et aux premiers jumeaux. De cette manière, la décision de savoir si l'amélioration apportée au premier individu est maintenue ou non est laissée à la sélection naturelle. Nous utilisons la deuxième stratégie, car la pression de sélection est déjà très forte. Les individus ignorés induisent néanmoins un biais sur l'exactitude de la probabilité de recherche locale. Si ces individus dupliqués sont nombreux, la probabilité observée peut devenir en effet plus petite que la probabilité choisie (car des individus initialement choisis pour la LS sont potentiellement rejetés par la vérification d'unicité). Nous adaptons alors cette stratégie en utilisant la probabilité comme une proportion du nombre d'individus et en appliquant la recherche locale à exactement ce quota de la population. Par conséquent, la probabilité est généralement respectée. Toutefois, dans le cas où le nombre de copies dans la population dépasse le nombre d'individus sur lesquels la recherche locale n'est pas appliquée, elle est réduite. La sélection des individus reste indépendante de leur *fitness*.

SGGA Contrairement aux deux stratégies précédentes, le but du GGA trié (SGGA) est de concentrer la recherche locale sur les meilleurs individus uniquement. Comme les algorithmes génétiques sont des méthodes bien adaptées pour l'exploration globale avec un léger déficit dans la convergence

vers l'optimum, cette stratégie pourrait donner un avantage supplémentaire. Les meilleurs individus méritent une recherche plus approfondie autour d'eux tandis que les autres continuent à explorer les autres régions de l'espace de recherche. Notre algorithme est basé sur une population triée selon la *fitness*, ce qui impose une légère augmentation du temps de calcul. Comme pour UGGA la probabilité de recherche locale est alors définie comme une proportion sur la taille de la population.

USGGA L'GGA unique et trié (USGGA) combine les deux variantes précédentes. Nous utilisons une population triée et appliquons la recherche locale aux meilleurs individus sans prendre compte leurs copies. Contrairement au UGGA et grâce au triage de la population, nous n'avons pas besoin de table de hachage. Pour l'identification des copies, il suffit de comparer l'individu actuel avec son précédent dans la population triée.

4.3 Expérimentation

Nous comparons expérimentalement les quatre stratégies présentées précédemment. Pour cela, il faut d'abord étudier l'influence des différentes valeurs de la probabilité de recherche locale sur les différents opérateurs de croisement avec lesquels nous combinons la recherche locale. Pour cela, nous choisissons une taille de graphe de 60 car pour des tailles plus petites, l'algorithme génétique sans recherche locale a déjà une précision quasi parfaite. Comme la recherche locale est assez coûteuse, elle n'améliore pas le temps de calcul pour les graphes plus petits (en particulier de taille 40 comme auparavant). Cela ne serait donc pas avantageux. À la taille 60, un GGA avec l'opérateur de croisement UPBX atteint une précision d'environ 67% contre 78% pour un GA steady-state (SSGA). Ces valeurs laissent de la place pour des améliorations sans être trop faibles.

Nous examinons le SSGA séparément des autres versions. D'une part, sa base, un EA *steady-state* au lieu d'un EA générationnel, est différente. D'autre part, comme on n'a qu'un seul individu à la fois, avec une population triée et qu'elle ne contient pas de copies par définition, la recherche locale s'intègre simplement après la création d'un nouvel individu avec la probabilité donnée.

Nous comparons les différentes stratégies avec des paramètres variant de 1% à 100% pour l'application de recherche locale sur les individus. Nous limitons le temps de calcul à travers un nombre maximal d'appels à la fonction d'évaluation. En effet, pour la version sans recherche locale nous permettons un total d'un million d'appels à la fonction d'évaluation.

Pour une comparaison juste entre des algorithmes avec ou sans recherche locale (et entre algorithmes avec LS, mais avec des probabilités différentes), il faut poser les mêmes limites sur la durée d'exécution totale indépendamment de la répartition entre recherche génétique et recherche locale. Nous utilisons les résultats sur la complexité théorique de la LS afin de calculer une approximation du nombre d'appels à la fonction d'évaluation sans recherche locale. Cela correspond au même effort de calcul qu'un algorithme intégrant la LS. Soit f le nombre d'appels à la fonction d'évaluation,

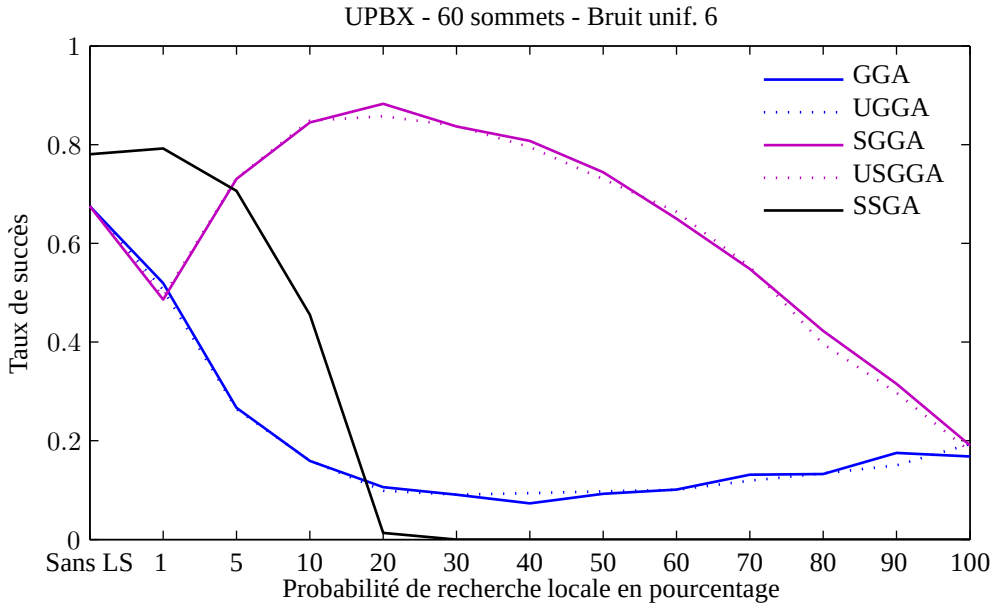

FIGURE 4.1 – Influence de la probabilité de recherche locale - croisement UPBX - par type de moteur d'évolution : GGA - générationnel ; UGGA - générationnel unique ; SGGA - générationnel trié ; USGGA - générationnel trié unique ; SSGA - *steady-state*

p_{RL} la probabilité de recherche locale et n la taille du graphe, le nombre d'équivalents d'appels est alors $f' = f(1 + 2p_{RL} \times n)$. Le nombre maximal des générations diminue alors naturellement avec la probabilité autant que le coût de la recherche locale l'impose. En bref, le nombre d'appels à la fonction d'évaluation diminue avec des probabilités de LS croissantes autant que le coût de la recherche locale augmente.

4.3.1 Croisement UPBX

La figure 4.1 illustre les résultats pour UPBX. Pour SSGA nous constatons que la précision diminue avec des paramètres élevés. A partir de 20%, elle baisse à près de 0. L'utilisation de la recherche locale n'est donc pas favorable. De même les versions UGGA et GGA n'en profitent pas, bien que l'on constate pour les probabilités très élevées (à partir de 40% - 50%) une légère régénération sur un niveau très faible. Quant aux versions triées, SGGA et USGGA, elles profitent de la recherche locale et atteignent des taux de succès de respectivement plus de 88% et 85%. L'amélioration est toutefois très dépendante de la probabilité choisie, avec un optimum autour de 20%. Il est intéressant de noter qu'une probabilité très faible, autour de 1%, fait perdre sensiblement de précision à l'algorithme. Quand on passe à 5%, cette perte est plus que compensée.

Moteur *steady-state* Les moteurs *steady-state* ne bénéficient quasiment pas de la recherche locale. Seulement avec un très petit taux, de 1%, nous observons une légère amélioration de la précision

(cf. Fig. 4.1, p. 116). Notre implémentation est basée sur une population sans copies dont le pire individu est remplacé par le nouveau descendant. Les autres individus sont affectés uniquement si leur *fitness* est inférieure à la *fitness* du nouvel individu. Dans ce cas, leur rang dans la population baisse d'une position. L'intégration répétée des bons individus mène à l'exclusion des individus moins bien placés, tandis que les meilleurs sont toujours maintenus.

La recherche locale a deux effets potentiels. D'une part, les descendants sont uniformisés : s'il y a une correspondance entre sommets spécifiques dont la distance est très petite, celle-ci est favorisée par la recherche locale et sera présente dans une grande partie des descendants, même si un optimum global se trouve avec une autre configuration. D'autre part, la recherche locale introduit dans la population des bonnes solutions avec très peu de différences aux individus déjà présents. La distribution des valeurs de *fitness* perd donc un peu de sa variance. De cette manière la pression de sélection, qui est déjà assez élevée pour ce type d'algorithme, est encore augmentée. Il est donc assez difficile pour un individu assez bon venant d'une autre région prometteuse de s'intégrer dans la population, même si près de lui se trouve une meilleure solution. La recherche devient alors plus locale et perd l'avantage de la bonne exploration des EA.

4.3.2 Croisement DPX

Le croisement DPX obtient déjà un taux de succès de 1 sans recherche locale. Une amélioration n'est plus possible. Nous nous intéressons donc à l'effet sur le temps de calcul. La figure 4.2 montre le temps de calcul en fonction de la probabilité de recherche locale. Nous constatons une baisse importante pour des probabilités assez faibles autour de 0,01 (pour les moteurs GGA et SSGA) ou 0,05 (pour les moteurs UGGA, SGGA, et USGGA). Pour des probabilités plus élevées, le temps de calcul augmente. Le gain est maximal pour les moteurs SGGA et USGGA.

4.3.3 Comportement des autres opérateurs de croisement

Tous les résultats présentés jusqu'ici ne sont applicables qu'avec l'opérateur UPBX et DPX. Les autres opérateurs montrent un comportement différent. UPMX dont la précision sans recherche locale est quasiment nulle pour les graphes de taille 60 reste encore très près de UPBX. Une probabilité de recherche locale faible, en combinaison avec les stratégies SGGA et USGGA, augmente le taux de succès jusqu'à 0,6. Contrairement au cas précédent, nous constatons une différence entre la version USGGA et SGGA. De meilleures performances sont obtenues avec la stratégie USGGA.

En ce qui concerne l'opérateur CX, l'ajout de la recherche n'est avantageux avec aucun des paramètres choisis. Donc, en général, il semble qu'une faible probabilité de recherche locale soit avantageuse, mais une vérification avec l'opérateur choisi peut être nécessaire.

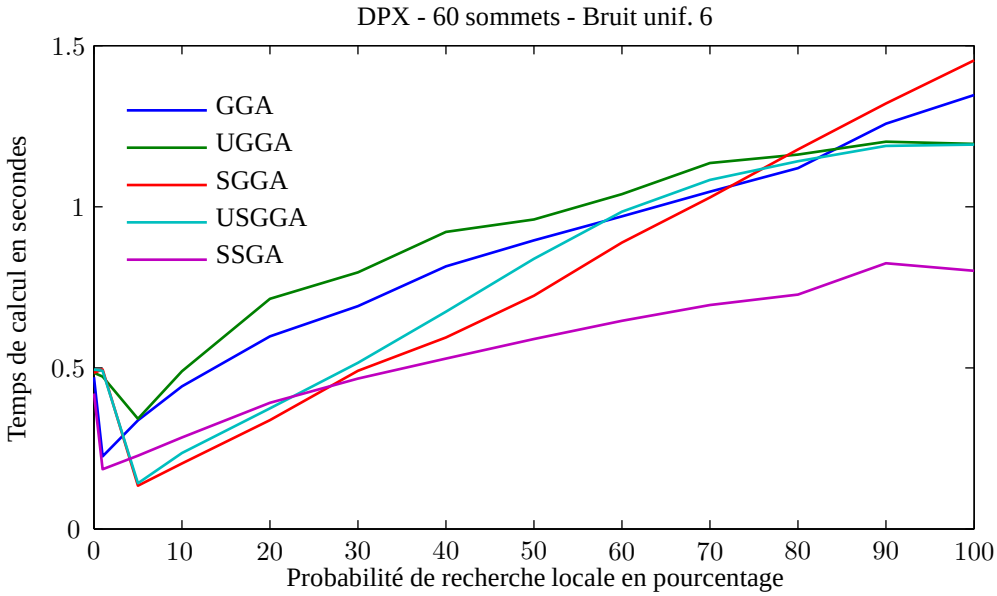

FIGURE 4.2 – Influence de la probabilité de recherche locale sur le temps de calcul - croisement DPX - par type de moteur d'évolution : GGA - générationnel ; UGGA - générationnel unique ; SGGA - générationnel trié ; USGGA - générationnel trié unique ; SSGA - *steady-state*

4.4 Bilan

Dans ce chapitre, nous avons étudié les effets de la recherche locale sur l'algorithme pour l'appariement évolutionnaire de graphes et constaté que les effets dépendent de l'opérateur de croisement. Nous avons observé que la recherche locale peut améliorer la précision. Si le taux de succès est déjà 1 alors une réduction du temps de calcul est possible. Toutefois, comme la complexité de la recherche locale par rapport à une évaluation de la fonction d'évaluation est très élevée, il est nécessaire d'utiliser une probabilité assez faible afin de garder un temps de calcul acceptable. Nous constatons qu'en pratique le rapport entre les temps d'exécution de la recherche locale et de la fonction d'évaluation est sensiblement moins important qu'il ne devrait l'être en théorie : la recherche locale est comparativement plus rapide qu'attendu. Il apparaît que les processeurs modernes utilisés effectuent efficacement des accès répétés à des zones proches en mémoire (tels que les accès nécessaires pour l'évaluation des petits changements dans la recherche locale), plutôt que des accès à des zones éloignées (lors du changement d'individu). Nous constatons en particulier une charge de processeur affichée plus proche de 100%.

Le résultat clef pour le design d'un algorithme mémétique est d'une part, qu'il faut choisir les meilleurs individus de la population pour la recherche locale. Parmi les différentes stratégies d'implémentation, les stratégies basées sur le tri de la population apportent une amélioration.

Chapitre 5

Modèles parallèles

Il existe plusieurs modèles de parallélisation pour les algorithmes évolutionnaires (Konfršt, 2004). Comme ils opèrent sur un ensemble de solutions dont les individus sont mis en concurrence, l'idée générale est de décomposer la population afin d'appliquer les opérations sur les différentes sous-populations en parallèle.

La seule opération qui ne permette pas de parallélisation sans changement des caractéristiques de l'algorithme génétique, est la sélection. Par contre, l'opération la plus importante est l'appel à la fonction d'évaluation qui a une complexité typiquement très élevée par rapport aux autres parties de l'algorithme. Le parallélisme global émerge de l'idée de préserver toutes les caractéristiques du GA et d'accélérer son exécution (Alba & Tomassini, 2002). Le schéma de sélection est centralisé tandis que l'évaluation des individus est faite en parallèle. Le gain apporté par la parallélisation des autres parties de l'algorithme, notamment les opérateurs génétiques, ne justifie en général pas l'effort comme l'affirment les mêmes auteurs.

Nous présentons d'abord une classification des modèles des algorithmes génétiques parallèles. Nous introduisons par la suite les limites théoriques de l'accélération du modèle maître-esclave. Dans la section 5.3, nous présentons les détails techniques de notre implémentation que nous utilisons par la suite afin d'étudier le gain pratique sur différents ordinateurs. Nous étudions notamment l'influence de la complexité du problème, à travers la taille de graphes et de population, ainsi que de différentes stratégies de communication entre les processus. Avant de conclure, nous introduisons deux possibilités de parallélisation au niveau applicatif.

5.1 Classification

Alba & Tomassini (2002) proposent une classification des différents modèles d'algorithmes génétiques parallèles. La parallélisation ne permet pas seulement l'accélération de l'algorithme mais, selon l'implémentation choisie, une variété d'avantages qui sont :

- Trouver plusieurs solutions au même problème

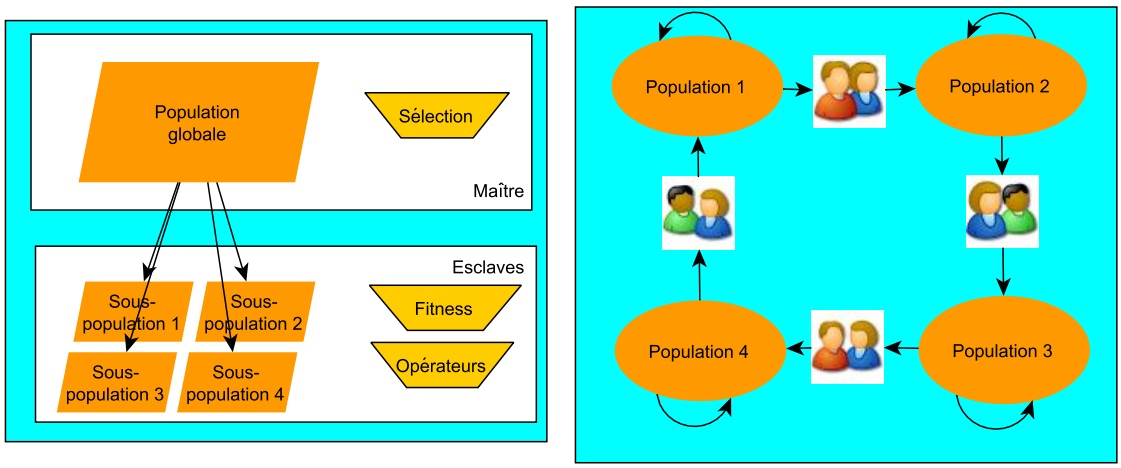

FIGURE 5.1 – Parallèlisme global et modèle en îlots.

- Recherche parallèle avec plusieurs points de départ dans l'espace de recherche
- Simple parallélisation comme îlots ou voisinages
- Recherche plus efficace même si l'on n'utilise pas de matériel parallèle
- Meilleure efficacité par rapport aux algorithmes séquentiels dans de nombreux cas
- Simple coopération avec d'autres stratégies de recherche
- Accélération du temps d'exécution

La classification des différentes approches se base d'une part sur la taille et le nombre des sous-populations et d'autre part sur leurs interactions. Les EA sont habituellement *panmictiques*, c'est-à-dire que les opérateurs sont appliqués à l'ensemble de la population, en particulier durant la sélection et le remplacement des individus où chaque individu peut s'accoupler avec n'importe quel autre.

On distingue ainsi des approches de granularité fine de celles d'une granularité élevée. L'interaction principale est la migration d'individus d'une sous-population vers une autre, basée sur des modèles de voisinage entre sous-populations. De même l'utilisation d'une décomposition hiérarchique est possible. De plus, les sous-populations peuvent être isolées les unes des autres, pendant la sélection et le croisement. En cas d'isolation, un individu ne peut être combiné qu'avec un autre individu de la même sous-population. Ainsi, l'algorithme n'est pas *panmictique*.

Dans la littérature, on trouve parfois des noms différents pour certains algorithmes, ainsi le modèle global est appelé maître-esclave dans Nowostawski & Poli (1999). Toutefois, le principe de la classification reste le même.

La figure 5.1 montre à gauche le principe du parallélisme global. Déjà l'initialisation peut être faite en parallèle. Puis, la fonction d'évaluation est calculée indépendamment pour chaque individu. La sélection globale nécessite le transfert des valeurs de *fitness* (et des chromosomes) au processeur maître. Une fois qu'il reçoit ces données, il détermine les parents pour la prochaine génération

et les envoie aux processeurs esclaves. Ceux-ci appliquent les opérateurs génétiques et calculent la fonction d'évaluation pour les nouvelles sous-populations. Finalement, les nouvelles valeurs de *fitness* sont envoyées au processeur maître et on répète la boucle jusqu'à atteindre le critère d'arrêt. Une variante est de paralléliser uniquement l'application de la fonction d'évaluation car il s'agit de l'opération la plus coûteuse.

Dans la même figure, à droite, l'idée générale du modèle en îlots (*island model*) est illustrée. Un îlot consiste en une sous-population isolée des autres, par la mer, et est associé à un processeur. Ainsi sur chaque îlot, un EA indépendant est exécuté. Périodiquement des individus traversent la mer afin d'arriver à l'îlot suivante. Le choix de ces individus peut se faire aléatoirement, ou selon la *fitness* ou même en fonction d'une notion d'âge. Le processus de changement de population est nommé migration. Cette modèle se comporte différemment d'un EA séquentiel avec la même taille de population globale (par exemple en ce qui concerne les caractéristiques suivantes : l'intensité de sélection, et le risque d'optima locaux).

5.2 Gain théorique

Le gain de temps maximal que l'on peut obtenir avec la parallélisation d'un algorithme génétique a été étudié par Cantú-Paz (2007, cf. aussi Cantú-Paz & Goldberg, 1999). L'accélération (*speedup*) parallèle est définie comme quotient entre le temps de calcul séquentiel et le temps parallèle $\left(\frac{t_s}{t_p}\right)$. Dans le cas idéal, la meilleure accélération possible est linéaire en nombre de processeurs, ce qui correspond à la division du temps de calcul par le nombre de processeurs utilisées. Il faut toutefois prendre en compte des temps de communication entre les processeurs. Si l'on considère le modèle global, le processeur maître (celui qui gère la sélection) communique les différentes sous-populations à traiter à chaque processeur esclave. Cette communication s'effectue en série car il ne peut communiquer avec les autres processeurs qu'un après l'autre.

Cantú-Paz (2007) suppose une distribution uniforme des individus sur les processeurs, c'est-à-dire que chaque processeur reçoit le même nombre d'individus. Cette hypothèse est en général raisonnable et rend le calcul plus facile. Elle présuppose que le temps de communication soit négligeable par rapport au temps d'évaluation de la fonction d'évaluation. Le temps pour l'application des opérateurs génétiques est ignoré, car considéré comme négligeable par rapport au calcul de la fonction d'évaluation.

Il faudrait revoir cette hypothèse si le temps de communication devenait élevé, comme par exemple si l'on souhaitait construire un algorithme distribué sur des ordinateurs distants ayant des latences plus longues. Ceci serait également nécessaire dans le cas rare où la quantité d'informations à transmettre est très lourde (selon la taille de la population et la longueur des chromosomes principalement) et le temps d'évaluation d'un individu est faible. On pourrait, dans ces cas encore, minimiser le temps d'inactivité tel que les premiers processeurs contactés reçoivent plus d'individus

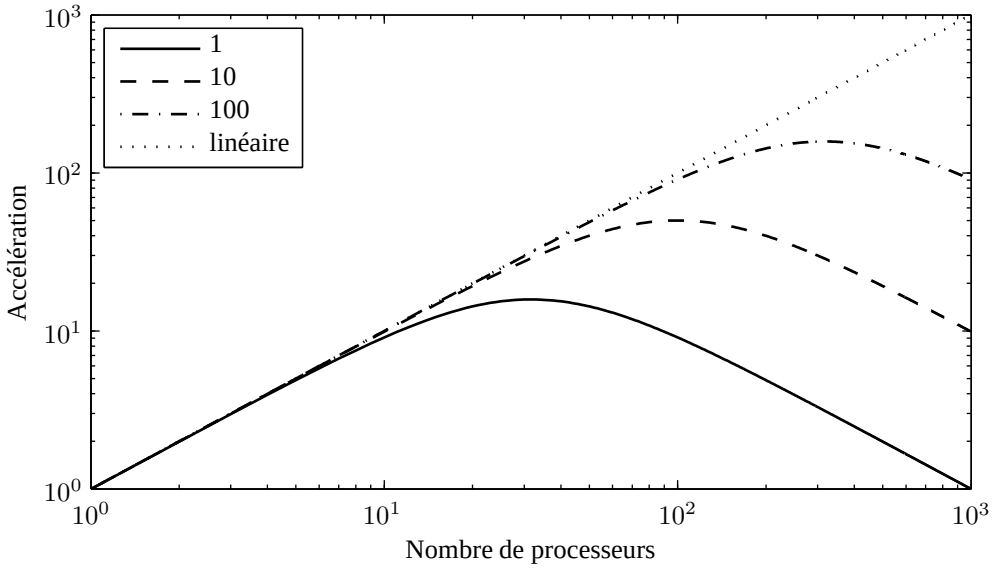

FIGURE 5.2 – Accélération théorique pour des différents rapports entre temps de calcul et temps de communication (Cantú-Paz & Goldberg, 1999; Cantú-Paz, 2007) en double échelle logarithmique. L'accélération linéaire est un cas idéalisé, uniquement atteignable avec un temps de communication de zéro.

que les derniers, dont notamment le processeur maître. Ce dernier ne peut commencer à travailler sur sa sous-population qu'après avoir terminé la communication avec le dernier processeur esclave.

Le premier résultat important est qu'il n'est pas toujours intéressant d'utiliser plus de processeurs. Le nombre de processeurs optimal peut être exprimé en fonction du rapport entre le temps de calcul de la fonction d'évaluation et le temps de communication. Quand on ajoute des processeurs, le temps de calcul diminue mais le temps de communication augmente. Soit s_p la taille de la population, t_f le temps de calcul d'un appel à la fonction d'évaluation et t_c le temps de communication par sous-population, le nombre optimal de processeurs est :

$$P = \sqrt{\frac{s_p t_f}{t_c}} \tag{5.1}$$

La figure 5.2 montre l'accélération possible en fonction des nombres de processeurs avec des paramètres différents pour le rapport $\frac{t_f}{t_c}$ en double échelle logarithmique. L'accélération maximale s'exprime en fonction du nombre optimal des processeurs comme $\frac{1}{2}P$. Dans ce cas, la moitié du temps est utilisée pour la communication et non pour le calcul. Ceci mène à deux conclusions. D'une part, on pourrait encore s'interroger sur la question d'optimiser la distribution de charge entre les processeurs en fonction du début de calcul (surtout au cas où $\frac{t_f}{t_c} = 1$). D'autre part, on va probablement préférer un nombre de processeurs moins élevé pour lequel l'accélération reste plus près de l'accélération linéaire et les processeurs individuels seront donc mieux exploités dans la

plupart des cas pratiques.

En ce qui concerne les autres modèles parallèles une estimation exacte de l'accélération devient plus compliquée. D'une part, chaque processeur exécute son propre algorithme génétique, ce qui élimine une partie du temps de communication [9] parce que les sous-populations ne sont pas échangées complètement. D'autre part, les propriétés de l'algorithme changent selon la topologie des sous-populations, ce qui rend nécessaire une adaptation des paramètres. En particulier, il n'y a plus d'équivalence avec un algorithme séquentiel. La sélection n'est plus globale, aussi le temps de convergence peut changer. La pression de la sélection change avec la migration. Si l'on choisit pour la migration par exemple les individus en fonction de leur *fitness*, ce qui est normalement le cas, on augmente directement la pression menant à une convergence plus rapide. Une comparaison des quatre stratégies de migration principales focalisée sur le changement de la pression sélective se trouve dans Cantú-Paz (2001). Le nombre optimal de processeurs reste asymptotiquement le même que dans le cas de la parallélisation globale (Cantú-Paz & Goldberg, 1999).

Dans certains cas, une accélération super-linéaire peut être observée mais pour les raisons précédentes ces comparaisons ne sont en général pas *honnêtes* vis à vis de la version séquentielle (Cantú-Paz, 2007). Une autre possibilité d'une meilleure accélération que celle théoriquement possible, peut être due au fait que les sous-populations rentrent complètement dans le cache du processeur tandis que la population globale nécessite l'utilisation de la mémoire vive qui est beaucoup plus lente.

5.3 Approche proposée

La parallélisation des EA et de notre algorithme pour le problème d'appariement de graphes en particulier est relativement simple à implémenter. Nous nous intéressons surtout à la parallélisation à l'intérieur de l'EA. La section 5.5 introduit d'autres possibilités au niveau applicatif. La population peut être divisée en sous-populations. Cette approche est très générale car non-dépendante d'une application précise.

Il n'existe qu'une seule opération qui nécessite l'intégralité de la population en même temps : la sélection. Plus précisément, ce n'est pas la population qui est indispensable, mais les valeurs de la *fitness* de chaque individu. Toutes les autres opérations, y compris les plus coûteuses comme la recherche locale et la fonction d'évaluation, sont définies sur les individus (à l'exception du croisement nécessitant deux parents pour créer un enfant). L'idée fondamentale est de démarrer autant de processus que l'on possède de processeurs après avoir sélectionné les individus qui participeront à la reproduction. Cet ensemble de parents est ainsi distribué également sur tous les processeurs avec approximativement le même nombre d'opérations de chaque type à exécuter. En conséquence, le temps de calcul sur chaque processeur sera à priori égal, ce que permet une exploitation quasi-optimale des ressources, car il n'y aura pas de temps d'attente après que les processus aient terminé.

9. Et potentiellement du temps d'inactivité sous conditions qu'il ne faille pas trop synchroniser.

En réalité, l'éventuel temps d'attente est une source majeure de perte de performance lors de la synchronisation. En ce qui concerne la synchronisation des données, nécessaire afin qu'aucun processus ne puisse corrompre les données des autres, elle peut être assez limitée. En effet, si l'on s'assure que le croisement, la mutation et la recherche locale créent des nouveaux individus au lieu d'altérer les existants, on n'a pas besoin de synchroniser le tout. Chaque processus possède désormais sa propre liste d'individus et sa liste des valeurs de *fitness* de ceux-ci.

Le surcoût provient de la fusion des listes afin de faire la sélection (étant faite par une liste d'indices au lieu de copier les individus). Après la sélection, les nouveaux parents sont redistribués parmi les différents processus. Le surcoût en mémoire est limité aux objets nécessaires pour démarrer les nouveaux processus ainsi que les références redondantes (qui sont nécessaires afin que chaque processus puisse accéder aux parents dont il a besoin) sur les individus.

Quant au générateur de nombres aléatoires, certains opérateurs nécessitent un grand nombre d'appels à lui. Comme il est nécessairement synchronisé, il risque d'être un goulot d'étranglement s'il y a des appels concurrents venant des différents processus. Afin d'éviter tout appel concurrent, chaque processus possède son propre générateur.

5.4 Gain pratique - expérimentation

Pour étudier la performance à large échelle de notre EA, nous utilisons une grappe de calcul hétérogène, constituée de 17 nœuds qui se basent sur six types de processeurs. Afin d'obtenir des résultats homogènes, nous utilisons en particulier les nœuds équipés de processeurs Intel Xeon X7350 Tigerton (2.93 GHz, 4 noyaux, 4Mo cache L2 par noyau) et Intel Xeon Tulsa (3.4 GHz, 2 noyaux, 2Mo cache L2 par noyau, 16Mo cache L3). Les nœuds utilisés ont 16 noyaux chacun. Le système d'exploitation est Linux. Il faut aussi noter que notre algorithme est écrit en Java. Par conséquent, un accès direct aux ressources n'est pas possible, on n'a notamment pas de contrôle sur la distribution des processus sur les processeurs. La performance est dépendante de l'implémentation de la machine virtuelle. Nous utilisons la Java Hotspot VM dans la version 1.6. Dans les dernières comparaisons avec d'autres langues de programmation, Java s'est montré extrêmement efficace (Amedro et al., 2008).

Nous avons choisi une base de graphes réels. Il s'agit des graphes extraits d'une base de données de 42000 molécules [10] au total. Nous utilisons plusieurs sous-ensembles contenant tous des molécules d'une taille spécifique. L'algorithme est lancé trente fois sur chaque paire de graphes après un filtrage qui consiste à éliminer les graphes ayant une distribution différente d'atomes (c'est-à-dire un nombre différent pour au moins un type d'atomes). En effet, il s'agit d'une mesure qui permet la limitation de l'espace de recherche basé sur une signature (cf. Sec. 2.8, p. 54). De plus, si deux graphes ont des signatures non-identiques, alors on n'a pas besoin de chercher un appariement, puisque l'on

10. NCI AIDS antiviral screen.

peut directement conclure qu'il s'agit de deux graphes différents. Dans l'autre cas uniquement, la recherche de l'appariement pour la décision d'isomorphisme est nécessaire.

La diminution du temps de calcul dépend en pratique de plusieurs facteurs. Dans cette section, nous étudions certains d'entre eux qui peuvent aider à décider si la parallélisation est avantageuse dans une application précise. La question principale étant : le gain en vitesse justifie-t-il le coût supplémentaire (notamment l'utilisation d'ordinateurs parallèles très coûteux) ? De manière générale, la parallélisation est avantageuse si le calcul à distribuer est assez lourd. En termes techniques, il faut un minimum de travail dans chaque processus pour justifier la distribution. Les processus longs sont préférables aux processus plus courts.

Le calcul principal se passe au niveau de la fonction d'évaluation. Il devient plus coûteux quand la taille des graphes augmente. Comme la parallélisation se fait par génération, la taille de la population joue aussi un rôle, car elle détermine combien d'appels à la fonction d'évaluation sont distribués.

5.4.1 Taille des graphes

Nous étudions d'abord l'influence de la taille de graphes. L'accélération pour des tailles de graphes de 22, 30 et 50 sommets est illustrée sur les figures 5.3 et 5.4. Les expériences sont effectuées en utilisant la taille de population 121, dont un individu élite sur lequel les opérateurs ne sont pas appliqués. De fait, cette valeur assure que le nombre d'opérations est divisible par la plupart de nombres de 1 à 8 (le nombre de processeurs) ce qui mène à des sous-populations de taille égale.

Nous constatons que l'accélération est effectivement plus forte pour des tailles plus élevées. Entre les résultats obtenus avec les processeurs Tulsa et Tigerton, on peut noter que les processeurs Tulsa donnent un gain supérieur. Les processeurs Tigerton, qui sont *a priori* environ deux fois plus rapides, ne donnent quasiment pas de gain en vitesse lorsque l'on passe d'un seul à deux. L'accélération est ensuite comme attendu. Toutefois, avec quatre processeurs, l'algorithme est seulement deux fois plus rapide. Pour des graphes de taille 22, le nombre optimal de processeurs est quatre (les gains que l'on observe en utilisant 5 et 6 processeurs étant négligeables). Quand la taille augmente, alors l'algorithme peut profiter de plus de processeurs (5 pour la taille 30, 8 à taille 50).

Les processeurs Tulsa profitent dès le début de l'ajout de processeurs. A part cela, nous constatons le même comportement qu'avec les processeurs Tigerton. L'accélération est plus grande, mais les processeurs sont *a priori* moins performants. En utilisant trois processeurs l'algorithme est déjà entre 2.5 fois (un peu moins pour la taille 22, un peu plus pour la taille 30) et presque 3 fois (pour la taille 50) plus rapide. Lorsque l'on utilise encore plus de processeurs, l'accélération s'éloigne de son optimum théorique.

En résumé, la parallélisation est plus avantageuse pour de grands graphes que pour des petits. Avec les tailles de graphes testées, nous observons un comportement presque idéal pour les petits

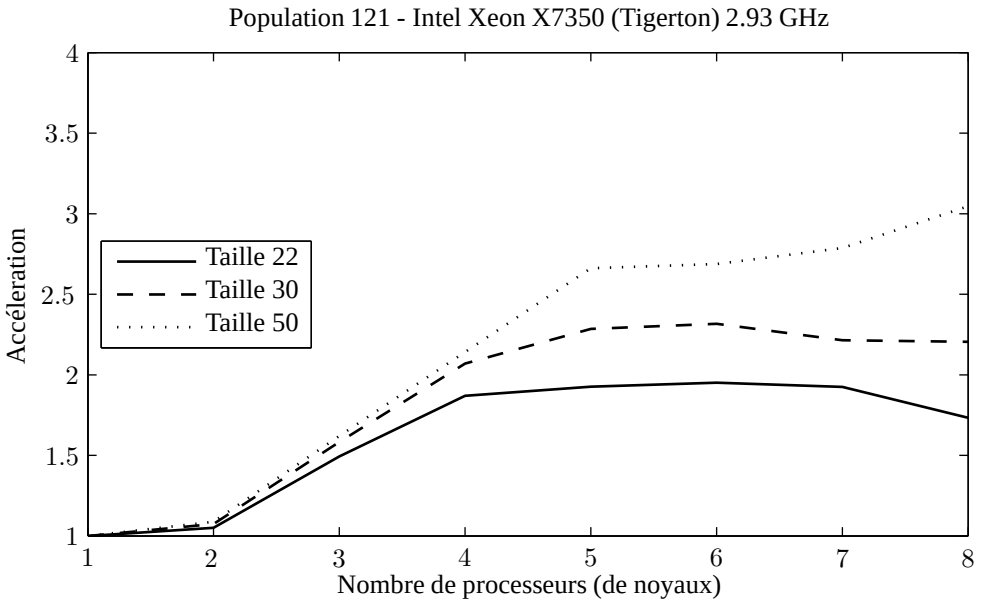

FIGURE 5.3 – Accélération par rapport à la taille des graphes (processeurs Tigerton).

nombres de processeurs. En effet, on s'approche plus du cas idéal avec des graphes plus grands. Quand on traite des problèmes plus complexes, le nombre de processeurs dont on peut profiter efficacement augmente.

5.4.2 Taille de la population

Nous étudions ensuite l'influence de la taille de la population. L'étude théorique [11] sur l'accélération maximale et sur le nombre optimal de processeurs suppose que la taille de la population soit assez grande (1000 individus). Cependant, les tailles qui mènent à des bonnes performances en termes de précision et temps de calcul (cf. Ch. 3, p. 59 et suivantes), ainsi que celles mentionnées dans la plupart des travaux, sont souvent plus petites. En effet, la taille de la population est choisie en fonction des performances de l'algorithme et non afin de permettre une meilleure accélération. Les différents applications et opérateurs nécessitent des tailles bien différentes (par exemple 400 pour un GGA-CX contre 10 pour un GGA-UOX). Il est donc néanmoins intéressant d'observer la relation entre la taille de la population et l'accélération. De même, les expériences sur la taille de population donnent une indication sur les effets prévisibles de la taille du problème. Comme en effet, l'effort de computation ne croît pas linéairement avec la taille de graphes, ceci est le cas pour la taille de la population.

Nous observons que les différences entre les types de processeurs subsistent. C'est-à-dire l'utilisation de deux processeurs Tigerton n'augmente que très peu la vitesse et l'accélération est de

11. cf. Sec. 5.2, p. 121.

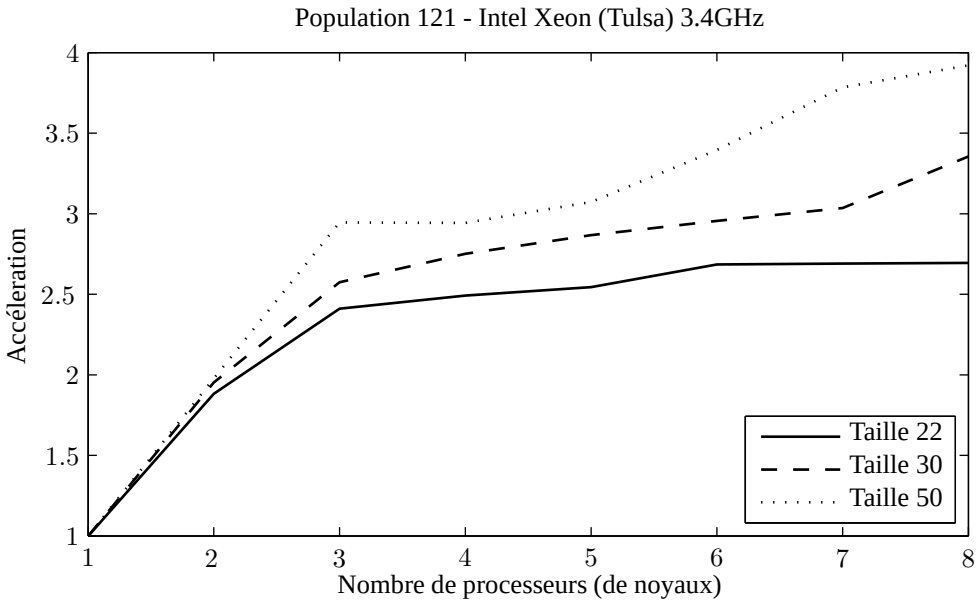

FIGURE 5.4 – Accélération par rapport à la taille des graphes (processeurs Tulsa).

manière générale plus faible que sur les processeurs Tulsa. Le comportement général reste néanmoins comparable.

La figure 5.5 montre l'effet de la taille de la population sur la vitesse pour un graphe de taille 22. Les tailles de population utilisées sont 100, 500, 1000 et 1500 individus. Nous constatons qu'une petite taille mène à une accélération plus faible, tandis qu'un algorithme utilisant une grande taille accélère plus. Pour la taille 1000 par exemple, l'algorithme est un peu moins de trois fois plus rapide avec trois processeurs. Quand on utilise huit processeurs, le gain n'augmente que peu.

De manière générale, sur les graphes de taille 20-50, l'utilisation de trois processeurs semble approprié. Nous pouvons conclure que l'application de la parallélisation vaut la peine quand les graphes sont grands et/ou la taille de la population est grande. Par contre, augmenter explicitement de la taille de la population afin de faire une bonne parallélisation ne paraît pas avantageux.

5.4.3 Communication entre processus

Après ces expériences générales, nous étudions aussi des optimisations possibles concernant la communication entre processeurs. De fait, quand on copie un grand nombre de chromosomes du processeur principal aux processeurs travailleurs, il se peut qu'une approche étape par étape accélère l'exécution car le temps d'attente diminue. L'idée que nous suivons consiste à envoyer uniquement les identifiants des parents et non les parents eux mêmes. De la sorte les processeurs peuvent commencer à travailler plus rapidement, y compris le processeur maître. On tente donc de réduire le temps d'inactivité.

Soit l la longueur des chromosomes et s_p la taille de la population. On a alors besoin d'envoyer

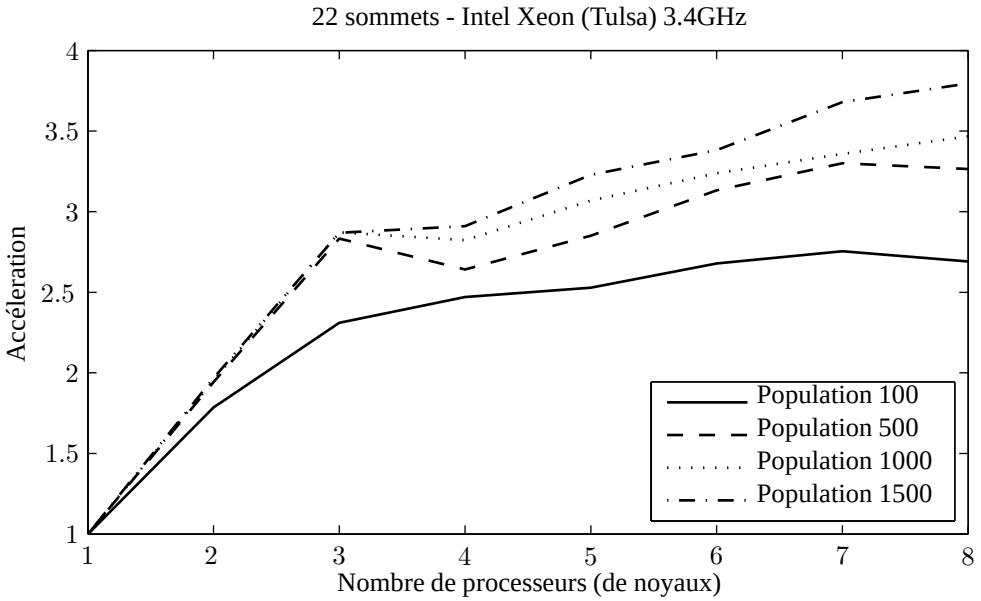

FIGURE 5.5 – Accélération par rapport à la taille de la population (processeurs Tulsa).

un tableau de taille $l \times s_p$. L'idée d'envoyer des identifiants réduit la taille du tableau à envoyer initialement à une valeur entre s_p et $2s_p$ selon la quantité de croisements à faire. On obtient notamment $2s_p$ quand chaque enfant a deux parents (quand le croisement s'applique à 100% de la population). On obtient s_p si aucun enfant n'est créé en utilisant le croisement.

Bien que l'idée générale soit de copier uniquement les identifiants, au moment d'appliquer les opérateurs, le processeur esclave a besoin de connaître les chromosomes. Nous étudions l'algorithme parallèle sur une seule machine dont la mémoire est partagée : le processeur esclave cherche alors lui-même les parents nécessaires pour la création du descendant suivant. Pour cela une synchronisation d'accès est inévitable car la population des parents n'existe qu'une seule fois dans la mémoire.

Un schéma très souvent utilisé dans les algorithmes génétiques est d'appliquer les opérateurs l'un après l'autre sur l'ensemble de la population. On commence avec le croisement. Une fois que ceci est fait sur toute la population, on applique la mutation sur les résultats, et ainsi de suite. Du point de vue de la synchronisation, le problème avec ce schéma, que nous appelons ordre classique par la suite, est que, comme le croisement est un opérateur très peu coûteux notamment par rapport à la recherche locale et l'appel à la fonction d'évaluation, les demandes des parents ne sont pas bien distribuées. Tous les processeurs esclaves essaient d'accéder aux parents au début de l'exécution. Ceci crée des blocages qui ralentissent l'algorithme. Le nombre de blocages (et ainsi le ralentissement dû à la synchronisation) augmente avec le nombre de processeurs. La taille de la population joue aussi un rôle : quand elle est faible, il y a peu de données à communiquer, aussi l'envoi de toute la matrice des parents n'est pas coûteux.

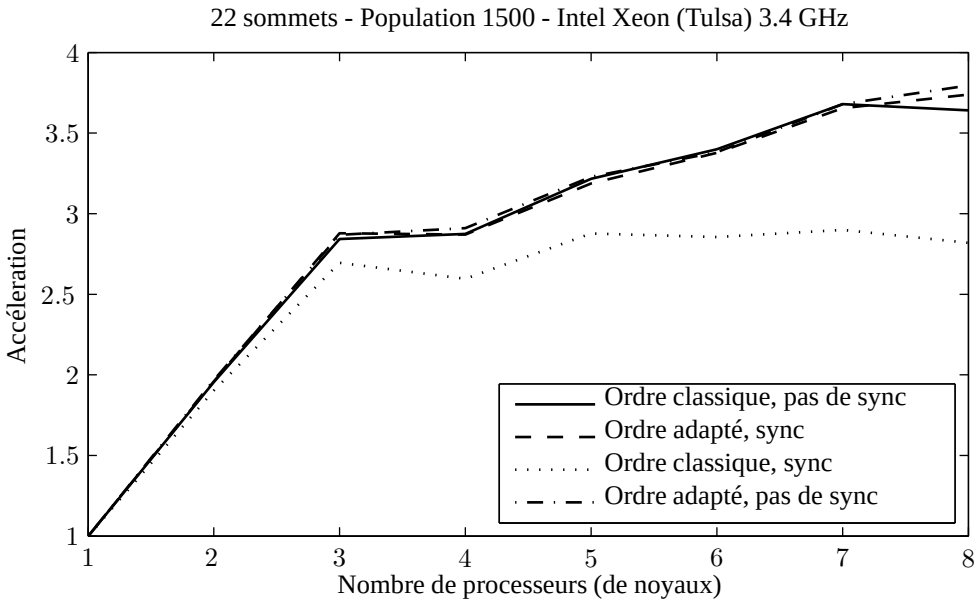

FIGURE 5.6 – Accélération par rapport à l'ordre (processeurs Tulsa).

Afin d'uniformiser la distribution des accès et donc limiter le nombre de blocages, nous proposons de changer l'ordre d'exécution. De fait, rien n'empêche de suivre un individu de la création par le croisement jusqu'au calcul de la fonction d'évaluation pour ensuite continuer avec le prochain. Nous effectuons donc une comparaison de ce nouvel ordre et de l'ordre classique, dans les deux cas sans ou avec synchronisation. L'absence de synchronisation implique que les parents sont copiés pour chaque processeur esclave par le processeur principal. Les figures 5.6 et 5.7 montrent les résultats obtenus. Nous constatons en premier lieu que la stratégie la moins bonne est l'ordre classique avec synchronisation. Ensuite nous observons que le changement de l'ordre est avantageux dans tous les cas (avec ou sans synchronisation).

Pour les processeurs Tulsa, les différences entre les trois versions (autres que l'ordre classique avec synchronisation) sont négligeables jusqu'à 7 processeurs parallèles. Quand on en utilise 8, nous constatons un léger avantage pour l'ordre adapté sans synchronisation, devant l'ordre adapté avec synchronisation.

Pour les processeurs Tigerton, les relations sont plus compliquées. Bien que l'ordre classique sans synchronisation montre le comportement le plus cohérent, l'ordre adapté montre une meilleure accélération (avec synchronisation). Sans synchronisation, l'accélération est aussi meilleure quand on utilise peu de processeurs. Toutefois, quand on passe à 7, l'accélération diminue. Pour l'ordre classique avec synchronisation, nous observons une accélération jusqu'à 5 processeurs, puis, la vitesse baisse beaucoup. La meilleure stratégie est l'ordre adapté avec synchronisation, mais les différences ne sont pas très élevées.

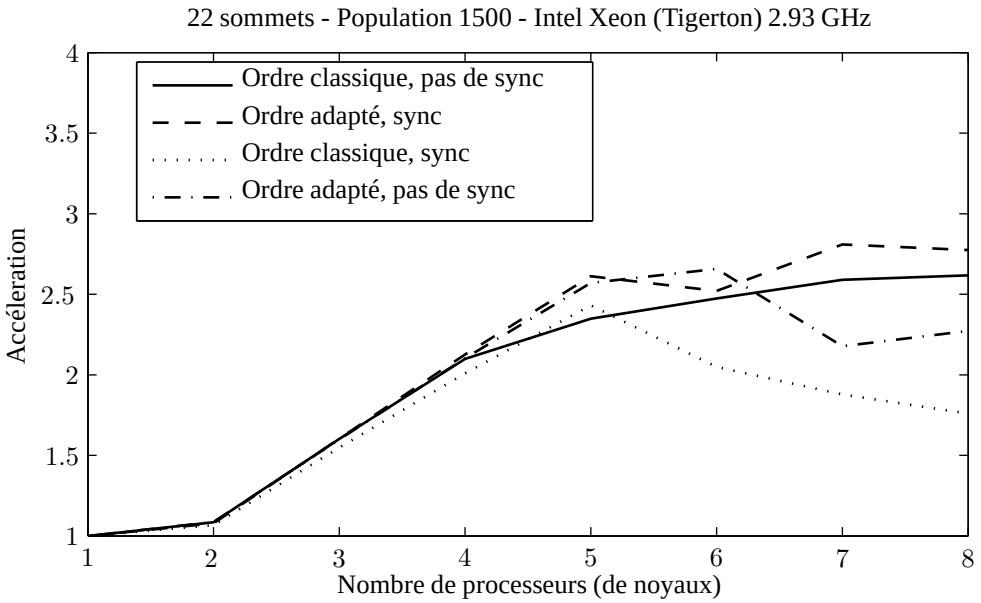

FIGURE 5.7 – Accélération par rapport à l'ordre (processeurs Tigerton).

5.5 Parallélisation au niveau applicatif

Recherche d'un graphe dans une base Dans certaines applications il est possible de paralléliser à un niveau plus haut. Par exemple si l'on souhaite identifier un graphe donné parmi un ensemble des graphes existants pour trouver celui qui correspond le mieux, on a besoin de plusieurs appariements. Dans le pire cas, il faut le comparer à tous les graphes de la base. Dans ce cas, au lieu de distribuer un algorithme sur les processeurs disponibles, on peut paralléliser au niveau de l'identification. En d'autres termes, on utilise l'algorithme génétique séquentiel mais on en utilise plusieurs instances sur différentes paires de graphes en parallèle. La parallélisation se situe alors au niveau applicatif. Les avantages principaux sont d'une part que les fils d'exécution sont plus longs et d'autre part que la communication entre processeurs est minimisée. Aussi dans le cadre de l'identification on peut observer l'évolution des distances : si par exemple une exécution arrive à une distance proche de 0, alors il n'est plus nécessaire de suivre les autres. On peut alors interrompre de façon anticipée tous les algorithmes qui tournent en fonction des résultats d'un seul. Ceci est particulièrement intéressant car nous observons que le nombre de générations est moins élevé quand il s'agit des graphes similaires que si l'on compare des graphes différents. Bien que ceci soit aussi vrai pour d'autres méthodes comme les approches basées sur la recherche de l'arbre A*, ils ne permettent pas d'obtenir une solution courante facilement.

Augmentation de la précision avec des populations indépendantes Il peut également être avantageux d'exécuter le même algorithme génétique sur une seule paire de graphes plusieurs fois afin

d'augmenter la probabilité de trouver la bonne solution. Par exemple, si l'on suppose que l'algorithme obtienne la solution optimale avec une probabilité p de 50%, alors la probabilité d'obtenir la solution optimale avec (au moins) un algorithme parmi n algorithmes identiques est de $1-(1-p)^n$. Pour $n = 2$ on obtient 75%. Comme $1-(1-p)^n$ tend asymptotiquement vers 1, on peut, avec le choix d'un nombre d'algorithmes parallèles suffisant, assurer qu'un trouve la solution optimale avec n'importe quelle probabilité $p < 1$ donnée.

5.6 Bilan

Bien que la parallélisation d'un algorithme génétique semble évidente, il existe de très nombreux détails techniques à prendre en compte afin que l'on puisse vraiment accélérer l'exécution. Par exemple, le confort d'un langage de programmation orienté objet comme JAVA cache certaines choses de bas niveau qui sont cependant importantes (optimisation de l'ordre de boucles, synchronisation, distribution des fils d'exécution sur les processeurs). Il est donc important de bien choisir les technologies utilisées et surtout de valider les résultats. Nos expériences montrent que la qualité de l'accélération dépend de la taille du problème et de la taille de la population. Avec les graphes de taille moyenne que nous testons, l'utilisation de 3-5 processeurs semble appropriée. Si l'on traite des graphes plus grands ou si l'on a besoin de tailles de population plus importantes, alors ce nombre peut augmenter et on peut effectivement profiter de plus de processeurs. Finalement, selon les applications, il est possible de paralléliser non au niveau des générations mais au niveau des algorithmes. Globalement, la parallélisation améliore la vitesse dans tous les cas. Il existe cependant une limite, théorique et pratique, du nombre maximal de processeurs dont l'algorithme peut profiter. En ce qui concerne la précision, et les autres propriétés de l'algorithme, elles restent inchangées car nous avons choisi une parallélisation globale. En effet, d'autres types de parallélisation sont possibles mais nécessitent une étude beaucoup plus approfondie car les propriétés de l'EA changent.

Chapitre 6

Garantie du résultat optimal

Malgré toutes les astuces que l'on peut utiliser afin d'améliorer la convergence des EA, ces derniers restent des méthodes stochastiques dont le résultat dépend des nombreuses variables aléatoires. Bien que l'on puisse augmenter la probabilité de trouver la solution optimale, on n'a pas de garantie d'optimalité en général [12]. Dans les chapitres précédents nous avons amélioré l'algorithme avec plusieurs autres heuristiques qui ne garantissent pas l'obtention de la solution optimale. Des méthodes dites optimales existent mais elles sont très lentes.

Notre proposition consiste à combiner l'algorithme génétique, qui obtient une bonne solution assez rapidement, avec une méthode exacte, qui est plus lente mais garantit la solution optimale (Bärecke & Detyniecki, 2007a). L'idée principale est d'utiliser une méthode de recherche d'arbre qui permet de limiter les chemins explorés suivant une estimation de la distance maximale acceptable entre les graphes. Il s'agit d'une distance pour laquelle on sait qu'il existe une solution correspondante : l'algorithme génétique donne une estimation de départ et accélère ainsi la recherche d'arbre.

Nous étudions d'abord la méthode optimale avec laquelle nous proposons de combiner, examinant en particulier son paramètre, la distance maximale tolérée. Nous détaillons ensuite l'algorithme hybride proposé.

6.1 Influence du seuil d'acceptation sur l'algorithme A*

Notre intérêt pour l'algorithme de Berretti et al. (2001) (cf. Sec. 1.5, p. 14) est basé sur le fait qu'il garantit la solution optimale. Son seul paramètre applicatif est le seuil maximal d'acceptation modélisant la différence maximale tolérée entre deux graphes, ce qui correspond dans le problème d'identification au seuil sur lequel deux graphes peuvent être considérés comme différents. On constate une corrélation directe entre ce seuil et le temps de calcul de l'algorithme. L'idée est

12. Dans certains cas des EA simples, on peut toutefois prouver l'optimalité mais sous l'hypothèse d'un temps de calcul infini (Rudolph, 1994).

que la définition d'un seuil permet d'élaguer l'ensemble des appariements candidats. Il serait alors souhaitable de réduire le seuil d'acceptation le plus possible afin d'obtenir les meilleures performances. Si le seuil est inférieur à la distance correspondante à la meilleure solution existante et donc à la distance réelle entre les graphes en question, alors l'algorithme ne parvient pas à trouver de solution. Dans ce cas, il valide simplement la non-existence d'une solution inférieure au seuil choisi et le temps de calcul est considérablement réduit.

Nous cherchons à établir une analyse plus profonde de la corrélation entre le seuil et le temps de calcul. Il est évident que le seuil est fortement dépendant de l'application concrète et en particulier de l'amplitude des distorsions possibles. Nous utilisons une base de graphes aléatoires car une telle base permet de fixer le niveau de bruit et ainsi de calculer une espérance de la distance réelle. En effet, nous cherchons à établir empiriquement un lien entre le temps de calcul et le rapport entre la distance réelle et le seuil. Cela nécessite une base contenant des paires de graphes avec des distances proches.

L'espérance mathématique de la distance entre les graphes se calcule en fonction de leur taille et du niveau de bruit utilisé lors de leur création. Soit σ l'écart-type de la variable aléatoire qui détermine le bruit ajouté, alors l'espérance de distance entre un graphe et le graphe correspondant perturbé est $E = E(d(g_1, g_2)) = \frac{\sqrt{3}}{4}\sigma n(n+1)$. Dans le cas du bruit uniforme le niveau de bruit est déterminé par la longueur de l'intervalle $[-\varepsilon; +\varepsilon]$. Afin d'assurer une espérance égale à celle du bruit gaussien, alors on calcule ε en fonction de σ : $\varepsilon = \sqrt{3}\sigma$.

Nous examinons en particulier l'intervalle centré sur cette espérance. Nous définissons vingt seuils équidistants dans l'intervalle $[0,1E + \tau, 2,0E + \tau]$. Nous excluons le cas d'une espérance de 0, qui correspond à $\sigma = \varepsilon = 0$, car il n'est pas intéressant en pratique. De fait, une espérance de 0 transforme l'isomorphisme inexact en un problème d'isomorphisme exact pour lequel des méthodes bien plus performantes existent (cf. Sec. 1.5, p. 14). Quand on lance l'algorithme de Berretti et al. avec un seuil qui correspond exactement à la distance entre les graphes, alors l'algorithme ne trouve pas de solution pour des raisons techniques (cf. Ann. E, p. 257). Nous introduisons donc une tolérance numérique. Une valeur suffisante pour le fonctionnement de tous les algorithmes dont en particulier l'A* est $\tau = 10^{-10}$.

6.1.1 Expérimentation

D'abord nous visons à établir un lien entre le temps de calcul et le seuil d'acceptation. Nous exprimons le seuil d'acceptation comme un multiple de l'espérance de la distance. En effet, ceci permet de voir l'influence d'un seuil plus ou moins proche de la distance réelle (que l'on ne connaît pas dans les cas réels). Deux cas sont à distinguer : d'une part, le seuil peut être supérieur ou égal à la distance réelle. Une solution sera alors trouvée. D'autre part, le seuil peut être inférieur à la distance réelle, et dans ce cas, l'algorithme ne peut pas trouver de solution, vérifiant ainsi la non-existence

d'une solution.

Nous constatons qu'en général, le temps de calcul augmente avec le seuil d'acceptation. Les figures 6.1 (bruit uniforme) et 6.2 (bruit gaussien) montrent les temps de calcul en fonction du seuil choisi. Nous utilisons une échelle logarithmique afin de pouvoir visualiser un domaine très large. Les lignes pointillées horizontales correspondent au temps de calcul sans seuil d'acceptation. Le temps est alors le temps nécessaire pour vérifier la non-existence d'une solution. Les lignes en trait plein correspondent au premier cas. Les lignes semi-pointillées indiquent les exécutions sans solution inférieure au seuil choisi.

Nous constatons que pour des seuils très petits (inférieures à $0,4E$) par rapport à la distance espérée les temps de calcul sont négligeables. Entre 40% et 60% de l'espérance, le temps de calcul augmente considérablement. Nous observons également que le niveau de bruit influence le début de la croissance du temps de calcul : un niveau plus élevé précipite la croissance. La différence entre des exécutions avec ou sans solution est négligeable : les courbes s'enchainent sans saut. Quand le seuil dépasse la distance, alors le temps de calcul progresse toujours avec un gradient dépendant du bruit : plus le bruit est fort plus la hausse est importante. Nous observons que le temps de calcul n'atteint pas celui de référence (sans seuil), même quand on met le seuil à deux fois la distance.

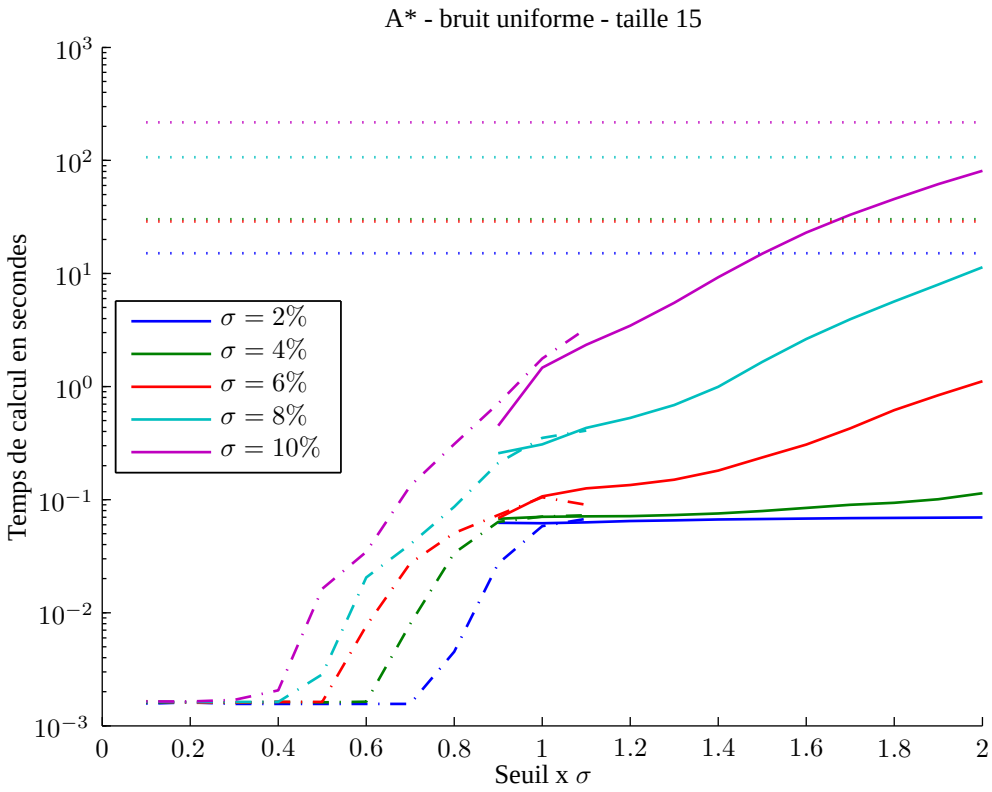

FIGURE 6.1 – Influence du seuil d'acceptation sur le temps de calcul - bruit uniforme.

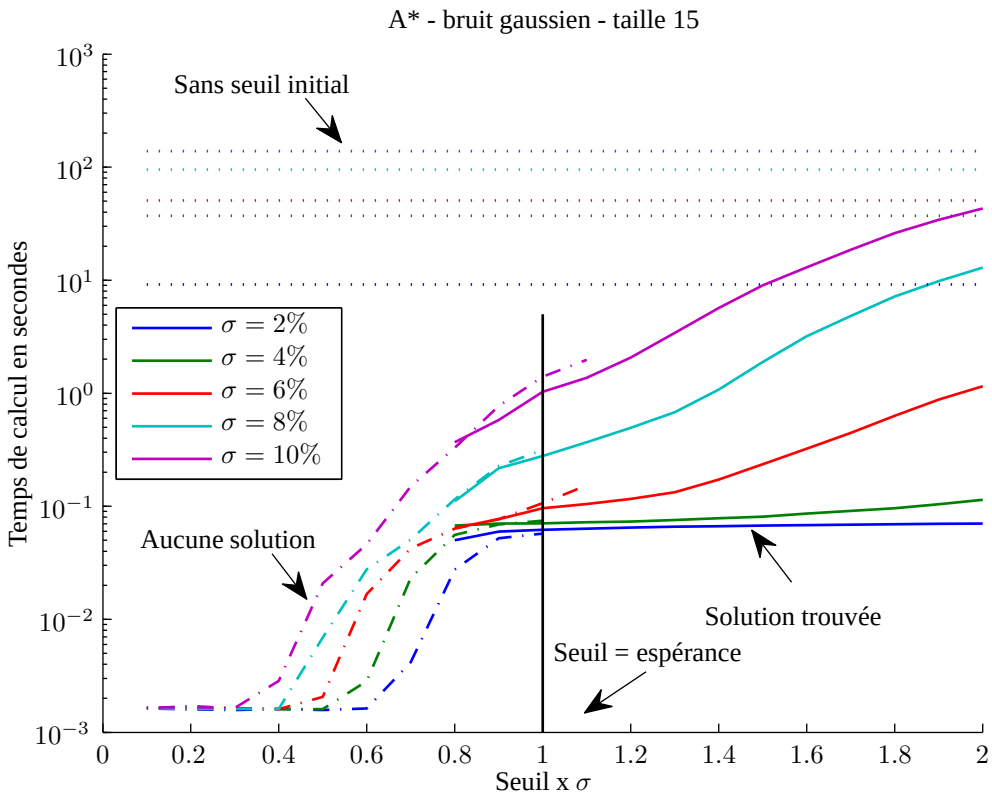

FIGURE 6.2 – Influence du seuil d'acceptation sur le temps de calcul - bruit gaussien.

Les tableaux 6.1 et 6.2 montrent la relation entre l'algorithme A* sans seuil et avec un seuil égal au double de l'espérance, dans un environnement parallèle et séquentiel respectivement. L'exécution en parallèle permet d'obtenir des résultats plus rapidement mais on ne peut pas exclure des effets de la parallélisation sur le mesure du temps. L'exécution séquentielle donne ainsi des résultats plus précis mais nous ne pouvons pas aller aussi loin en termes de taille de graphes et de niveau de bruit. Nous constatons que les temps de calcul avec seuil sont largement inférieurs à ceux sans seuil. Dans tous les cas, le temps de calcul augmente de manière très importante avec le niveau de bruit et la taille des graphes, ce qui rend l'utilisation de l'algorithme pratiquement impossible pour des graphes de taille 20. Pour la version séquentielle nous avons utilisé un type de processeur qui est environ deux fois plus rapide que le type utilisé dans la version parallèle, par contre on n'en utilise qu'un seul à la fois au lieu de 16. Par conséquent la version parallèle fournit encore des résultats pour des problèmes environ huit fois plus complexes (en termes du temps de calcul nécessaire). Nous observons également une différence entre les deux types de bruit, le temps de calcul avec du bruit gaussien est inférieur au temps avec du bruit uniforme. En effet, il s'agit du temps moyen sur l'ensemble de 50 paires de graphes. Les distances entre les paires de graphes sont plus concentrées autour de l'espérance quand on utilise du bruit gaussien que quand on utilise du bruit uniforme. Pour

	A* sans seuil					Double de l'espérance				
Bruit	2%	4%	6%	8%	10%	2%	4%	6%	8%	10%
Taille	Bruit uniforme									
13	3,8	7,1	15,4	17,2	39,1	0,1	0,1	0,4	3,4	19,7
14	10,6	34,7	58,6	98,4	127,5	0,1	0,2	1,4	9,3	54,6
15	36,0	69,9	82,9	302,7	495,5	0,2	0,2	3,2	31,7	191,0
16	230,9	253,7	954,9	749,4	3085,0	0,2	0,6	8,5	102,4	789,7
	Bruit gaussien									
13	4,4	5,5	6,2	11,9	32,2	0,1	0,1	0,3	2,4	12,3
14	5,1	12,0	44,8	36,9	139,3	0,1	0,2	1,2	6,0	39,7
15	17,8	107,4	125,6	240,2	354,8	0,1	0,3	3,3	31,9	120,6
16	89,9	148,7	702,6	615,0	3457,2	0,3	0,7	8,0	70,8	925,3

Tableau 6.1 – Temps de calcul en secondes de l'algorithme A* sans seuil - version parallèle - sur Intel Xeon Tulsa.

ces dernières notamment, les valeurs extrêmes sont plus éloignées de l'espérance. Comme le temps de calcul est exponentiel avec la distance, alors ces valeurs extrêmes ont une forte influence sur la moyenne.

Dans cette section, nous avons effectué des expérimentations exhaustives avec des seuils différents afin de quantifier la correspondance entre le temps de calcul et le seuil choisi. Nous constatons d'une part que la convergence de l'algorithme A* est accélérée énormément si l'on fournit une bonne estimation du seuil. Ceci est aussi vrai si l'on surestime la distance (jusqu'à un facteur de deux, ce qui est toujours une bonne estimation dans des applications concrètes ayant une variance forte et *a priori* inconnue des données). D'autre part, le temps de calcul pour les seuils inférieurs à la distance, ne fournissant pas de solutions, tend à être très faible. Par contre, un problème majeur est que l'on ne connaît pas en pratique l'espérance de la distance en avance, ce qui est nécessaire afin de fixer les seuils.

	A* sans seuil					Double de l'espérance				
Bruit	2%	4%	6%	8%	10%	2%	4%	6%	8%	10%
Taille	Bruit uniforme									
13	1,6	2,6	5,7	7,0	15,6	0,03	0,04	0,14	1,3	7,5
14	3,8	12,0	20,0	42,0	52,4	0,05	0,07	0,45	3,6	22,1
15	15,1	30,2	28,8	106,7	217,0	0,07	0,11	1,12	11,4	81,0
	Bruit gaussien									
13	1,6	2,5	3,0	5,0	14,2	0,03	0,04	0,17	1,0	5,7
14	2,2	4,2	14,8	18,5	54,4	0,05	0,06	0,39	2,9	15,9
15	9,2	37,1	50,8	95,3	138,8	0,07	0,11	1,15	12,9	43,2

Tableau 6.2 – Temps de calcul en secondes de l'algorithme A* sans seuil - version sequentielle - sur Intel Xeon X7350.

6.2 Stratégie de doublement

Dans la pratique, la définition d'une espérance de la distance et ainsi d'un seuil approprié revient presque à la résolution du problème lui-même et est donc souvent impossible. Nous essayons de contourner ce problème avec une version itérative qui commence par un seuil très petit, sous-estimant la distance réelle, puis exécute l'algorithme à plusieurs reprises en augmentant le seuil jusqu'à ce qu'une solution soit trouvée.

L'idée de la version itérative de l'algorithme A* vient des observations faites dans la section précédente. D'une part, le temps de calcul pour de petits seuils est négligeable, celui d'un seuil égal à deux fois la distance est inférieur au temps sans seuil. D'autre part, le temps de calcul augmente plus que linéairement avec le seuil. Pour cette raison, nous espérons que la somme de plusieurs exécutions avec des seuils assez petits montre des avantages par rapport à l'exécution sans seuil (ou avec un seuil conservateur, c'est-à-dire très lointain afin de n'exclure aucune solution possible).

Nous devons alors définir une stratégie d'augmentation du seuil. L'augmentation ne doit pas être trop petite, sinon de nombreuses exécutions seraient nécessaires jusqu'à ce que l'on trouve la solution, ce qui augmenterait le temps de calcul de façon significative. Elle ne doit pas non plus être trop grande, sinon le pas pourrait mener à un seuil conservateur que l'on pourrait peut-être déjà fixer auparavant et les exécutions précédentes n'auraient aucun effet positif.

Nous proposons de doubler la valeur du seuil à chaque étape. D'une part, cette stratégie est exponentielle : même si la sous-estimation est importante, relativement peu d'étapes seront nécessaires afin de passer au-delà de la distance. D'autre part, la surestimation maximale que l'on peut avoir avec cette approche est inférieure à deux fois la distance : en particulier, si l'avant-dernier seuil atteint presque la distance, le prochain et dernier sera un peu inférieur à deux fois celle-ci. Comme nous avons vu dans la section précédente, un tel seuil apporte déjà un gain important.

Les figures 6.3 et 6.4 illustrent les résultats obtenus par cette méthode par rapport à la version initiale. La sous-estimation initiale est d'un facteur de 50 ce que nous jugeons assez important pour la pratique. Par conséquent, sept itérations seront nécessaires en moyenne (pour la première itération on utilise le seuil initial, puis on l'augmente 6 fois, $2^6 = 64 > 50$). Les courbes à trait plein correspondent à la stratégie de doublement tandis que celles à trait pointillé correspondent à l'A* sans seuil initial.

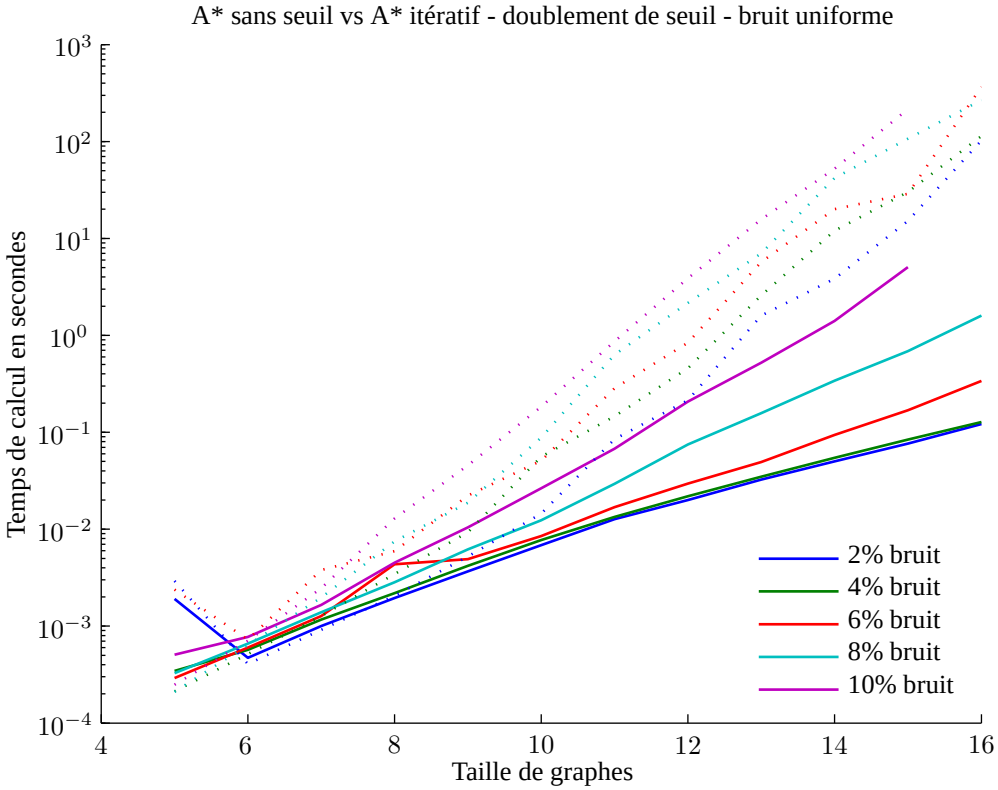

FIGURE 6.3 – Stratégie de doublement, bruit uniforme.

Nous constatons un gain considérable par rapport à l'approche sans seuil puisque le temps de calcul reste inférieur à 10 secondes alors qu'il atteignait plus de 100 secondes auparavant. Une telle stratégie est alors envisageable si l'on ne possède pas d'estimation d'un seuil d'acceptation, ce qui est souvent le cas pour les applications réelles. Toutefois, nous constatons que la différence de temps de calcul augmente exponentiellement avec la taille des graphes. En pratique, les applications sont confrontées à une limitation de la taille maximale de graphes que l'on peut traiter. Nous pouvons alors pousser la taille maximale vers des valeurs un peu plus importantes, mais sans effectivement enlever la limitation sur la taille.

On peut s'interroger sur la valeur du paramètre qui détermine la sous-estimation initiale choisi. Ici, nous pouvons utiliser un paramètre qui dépend de l'espérance, ce qui n'est pas possible en pra-

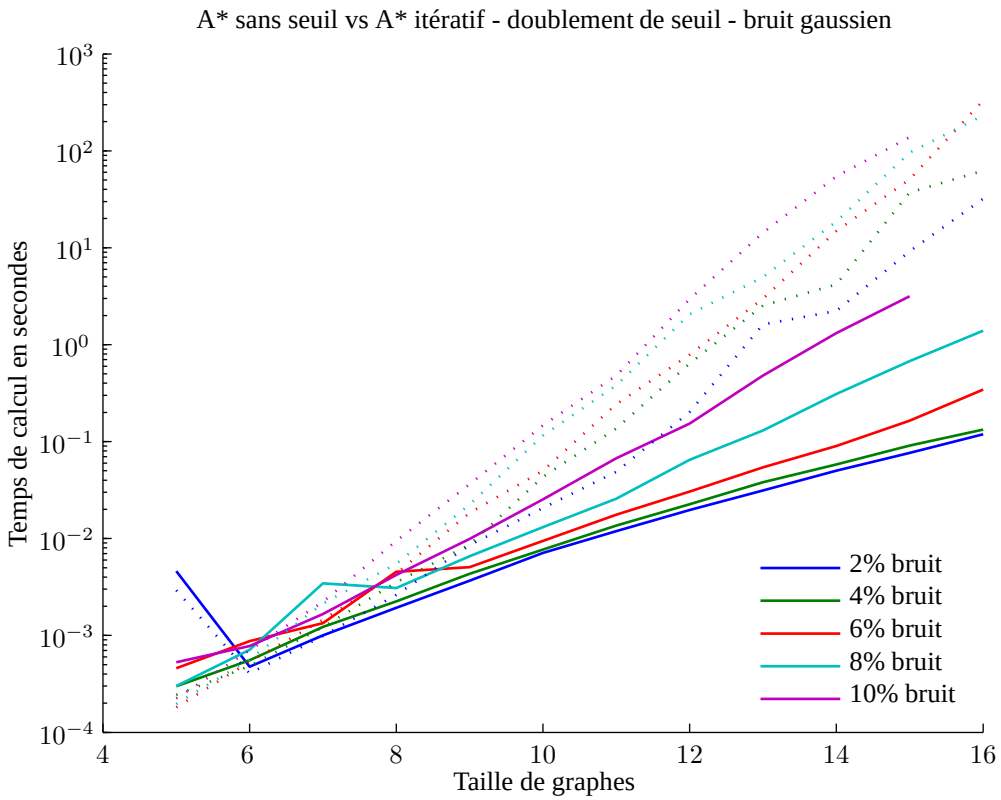

FIGURE 6.4 – Stratégie de doublement, bruit gaussien.

tique. La valeur exacte est cependant secondaire sous condition qu'elle soit inférieure à la plus petite distance, puisqu'avec la stratégie de doublement le seuil accroît exponentiellement et atteint donc la région de recherche très rapidement même si la distance réelle est plusieurs ordres de grandeurs plus loin. Par exemple, si l'on a choisi un facteur de 1000, qui correspond à un seuil initial $\frac{1}{1000}$ fois la distance, alors on dépasse la distance et on trouve la solution après seulement 11 itérations. Pour un facteur d'un million on calcule 21 itérations. Ce qui est beaucoup plus important dans ce contexte est le montant du dépassement dans la dernière itération. La figure 6.5 montre le temps de calcul de la version itérative avec des différents seuils initiaux. Les boîtes à moustaches montrent la distribution des temps de calcul sur les paires de graphes de taille 20 avec un niveau de bruit moyenne ($\sigma = 6$). A des fins de comparaison nous avons ajouté, à droite, la distribution du temps de calcul pour la stratégie $F2$ qui utilise un seuil fixe de deux fois l'espérance. Quant à l'algorithme A* sans seuil, il n'est plus applicable pour cette taille de graphes.

Nous constatons que la courbe montre des formes répétitives : le comportement entre 8 (2^3) et 16 (2^4) est équivalent à celui entre 16 et 32 (2^5). Les médianes sont les plus basses pour des puissances de deux. En revanche la dispersion vers des valeurs plus grandes est maximale. De fait, pour la moitié de l'échantillon, le seuil dans la dernière itération est très proche de la distance réelle

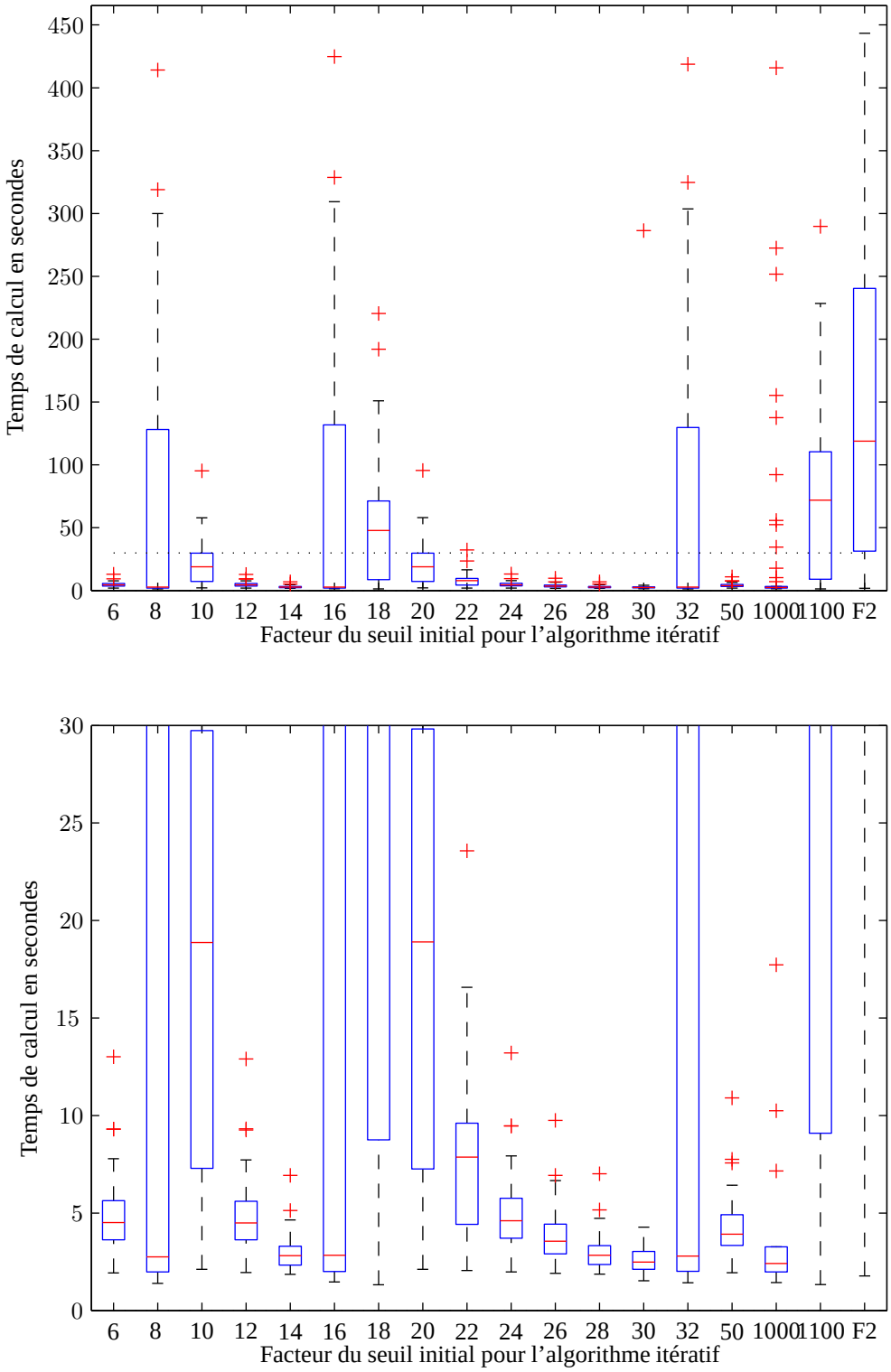

FIGURE 6.5 – Seuil initial pour la stratégie de doublement, taille de graphes 20, bruit uniforme $\sigma = 6\%$. En haut : domaine complet. En bas : restriction du temps de calcul sur l'intervalle $[0, 30]$

tandis que pour le reste elle est égale à son double. De fait, le pire cas pour l'algorithme serait de prendre une valeur initiale légèrement supérieure (et donc un facteur légèrement plus petit qu'une puissance de deux).

Notre étude montre que si l'on applique la stratégie de doublement, alors le temps de calcul dépend très fortement de la proximité du facteur à une puissance de 2 plutôt que du niveau de sous-estimation (par exemple le temps de calcul avec un facteur 6 est quasiment identique à celui du facteur 12, 24, et très proche du 50). En général, on ne connaît pas l'espérance exacte de la distance. En effet, chaque paire de graphes a une autre distance. Si l'on fixe un seuil pour une base entière, alors le facteur associé à ce seuil est différent pour chaque graphe. Pour la base utilisée, on peut noter que la variance des distances de graphes est très limitée (la distance moyenne sur toutes les paires de graphes est : $\overline{d(G_{i,1}, G_{i,2})} = 10,85$; pour la variance on obtient : $\sigma = 0,42(3,85\%)$; ainsi pour le minimum et maximum $\min(d(G_{i,1}, G_{i,2})) = 9,69$; $\max(d(G_{i,1}, G_{i,2})) = 11,69$; $\mathbb{E} = 10,91$) : les distances se concentrent autour de l'espérance. Par conséquent le facteur de sous-estimation de la distance est similaire pour chaque paire de graphes.

Pour des bases réelles, on obtient une distribution sur un plage de facteurs, pour chaque paire de graphes, le niveau de la sous-estimation est différent. La distribution des temps de calcul sur une grande base est donc quasiment indépendante du seuil choisi. Par analogie avec notre expérimentation, elle se compose de tous les groupes des diagrammes. Comme le temps est légèrement supérieur au temps de calcul avec un seuil fixe de deux fois la distance uniquement dans le pire cas et dans beaucoup d'autres cas inférieur à celui-ci, alors le temps de calcul moyen de la stratégie de doublement est souvent nettement inférieur à celui du seuil fixe (nous effectuons une comparaison après l'introduction de l'algorithme hybride dans la Fig. 6.6, p. 143).

6.3 Algorithme hybride

La combinaison des algorithmes approximatifs avec des algorithmes exhaustifs est possible lorsque l'algorithme exhaustif profite de la connaissance d'une solution candidate assez bonne. L'idée principale consiste à exécuter les deux algorithmes l'un après l'autre en transmettant la meilleure solution candidate de l'algorithme approximatif comme point de départ à l'algorithme exhaustif. Celui-ci la vérifie ensuite. Nous avons choisi l'algorithme de Berretti et al. (2001) comme méthode exhaustive et notre approche génétique comme approche approximative. D'abord, on applique un algorithme génétique. Nous utilisons l'algorithme générationnel avec le croisement UPBX et les paramètres *optimaux* (cf. Tab. 3.1, p. 95). La distance entre les graphes obtenue par le meilleur individu est évidemment une borne supérieure de la distance optimale. Ensuite on exécute l'algorithme de Berretti en utilisant cette valeur comme limite d'acceptation [13]. À la fin, on garantit donc

13. Des erreurs d'arrondi venant de la représentation des nombres à virgule flottante (cf. Ann. E, p. 257), pouvant être considérées comme bruit numérique, imposent l'introduction d'une tolérance numérique. Nous utilisons 10^{-15} en double

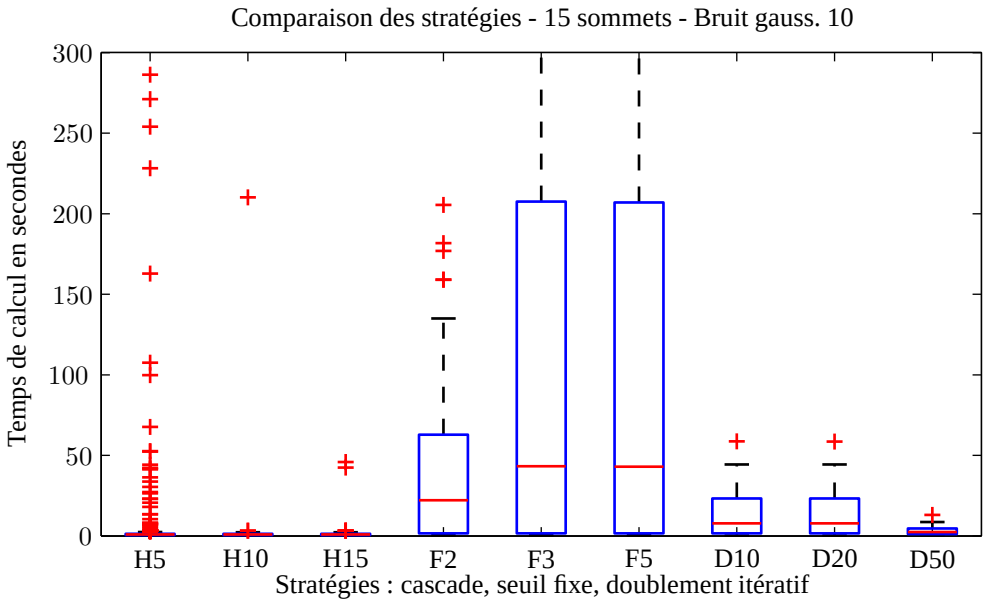

FIGURE 6.6 – Comparaison des différents stratégies, taille 15, bruit gaussien, niveau 10.

le résultat optimal. L'avantage en comparaison avec la méthode de Berretti simple, qui est basée sur la stratégie de recherche d'arbre A* d'Ullmann (1976), est que plusieurs sous-arbres peuvent être exclus plus tôt. Ainsi, le temps de calcul est fortement réduit, ce qui permet le traitement de graphes plus grands.

La figure 6.6 montre le temps de calcul des différentes méthodes sur des graphes de taille 15 soumis à un bruit gaussien avec un écart-type $\sigma = 0,1$. L'approche hybride qui combine l'algorithme génétique et la méthode A* est dénommée H. Le numéro correspond au nombre de générations sans changement que l'on utilise comme critère d'arrêt. La médiane du temps de calcul se situe dans les trois cas autour de 0,95 secondes : on obtient précisément 0,9523 pour H5, 0,9526 pour H10, et 0,9413 pour H15. Nous constatons néanmoins que le nombre des valeurs déviantes est visiblement réduit quand on choisit un critère d'arrêt plus relâché. Il est donc préférable d'attendre quelques générations de plus afin de permettre à l'algorithme génétique de converger.

Les trois colonnes notées F au centre correspondent à l'approche à seuil fixe. Nous avons choisi des seuils de deux, trois et cinq fois l'espérance mathématique. Ces deux dernières distributions sont quasi-identiques tandis que le temps de calcul est significativement plus petit quand on utilise le double de l'espérance.

Quant à la stratégie de doublement itérative (lettre D), nous commençons avec des seuils entre 10 et 50 fois plus petits que l'espérance. Nous constatons des temps de calcul très similaires avec les paramètres 10 et 20. En revanche le temps de calcul devient beaucoup plus petit quand on utilise le paramètre 50. Ceci peut paraître étrange au premier regard. En effet, quand on sous-estime le

précision.

	Bruit uniforme			Bruit gaussien		
Stratégie	$\sigma = 0,02$	$\sigma = 0,06$	$\sigma = 0,1$	$\sigma = 0,02$	$\sigma = 0,06$	$\sigma = 0,1$
H15	0,080	**0,13**	**2,0**	0,078	**0,10**	**0,94**
F2	**0,071**	0,72	67	**0,072**	1,0	22
F3	0,080	3,0	130	0,075	2,9	43
F5	0,078	5,3	130	0,074	5,2	43
D10	0,077	0,27	18	0,10	0,35	7,8
D20	0,078	0,27	18	0,10	0,35	7,8
D50	0,076	0,16	3,6	0,076	0,16	2,4

Tableau 6.3 – Médianes du temps de calcul en secondes, taille de graphes 15.

seuil d'un facteur de 10, alors on effectue en moyenne 6 itérations, pour les facteurs 20 et 50, on effectue respectivement 7 et 8 itérations. Le point commun entre les facteurs 10 et 20 est le coût de surestimation pendant la dernière itération. Soit a le facteur de sous-estimation et t le numéro de l'itération, alors le seuil se détermine par $2^{t-1} \times \frac{\mathbb{E}}{a}$. Pour la première itération dont le seuil est supérieur à l'espérance, il est 60% plus grand pour les facteurs 10 et 20, et seulement 28% pour le facteur 50. L'ajout d'itérations est beaucoup moins important.

Globalement, quand on compare les trois approches, on constate que l'algorithme hybride est toujours le plus rapide. Le deuxième est la stratégie de doublement suivie par A* avec seuil fixe.

La figure 6.7 montre le comportement des stratégies sur des graphes avec bruit uniforme. On constate que les temps de calcul sont plus élevés qu'en utilisant un bruit gaussien. Les médianes sont 1,75 au lieu de 0,94 secondes pour H15, 67,42 au lieu de 22,13 secondes pour F2, et 2,44 au lieu de 3,63 secondes pour D50. L'avantage de l'algorithme hybride s'accentue donc encore plus.

On peut noter également que les différences augmentent particulièrement avec le niveau de bruit (cf. Tab. 6.3, p. 144). Nous constatons que pour des niveaux de bruit très faibles, la médiane du temps de calcul est inférieure à un dixième d'une seconde.

Si l'on considère des graphes plus grands, par exemple de taille 25, on constate une médiane de 1,74 secondes pour l'algorithme hybride (H15) contre 1,83 secondes pour l'approche fixe (F2). La stratégie de doublement (D50) nécessite 1,73 secondes. Malgré la similitude des médianes, les variances sont plus élevées pour l'approche à seuil fixe. Par conséquent, l'utilisation des autres stratégies permet de réduire le nombre de valeurs déviantes.

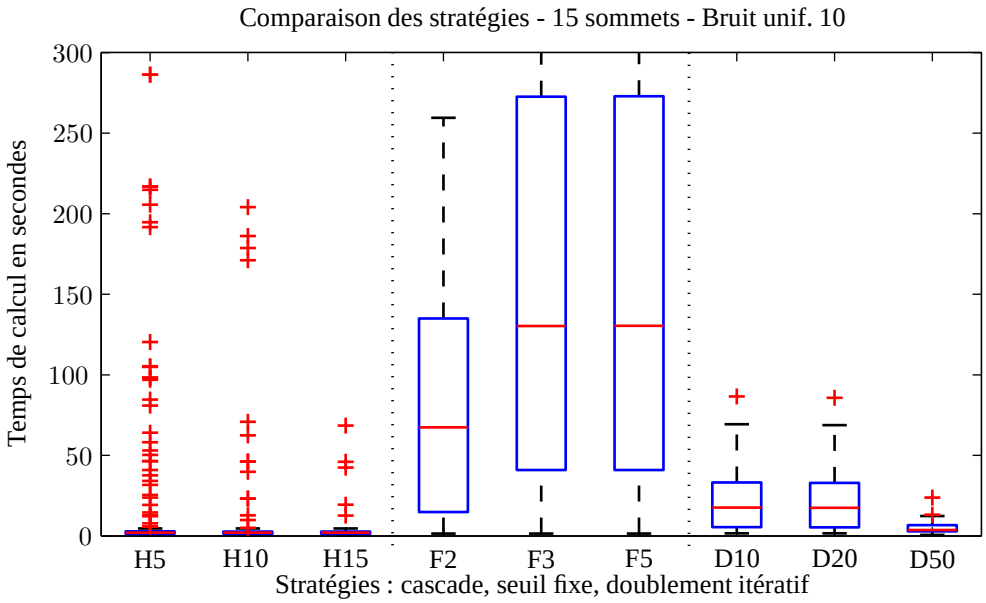

FIGURE 6.7 – Comparaison des différents stratégies, taille 15, bruit uniforme, niveau 10.

6.4 Bilan

Si l'on souhaite appliquer des algorithmes de recherche d'arbre (basés notamment sur A*) sur un problème inexact en pratique, il faut absolument limiter l'espace de recherche (d'une manière ou d'une autre). En général, ceci est fait en se limitant à des graphes très petits. Des algorithmes du type A* permettent aussi une limitation basée sur une distance maximale acceptable. Ceci mène à deux problèmes principaux : d'une part, la détermination du seuil qui n'est pas triviale. D'autre part, si la distance réelle est supérieure au seuil, on ne trouve aucun résultat. Nous avons comparé ici trois stratégies différentes. Premièrement, nous avons utilisé un seuil fixe établi sur la base de l'espérance mathématique de la distance. La deuxième stratégie se base sur un seuil adaptatif, initialisé à une valeur faible puis doublé jusqu'à ce que l'algorithme trouve une solution. On n'aura donc jamais un seuil dépassant deux fois la distance. Bien que cette approche ne soit pas toujours plus performante que celle du seuil fixe, notamment si le seuil utilisé par cette dernière est très strict, l'avantage principal réside dans la non-nécessité de connaître l'espérance mathématique de la distance. Ceci évite à l'utilisateur une étude détaillée des variances dans son domaine d'application. Aussi, une solution est toujours identifiée même si la distance est supérieure à l'estimation préalable. La troisième stratégie consiste à utiliser l'algorithme génétique afin de déterminer le seuil. Cette approche est encore plus générale, dans le sens que l'on peut l'appliquer sur de très nombreux problèmes réels, et on obtient un seuil par instance et non général. Elle est particulièrement avantageuse dans des environnements très bruités.

En ce qui concerne les algorithmes génétiques, bien que leur précision soit très élevée, il n'est

pas possible de garantir l'obtention de la solution optimale. Pourtant, une vérification de la solution par un algorithme A* est possible. Nous pouvons observer qu'en ajoutant cette étape, une garantie peut être formulée et le temps de calcul reste très bas par rapport à l'application de la recherche d'arbre simple. En effet, en fournissant une solution sous-optimale mais pas trop éloignée de l'optimum comme seuil à l'algorithme de Berretti-Ullmann, on peut accélérer celui-ci sensiblement. De cette façon, nous constatons qu'en moyenne l'exécution de l'algorithme génétique puis de l'algorithme de Berretti prend beaucoup moins de temps de calcul que l'exécution de l'algorithme de Berretti simple. Toutefois, le temps de calcul dans le pire cas reste exponentiel, notamment quand l'algorithme génétique ne trouve pas de solution sous-optimale assez bonne pour accélérer la recherche.

Chapitre 7

Applications

Dans ce chapitre, nous illustrons les résultats de notre algorithme sur trois applications concrètes. Nous introduisons d'abord le contexte général des applications aux images. Ensuite, nous présentons les résultats pour deux applications dans ce domaine. La première s'inscrit dans le cadre de l'identification de personnes. Nous apparions des graphes extraits des photos en deux dimensions avec ceux obtenus à partir des modèles en trois dimensions de leurs visages. La deuxième application vise l'identification d'objets basée sur des graphes *a priori* sans attributs. Enfin, nous illustrons le fait que notre algorithme fonctionne également pour le problème d'isomorphisme exact, dans le cas de molécules.

7.1 Images

Le recherche d'images est souvent associée à des sémantiques de haut niveau. Les méthodes actuelles cherchant à réduire le fossé sémantique incluent l'utilisation d'ontologies, l'apprentissage automatique des relations entre les caractéristiques de bas niveau avec des concepts sémantiques, les boucles de pertinence, des *templates* sémantiques et la fusion de données, notamment le texte associé à une image (un résumé de toutes ces méthodes est donné dans Liu et al., 2007). Nous proposons d'exploiter une information de relations représentées par des graphes.

Nous introduisons d'abord des pistes générales de l'utilisation de graphes dans le domaine d'images avant d'illustrer nos applications à deux problèmes précis.

7.1.1 Contexte général

Suivant l'idée qu'il y a plusieurs « objets » sur une image qui sont en relation les uns avec les autres, on essaie de découper les images et de construire un graphe à partir de ses parties. Il existe deux méthodes pour identifier les entités (sommets) dans l'image. D'une part, la segmentation identifie des régions (Shi & Malik, 2000; Boykov et al., 2001; Boykov & Kolmogorov, 2004;

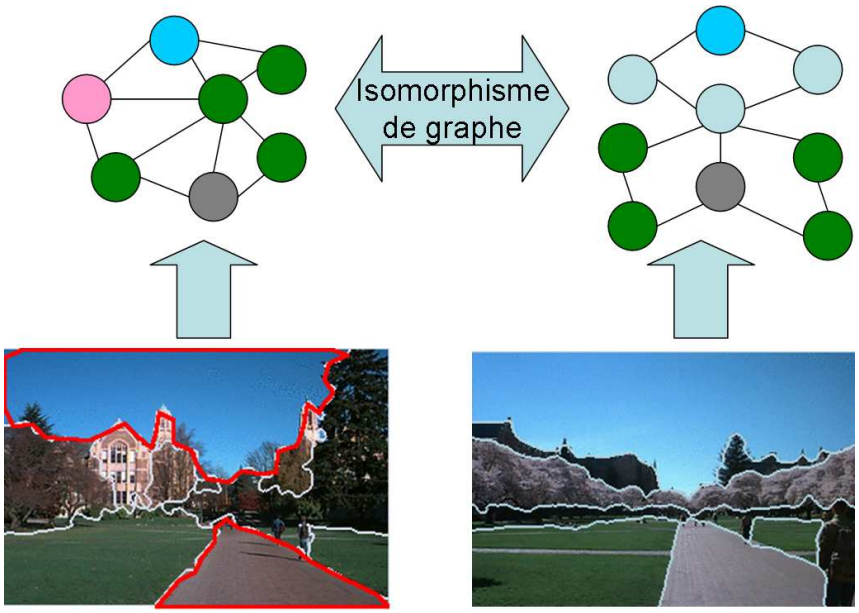

FIGURE 7.1 – Isomorphisme de graphe à partir des régions, images de Omhover & Detyniecki (2006).

Felzenszwalb & Huttenlocher, 2004). D'autre part, des approches d'extraction des points d'intérêts identifient un ensemble de points pertinents (Mikolajczyk & Schmid, 2002, 2005). Dans le cas de la segmentation, nous sommes confrontés à trois défis : la définition des régions et donc la segmentation automatique de l'image, le choix des caractéristiques à extraire pour chaque région et finalement la question de la description des relations entre régions. La figure 7.1 montre schématiquement le problème de l'isomorphisme de sous-graphes entre deux images. Pour nos expérimentations, nous considérons la description de l'image comme graphe comme étant donnée, afin que nous puissions nous concentrer pleinement sur le problème d'optimisation de l'appariement.

En ce qui concerne la modélisation par points d'intérêts, la définition des relations devient plus délicate. On obtient en premier lieu, un ensemble de points, non connectés, et sans notion explicite de voisinage. Il faut donc connecter les points afin d'obtenir un graphe. Une méthode courante est l'utilisation d'un algorithme de triangulation, par exemple une triangulation de Delaunay (Luo & Hancock, 2001). La construction d'un graphe de voisinage est également possible. La question qui se pose par la suite est : est-ce que l'introduction des relations apporte plus d'informations ? A notre avis ceci n'est pas le cas. Par ailleurs, si l'on considère une triangulation de Delaunay, la structure du graphe peut être sensiblement modifiée par un petit déplacement d'un seul point. Ceci limite la robustesse de l'approche qui est très importante pour l'application aux images.

Dans le domaine de la reconnaissance de formes, on parle aussi d'appariement des graphes pour un problème spécifique : on pourrait le dénommer l'appariement de dessins de traits, une application principale étant la reconnaissance optique de caractères (Auwatanamongkol, 2007). Le terme graphe

est utilisé dans son sens géométrique. Il est toujours planaire (l'intersection de deux traits est identifiée comme un sommet durant l'étape d'extraction de graphe). De plus, les angles entre les traits sont très souvent utilisés (pour des graphes généraux la notion d'angles n'existe pas). Le modèle est donc très sensible aux déformations et beaucoup plus spécifique que le problème d'isomorphisme de graphes.

Un autre problème similaire est le problème de l'appariement d'ensembles de points (*point pattern matching*, Gold et al., 1998; Caetano et al., 2006). Il consiste à trouver une correspondance entre deux ensembles de points dans un espace euclidien à deux ou trois dimensions. Par rapport au problème d'appariement de graphes il y a deux différences clefs. D'une part, il n'y a pas d'arêtes. D'autre part, la correspondance recherchée est une isométrie (approximative dans le cas inexact). L'application la plus courante est l'appariement des points d'intérêts, comme par exemple des SIFT (Lowe, 2004). En pratique le problème est souvent inexact et requiert des méthodes d'optimisation. Les méthodes utilisées incluent des approches basées sur les plus proches voisins (Beis & Lowe, 1997, 1999), des approches qui estiment la position et l'orientation des objets dans l'image et en dérivent les correspondances par exemple en utilisant *softassign* (Gold et al., 1998), ou encore des méthodes spectrales (Carcassoni & Hancock, 2003). Demirci et al. (2006) abordent le problème des appariements *many-to-many* avec l'algorithme EMD (*Earth Mover's Distance*). Caetano et al. (2006a ; 2006b) proposent un changement de point de vue intéressant : au lieu d'utiliser un modèle exact et des algorithmes inexacts, ils suggèrent l'utilisation d'un modèle inexact et d'un algorithme exact qui a une complexité polynomiale grâce aux contraintes imposées.

En ce qui concerne les autres applications existantes, on peut citer la combinaison des graphes avec la logique floue qui peut intervenir non seulement dans la description d'images par le graphe mais aussi dans l'algorithme d'appariement (Krishnapuram et al., 2004). Les modèles de graphes sont également utilisés pour la régularisation des images de couleur, notamment afin de réduire le bruit et d'améliorer la qualité (Lezoray et al., 2007).

On peut également passer au traitement des vidéos. Chevalier et al. (2007) proposent l'utilisation des graphes d'adjacence de régions qui changent avec le temps pour la reconnaissance d'objets dans des vidéos de basse résolution. Enfin, on peut modéliser l'aspect temporel de la vidéo avec un graphe, par exemple pour le résumer (Ngo et al., 2005).

7.1.2 Images 3D

Notre première application dans le domaine des images se place dans le cadre de l'identification de personnes : nous utilisons une base fixée de visages à reconnaître. Nous disposons concrètement des modèles 3D des visages et tentons d'identifier la bonne personne avec une photo en deux dimensions uniquement. Il s'agit d'un problème d'isomorphisme de sous-graphes parce que des occlusions sont omniprésentes. De plus, le niveau de bruit est très élevé car les photos et les modèles sont issus

de deux processus différents. Ceci rend l'application des méthodes exactes très délicate et favorise plutôt notre approche génétique.

La base contient les modèles 3D de 61 visages ainsi que plusieurs photographies de chaque personne. Berretti et al. (2008) nous fournissent avec la description de chaque image sous forme d'un graphe. Les attributs des nœuds et des arêtes sont des vecteurs de taille 27. De fait, ceci correspond à un coefficient par relation spatiale pour chaque direction. Les graphes ont été réduits afin de pouvoir être traités avec les méthodes existantes et contiennent en général 12 sommets. Ils sont donc assez petits, ce qui défavorise l'application des algorithmes génétiques à cause de leur coût constant d'initialisation. Par ailleurs, nous sont données les mesures de distances pour les sommets et les arêtes, correspondantes à cette description.

Pour chaque modèle 3D, deux vues frontales sont disponibles, la première avec une expression neutre et la deuxième avec un sourire. Les correspondances sont connues. Avant de se poser la question de la qualité d'un algorithme en particulier, il faut d'abord étudier la pertinence de la description. En effet, si l'on fait un classement des plus proches modèles par rapport à une image donnée, il est fortement souhaitable que le meilleur modèle décrive la même personne. Nous constatons que si l'on utilise la totalité des sommets de chaque graphe, alors ceci n'est pas toujours le cas. Par conséquent, on peut se tromper de la personne, même si l'on a le résultat optimal numérique. Toutefois, si l'on considère uniquement les sous-graphes composés des premiers neuf nœuds dans l'ordre donné, alors la personne peut être identifiée correctement dans tous les cas (pour cette base) si son expression est neutre. Si la personne sourit, cela devient moins pertinent.

Nous effectuons deux tests séparés. La différenciation se fait selon l'existence d'une correspondance réelle entre les images associées aux graphes. Les cas sont nommés par conséquent positifs, si les graphes correspondent à la même personne et négatifs, sinon.

Notre référence est l'algorithme A* (Berretti et al., 2001) qui garantit l'optimalité de résultats. Il est très rapide dans le cas positif, où les distances sont très petites. Dans le cas négatif, il converge beaucoup plus lentement du fait de la distance qui conduit à des seuils internes plus élevés. Dans le cas positif, 90% des exécutions se terminent dans un délai très court. Cependant, le temps de convergence des 10% restants est très variable. Dans le cas négatif, la variance du temps est beaucoup plus élevée.

La précision des approches génétiques est de 100%, avec 30 tests lancés par paire (2D/3D) : parmi environ 2000 exécutions indépendantes faites par opérateur, nous n'avons pas constaté une seule erreur. Les figures suivantes (7.2 à 7.6, p. 151 à 153) illustrent les résultats avec une sélection des différents opérateurs. Nous constatons que les temps de calcul sont très faibles, de l'ordre de 50 millisecondes [14].

14. Avec cet ordre de grandeur, il est important de vérifier la précision des mesures de temps utilisées. La précision de l'horloge dépend du système d'exploitation, Windows XP, par exemple, a une résolution de 15 millisecondes. Nous avons effectué les tests sous des noyaux Linux fournissant une précision d'une milliseconde. Afin d'éviter toute influence

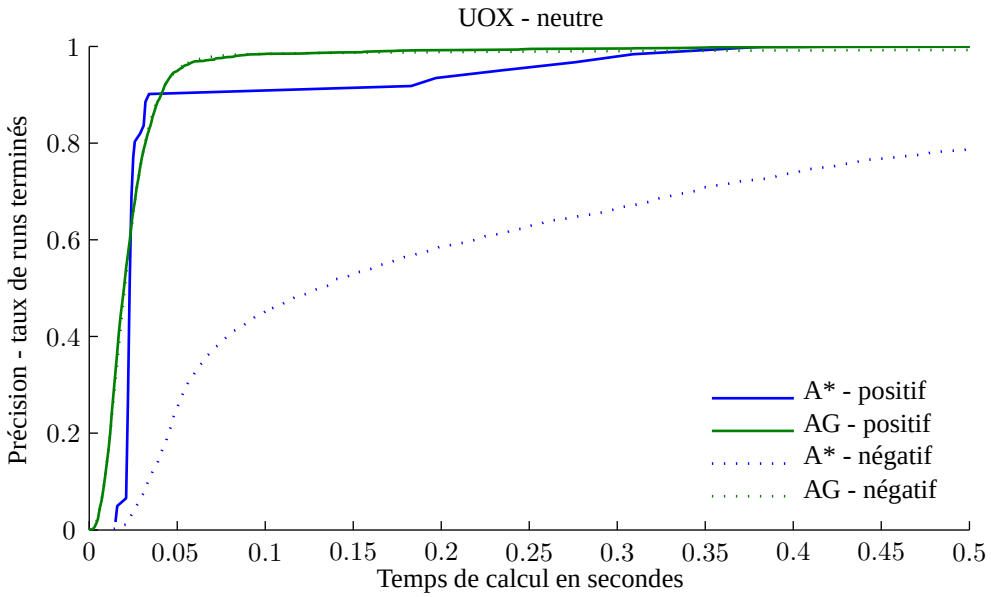

FIGURE 7.2 – Images 3D - expression neutre - UOX.

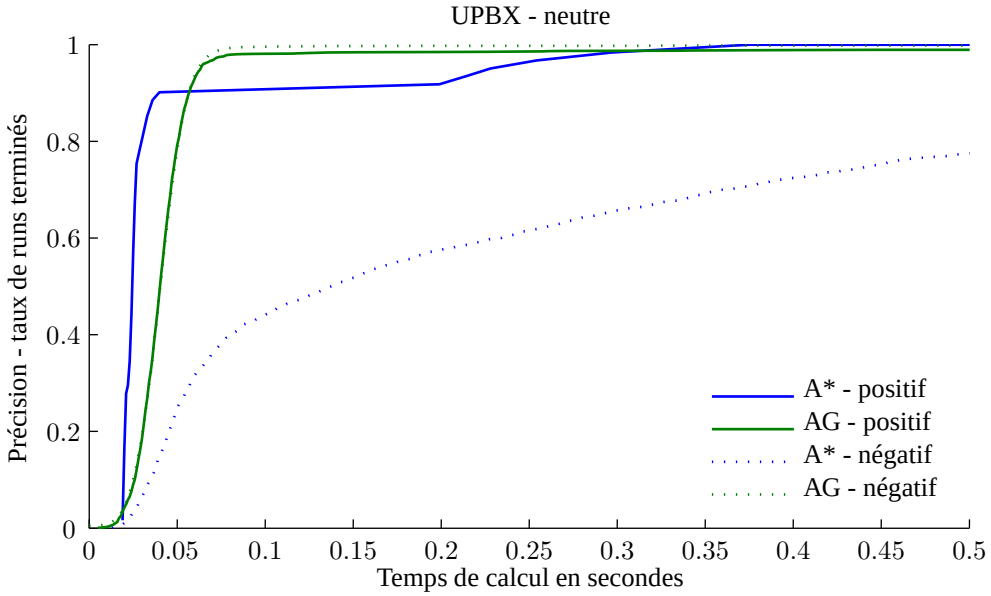

FIGURE 7.3 – Images 3D - expression neutre - UPBX.

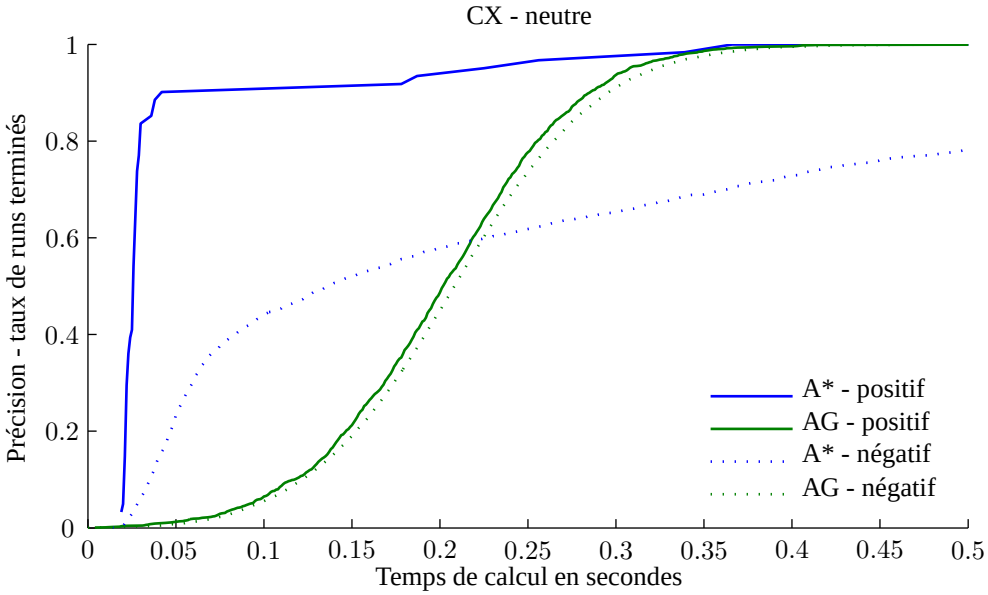

FIGURE 7.4 – Images 3D - expression neutre - CX.

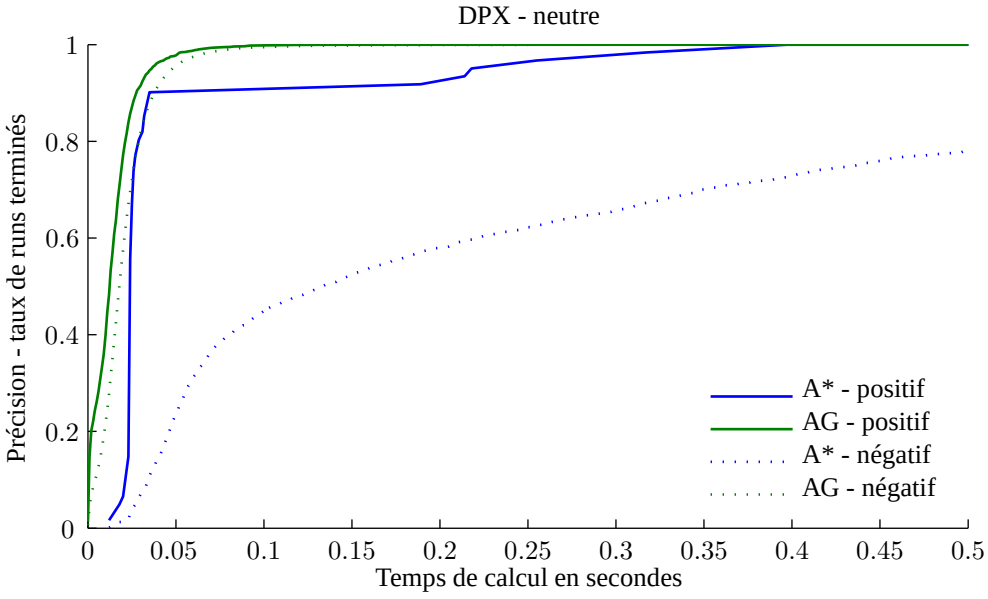

FIGURE 7.5 – Images 3D - expression neutre - DPX.

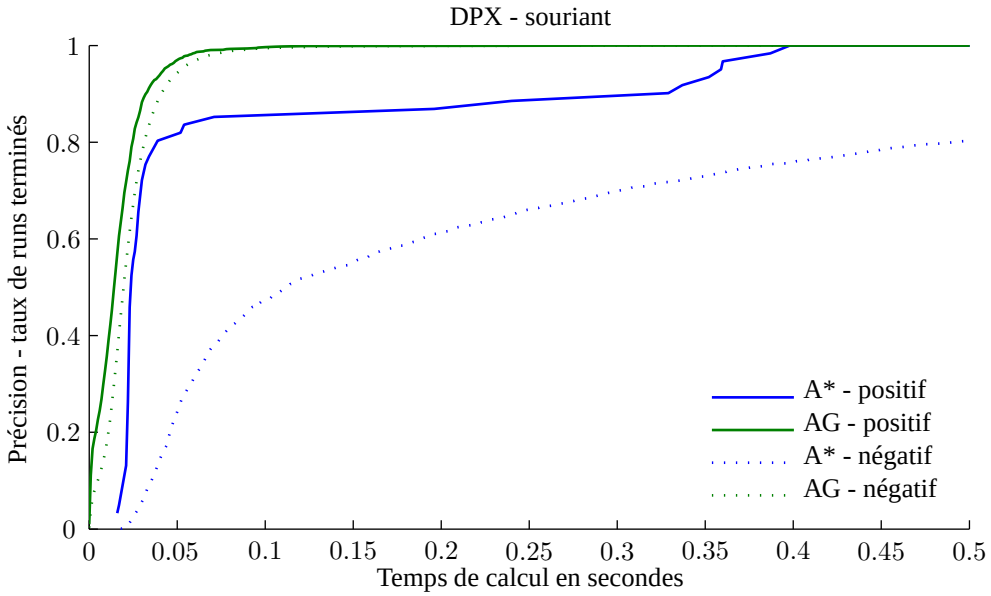

FIGURE 7.6 – Images 3D - expression souriante - DPX.

7.1.3 COILDel

La base Columbia Object Image Library (COIL) est une base d'images destinée à la reconnaissance de formes (Nene et al., 1996). Elle contient 100 objets différents. Chaque objet est photographié sous tous les angles entre 0°et 355°par pas de 5°. Il y a donc un total de 72 images par objet. Les objets se trouvent isolés sur les photos sur fond noir. Il n'y a donc pas d'effets d'occlusion et pas de nécessité d'identifier quel est (et où se trouve) l'objet dans l'image.

Riesen & Bunke (2008) utilisent un détecteur de contours afin d'extraire un ensemble de points de chaque image. Une triangulation de Delaunay connecte par la suite les points afin de former un graphe. Ce dernier est non orienté et ne contient pas d'étiquettes. Riesen & Bunke utilisent la distance d'édition de graphes pour déterminer la distance entre deux graphes. Avec un k plus proches voisins (k-NN) basé sur cette mesure de distance, les auteurs obtiennent une précision de 93,3%. Au premier regard ceci peut paraître très bon, mais il faut savoir que l'ensemble d'apprentissage contient tous les objets à intervalles de rotation de 15°. Si l'on souhaite classer un objet de l'ensemble de test, on sait qu'il y a exactement le même objet à 5°et à 10°de rotation dans l'ensemble d'apprentissage. Ces deux exemples sont tellement proches de la requête que la probabilité qu'ils soient les deux voisins les plus proches est assez élevée, quels que soient les descripteurs visuels et la distance. Si l'on choisit par exemple $k = 3$, alors même s'il y a un autre objet plus proche que ces deux exemples, la classification sera correcte. En pratique, par contre, il serait intéressant de voir si l'on peut classer un objet sous la condition qu'il n'existe qu'une seule fois dans la base (avec une rotation arbitraire) ;

gênant la précision, nous avons effectué les tests dans un environnement séquentiel sur un seul processeur.

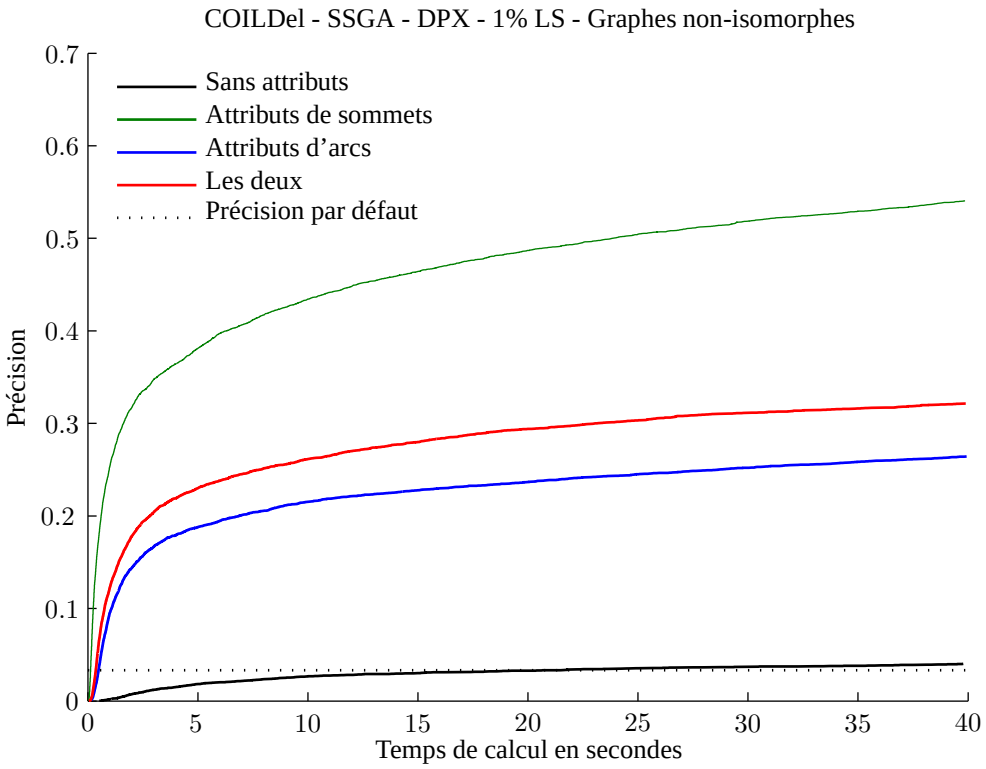

FIGURE 7.7 – Histogramme DPX - COILDel.

d'autant plus qu'afin d'exécuter le k-NN, il faut *a priori* comparer chaque nouvel objet à chaque objet de la base d'apprentissage. Il est en général possible d'éliminer une partie de ses comparaisons en structurant la base. Le scénario utilisé nécessite 2400 (moins ce que l'on peut éliminer) calculs de la distance d'édition de graphe qui est NP-difficile. En revanche, si chaque objet n'existait qu'une fois, un maximum de 100 comparaisons serait suffisant.

La structure des graphes construits par une méthode de triangulation est fortement perturbée par des changements de points. Un très léger déplacement d'un point peut mener au changement de pratiquement tous les arcs autour de lui. Ainsi l'introduction ou la suppression d'un point peut avoir des effets même sur des points lointains. On peut se poser la question de savoir si un graphe issu d'une triangulation apporte un gain d'information par rapport à l'ensemble de points sur lequel il se base. Il est évident qu'une approche structurelle est beaucoup plus adaptée à ce problème qu'une approche dont les déformations structurelles possibles sont limitées.

Notre objectif est d'étudier le problème d'isomorphisme sur des données réelles. Nous appliquons donc notre algorithme sur ce problème, même si sa limitation sur certains changements structurels ne semble pas avantageuse.

Nous utilisons quatre mesures de distance différentes. Dans la première, chaque différence d'arc (présent-non présent) augmente la distance de un. Dans la deuxième, nous attribuons à chaque arc

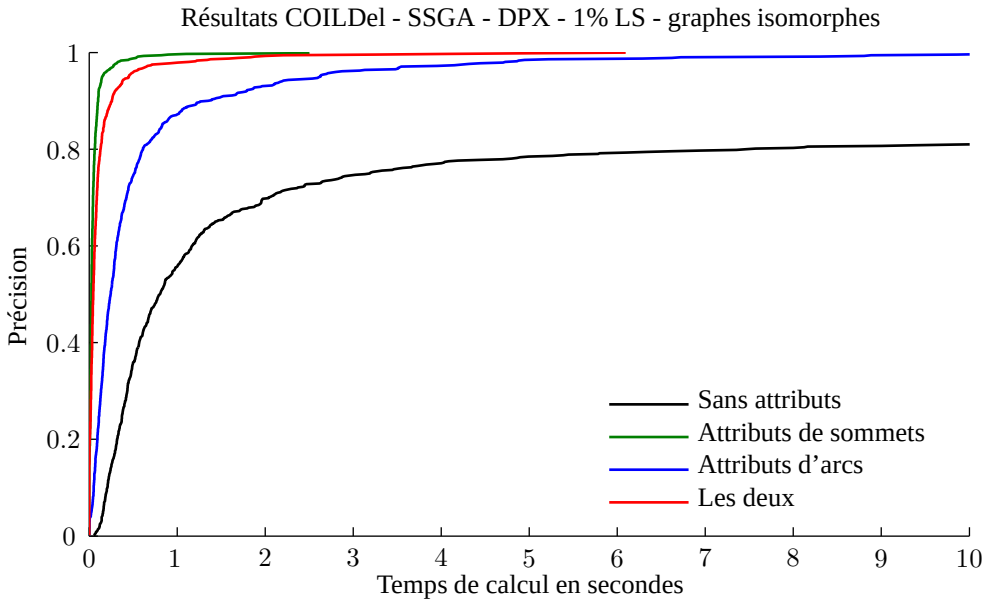

FIGURE 7.8 – Histogramme DPX - COILDel - cas isomorphes.

la distance euclidienne entre les sommets qu'il connecte. La troisième distance prend en compte les coordonnées des sommets et en calcule la distance euclidienne. La dernière est une somme pondérée de la deuxième et de la troisième.

Nous nous intéressons surtout à la question de savoir si l'on retrouve la bonne distance entre deux graphes de la base donnés et non à la précision d'une éventuelle classification qui serait l'étape suivante. D'abord, il faut noter que, dans la base, il n'y a pas de graphes exactement isomorphes, sauf quand on compare un graphe à lui-même, et que nous ne possédons pas de valeurs de référence : les distances sont *a priori* inconnues. L'application d'une méthode exhaustive afin de les déterminer prendrait trop de temps, étant donné la taille de graphes (en moyenne 21 sommets avec un maximum de 77). Nous prenons donc la meilleure valeur trouvée pour une paire de graphes parmi les 30 exécutions de l'algorithme comme référence. Ceci ne donne pas de garantie sur l'optimalité mais il est probable que, si cette valeur est trouvée souvent, elle est la bonne. La précision par défaut est alors $1/30$, ce qui ne veut pas dire que les solutions trouvées sont réellement exactes. Par contre, si l'on n'arrive même pas à retrouver le même optimum lors de plusieurs exécutions, on peut conclure que l'algorithme fonctionne mal sur ce problème. Pour limiter le temps de calcul, nous effectuons les tests préliminaires uniquement sur les images qui se trouvent dans l'ensemble d'apprentissage, défini par Riesen & Bunke, pour un seul objet. Au total nous effectuons 9000 exécutions de l'algorithme.

La figure 7.7 montre l'histogramme de la précision globale en fonction du temps, plus précisément en fonction du nombre d'appels à la fonction d'évaluation. Nous constatons que la précision reste très faible malgré un temps considérable donné à l'algorithme (100 millions d'évaluations de la fonction d'évaluation, ce qui prend autour de 20h sur 8 processeurs par mesure, donc plus de 500h

de temps de calcul pour un processeur). Le résultat obtenu est similaire à celui d'une recherche uniforme pour des graphes sans attributs. La meilleure performance (sur un niveau de précision toujours très faible) est obtenue avec la distance euclidienne entre les coordonnées des sommets. Quant aux mesures utilisant des attributs sur les arcs, elles se trouvent entre les deux. En général, on peut conclure que les différences de structure évitent la convergence, surtout en combinaison avec une fonction d'évaluation ayant de nombreux optima locaux dispersés dans l'espace (les triangulations sont très sensibles au bruit). Aussi, au lieu d'exécuter notre algorithme génétique sur un graphe créé de cette façon, il apparaît préférable d'effectuer directement l'alignement entre les points.

En ce qui concerne les cas dont la distance est zéro, c'est-à-dire les graphes identiques, alors la précision atteint 100%, si l'on prend en compte les attributs de sommets. Le temps de convergence a aussi un ordre de grandeur beaucoup plus petit que dans le cas précédent. Si l'on n'utilise que les attributs d'arcs, alors il reste 20% des cas où la solution n'est pas trouvée. La distance sans attributs, l'algorithme ne converge toujours pas.

7.2 Molécules

Nous mettons en œuvre notre algorithme sur une base de molécules. Il s'agit d'une base publique du *National Cancer Institute* contenant des composés chimiques qui sont susceptibles d'avoir des effets positifs pour la lutte contre le SIDA. Les données sont sous la forme d'un fichier au format MDL SDFILE. Cette base contient 42687 molécules avec des tailles entre 2 et 438 atomes. La taille moyenne est de 45,7 atomes. Le problème ici est celui d'un isomorphisme de graphes exact, c'est-à-dire que, dès qu'un élément ou une liaison diffère, cela entraîne un changement brusque des propriétés de la molécule.

Avant de passer à la résolution du problème de l'isomorphisme de graphes, nous pouvons exploiter certaines caractéristiques des molécules. Les attributs des sommets ainsi que ceux des arêtes appartiennent à un domaine discret et fini : les sommets sont des éléments chimiques ; leurs liaisons sont décrites par des nombres entiers généralement petits.

7.2.1 Filtrage par signature

Nous effectuons un filtrage préalable basé sur la comparaison des signatures des molécules. La signature atomique comporte uniquement le nombre de sommets de chaque élément chimique (l'ordre étant donné par le tableau périodique des éléments). Nous obtenons par exemple 2H1O et C1O2 pour l'eau et le dioxyde de carbone respectivement. Si deux molécules n'ont pas la même signature, alors elles ne peuvent pas être isomorphes. Dans le cas contraire, il faut toutefois procéder à l'appariement structurel. Nous avons obtenu 27381 signatures différentes. Le tableau 7.1 montre en détail la distribution des molécules selon leurs signatures. On construit des groupes contenant les

molécules de même signature, le tableau indique le nombre de groupes de chaque taille : ainsi avec la signature atomique, on obtient 20370 groupes à une molécule soit 20370 molécules qui peuvent être identifiées directement et pour lesquelles la signature est un critère suffisant. Par contre, on observe aussi 3846 groupes à deux molécules, c'est-à-dire 3846 signatures pour lesquelles deux molécules possibles sont à tester, 1382 groupes à trois molécules, pour lesquelles trois possibilités sont à tester, etc. Le nombre maximal de molécules de même signature est 20. Par conséquent, il n'est jamais nécessaire d'effectuer plus de 20 comparaisons structurelles pour identifier une molécule (ou décider qu'elle n'est pas encore présente dans la base). En fait, sans l'extraction d'une signature, quand les molécules sont filtrées par leur taille uniquement, le nombre d'appariements nécessaires dépasse mille selon la taille (cf. la deuxième colonne du Tab. 7.2, p. 161). Aussi les graphes les plus grands nécessitent moins de comparaisons : par exemple les groupes avec le plus grand graphe sont (taille de graphes, nombre de graphes ayant une signature atomique identique) : (245,3),(228,2),(221,2),(217,3),(216,2),(205,2). La totalité des groupes contenant au moins dix molécules comporte des molécules entre 22 et 69 atomes.

Il est possible de rendre la signature plus pertinente en utilisant plus d'informations sur les sommets que l'élément chimique. L'idée générale est de substituer aux étiquettes de sommets des étiquettes plus discriminantes. Aussi, dans l'algorithme Nauty [15] de McKay (1981) un ordonnancement des sommets basés sur ces nouvelles étiquettes est utilisé. Les principales informations que l'on peut utiliser pour enrichir les étiquettes sont les étiquettes des arêtes incidentes ainsi que les étiquettes des sommets voisins. Zampelli et al. (2007) proposent de renforcer l'étiquetage en regardant les étiquettes des voisins dans un rayon plus grand : au lieu de prendre en compte uniquement les voisins directs, ils prennent en compte tous les sommets ayant au maximum une distance α du sommet donné.

Nous proposons deux étiquetages de ce type, qui mènent à la signature cumulée et la signature binaire respectivement. Les étiquettes se calculent à partir de l'élément chimique et du degré, c'est-à-dire le nombre de liaisons partant de cet atome. Si l'on considère uniquement les liaisons covalentes qui sont les plus courantes dans des molécules (une exception étant par exemple les cycles aromatiques), il existe des liaisons doubles et des liaisons triples. Le nombre total de liaisons dépend le plus souvent des propriétés chimiques de l'atome uniquement et n'apporte donc pas d'information complémentaire. Un étiquetage par le degré sous forme du nombre total de liaisons (la signature cumulée) ne peut donc pas (sauf exceptions) beaucoup améliorer le filtrage. Si l'on suppose par contre un étiquetage qui compte les atomes auxquels l'atome est connecté directement (que l'on appelle signature binaire car les liaisons sont binarisées), ceci peut différer pour un même atome, car il change de valeur selon la structure. L'acide éthanoïque (cf. Fig. 7.9, p. 158) contient deux carbones et deux oxygènes. Le nombre total de liaisons est quatre pour les carbones et deux pour les oxygènes (chiffres en index sur la partie gauche). Par conséquent, nous ne pouvons pas

15. Disponible à l'adresse : http ://cs.anu.edu.au/ bdm/nauty/.

distinguer entre les deux atomes de carbone avec ce degré. Par contre, si l'on utilise le nombre des voisins, c'est-à-dire si l'on compte uniquement s'il y a une liaison d'un type quelconque ou non, alors les atomes deviennent discernables. Les valeurs (4 et 3 respectivement) sont données en index du symbole chimique sur la partie droite.

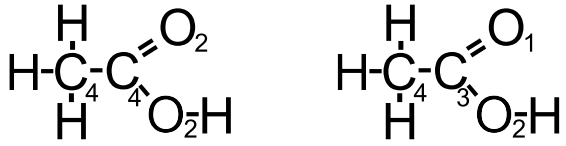

FIGURE 7.9 – Acide éthanoïque (acétique).

Filtrage par signature

Nous examinons par la suite jusqu'à quel point le filtrage de la base est possible avec les différentes signatures. Le tableau 7.1 illustre le nombre de groupes répartis selon leur taille. La première ligne contient notamment le nombre de molécules qui peuvent être identifiées directement par la signature. La deuxième ligne contient le nombre de paires de molécules qui ne sont pas distinguables, etc. Nous constatons que même avec la signature qui mène au filtrage le plus faible, aucune molécule ne nécessite plus de 20 comparaisons. La signature cumulée améliore légèrement le filtrage. Quant à la signature binaire, le nombre de molécules détectées directement augmente de 50% et la taille maximale d'un groupe se réduit à 14. La dernière ligne du tableau donne le nombre moyen de comparaisons à faire par molécule (et non par groupe). Il faut noter que même si la signature de la molécule est unique, nous considérons que l'appariement est quand même nécessaire, car il pourrait s'agir d'une molécule inconnue dans la base.

7.2.2 Approche par couleur

Les mêmes idées appliquées afin de filtrer la base et réduire le nombre d'appariements à effectuer peuvent être utilisées pour réduire l'espace de recherche du calcul de l'appariement en soi. Ceci permet alors de réduire le temps de calcul de l'appariement lui-même. En effet, dans la procédure de recherche nous pouvons utiliser les étiquettes afin de formuler des contraintes dures (parce que nous nous situons dans un cas exact) : intuitivement, un atome du type A ne peut être apparié qu'avec un atome du même type.

Le tableau 7.2 montre la réduction de l'espace de recherche que l'on obtient sur la base. Pour donner un exemple : la taille de l'espace de recherche pour l'appariement de la molécule de l'acide éthanoïque dont la taille est 8, sans utilisation d'étiquettes, est $8! = 40320$. Il existe donc 40320 appariements possibles. Quand on introduit la contrainte d'apparier uniquement les mêmes atomes, le nombre d'appariements diminue à $4!2!2! = 96$.

On peut encore faciliter la recherche en utilisant des caractéristiques plus discriminantes comme contraintes, comme dans les signatures binaires. L'étiquetage par degré binaire arrive à discriminer les deux C ainsi que les deux O, il n'y a que $4!1!1!1!1! = 24$ appariements possibles. Par contre, l'espace de recherche reste le même pour l'étiquetage par degré cumulé parce que l'on ne distingue pas les atomes O et C.

Nous calculons la taille de l'espace de recherche pour chaque atome afin de déduire un résumé pratique sur la base. Pour chaque étiquetage la taille minimale, moyenne et maximale de l'espace de recherche sont indiquées dans le tableau 7.2. Selon la réduction apportée, les trois étiquetages sont dans l'ordre suivant : signature atomique, signature cumulée, signature binaire. Néanmoins, le nombre d'appariements minimal de l'étiquetage cumulé est parfois inférieur à celui de l'étiquetage binaire, notamment pour les tailles 30, 50, 60 et 80. Quant à la moyenne et au maximum, ceci n'est pas le cas. Il existe alors des molécules qui peuvent être mieux différenciées avec le degré cumulé qu'avec le degré binaire. Il s'agit probablement de molécules ayant des liaisons spécifiques non covalentes. Comme elles sont assez rares, l'influence sur l'ensemble est négligeable.

Taille de groupes	Nombre de groupes		
	Signature atomique	Signature cumulée	Signature binaire
1	20370	22322	31168
2	3846	3728	3557
3	1382	1260	795
4	670	593	222
5	375	328	111
6	251	227	41
7	157	137	17
8	110	88	5
9	76	64	7
10	45	36	6
11	29	26	1
12	25	19	2
13	14	13	0
14	8	7	1
15	6	5	0
16	2	3	0
17	5	5	0
18	6	5	0
19	3	3	0
20	1	1	0
Moyenne	2.82	2.61	1.48

Tableau 7.1 – Distribution des molécules selon leurs signatures : nombre de groupes de molécules de même signature de chaque taille de groupe. La moyenne correspond au nombre moyen de comparaisons qu'il faut effectuer pour vérifier si la molécule est dans la base.

Taille	Nombre		Signature atomique			Signature cumulée			Signature binaire		
			min	moy	max	min	moy	max	min	moy	max
10	21	3,63E006	7,20E001	2,18E003	1,73E004	7,20E001	2,17E003	1,73E004	1,20E001	2,59E002	1,44E003
20	389	2,43E018	4,15E005	3,18E014	1,22E017	4,15E004	4,43E010	1,61E012	4,15E004	2,21E010	1,61E012
30	1144	2,65E032	9,93E014	2,26E024	2,42E027	1,79E009	4,74E021	1,82E024	6,69E010	8,96E020	4,41E023
40	1118	8,16E047	1,97E023	5,31E035	3,79E038	7,93E021	5,78E033	5,41E035	5,21E019	1,97E033	5,41E035
50	798	3,04E064	3,15E034	4,03E049	1,61E052	8,07E022	9,75E046	3,23E049	1,76E028	2,60E046	1,14E049
60	448	8,32E081	8,65E046	1,50E062	2,67E064	3,38E035	1,50E062	2,67E064	1,43E037	2,83E061	2,54E063
70	238	1,20E100	2,08E058	3,00E086	7,14E088	1,05E055	8,26E076	1,24E079	1,17E050	6,53E076	1,24E079
80	114	7,16E118	1,37E071	1,29E092	6,59E093	1,27E059	1,29E092	6,59E093	1,85E062	3,79E089	3,57E091
90	76	1,49E138	7,72E088	9,38E106	6,68E108	6,60E086	9,38E106	6,68E108	2,02E081	3,01E101	1,66E103
100	35	9,33E157	3,99E103	1,06E120	2,10E121	3,99E103	1,06E120	2,10E121	2,13E092	2,79E116	8,46E117
110	32	1,59E178	2,11E125	1,26E143	4,05E144	3,77E123	1,26E143	4,05E144	3,51E110	2,81E135	8,85E136
120	31	6,69E198	4,87E135	1,49E158	4,62E159	4,87E135	1,49E158	4,62E159	1,22E118	2,01E154	6,23E155
130	18	6,47E219	1,79E150	3,29E172	4,18E173	1,79E150	3,29E172	4,18E173	1,14E132	2,62E170	4,71E171
140	9	1,35E241	9,82E173	1,03E191	8,06E191	9,82E173	1,03E191	8,06E191	7,63E155	9,55E178	8,60E179
150	6	5,71E262	6,35E181	9,46E200	3,85E201	6,35E181	9,46E200	3,85E201	5,50E160	3,61E183	1,81E184
160	6	4,71E284	2,62E208	3,55E219	1,63E220	2,62E208	3,55E219	1,63E220	1,19E189	5,52E215	1,66E216
170	3	7,26E306	8,11E218	9,75E235	2,93E236	8,11E218	9,75E235	2,93E236	2,68E196	9,75E235	2,93E236
180	3	2,01E329	2,35E242	2,07E245	3,10E245	2,35E242	2,07E245	3,10E245	1,21E219	1,53E224	2,30E224
190	2	9,68E351	2,49E271	2,36E289	4,73E289	2,49E271	2,36E289	4,73E289	1,32E249	3,02E268	6,05E268
200	2	7,89E374	1,49E269	9,20E278	1,84E279	4,97E268	9,20E278	1,84E279	4,65E242	2,77E254	5,53E254
210	2	1,06E398	4,17E299	1,23E307	2,46E307	4,17E299	1,23E307	2,46E307	6,12E279	1,25E294	2,49E294
220	1	2,28E421	4,31E323	4,31E323	4,31E323	4,31E323	4,31E323	4,31E323	3,05E304	3,05E304	3,05E304
230	1	7,76E444	1,06E334	1,06E334	1,06E334	1,06E334	1,06E334	1,06E334	5,95E310	5,95E310	5,95E310
250	1	3,23E492	1,10E366	1,10E366	1,10E366	1,10E366	1,10E366	1,10E366	1,24E339	1,24E339	1,24E339
390	1	6,84E842	9,51E648	9,51E648	9,51E648	9,51E648	9,51E648	9,51E648	7,86E589	7,86E589	7,86E589

Tableau 7.2 – Comparaison de la taille de l'espace de recherche en utilisant les différentes signatures.

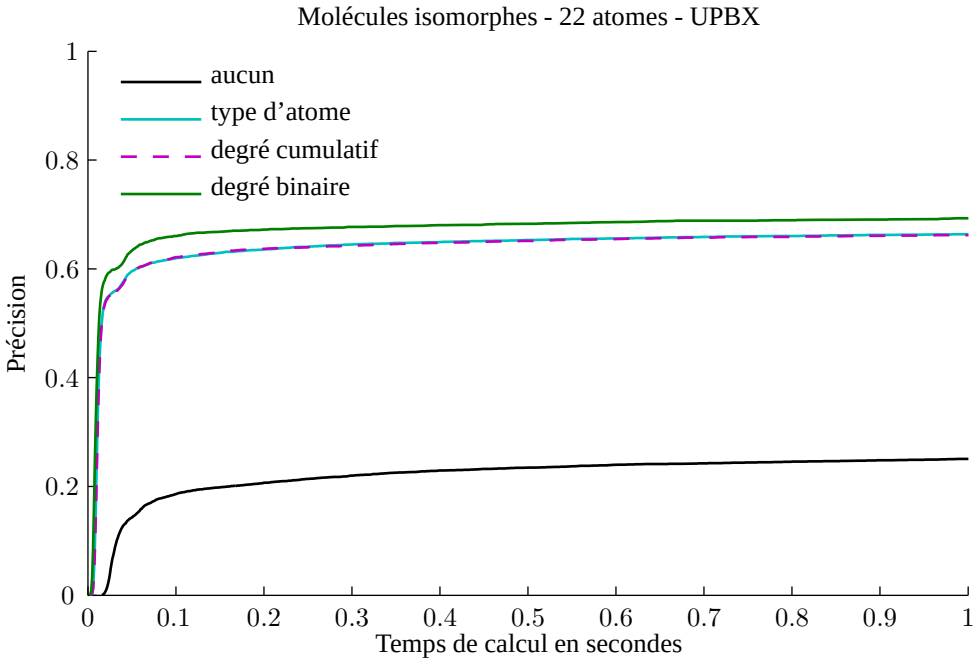

FIGURE 7.10 – Appariement de molécules isomorphes en utilisant des différentes signatures.

Nous examinons par la suite l'effet de l'étiquetage en combinaison avec notre algorithme génétique. Nous utilisons toutes les molécules de taille 22. Le choix est basé sur le nombre de molécules similaires après le filtrage par signature. En effet, la base contient plus de mille molécules de taille 22. Par la suite, il y a un nombre suffisant de molécules non-filtrables même quand on utilise la signature la plus discriminante (la signature binaire). Nous distinguons le cas positif et le cas négatif. Il s'agit du cas positif si les molécules à apparier sont isomorphes (exactement). En pratique, on apparie une molécule avec elle-même. Dans le cas négatif, les molécules ne sont pas isomorphes, mais l'appariement est nécessaire car les signatures sont identiques. C'est en effet dans ce cas que nous avons besoin d'un nombre assez grand de molécules ayant la même taille (et la même signature). Beaucoup d'algorithmes montrent un comportement différent selon le cas positif ou négatif, en particulier l'A*. Le cas négatif est beaucoup plus complexe. Les résultats avec l'opérateur UPBX sont donnés dans la figure 7.10 (cas positif) et la figure 7.11 (cas négatif).

Il faut noter que si l'on se trouve dans le cas inexact, et on cherche à calculer une distance, il est possible, selon la définition du coût d'une substitution, que l'alignement d'un atome de type A avec un atome de type B donne une distance inférieure que celle obtenue si l'on respecte les types d'atomes. Dans un tel cas, le filtrage par signature et l'approche par couleur excluent l'appariement optimal. Par conséquent, les valeurs des distances « optimales » peuvent différer selon le type de signature.

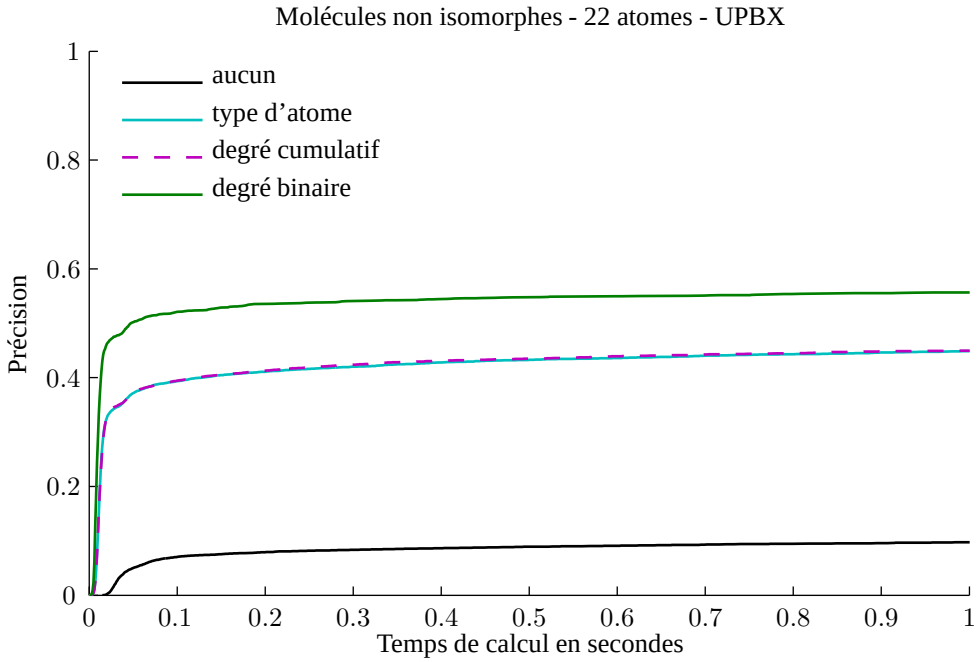

FIGURE 7.11 – Appariement de molécules non isomorphes en utilisant des différentes signatures.

7.3 Bilan

Dans ce chapitre, nous avons illustré trois applications concrètes de notre algorithme. Nous constatons des grandes différences dans les performances. En fait, il s'agit de trois problèmes très différents. Quant à l'application de l'identification de personnes basée sur des modèles en trois dimensions, nous nous situons dans un contexte d'isomorphisme inexact de sous-graphes. Il y a très peu de perturbations structurelles, mais des perturbations importantes au niveau des attributs. Dans ce cas, notre algorithme montre des très bonnes performances.

Dans l'application sur la base COIL-Del, les graphes sont soumis à des perturbations structurelles importantes mais n'ont *a priori* pas d'attributs. Dans ce cas, nous observons des performances assez faibles. Toutefois, il est possible d'augmenter la précision en introduisant des attributs tels que la distance entre les points d'intérêt de l'image ou encore leurs coordonnées.

En ce qui concerne les molécules, il s'agit d'un problème d'isomorphisme exact pour des graphes avec des étiquettes discrètes. Afin de filtrer la base et de restreindre l'espace de recherche, nous introduisons trois types de signatures des sommets. Nous constatons que les résultats sans signatures sont assez faibles tandis qu'avec les signatures ils deviennent acceptables.

Nous pouvons conclure que notre algorithme peut être appliqué à tous les problèmes d'isomorphisme de graphes, soit l'isomorphisme exact ou inexact, soit l'isomorphisme des graphes de même taille ou l'isomorphisme de sous-graphes. Quant à l'isomorphisme exact, il est important de noter que, pour ce cas, il existe des méthodes exhaustives bien plus performantes. Les meilleures perfor-

mances sont obtenues sur des problèmes d'isomorphisme inexact. Notre algorithme n'est dans ce cas pas sensible aux perturbations des valeurs des étiquettes mais plutôt aux perturbations structurelles. Contrairement aux algorithmes de recherche d'arbre, le temps de calcul diffère très peu entre des cas où les graphes correspondent et des cas où des graphes ne correspondent pas. Pour d'autres méthodes, notamment l'algorithme A*, le temps de calcul est beaucoup plus élevé dans le deuxième cas.

Chapitre 8

Conclusion et perspectives

Contribution

Nous avons considéré une approche génétique pour la résolution du problème d'isomorphisme de graphes. Nous avons réalisé des études à la fois des composants internes de l'algorithme génétique, de leurs interactions et d'autres méthodes. Dans ce cadre, nous avons effectué une comparaison exhaustive des opérateurs et étudié plusieurs aspects différents pour améliorer les résultats. Nous nous sommes intéressés en particulier à la combinaison avec des heuristiques qui n'entraînent pas de contraintes spécifiques, dédiées à des applications concrètes. Par conséquent, nous pouvons appliquer presque toutes les méthodes présentées ici à toutes les applications qui peuvent être modélisées par l'un des quatre problèmes d'appariement de graphes, définis par les quatre combinaisons possibles des termes opposés exact, inexact et graphe, sous-graphe.

Opérateurs

Nous avons proposé au total cinq nouveaux opérateurs de croisement, deux opérateurs de permutation, PBX et UPBX, et trois opérateurs intelligents, DPX, SDPX et NDPX, qui incluent une heuristique gloutonne. Nous avons mené une étude complète et exhaustive pour le paramétrage, compris au sens large et incluant le type de moteur d'évolution, de ces opérateurs ainsi que pour les opérateurs existants afin de pouvoir les comparer entre eux. En effet, les paramètres optimaux ainsi que la sensibilité à leur égard changent selon l'opérateur.

Nous avons montré que les opérateurs proposés ont des performances qualitativement et quantitativement supérieures à celles des autres opérateurs. Parmi les opérateurs proposés, les opérateurs intelligents fournissent les meilleurs résultats.

Nous avons réalisé une étude complémentaire sur la mutation, dans laquelle nous avons montré que l'application d'une mutation plus perturbatrice au lieu d'une mutation élémentaire n'améliore guère les résultats.

Algorithme *mémétique*

Afin d'augmenter le taux de succès pratique et de permettre à l'algorithme génétique d'exploiter les solutions prometteuses, nous avons étudié sa combinaison avec une heuristique de recherche locale. Il est généralement admis que les algorithmes génétiques profitent de l'intégration d'heuristiques spécialisées au problème donné. Les expériences exhaustives que nous avons menées sur une heuristique particulière ont permis de souligner que celle-ci peut effectivement améliorer la précision pour certains opérateurs, mais aussi réduire le temps de calcul, en particulier en combinaison avec les opérateurs intelligents.

Parallélisation

Nous avons également réalisé une implémentation parallèle de l'algorithme dans le but d'accélérer l'exécution. Le gain, estimé comme assez limité en théorie, s'est avéré plus limité encore dans la pratique, en particulier en ce qui concerne les appariements de graphes de petite taille. On peut pourtant profiter des avantages de la parallélisation lorsque la complexité du problème augmente.

Garantie du résultat optimal

La garantie de l'optimalité des résultats est un aspect plus théorique par rapport à ce qui précède. Nous avons proposé d'abord des stratégies qui peuvent accélérer l'algorithme A*, avant de proposer la combinaison de ce dernier avec notre approche évolutionnaire. L'algorithme génétique sert alors à fournir une première bonne estimation pour l'algorithme de recherche exhaustive qui vérifie ensuite s'il existe encore une meilleure solution. Nous avons montré que le temps de calcul de cette approche hybride est inférieur à celui de l'approche A* seule. L'approche hybride n'est que très légèrement plus lente que l'algorithme génétique.

Applications

Enfin, nous avons testé notre algorithme sur trois applications représentant différents aspects du problème d'isomorphisme de graphes afin de démontrer effectivement sa généralité. Nous l'avons comparé en particulier avec l'algorithme A* sur des graphes décrivant des modèles faciaux. Nous avons aussi montré que l'algorithme génétique fournit des résultats supérieurs à ce dernier, en termes de taille de graphes que l'on peut traiter ainsi qu'en termes de temps de calcul pour une application réelle avec des petit graphes, bien que ces derniers aient dû favoriser A*.

Pour de nombreux algorithmes, dont A*, on observe une incohérence entre le cas où l'on apparie deux graphes correspondants au même objet et le cas où l'on apparie deux graphes provenant d'objets différents (*match time* et *no match time*). Pour les approches exhaustives comme A*, ce dernier cas est généralement beaucoup plus difficile et le temps de calcul est largement supérieur. Nous

constatons que notre approche ne montre pas de sensibilité à ce propos, à l'exception des opérateurs de croisement intelligents pour lesquels nous observons une augmentation très faible du temps de calcul dans le cas négatif, mais qui reste négligeable par rapport à celui de A*.

Perspectives

Nos travaux ouvrent des perspectives vers trois directions principales que nous détaillons ci-dessous. Premièrement, à un niveau applicatif, on peut s'interroger sur la pertinence et les particularités de la description des différents types de données sous forme de graphes. Deuxièmement, il serait avantageux d'approfondir les combinaisons et interactions entre les différents aspects évoqués, et isolément d'y inclure des méthodes adaptatives. Enfin, l'étude d'une généralisation additionnelle de notre approche sur les appariements multivoques permettrait de traiter des problèmes plus complexes encore.

Pertinence de la représentation des données

Nous avons constaté, dans le cas de l'appariement d'images en deux dimensions avec des modèles en trois dimensions, que les modèles de graphes sont aujourd'hui assez réduits afin de pouvoir résoudre les problèmes associés avec les algorithmes actuels. De plus, il s'avère que le modèle qui correspond à la plus petite distance entre les graphes de visages ne correspond pas toujours à la même personne, ce qui pose un problème d'importance. Une solution peut consister à augmenter l'expressivité de la représentation. En général, cela augmente le coût du calcul.

Les travaux que nous avons menés ont permis de définir un outil performant pour traiter efficacement des graphes plus grands. Cependant, en même temps que l'on augmente l'expressivité, on risque de perdre en généralité, car les descriptions deviennent de plus en plus dépendantes de l'application concrète. De plus, les effets du bruit s'accentuent sur des modèles de granularité de plus en plus fine. Il serait donc avantageux d'étudier en détail toute la chaîne de la représentation afin de s'assurer de la pertinence de la représentation obtenue. Il serait particulièrement intéressant d'appliquer des méthodes d'apprentissage pour déterminer les paramètres de la fonction à optimiser. Dans un cadre applicatif il s'agit en particulier des coûts et poids associés à la correction d'erreurs aux niveaux des attributs et de la structure du graphe.

Interactions et adaptivité

Nous avons observé que, dans un algorithme génétique, toutes les parties, comme par exemple les opérateurs et la recherche locale, interagissent.

Expérimentations complémentaires En particulier, les interactions entre la recherche locale et les différents opérateurs de croisement méritent une analyse plus approfondie.

En ce qui concerne la stratégie de parallélisation, nous avons choisi la stratégie maître-esclave afin de limiter les interactions avec les autres parties de l'algorithme. Une approche différente, comme par exemple le modèle en îlots, change les propriétés de l'algorithme évolutionnaire sur lequel il est basé et peut améliorer les résultats. Dans ce cas, une révision des méthodes de recherche locale et des opérateurs utilisés pourrait mener à des résultats intéressants.

Des études plus approfondies sont également envisageables en ce qui concerne le critère d'arrêt et des stratégies de sélection autres que la sélection par tournoi. Enfin, nous aimerions comparer notre approche avec les résultats d'autres méta-heuristiques comme les colonies de fourmis, qui ont été utilisées pour résoudre le problème d'isomorphisme de graphes.

Adaptivité Il nous paraît particulièrement intéressant de combiner la recherche locale avec la mutation *scramble* car les deux approches sont complémentaires. La recherche locale exploite les solutions trouvées, tandis que la mutation *scramble* permet une exploration plus dispersée que la mutation d'échange.

Une autre piste est l'utilisation d'un algorithme adaptatif dont le comportement change selon les différentes étapes d'exécution. Il existe par exemple des stratégies de contrôle de paramètres en fonction des mesures dynamiques de la population, comme par exemple celle de sa diversité.

Dans ce cadre, on pourrait, par exemple, adapter le type de mutation ainsi que, s'il s'agit de la mutation *scramble*, son paramètre. Si la diversité est faible, alors on utilisera une mutation plus perturbatrice afin de l'augmenter. Dans le cas contraire, on utilisera la mutation d'échange afin de mieux exploiter localement les environs des individus. Ce choix pourrait également être effectué au niveau local, plutôt que global, en regardant au lieu de la diversité le nombre d'individus dans le voisinage proche de l'individu qui subit la mutation. S'il existe de nombreux autres individus dans ce voisinage, alors il est plus pertinent d'appliquer plus de perturbations pour explorer une autre partie de l'espace de recherche.

Généralisation à des appariements multivoques

Bien que notre approche soit robuste aux perturbations d'attributs comme nous l'avons observé dans le cadre de l'application aux images en trois dimensions, elle est sensible aux perturbations structurelles comme nous l'avons constaté dans les applications aux images de la base COIL et aux molécules. Il existe même d'autres perturbations comme par exemple celle des effets de granularité. En effet, dans de nombreuses applications, les graphes sont extraits automatiquement à partir de données du domaine. Ainsi, dans le domaine des images, ils sont obtenus à partir de méthodes de segmentation automatique. Ces méthodes ne fournissent pas des résultats parfaits, ce qui peut

mener, dans le cas d'images, à des effets de sous ou sur-segmentation. Dans d'autres domaines, on peut également trouver des descriptions à différents niveaux de granularité.

Une piste intéressante pour augmenter la robustesse à tous ces changements est l'étude des appariements multivoques. Ces derniers permettent d'apparier un sommet à plusieurs autres sommets. Par conséquent, il ne s'agit pas d'une injection comme dans le cadre des appariements univoques. La représentation par permutation ne pouvant donc être appliquée, d'autres représentations doivent être recherchées pour ce type de problème, mais généralement le changement de la représentation entraîne le changement de tous les opérateurs génétiques. Néanmoins, nous pouvons constater dans l'état de l'art que, dans le domaine des algorithmes génétiques, une meilleure représentation mène souvent à des résultats largement supérieurs et que ceci constitue donc une voie prometteuse.

Dans un premier temps, on pourrait étudier l'ajout d'une étape complémentaire de prétraitement qui identifierait les perturbations possibles, notamment la fusion et la division de sommets mais aussi la substitution d'arêtes. Comme l'identification de ces perturbations peut être un autre problème d'optimisation lié à l'isomorphisme, l'application d'une stratégie co-évolutionnaire combinant notre approche actuelle avec une méthode génétique qui optimiserait le coût des perturbations semble une voie prometteuse. Celle-ci permettrait en effet de conserver la représentation des permutations par le chromosome modélisant l'appariement univoque, tandis que les relations multivoques seraient traitées par le second chromosome.

Bibliographie

E. Alba & M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Trans. on Evolutionary Computation*, 6(5):443–462, 2002.

H. Almohamad & S. Duffuaa. A Linear Programming Approach for the Weighted Graph Matching Problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):522–525, 1993.

R. Ambauen, S. Fischer, & H. Bunke. Graph Edit Distance with Node Splitting and Merging, and its Application to Diatom Identification. Dans E. Hancock & M. Vento, Éds., *IAPR Workshop GbRPR*, vol. 2726 de *LNCS*, pages 95–106. Springer, 2003.

B. Amedro, V. Bodnartchouk, D. Caromel, C. Delbé, F. Huet, & G. L. Taboada. Current State of Java for HPC. Rapport technique 0353, INRIA Sophia Antipolis, 2008.

P. J. Artymiuk, R. V. Spriggs, & P. Willett. Graph Theoretic Methods for the Analysis of Structural Relationships in Biological Macromolecules. *Journal of the American Society for Information Science and Technology*, 56(5):518–528, 2005.

A. Auger. Convergence results for the (1, λ)-SA-ES using the theory of ϕ-irreducible Markov chains. *Theoretical Computer Science*, 334:35–69, 2005.

A. Auger & O. Teytaud. Continuous Lunches are free plus the Design of Optimal Optimization Algorithms. *Algorithmica*, 2008.

S. Auwatanamongkol. Inexact Graph Matching using a Genetic Algorithm for Image Recognition. *Pattern Recognition Letters*, 28:1428–1437, 2007.

L. Babai, D. Grigoryev, & D. Mount. Isomorphism of Graphs with Bounded Eigenvalue Multiplicity. Dans *Proceedings of the fourteenth annual ACM symposium on Theory of computing (STOC)*, pages 310–324. ACM Press, New York, NY, USA, 1982. ISBN 0-89791-070-2. doi: http://doi.acm.org/10.1145/800070.802206.

T. Bäck, U. Hammel, & H.-P. Schwefel. Evolutionary Computation: Comments on the History and Current State. *IEEE Trans. on Evolutionary Computation*, 1(1):3–17, 1997.

J. S. Beis & D. G. Lowe. Shape Indexing using Approximate Nearest-Neighbor Search in High-Dimensional Spaces. Dans *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1000–1006. IEEE Computer Society, San Juan, Puerto Rico, 1997.

J. S. Beis & D. G. Lowe. Indexing without Invariants in 3D Object Recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(10):1000–1015, 1999.

E. Bengoetxea, P. Larrañaga, I. Bloch, A. Perchant, & C. Boeres. Inexact Graph Matching by means of Estimation of Distribution Algorithms. *Pattern Recognition*, 35:2867–2880, 2002.

S. Berretti, A. Del Bimbo, & P. Pala. 3D Face Recognition by Spatial Arrangement of Iso-Geodesic Surfaces. Dans *3DTV Conference: The True Vision - Capture, Transmission and Display of 3D Video*. IEEE, 2008.

S. Berretti, A. Del Bimbo, & E. Vicario. Efficient Matching and Indexing of Graph Models in Content-Based Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1089–1105, 2001.

M. Birattari, T. Stützle, L. Paquete, & K. Varrentrapp. A Racing Algorithm for Configuring Metaheuristics. Dans *Proc. GECCO'02*. 2002.

T. Blickle & L. Thiele. A Comparison of Selection Schemes used in Evolutionary Algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.

M. C. Boeres, C. C. Ribeiro, & I. Bloch. A Randomized Heuristic for Scene Recognition by Graph Matching. Dans C. Ribeiro & S. Martins, Éds., *WEA*, vol. 3059 de *LNCS*, pages 100–113. Springer, 2004.

A. Bouvier, F. Le Lionnais, & M. George. *Dictionnaire des Mathématiques*. Presses Universitaires de France, 2009.

Y. Boykov & V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

Y. Boykov, O. Veksler, & R. Zabih. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

T. Bärecke. Rapport sur les Résultats d'Agrégation des Mesures de plus de 3 Composantes des Coûts - Cas d'Étude: Du Puits à la Roue. Rapport technique, Université Pierre et Marie Curie - Paris6 UMR 7606, DAPA, LIP6, 2007.

T. Bärecke & M. Detyniecki. Combining Exhaustive and Approximate Methods for Improved Sub-Graph Matching. Dans *Proc. of the Int. Workshop on Advances in Pattern Recognition*. Springer, 2007a.

T. Bärecke & M. Detyniecki. Memetic Algorithms for Inexact Graph Matching. Dans *Proc. of the IEEE Congress on Evolutionary Computation - CEC'07*. IEEE, Singapore, 2007b.

H. Bunke. On a Relation between Graph Edit Distance and Maximum Common Subgraph. *Pattern Recognition Letters*, 18:689–694, 1997.

H. Bunke. Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. *IEEE Transactions on Pattern Analysis and Mach Intelligence*, 21(9):917–922, 1999. ISSN 0162-8828. doi:http://dx.doi.org/10.1109/34.790431.

H. Bunke, P. Foggia, C. Guidobaldi, & M. Vento. Graph Clustering using the Weighted Minimum Common Supergraph. Dans E. Hancock & M. Vento, Éds., *IAPR Workshop GbRPR*, vol. 2726 de *LNCS*, pages 235–246. Springer, 2003.

H. Bunke & K. Shearer. A Graph Distance Metric based on the Maximal Common Subgraph. *Pattern Recognition Letters*, 19:255–259, 1998.

L. S. Buriol, P. M. França, & P. Moscato. A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem. *Journal of Heuristics*, 10(5):483–506, 2004.

K. Burjorjee. Sufficient Conditions for Coarse-Graining Evolutionary Dynamics. Dans *Foundations of Genetic Algorithms*, vol. 4436 de *LNCS*, pages 35–53. Springer, 2007.

T. Caelli & S. Kosinov. Inexact Graph Matching using Eigen-Subspace Projection Clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):329–354, 2004.

T. S. Caetano & T. Caelli. Approximating the Problem, not the Solution: An Alternative View of Point Set Matching. *Pattern Recognition*, 39(4):552–561, 2006.

T. S. Caetano, T. Caelli, D. Schuurmans, & D. A. C. Barone. Graphical Models and Point Pattern Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1646–1663, 2006.

V. Campos, M. Laguna, & R. Martí. Context-Independent Scatter and Tabu Search for Permutation Problems. *INFORMS Journal on Computing*, 17(1):111–122, 2005.

E. Cantú-Paz. Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms. *Journal of Heuristics*, 7(4):311–334, 2001.

E. Cantú-Paz. Parameter Setting in Parallel Genetic Algorithms. Dans F. G. Lobo, C. F. Lima, & Z. Michalewicz, Éds., *Parameter Setting in Evolutionary Algorithms*, vol. 54 de *Studies in Computational Intelligence*. Springer, 2007.

E. Cantú-Paz & D. E. Goldberg. On the Scalability of Parallel Genetic Algorithms. *Evolutionary Computation*, 7(4):429–449, 1999.

M. Carcassoni & E. R. Hancock. Weighted Graph-Matching using Modal Clusters. Dans *GbRPR*, pages 260–269. 2001.

M. Carcassoni & E. R. Hancock. Spectral Correspondence for Point Pattern Matching. *Pattern Recognition*, 36(1):193–204, 2003.

J. Cervantes & C. R. Stephens. « Optimal » Mutation Rates for Genetic Search. Dans *GECCO*, pages 1313–1320. 2006.

R. M. Cesar, Jr., E. Bengoetxea, I. Bloch, & P. Larrañaga. Inexact Graph Matching for Model-Based Recognition: Evaluation and Comparison of Optimization Algorithms. *Pattern Recognition*, 38(11):2099–2113, 2005.

P.-A. Champin & C. Solnon. Measuring the Similarity of Labeled Graphs. Dans K. Ashley & B. D.G., Éds., *ICCBR*, vol. 2689 de *LNCS*, pages 80–95. Springer, 2003.

F. Chevalier, J.-P. Domenger, J. Benois-Pineau, & M. Delest. Retrieval of Objects in Video by Similarity based on Graph Matching. *Pattern Recognition Letters*, 28:939–949, 2007.

F. R. Chung. *Spectral Graph Theory*. Numéro 92 dans CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

V. Cicirello & S. Smith. Modeling GA Performance for Control Parameter Optimization. Dans *GECCO-2000: Proc. of the Genetic and Evolutionary Computation Conference*, pages 235–242. Morgan Kaufmann Publishers, 2000.

V. A. Cicirello. Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respects Absolute Position. Dans *GECCO*. Seattle, Washington, USA, 2006.

D. Conte, P. Foggia, C. Sansone, & M. Vento. Thirty Years of Graph Matching in Pattern Recognition. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

L. P. Cordella, P. Foggia, C. Sansone, & M. Vento. A (Sub)Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, 2004. ISSN 0162-8828. doi:http://dx.doi.org/10.1109/TPAMI.2004.75.

A. D. Cross & E. R. Hancock. Convergence of a Hill Climbing Genetic Algorithm for Graph Matching. Dans E. Hancock & M. Pelillo, Éds., *EMMCVPR*, vol. 1654 de *LNCS*, pages 221–236. 1999.

A. D. J. Cross, R. C. Wilson, & E. R. Hancock. Inexact Graph Matching using Genetic Search. *Pattern Recognition*, 30(6):953–970, 1997.

L. Davis. Applying Adaptive Algorithms to Epistatic Domains. Dans *Proc. Int. Joint Conf. on Artificial Intelligence*. 1985.

L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

B. Dawson. Comparing Floating Point Numbers. http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm, 2008. Consulté le 14/11/2008.

K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Thèse de doctorat, University of Michigan, 1975.

K. A. De Jong & J. Sarma. Generation Gaps Revisited. Dans *Foundations of Genetic Algorithms*. Morgan Kaufmann, 1993.

M. F. Demirci, A. Shokoufandeh, Y. Keselman, L. Bretzner, & S. Dickinson. Object Recognition as Many-to-Many Feature Matching. *International Journal of Computer Vision*, 69(2):203–222, 2006.

A. Deruyver, Y. Hodé, E. Leammer, & J.-M. Jolion. Adaptive Pyramid and Semantic Graph: Knowledge Driven Segmentation. Dans *GbRPR*, vol. 3434 de *LNCS*, pages 213–222. Springer, 2005.

P. Dickinson, M. Kraetzl, H. Bunke, M. Neuhaus, & A. Dadej. Similarity Measures for Hierarchical Representations of Graphs with Unique Node Labels. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):425–442, 2004.

R. Diestel. *Graph Theory*, vol. 173 de *Graduate Texts in Mathematics*. Springer, 3ème édition, 2005.

A. Eiben, Z. Michalewicz, M. Schoenauer, & J. Smith. Parameter Control in Evolutionary Algorithms. Dans J. Kacprzyk, Éd., *Parameter Setting in Evolutionary Algorithms*, Studies in Computational Intelligence, chapitre Parameter Control in Evolutionary Algorithms, pages 19–46. Springer, 2007.

A. Eiben & J. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2nd édition, 2007.

A. E. Eiben, R. Hinterding, & Z. Michalewicz. Parameter Control in Evolutionary Algorithms. *IEEE Trans. on Evolutionary Computation*, 3(2):124–141, 1999.

M. Eshera & K.-S. Fu. A Graph Distance Measure for Image Analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 14:398–407, 1984.

P. F. Felzenszwalb & D. P. Huttenlocher. Efficient Graph-Based Image Segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

I. Filotti & J. Mayer. A Polynomial-Time Algorithm for Determining the Isomorphism of Graphs of Fixed Genus. Dans *Proc. 12th Ann. ACM Symposium on Theory of Computing*, pages 236–243. 1980.

L. Finkelstein & S. Markovitch. Optimal Schedules for Monitoring Anytime Algorithms. *Artificial Intelligence - Special Issue on Computational Tradeoffs under Bounded Resources*, 126(1-2):63–108, 2001.

P. Foggia, C. Sansone, & M. Vento. A Performance Comparison of Five Algorithms for Graph Isomorphism. Dans *GbRPR*, pages 188–199. 2001.

S. Fortin. The Graph Isomorphism Problem. Rapport technique, University of Alberta, Canada, 1996.

B. Freisleben & P. Merz. New Genetic Local Search Operators for the Traveling Salesman Problem. Dans *Parallel Problem Solving from Nature - PPSN IV*, pages 890–899. 1996.

M. Garey & D. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman and Company, 1979.

S. Günter & H. Bunke. Self-Organizing Map for Clustering in the Graph Domain. *Pattern Recognition Letters*, 23:405–417, 2002.

S. Günter & H. Bunke. Validation Indices for Graph Clustering. *Pattern Recognition Letters*, 24:1107–1113, 2003.

C. Godsil & G. Royle. *Algebraic Graph Theory*, vol. 207 de *Graduate texts in mathematics*. Springer, 2001.

S. Gold & A. Rangarajan. A Graduated Assignment Algorithm for Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(4):377–388, 1996.

S. Gold, A. Rangarajan, C.-P. Lu, P. Suguna, & E. Mjolsness. New Algorithms for 2D and 3D Point Matching: Pose Estimation and Correspondence. *Pattern Recognition*, 31(8):1019–1031, 1998.

D. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison Wesley, Reading, MA, 1989.

D. Goldberg. What every Computer Scientist should know about Floating-Point Arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991a.

D. E. Goldberg. Genetic Algorithms, Noise, and the Sizing of Populations. Rapport technique 91010, University of Illinois, 1991b.

D. E. Goldberg & R. Lingle. Alleles, Loci and the Traveling Salesman Problem. Dans *Proc. 1st Int. Conf. on Genetic Algorithms*, pages 154–159. 1985.

M. Gori, M. Maggini, & L. Sarti. Exact and Approximate Graph Matching using Random Walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1100–1111, 2005. ISSN 0162-8828. doi:http://dx.doi.org/10.1109/TPAMI.2005.138.

J. J. Grefenstette. Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

T. Gärtner, P. Flach, & S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. Dans B. Schölkopf & M. K. Warmuth, Éds., *Learning Theory and Kernel Machines*, vol. 2777 de *Lecture Notes in Artificial Intelligence*, pages 129–143. Springer, 2003.

E. A. Hansen & S. Zilberstein. Monitoring and Control of Anytime Algorithms: A Dynamic Programming Approach. *Artificial Intelligence - Special Issue on Computational Tradeoffs under Bounded Resources*, 126(1-2):139–157, 2001.

N. Hansen, S. Müller, & P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

Z. Harchaoui & F. Bach. Image Classification with Segmentation Graph Kernels. Dans *CVPR*. IEEE, 2007.

G. Harik, E. Cantú-Paz, D. E. Goldberg, & B. L. Miller. The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations. *Evolutionary Computation*, 7(3):231–253, 1999.

G. R. Harik & F. G. Lobo. A Parameter-less Genetic Algorithm. Dans *Proc. of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 258–265. Morgan Kaufman, 1999.

V. Heidrich-Meisner & C. Igel. Hoeffding and Bernstein Races for Selecting Policies in Evolutionary Direct Policy Search. Dans *Proc. Int. Conf. on Machine Learning (ICML)*. 2009.

D. Hidovic & M. Pelillo. Metrics for Attributed Graphs based on the Maximal Similarity Common Subgraph. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):299–313, 2004.

Y. Ho & D. Pepyne. Simple Explanation of the No-Free-Lunch Theorem and its Implications. *Journal of Optimization Theory and Applications*, 115(3):549–570, 2002.

J. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

J. Hopcroft & R. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

J. Hopcroft & J. Wong. A Linear Time Algorithm for the Isomorphism of Planar Graphs. Dans *Proc. 6th Annual ACM Symp. on Theory of Computing*, pages 172–184. 1974.

E. Horvitz & S. Zilberstein. Computational Tradeoffs under Bounded Resources. *Artificial Intelligence - Special Issue on Computational Tradeoffs under Bounded Resources*, 126(1-2):1–4, 2001.

IEEE754. *IEEE Standard for Binary Floating-Point Arithmetic*. Standards Committee of the IEEE Computer Society, 2008. doi:10.1109/IEEESTD.2008.4610935.

C. Igel & M. Toussaint. A No-Free-Lunch Theorem for Non-Uniform Distributions of Target Functions. *Journal of Mathematical Modelling and Algorithms*, 3:313–322, 2004.

A. Inokuchi, T. Washio, & H. Motoda. A General Framework for Mining Frequent Subgraphs from Labeled Graphs. *Fundamenta Informaticae*, 66:53–82, 2005.

C. Irniger & H. Bunke. Decision Trees for Error-Tolerant Graph Database Filtering. Dans L. Brun & M. Vento, Éds., *GbRPR*, vol. 3434 de *LNCS*, pages 301–311. Springer, 2005.

C.-A. M. Irniger. *Graph Matching - Filtering Databases of Graphs Using Machine Learning Techniques*. Thèse de doctorat, Institut für Informatik und angewandte Mathematik, Universität Bern, 2005.

H. Ishibuchi & T. Murata. A Multi-Objective Genetic Local Search Algorithm and its Application to Flowshop Scheduling. *IEEE Transactions on Systems, Man, and Cybernetics, Appl. Rev.*, 28(3):392–403, 1998.

H. Ishibuchi, T. Yoshida, & T. Murata. Balance between Genetic Search and Local Search in Memetic Algorithms for Multiobjective Permutation Flowshop Scheduling. *IEEE Trans. on Evolutionary Computation*, 7(2):204–223, 2003.

B. J. Jain & F. Wysotzki. Solving Inexact Graph Isomorphism Problems using Neural Networks. *Neurocomputing*, 63:45–67, 2005.

A. Katok, B. Hasselblatt, & L. Mendoza. *Introduction to the Modern Theory of Dynamical Systems*. Cambridge University Press, 1997.

J. Köbler, U. Schöning, & J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, Boston, 1993.

K. G. Khoo & P. N. Suganthan. Structural Pattern Recognition using Genetic Algorithms with Specialized Operators. *IEEE Trans. on Systems, Man, and Cybernetics - Part B: Cybernetics*, 33(1):156–165, 2003.

R. I. Kondor & J. Lafferty. Diffusion Kernels on Graphs and other Discrete Structures. Dans *Proc. of the ICML*. 2002.

Z. Konfršt. Parallel Genetic Algorithms: Advances, Computing Trends, Applications and Perspectives. Dans *Proc. 18th Int. Parallel and Distributed Processing Symposium (IPDPS'04)*. IEEE, 2004.

N. Krasnogor & J. Smith. A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, 2005.

R. Krishnapuram, S. Medasani, S.-H. Jung, Y.-S. Choi, & R. Balasubramaniam. Content-based Image Retrieval based on a Fuzzy Approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1185–1199, 2004.

W. G. Kropatsch, Y. Haxhimusa, & A. Ion. Multiresolution Image Segmentations in Graph Pyramids. Dans *Applied Graph Theory in Computer Vision and Pattern Recognition*, vol. 52 de *Studies in Computational Intelligence*. Springer, 2007.

P. Larranaga, C. Kuijpers, R. Murga, I. Inza, & S. Dizdarevic. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review*, 13:129–170, 1999.

O. Lezoray, A. Elmoataz, & S. Bougleux. Graph Regularization for Color Image Processing. *Computer Vision and Image Understanding*, 107:38–55, 2007.

Y. Li, D. Blostein, & P. Abolmaesumi. Asymmetric Inexact Matching of Spatially-Attributed Graphs. Dans L. Brun & M. Vento, Éds., *GbRPR*, vol. 3434 de *LNCS*, pages 253–262. Springer, 2005.

Y. Liu, D. Zhang, G. Lu, & W.-Y. Ma. A Survey of Content-based Image Retrieval with High-Level Semantics. *Pattern Recognition*, 40:262–282, 2007.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, & C. Watkins. Text Classification using String Kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

E. Luks. Isomorphism of Graphs of Bounded Valence can be tested in Polynomial Time. *Journal of Computer and System Sciences*, 25:42–65, 1982.

B. Luo & E. R. Hancock. Structural Graph Matching Using the EM Algorithm and Singular Value Decomposition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(10):1120–1136, 2001.

S. Marini, M. Spagnuolo, & B. Falcidieno. From Exact to Approximate Maximum Common Subgraph. Dans L. Brun & M. Vento, Éds., *GbRPR*, vol. 3434 de *LNCS*, pages 263–272. Springer, 2005.

O. Maron & A. Moore. Hoeffding Races: Accelerating Model Selection Search for Classification and Function Approximation. Dans *Proc. Advances in Neural Information Processing Systems*. 1994.

M. Matsumoto & T. Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998. ISSN 1049-3301. doi:http://doi.acm.org/10.1145/272991.272995.

J. McCall. Genetic Algorithms for Modelling and Optimisation. *Journal of Computational and Applied Mathematics*, 184(1):205–222, 2005.

R. McGill, J. W. Tukey, & W. Larsen. Variations of Box Plots. *The American Statistician*, 32:12–16, 1978.

B. D. McKay. Practical Graph Isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

P. Merz. *Memetic Algorithms for Combinatorial Optimization Problems: Fitness Landscapes and Effective Search Strategies*. Thèse de doctorat, Parallel Syst. Res. Group, Dept. Electr. Eng. Comput. Sci., Univ. Siegen, Germany, 2000.

P. Merz. Advanced Fitness Landscape Analysis and the Performance of Memetic Algorithms. *Evolutionary Computation, Special Issue on Memetic Evolutionary Algorithms*, 12(3):303–326, 2004.

P. Merz & B. Freisleben. A Genetic Local Search Approach to the Quadratic Assignment Problem. Dans *Proc. of the 7th Int. Conf. on Genetic Algorithms*, pages 465–472. Morgan Kaufmann, East Lansing, MI, USA, 1997.

P. Merz & B. Freisleben. Fitness Landscapes, Memetic Algorithms, and Greedy Operators for Graph Bipartitioning. *Evolutionary Computation*, 8(1):61–91, 2000.

P. Merz & B. Freisleben. Memetic Algorithms for the Traveling Salesman Problem. *Complex Systems*, 13(4):297–345, 2001.

B. T. Messmer & H. Bunke. Efficient Subgraph Isomorphism Detection - A Decomposition Approach. *IEEE Trans. on Knowledge and Data Engineering*, 12(2):307–323, 2000.

H. Mühlenbein. How Genetic Algorithms Really Work - Part I: Mutation and Hillclimbing. Dans *Proc. Conf. Parallel Problem Solving from Nature*, pages 15–25. 1992.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, Berlin, Germany, 3 édition, 1996.

K. Mikolajczyk & C. Schmid. An Affine Invariant Interest Point Detector. Dans *Proc. 7th European Conference on Computer Vision (ECCV)*, pages 128–142. Copenhagen, Denmark, 2002.

K. Mikolajczyk & C. Schmid. A Performance Evaluation of Local Descriptors. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 27(10):1615–1630, 2005.

G. Miller. Isomorphism Testing for Graphs of Bounded Genus. Dans *Proc. 12th Ann. ACM Symposium on Theory of Computing*, pages 225–235. 1980.

P. Moscato. On Evolution, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Tech. Report. Caltech Concurrent Computation Program 826, California Institute of Technology, Pasadena, CA, 1989.

T. Motzkin & E. Straus. Maxima for Graphs and a new Proof of a Theorem of Turán. *Canadian Journal of Mathematics*, 17:533–540, 1965.

R. Myers & E. R. Hancock. Least-Commitment Graph Matching with Genetic Algorithms. *Pattern Recognition*, 34(2):375–394, 2001.

R. O. Myers. *Genetic Algorithms for Ambiguous Labelling Problems*. Thèse de doctorat, Department of Computer Science, University of York, 1999.

Y. Nagata & S. Kobayashi. Edge Assembly Crossover: A High-Power Genetic Algorithm for the Traveling Salesman Problem. Dans T. Bäck, Éd., *Proc. 7th Int. Conf. on GAs*, pages 450–457. Morgan Kaufman, 1997.

V. Nannen & A. Eiben. Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. Dans *IJCAI'07*, pages 975–980. 2007.

S. Nene, S. Nayar, & H. Murase. Columbia Object Image Library (COIL-100). Rapport technique CUCS-006-96, Columbia University, 1996.

A. Neubauer. Theory of the Simple Genetic Algorithm with α-Selection. Dans *Genetic and Evolutionary Computation Conference*, pages 1009–1016. ACM, 2008.

M. Neuhaus & H. Bunke. Automatic Learning of Cost Functions for Graph Edit Distance. *Information Sciences*, 177(1):239–247, 2007.

C.-W. Ngo, Y.-F. Ma, & H.-J. Zhang. Video Summarization and Scene Detection by Graph Modeling. *IEEE Trans. on Circuits and Systems for Video Technology*, 15(2):296–305, 2005.

H. D. Nguyen, I. Yoshihara, K. Yamamori, & M. Yasunaga. Implementation of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems. *IEEE Trans. on Systems, Man, and Cybernetics-Part B*, 37(1):92–99, 2007.

M. Nowostawski & R. Poli. Parallel Genetic Algorithm Taxonomy. Dans *KES*. 1999.

I. Oliver, D. Smith, & J. Holland. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. Dans *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, pages 224–230. Mahwah, NJ, USA, 1987.

J.-F. Omhover & M. Detyniecki. Fast Gradual Matching Measure for Image Retrieval based on Visual Similarity and Spatial Relations. *Int. J. Intell. Syst.*, 21(7):711–723, 2006.

Y. Ong, M. Lim, N. Zhu, & K. Wong. Classification of Adaptive Memetic Algorithms: A Comparative Study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(1):141–152, 2006.

M. Pelillo. Replicator Equations, Maximal Cliques, and Graph Isomorphism. *Neural Computation*, 11:1933–1955, 1999.

R. C. Read & D. G. Corneil. The Graph Isomorphism Disease. *Journal of Graph Theory*, 1:339–363, 1977.

I. Rechenberg. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart-Bad Cannstadt, Germany, 1973.

C. Reeves & J. E. Rowe. *Genetic Algorithms - Principles and Perspectives, A Guide to GA Theory*. Kluwer Academic Publishers, 2003.

J.-M. Renders & S. P. Flasse. Hybrid Methods using Genetic Algorithms for Global Optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 26(2):243–258, 1996.

R. Reynolds. An Introduction to Cultural Algorithms. Dans *3rd Conf. on Evolutionary Programming*. 1994.

K. Riesen & H. Bunke. IAM Graph Database Repository for Graph based Pattern Recognition and Machine Learning. Dans *Structural, Syntactic, and Statistical Pattern Recognition*, vol. 5342 de *LNCS*, pages 287–297. Springer, 2008.

A. Rogers & A. Prügel-Bennett. Genetic Drift in Genetic Algorithm Selection Schemes. *IEEE Trans. on Evolutionary Computation*, 3(4):298–303, 1999.

J. E. Rowe, M. D. Vose, & A. H. Wright. Reinterpreting No Free Lunch. *Evolutionary Computation*, 17(1), 2008.

G. Rudolph. Convergence Analysis of Canonical Genetic Algorithms. *IEEE Trans. on Neural Networks*, 5(1):96–101, 1994.

G. Rudolph. Convergence Analysis of Canonical Genetic Algorithms. Dans *Proc. IEEE Conf. on Evolutionary Computation*. 1996.

O. Sammoud, S. Sorlin, C. Solnon, & K. Ghédira. A Comparative Study of Ant Colony Optimization and Reactive Search for Graph Matching Problems. Dans J. Gottlieb & G. R. Raidl, Éds., *EvoCOP*, vol. 3906 de *LNCS*, pages 234–246. Springer, Budapest, Hungary, 2006.

A. Sanfeliu & K.-S. Fu. A Distance Measure between Attributed Relational Graphs for Pattern Recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):353–362, 1983.

H.-P. Schwefel. *Numerical Optimization of Computer Models*. Wiley, 1981.

J. Shapiro. *Theoretical Aspects of Evolutionary Computation*, chapitre Statistical mechanics theory of genetic algorithms, pages 87–108. Springer, 2001.

J. Shi & J. Malik. Normalized Cuts and Image Segmentation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

A. Shokoufandeh, D. Macrini, S. J. Dickinson, K. Siddiqi, & S. W. Zucker. Indexing Hierarchical Structures using Graph Spectra. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1125–1140, 2005.

M. Singh, A. Chatterjee, & S. Chaudhury. Matching Structural Shape Descriptions using Genetic Algorithms. *Pattern Recognition*, 30(9):1451–1462, 1997.

S. Smit & A. Eiben. Comparing Parameter Tuning Methods for Evolutionary Algorithms. Dans *IEEE Congress on Evolutionary Computation (CEC)*. 2009.

J. E. Smith. Coevolving Memetic Algorithms: A Review and Progress Report. *IEEE Trans. on Systems, Man, and Cybernetics-Part B*, 37(1):6–17, 2007.

A. J. Smola & R. Kondor. Kernels and Regularization on Graphs. Dans B. Schölkopf & M. K. Warmuth, Éds., *Learning Theory and Kernel Machines*, vol. 2777 de *Lecture Notes in Artificial Intelligence*, pages 144–158. Springer, 2003.

S. Sorlin, C. Solnon, & J.-M. Jolion. A Generic Graph Distance Measure Based on Multivalent Matchings. Dans A. Kandel, H. Bunke, & M. Last, Éds., *Applied Graph Theory in Computer Vision and Pattern Recognition*, vol. 52 de *Studies in Computational Intelligence*, chapitre A Generic Graph Distance Mesasure Based on Multivalent Matchings, pages 151–181. Springer, 2007.

T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, & C. Whitley. A Comparison of Genetic Sequencing Operators. Dans R. Belew & L. Booker, Éds., *Proc. of the 4th International Conference on Genetic Algorithms*, pages 69–76. Morgan Kaufman, San Mateo, CA, 1991.

C. R. Stephens & J. Cervantes. Just What are Building Blocks? Dans *Foundations of Genetic Algorithms*, vol. 4436 de *LNCS*, pages 15–34. Springer, 2007.

M. J. Streeter. Two Broad Classes of Functions for which a No Free Lunch Result does not Hold. Dans *Genetic and Evolutionary Computation - GECCO*, vol. 2724 de *LNCS*, pages 1418–1430. 2003.

P. N. Suganthan. Structural Pattern Recognition using Genetic Algorithms. *Pattern Recognition*, 35(9):1883–1893, 2002.

G. Syswerda. Schedule Optimization using Genetic Algorithms. Dans L. Davis, Éd., *Handbook of Genetic Algorithms*, pages 332–349. Van Nostrand Reinhold, New York, 1991.

N. Thome, D. Merad, & S. Miguet. Learning Articulated Appearance Models for Tracking Humans: A Spectral Graph Matching Approach. *Signal Processing: Image Communication*, 23(10):769–787, 2008. doi:doi:10.1016/j.image.2008.09.003.

J. Torán. On the Hardness of Graph Isomorphism. *SIAM Journal on Computing*, 33(5):1093–1108, 2004.

R. Torres-Velázquez & V. Estivill-Castro. A Memetic Algorithm Guided by Quicksort for the Error-Correcting Graph Isomorphism Problem. Dans S. Cagnoni, Éd., *EvoWorkshops*, pages 173–182. 2002.

W. Tsai & K.-S. Fu. Error-Correcting Isomorphism of Attributed Relational Graphs for Pattern Analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 9:757–768, 1979.

J. W. Tukey. *Exploratory Data Analysis*. Addison Wesley, 1977.

J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 23(1):31–42, 1976. ISSN 0004-5411. doi:http://doi.acm.org/10.1145/321921.321925.

S. Umeyama. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(5):695–703, 1988.

P. F. Velleman & D. C. Hoaglin. *Applications, Basics, and Computing of Exploratory Data Analysis*. Duxbury Press, 1981.

M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, Massachusetts, 1999.

Y.-K. Wang, K.-C. Fan, & J.-T. Horng. Genetic-Based Search for Error-Correcting Graph Isomorphism. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(4):588–597, 1997.

R. C. Wilson & E. R. Hancock. Structural Matching by Discrete Relaxation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19:634–648, 1997.

D. H. Wolpert & W. G. Macready. No Free Lunch Theorems for Search. Rapport technique SFI-TR-95-02-010, The Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM, USA, 1996.

D. H. Wolpert & W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

D. H. Wolpert & W. G. Macready. Coevolutionary Free Lunches. *IEEE Trans. on Evolutionary Computation*, 9(6):721–735, 2005.

B. Yuan & M. Gallagher. Statistical Racing Techniques for Improved Empirical Evaluation of Evolutionary Algorithms. Dans *Parallel Problem Solving from Nature - PPSN*. 2004.

S. Zampelli, Y. Deville, C. Solnon, & P. Dupont. Filtering for Subgraph Isomorphism. Dans *Int. Conf. on Principles and Practice of Constraint Programming*, vol. 4741 de *LNCS*, pages 728–742. Springer, 2007.

Annexes

Annexe A

Sensibilité au bruit des paramètres

A.1 Moteur générationnel

A.1.1 Bruit uniforme

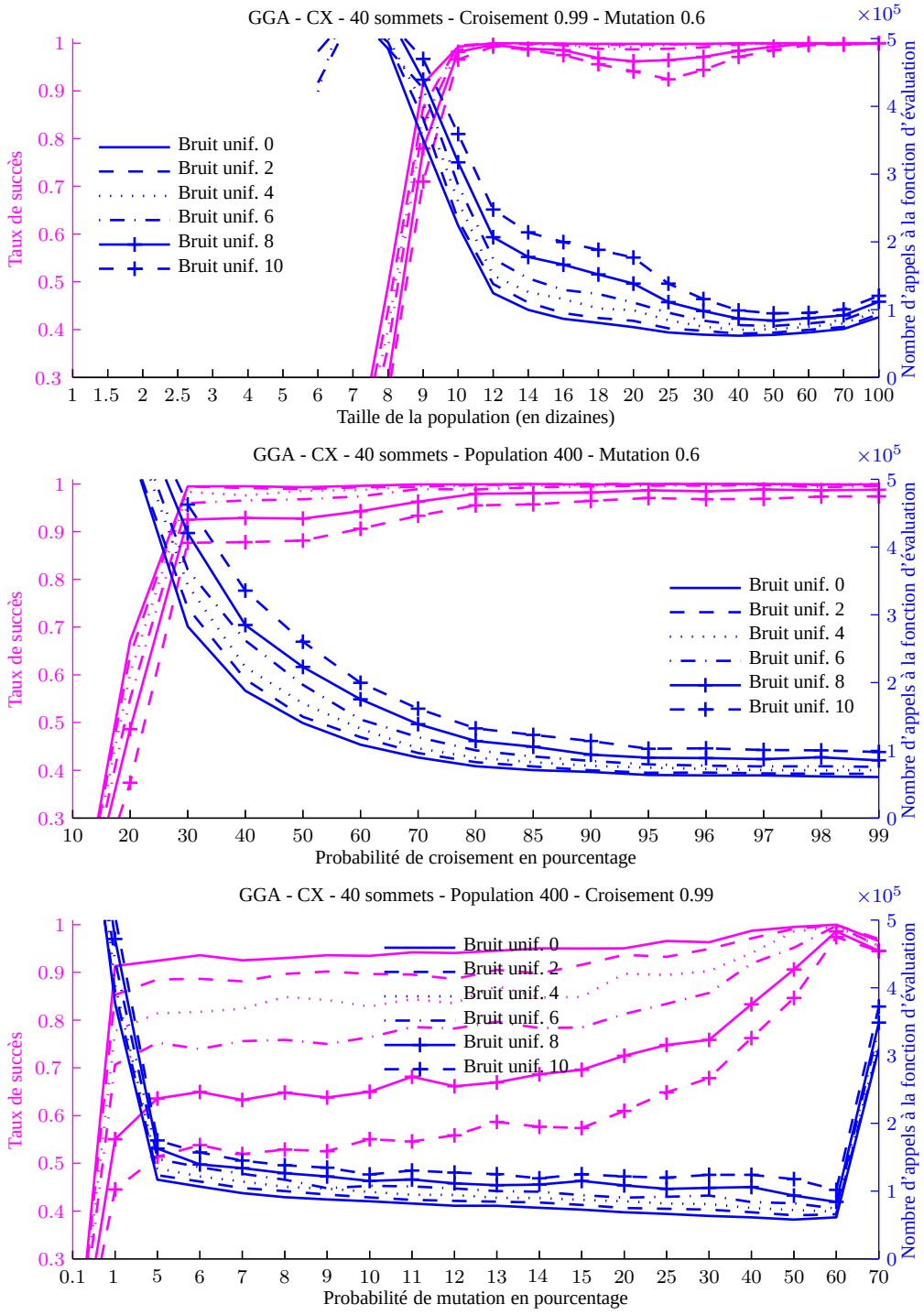

FIGURE A.1 – Sensibilité au bruit uniforme des paramètres du CX dans un moteur générationnel.

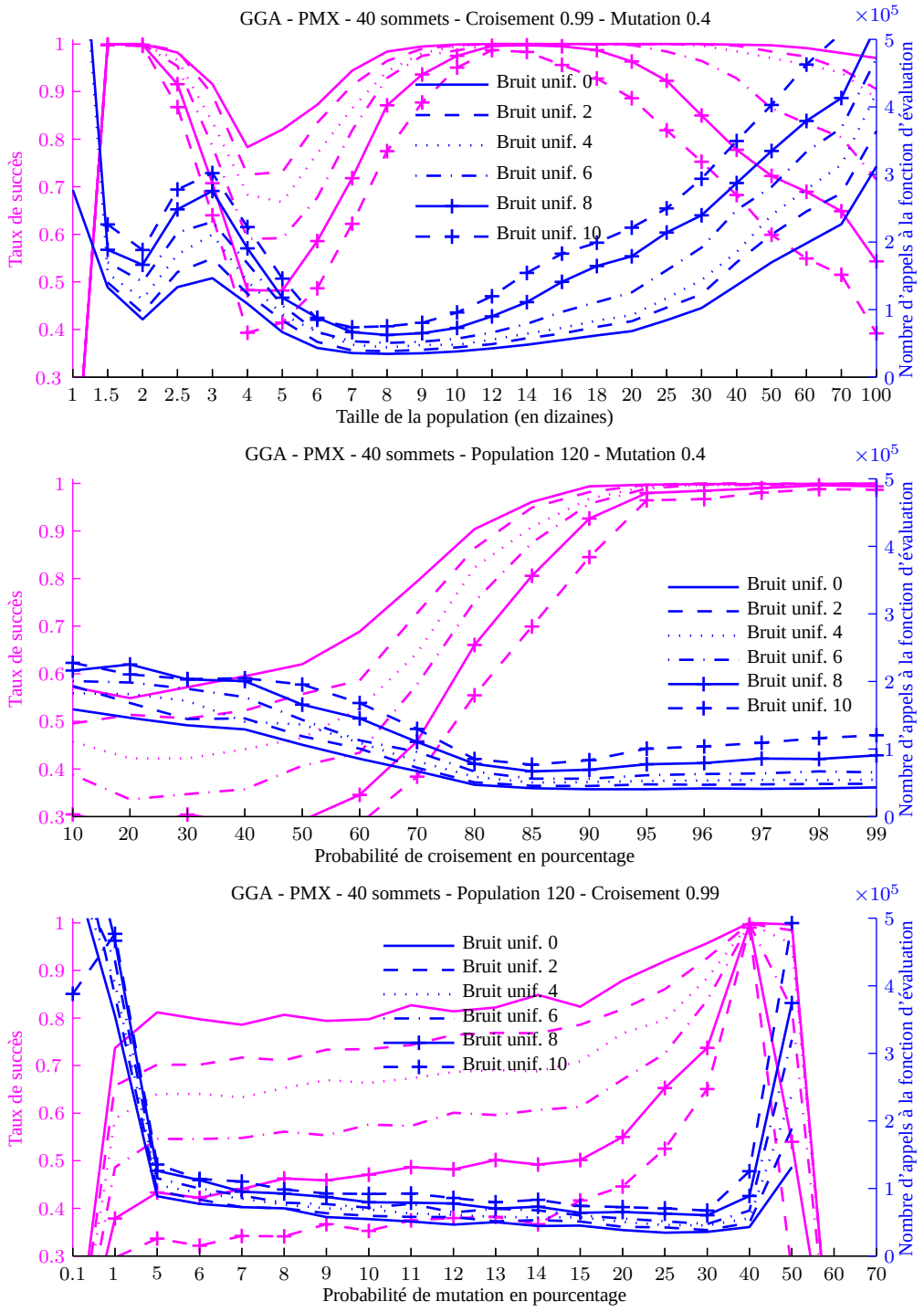

FIGURE A.2 – Sensibilité au bruit uniforme des paramètres du PMX dans un moteur générationnel.

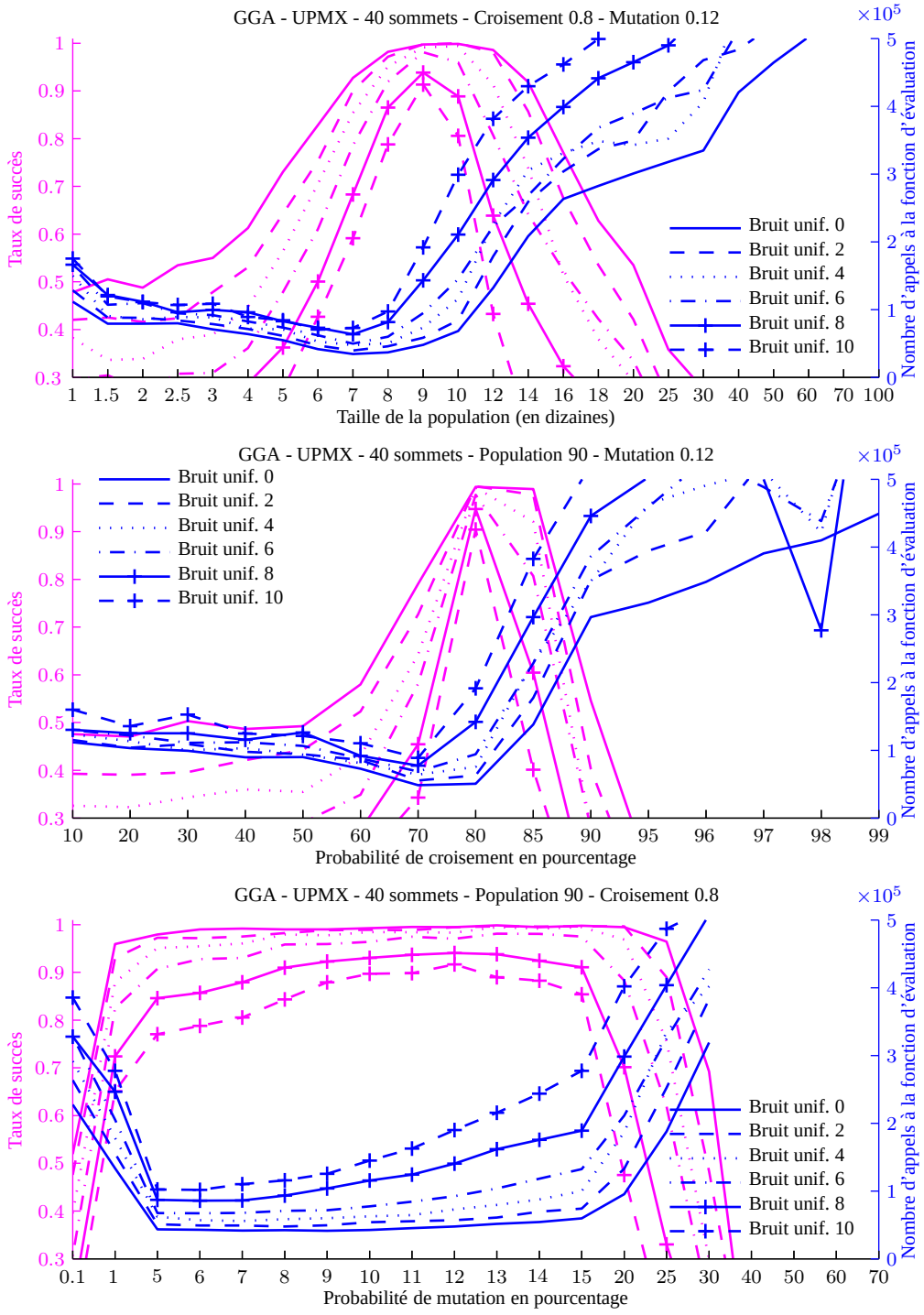

FIGURE A.3 – Sensibilité au bruit uniforme des paramètres du UPMX dans un moteur générationnel.

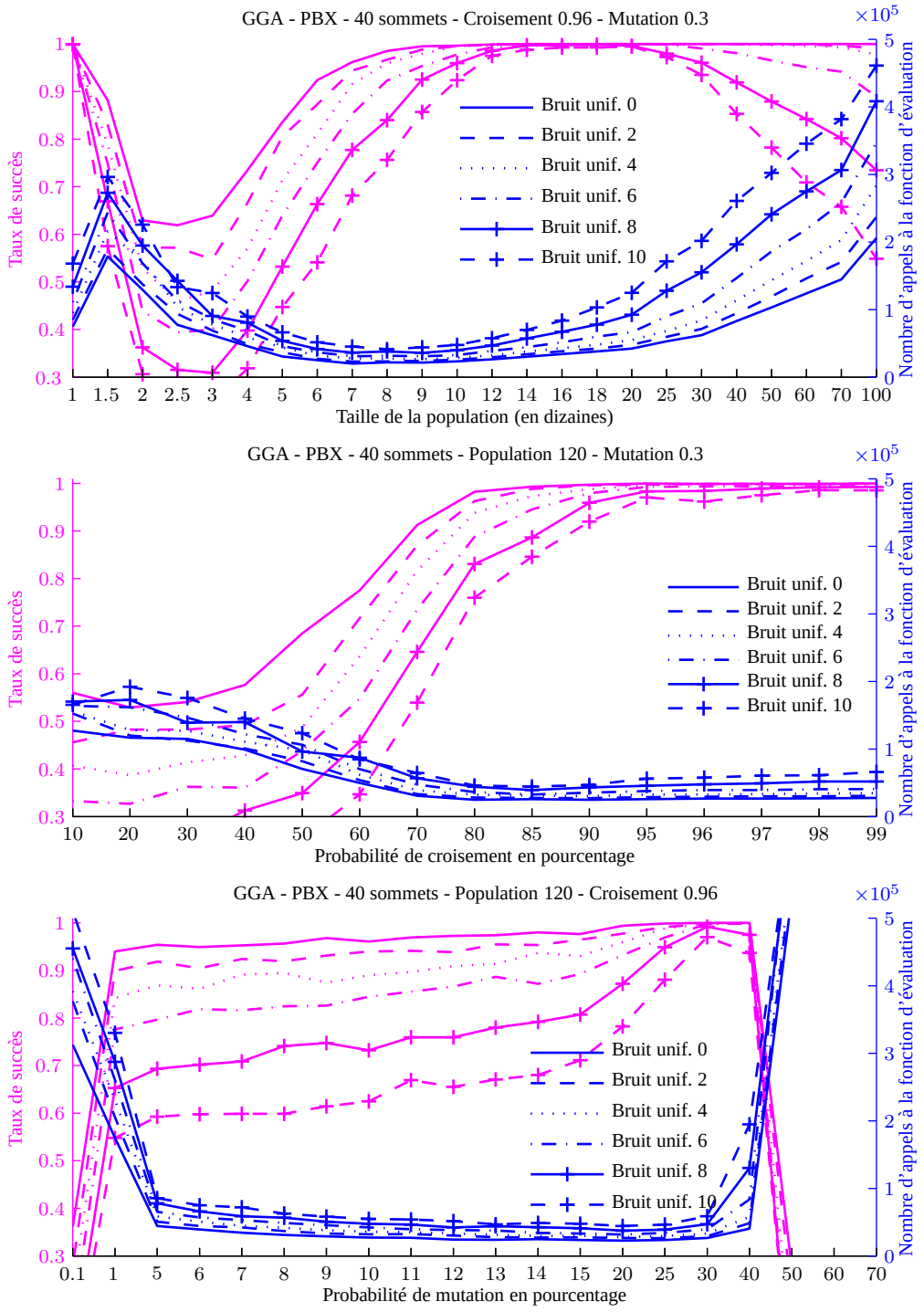

FIGURE A.4 – Sensibilité au bruit uniforme des paramètres du PBX dans un moteur générationnel.

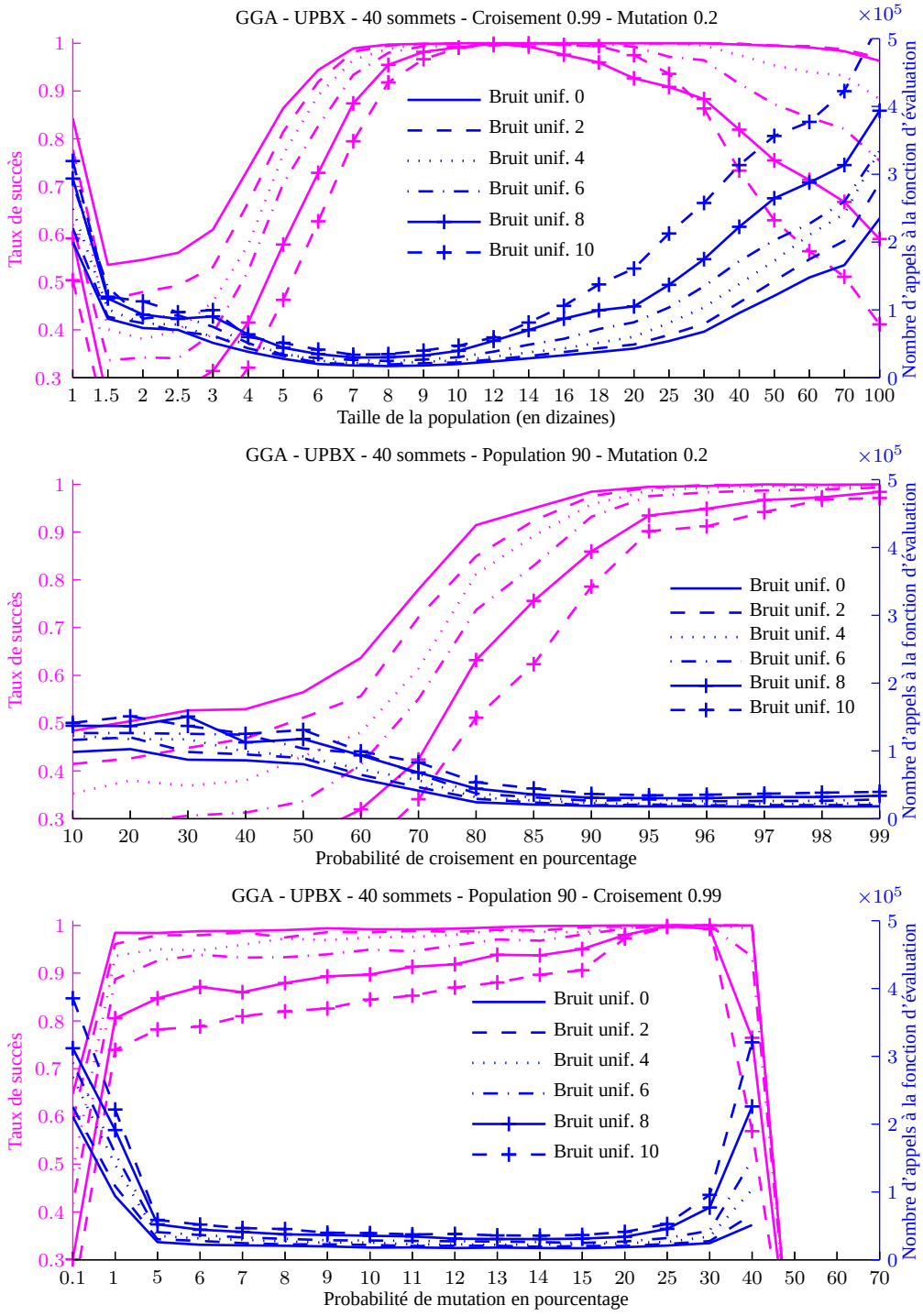

FIGURE A.5 – Sensibilité au bruit uniforme des paramètres du UPBX dans un moteur générationnel.

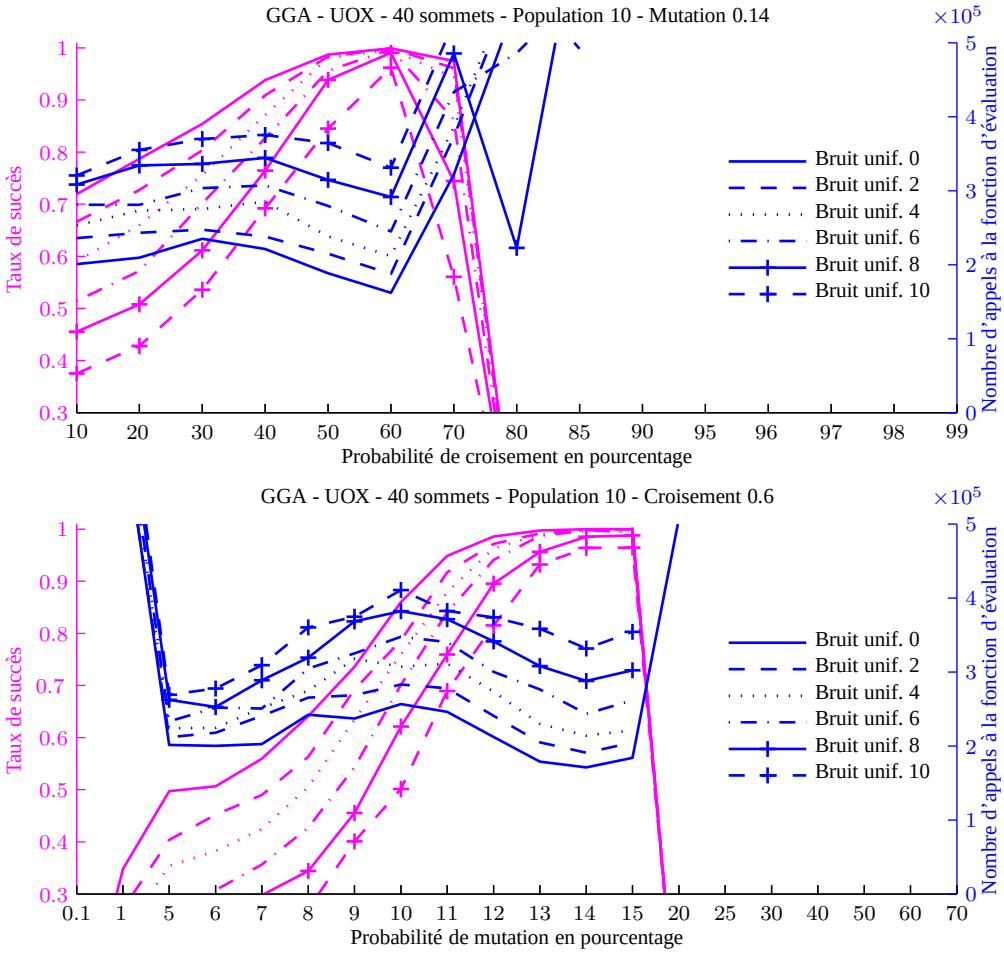

FIGURE A.6 – Sensibilité au bruit uniforme des paramètres du UOX dans un moteur générationnel.

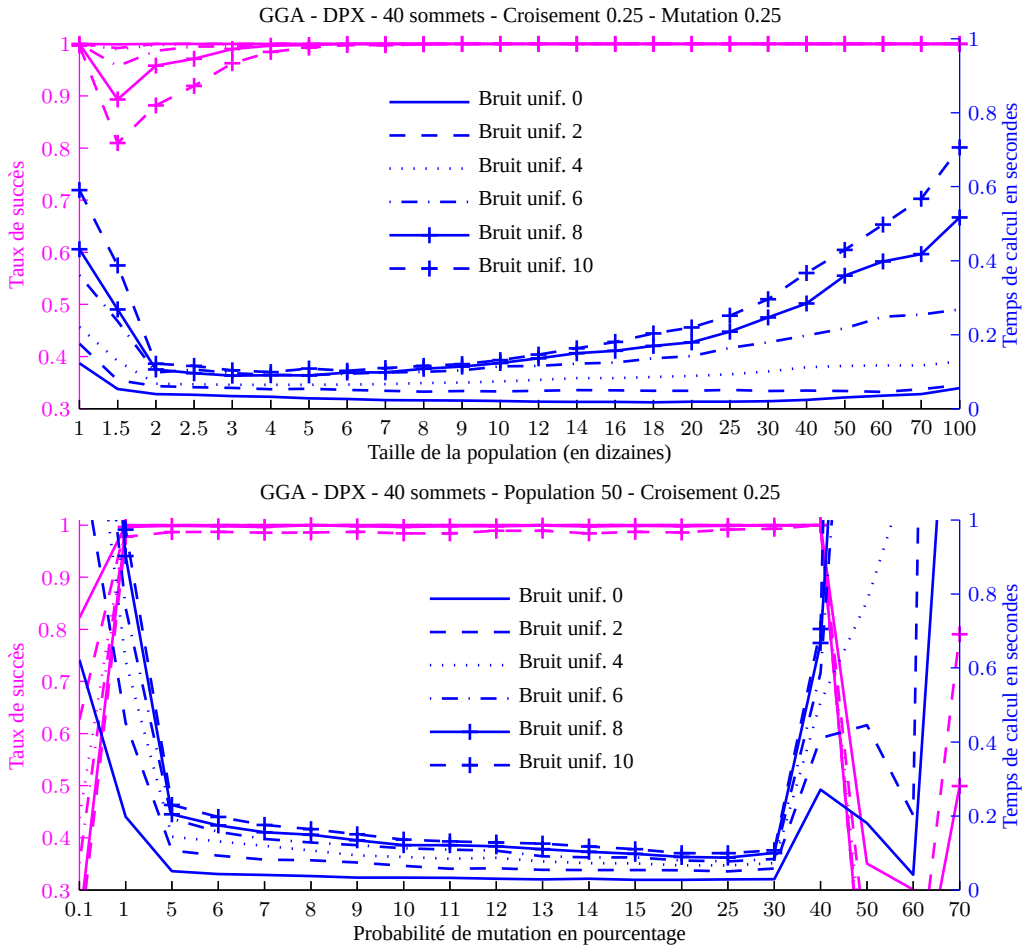

FIGURE A.7 – Sensibilité au bruit uniforme des paramètres du DPX dans un moteur générationnel.

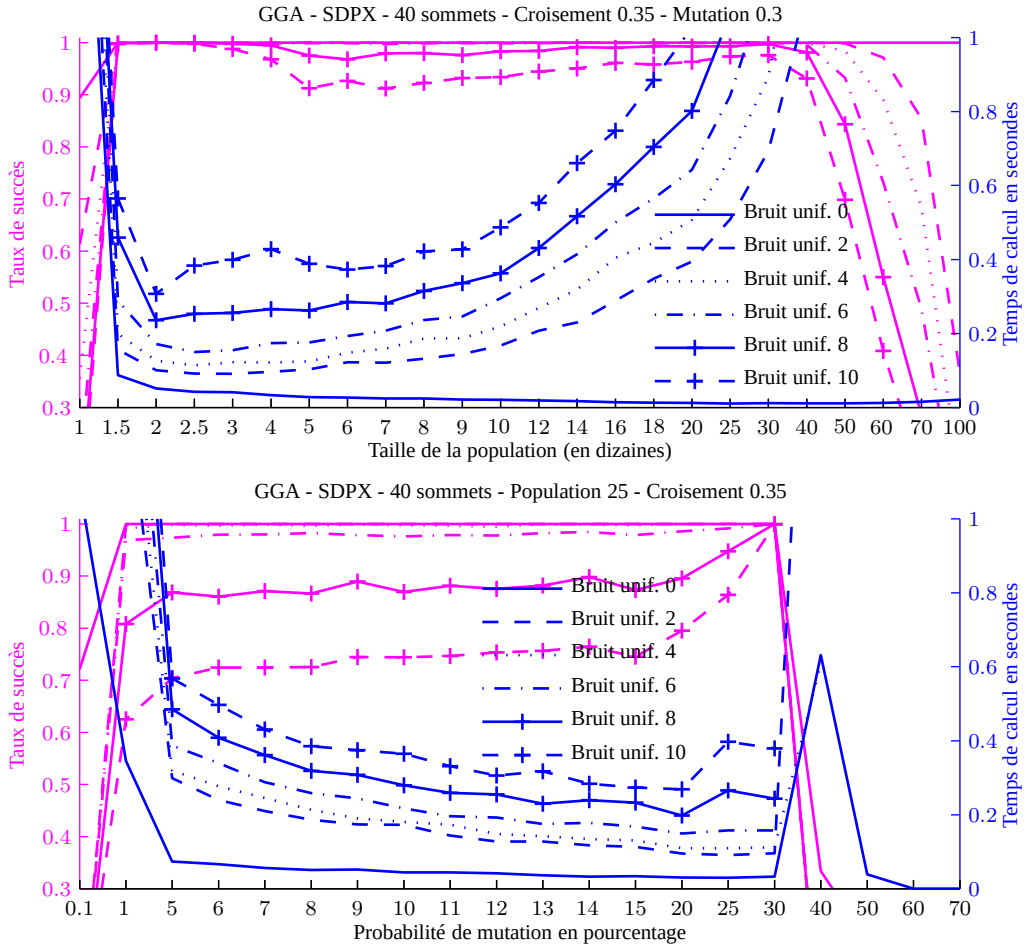

FIGURE A.8 – Sensibilité au bruit uniforme des paramètres du SDPX dans un moteur générationnel.

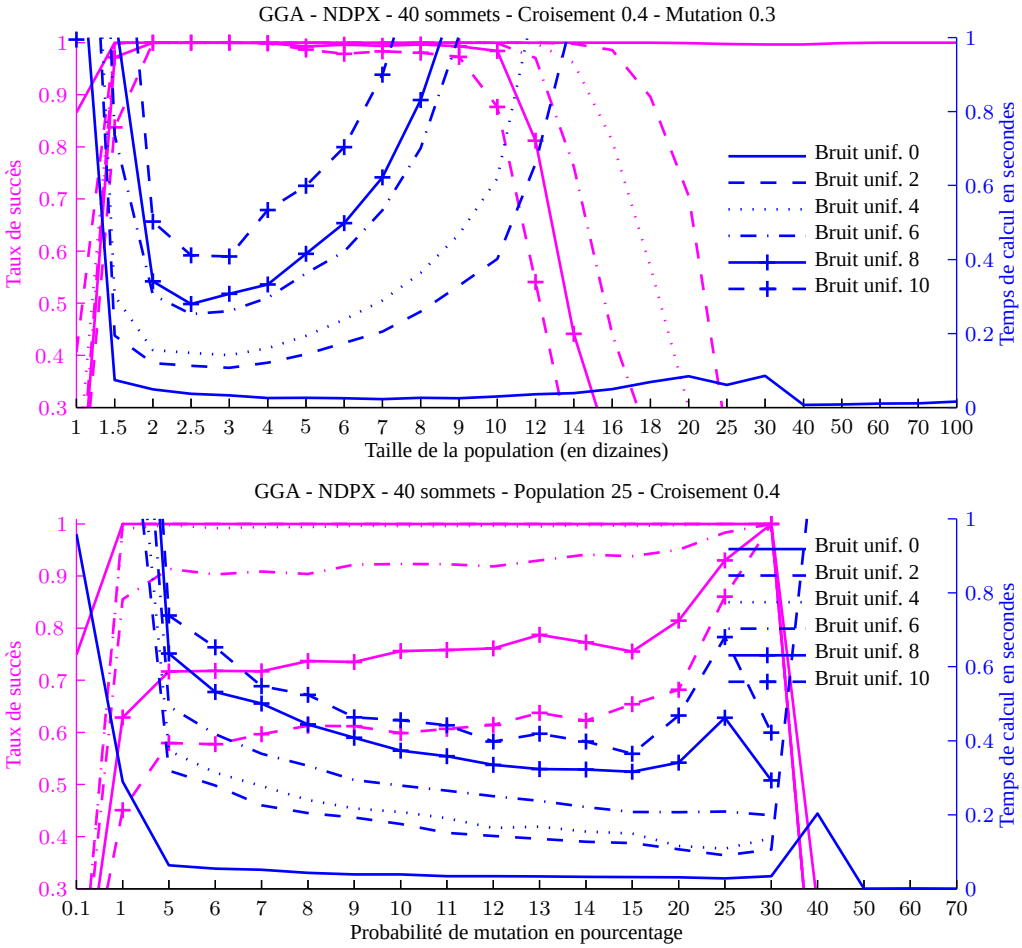

FIGURE A.9 – Sensibilité au bruit uniforme des paramètres du NDPX dans un moteur générationnel.

A.1.2 Bruit gaussien

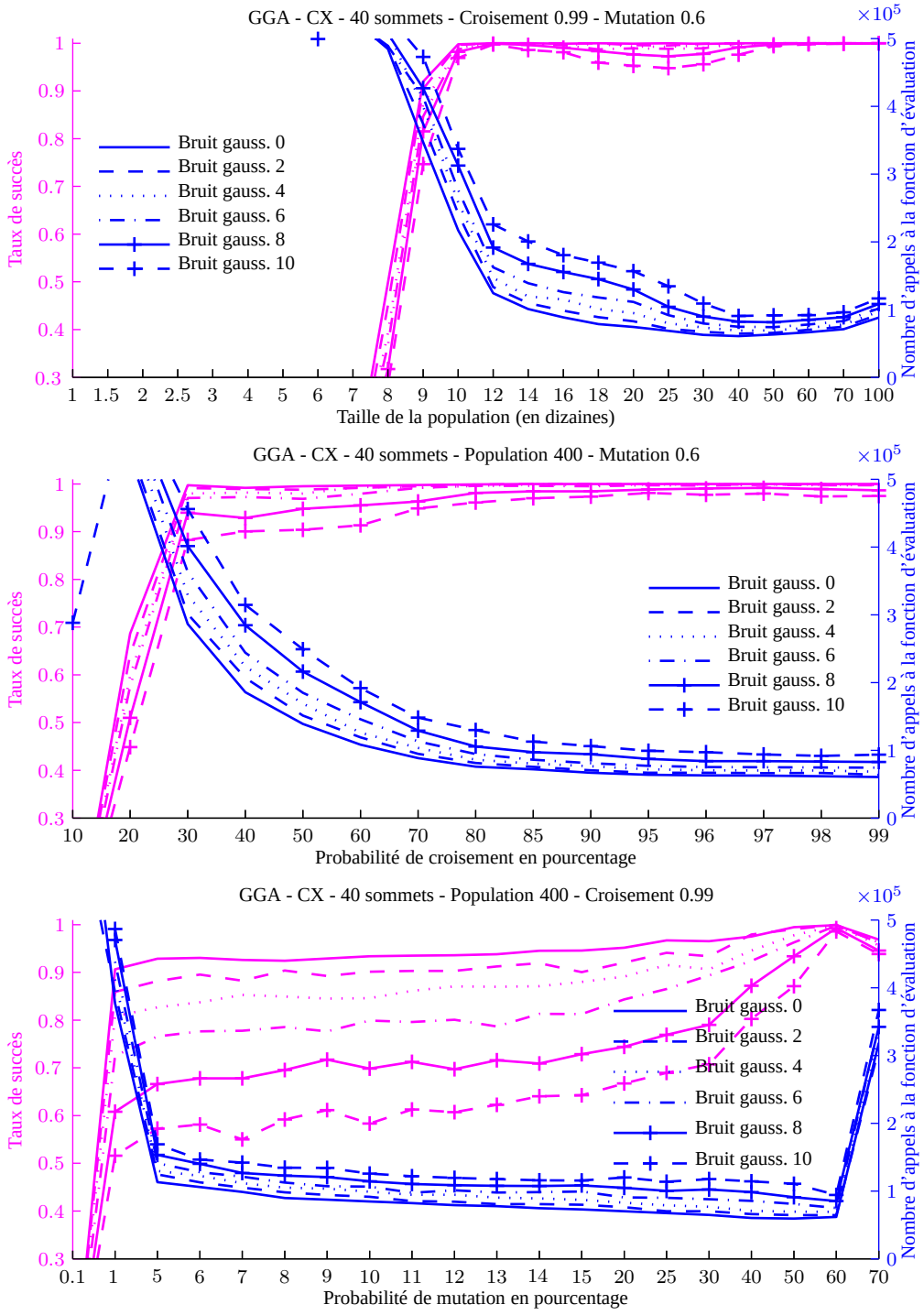

FIGURE A.10 – Sensibilité au bruit gaussien des paramètres du CX dans un moteur générationnel.

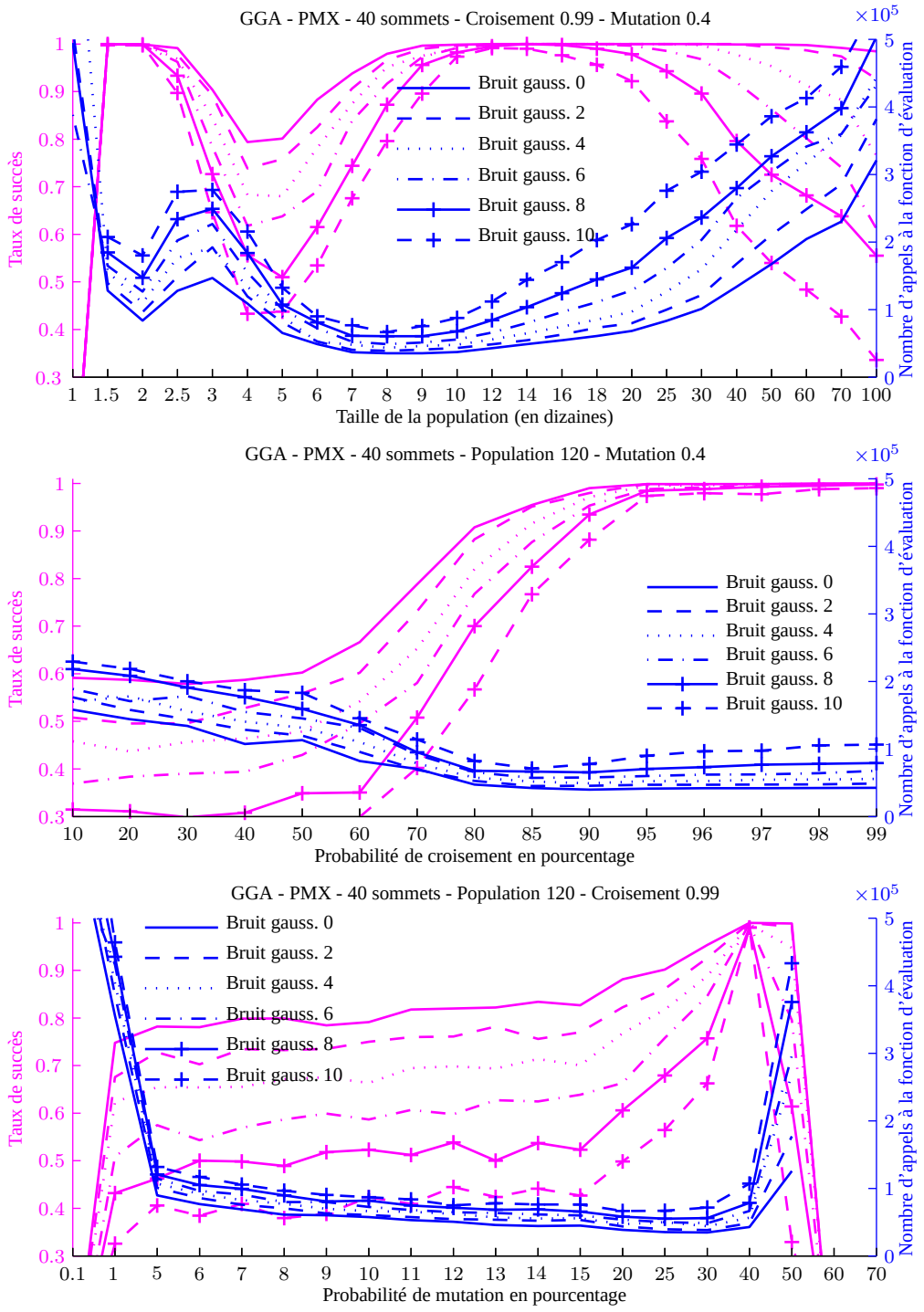

FIGURE A.11 – Sensibilité au bruit gaussien des paramètres du PMX dans un moteur générationnel.

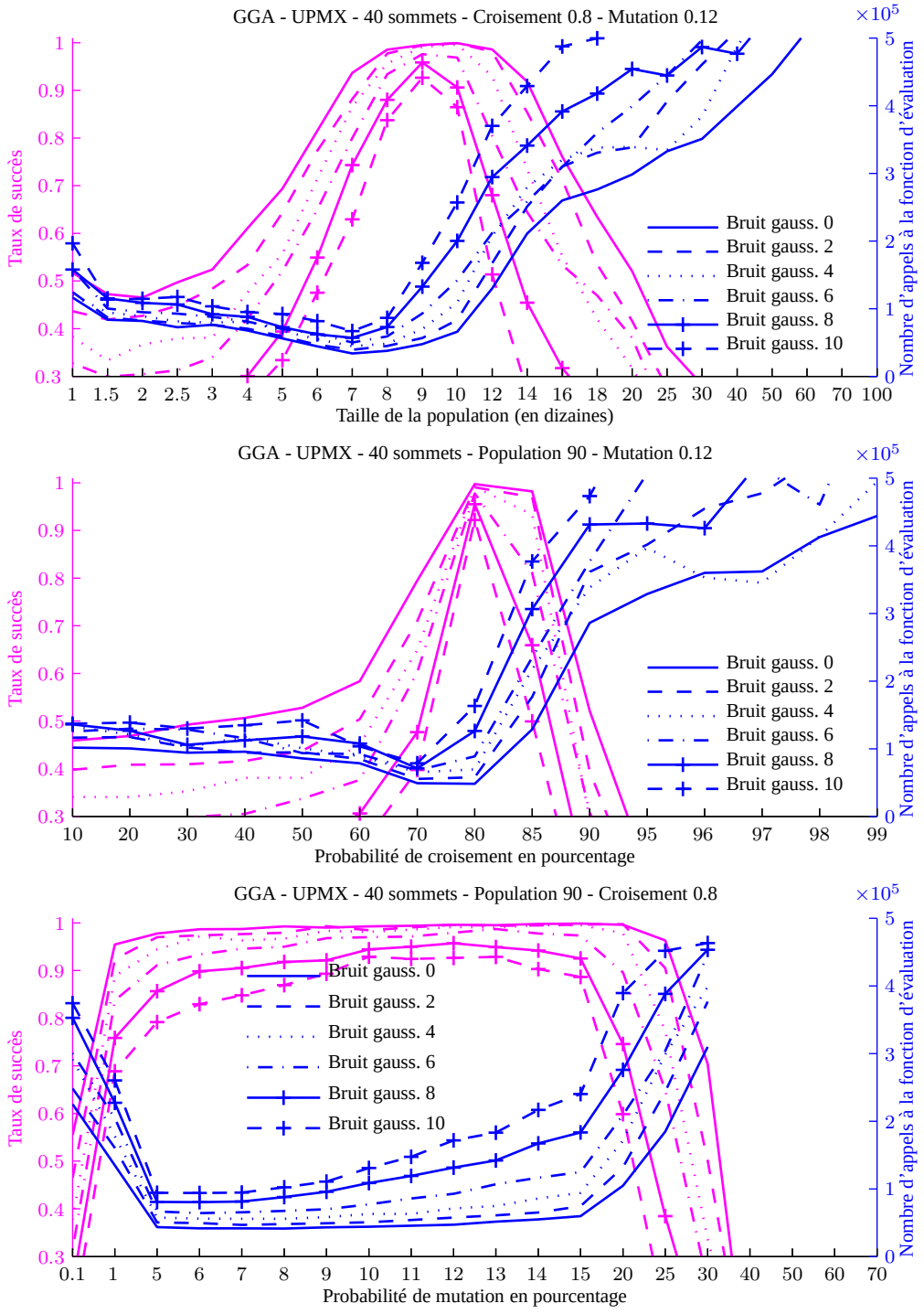

FIGURE A.12 – Sensibilité au bruit gaussien des paramètres du UPMX dans un moteur générationnel.

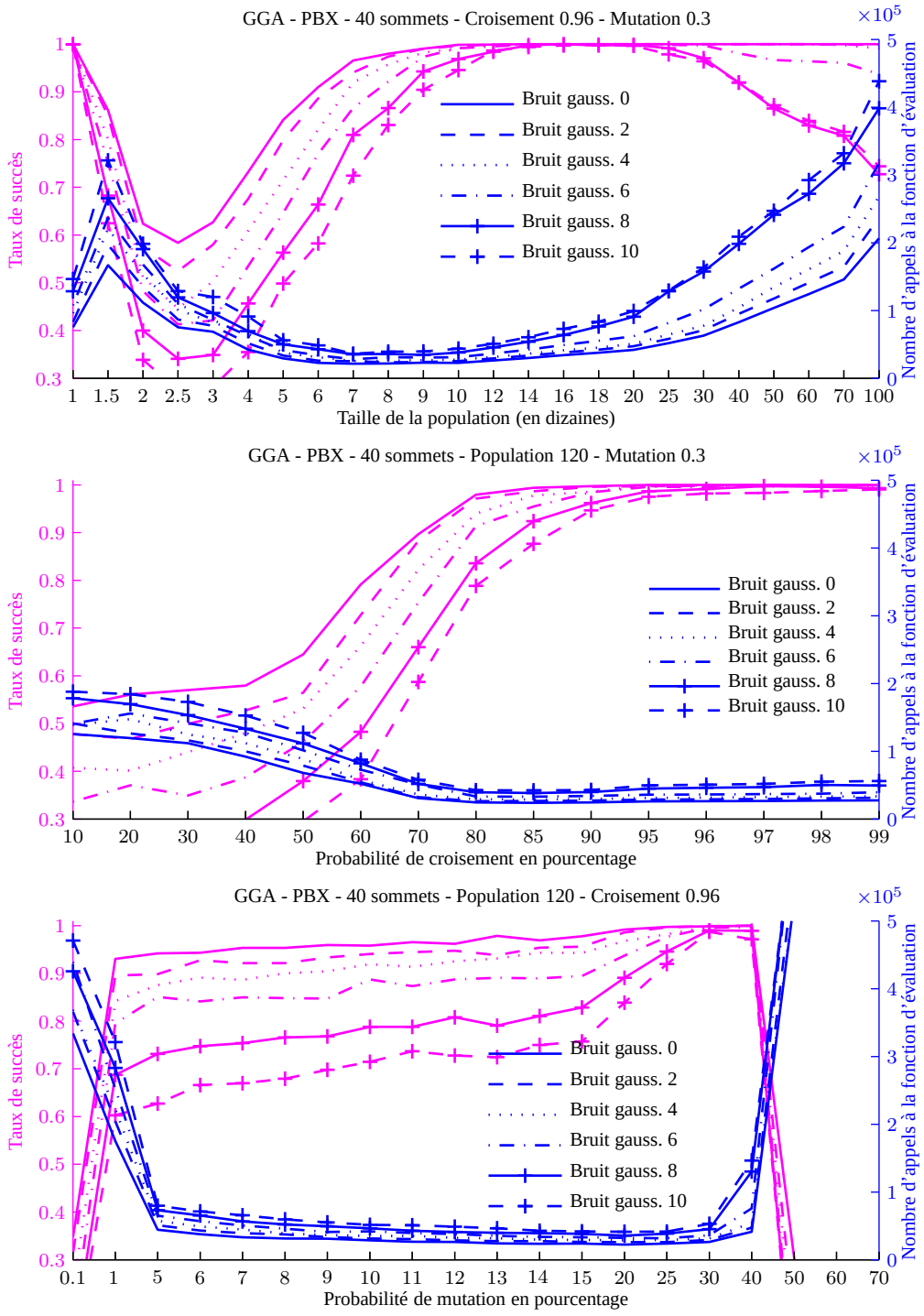

FIGURE A.13 – Sensibilité au bruit gaussien des paramètres du PBX dans un moteur générationnel.

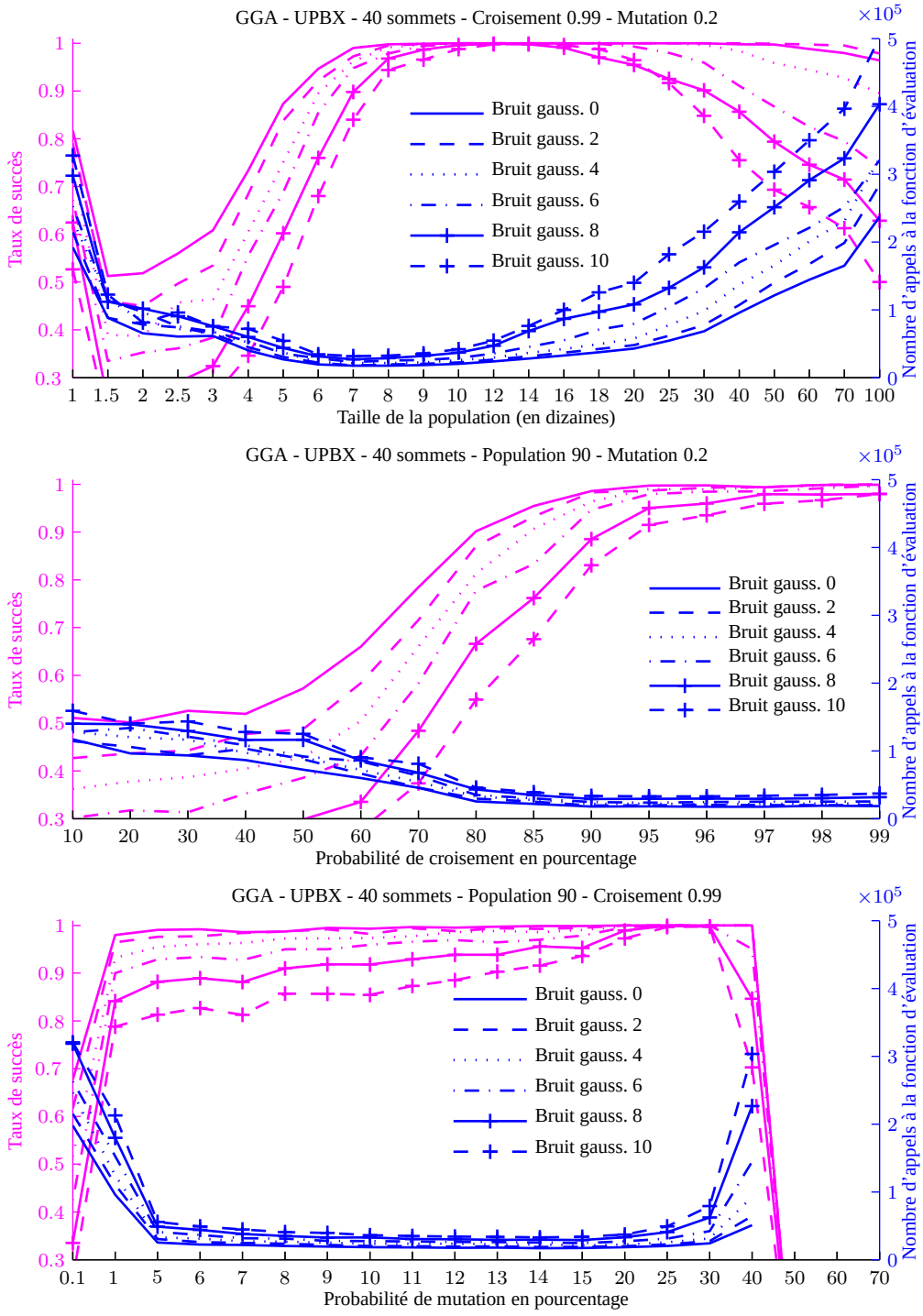

FIGURE A.14 – Sensibilité au bruit gaussien des paramètres du UPBX dans un moteur générationnel.

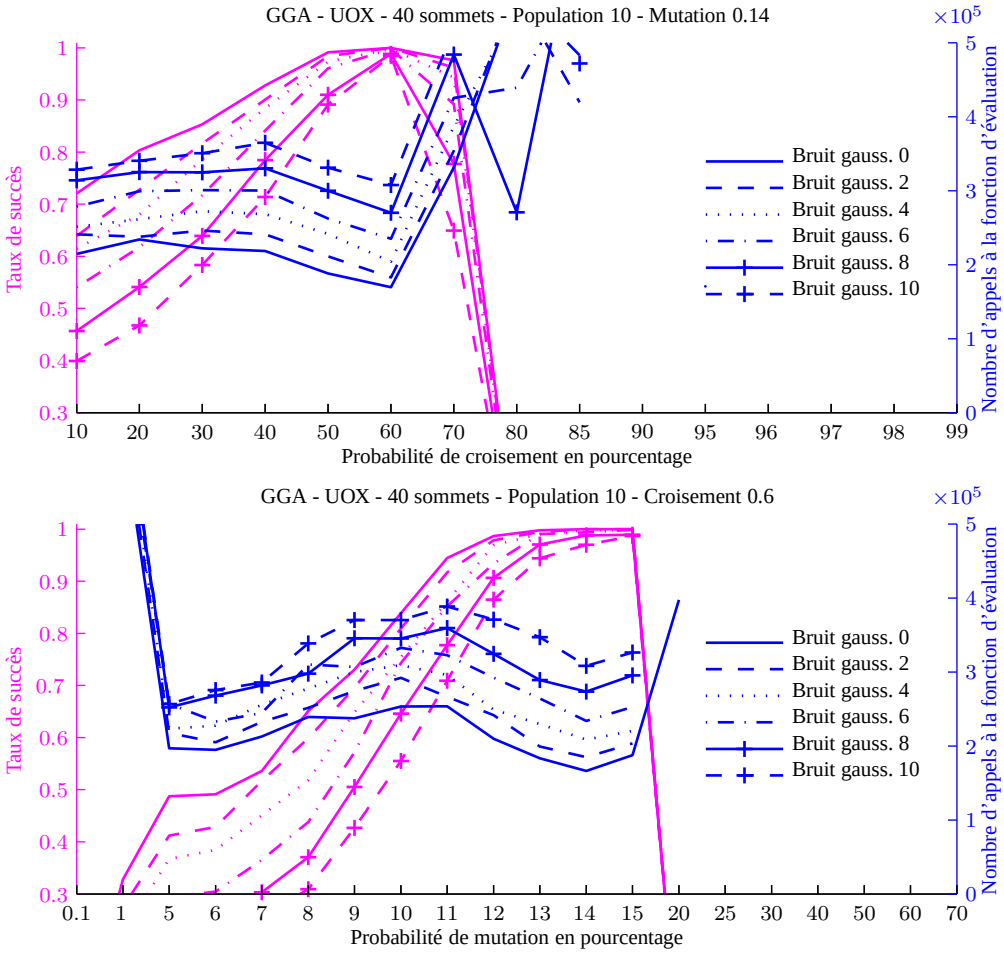

FIGURE A.15 – Sensibilité au bruit gaussien des paramètres du UOX dans un moteur générationnel.

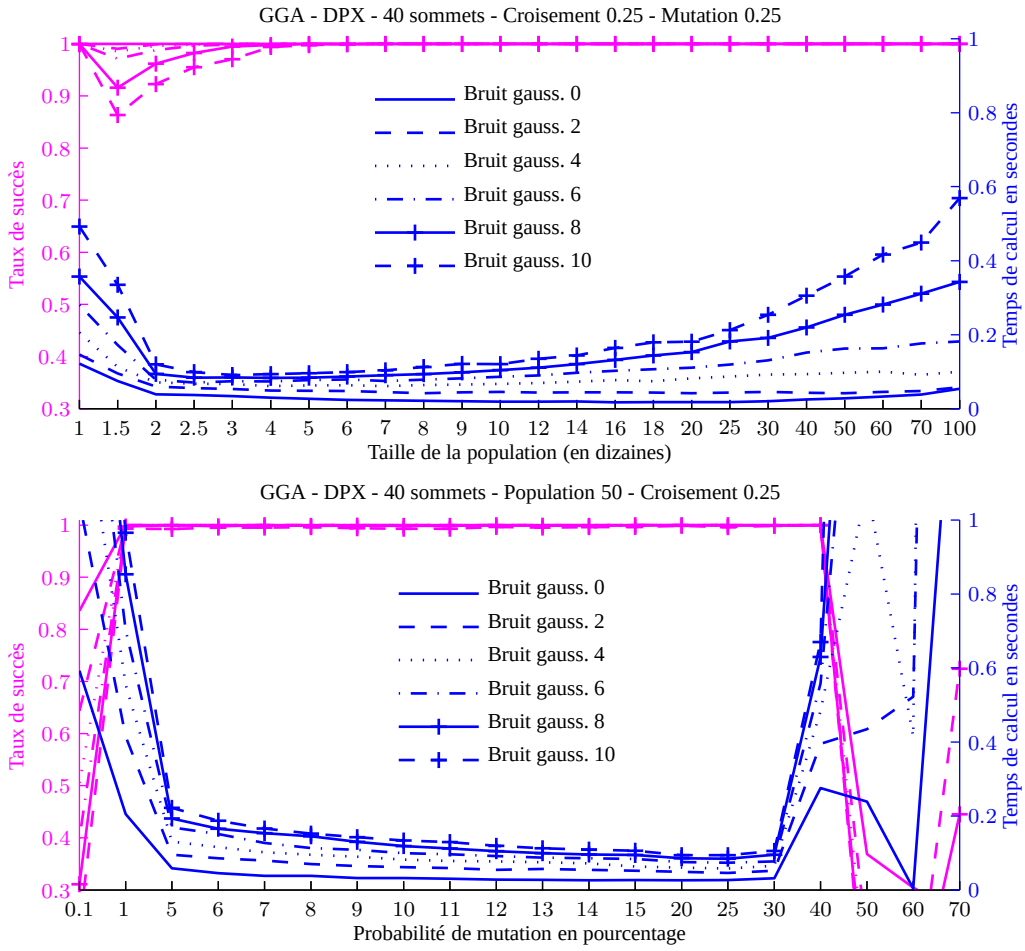

FIGURE A.16 – Sensibilité au bruit gaussien des paramètres du DPX dans un moteur générationnel.

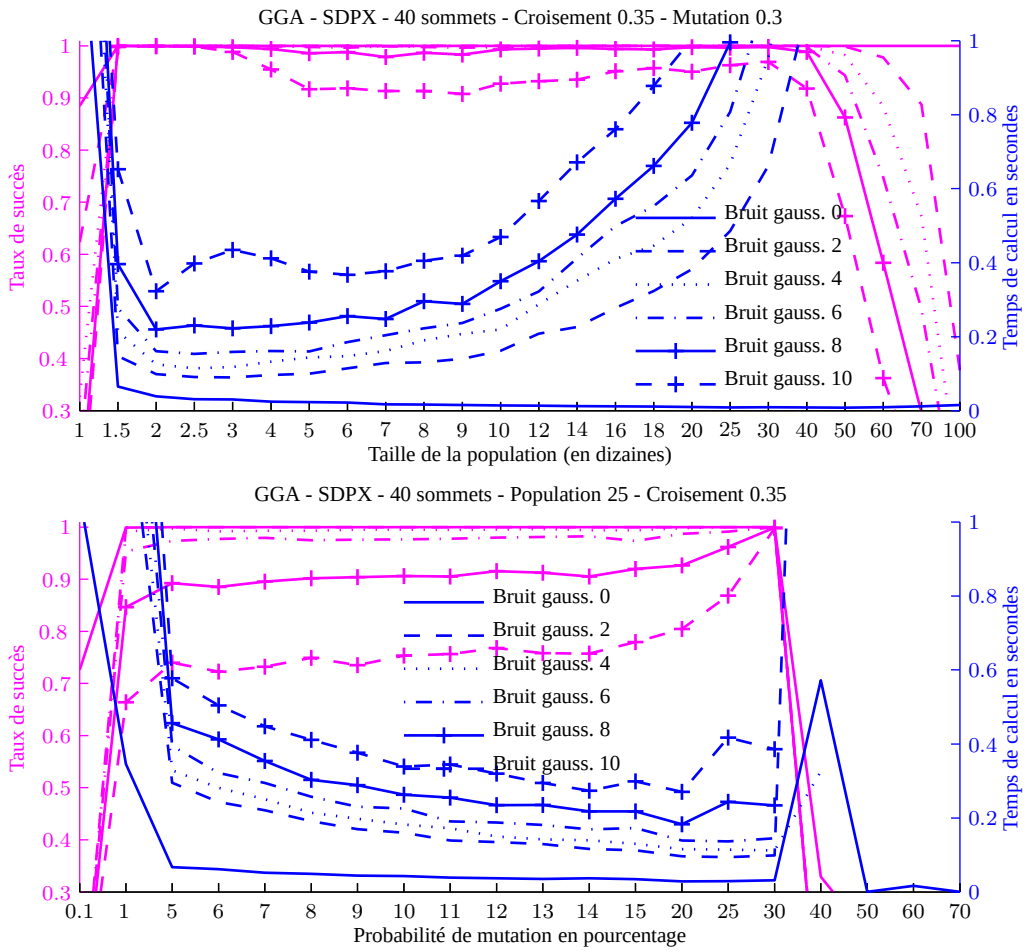

FIGURE A.17 – Sensibilité au bruit gaussien des paramètres du SDPX dans un moteur générationnel.

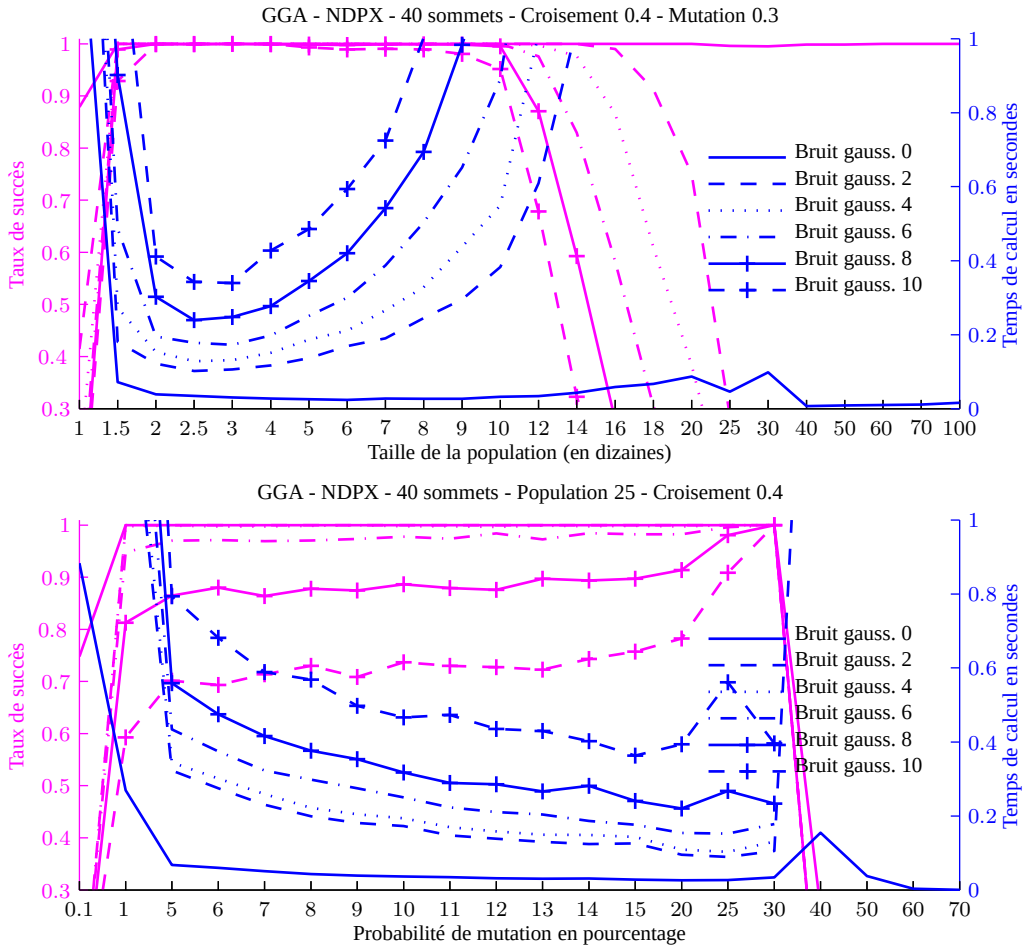

FIGURE A.18 – Sensibilité au bruit gaussien des paramètres du NDPX dans un moteur générationnel.

A.2 Moteur *steady-state*

A.2.1 Bruit uniforme

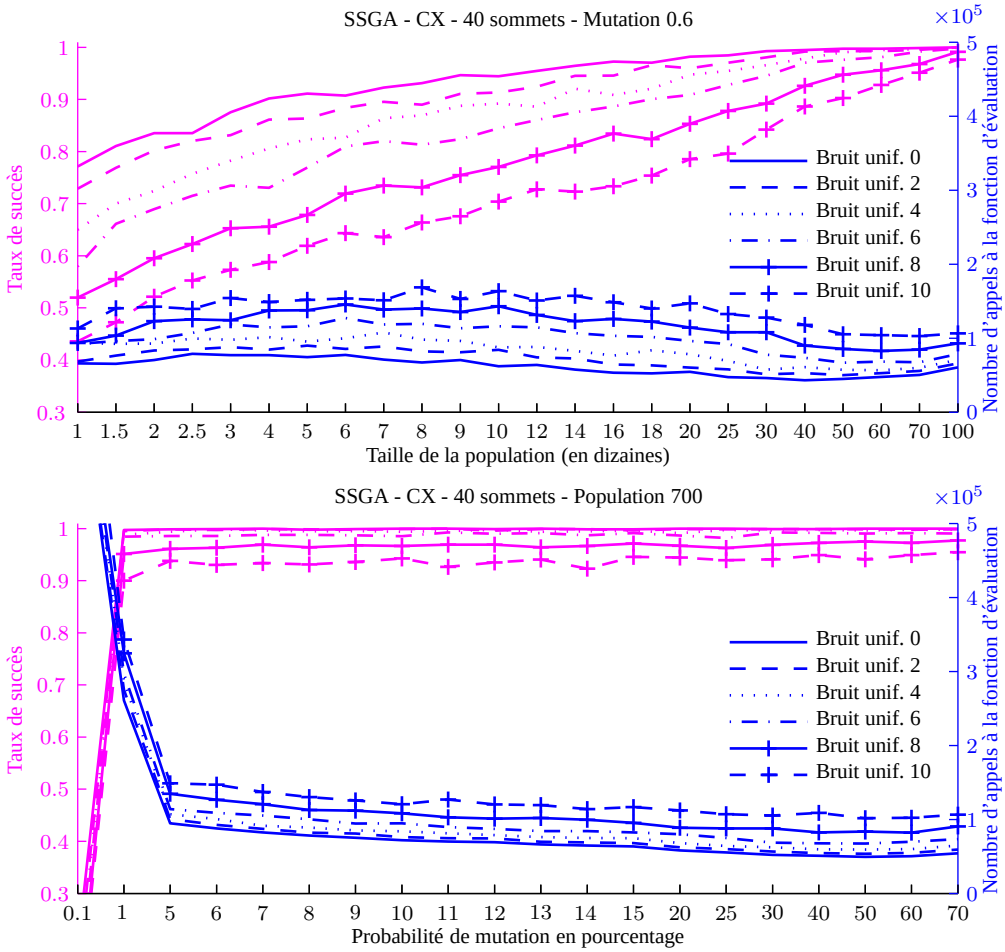

FIGURE A.19 – Sensibilité au bruit uniforme des paramètres du CX dans un moteur *steady-state*.

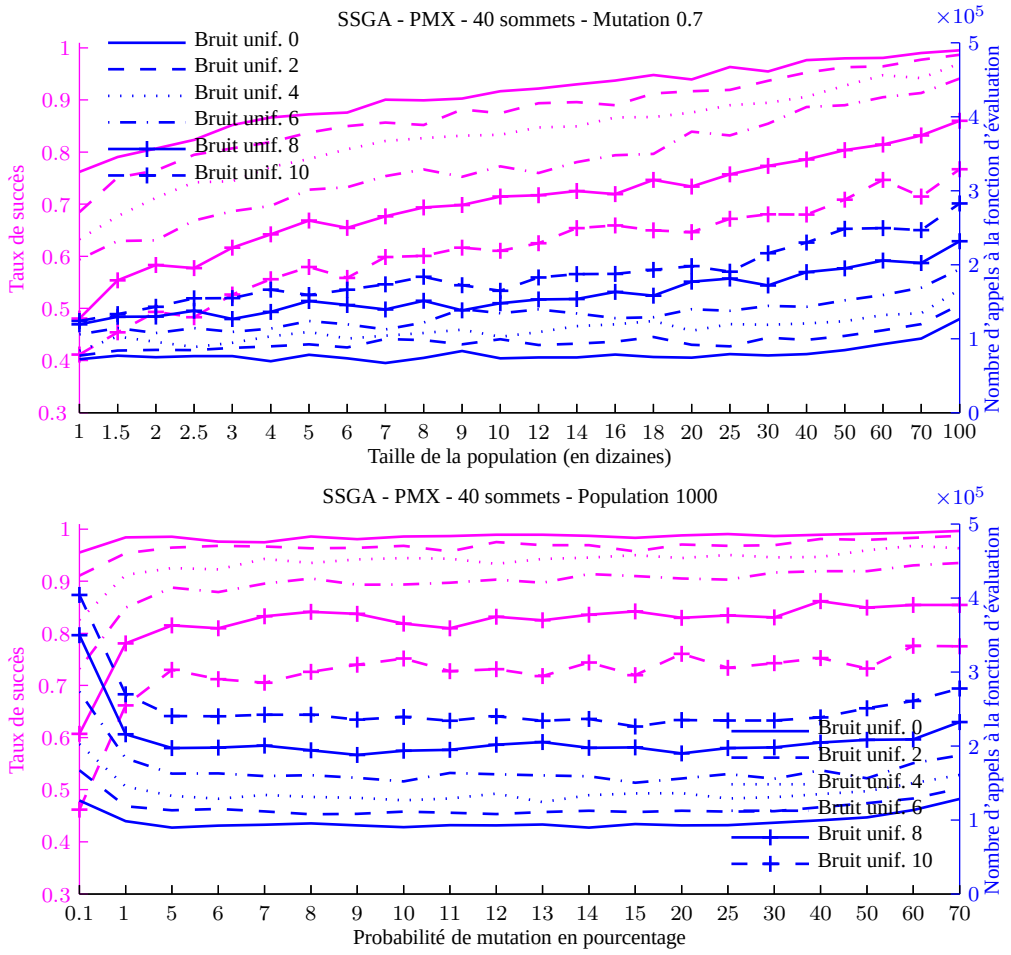

FIGURE A.20 – Sensibilité au bruit uniforme des paramètres du PMX dans un moteur *steady-state*.

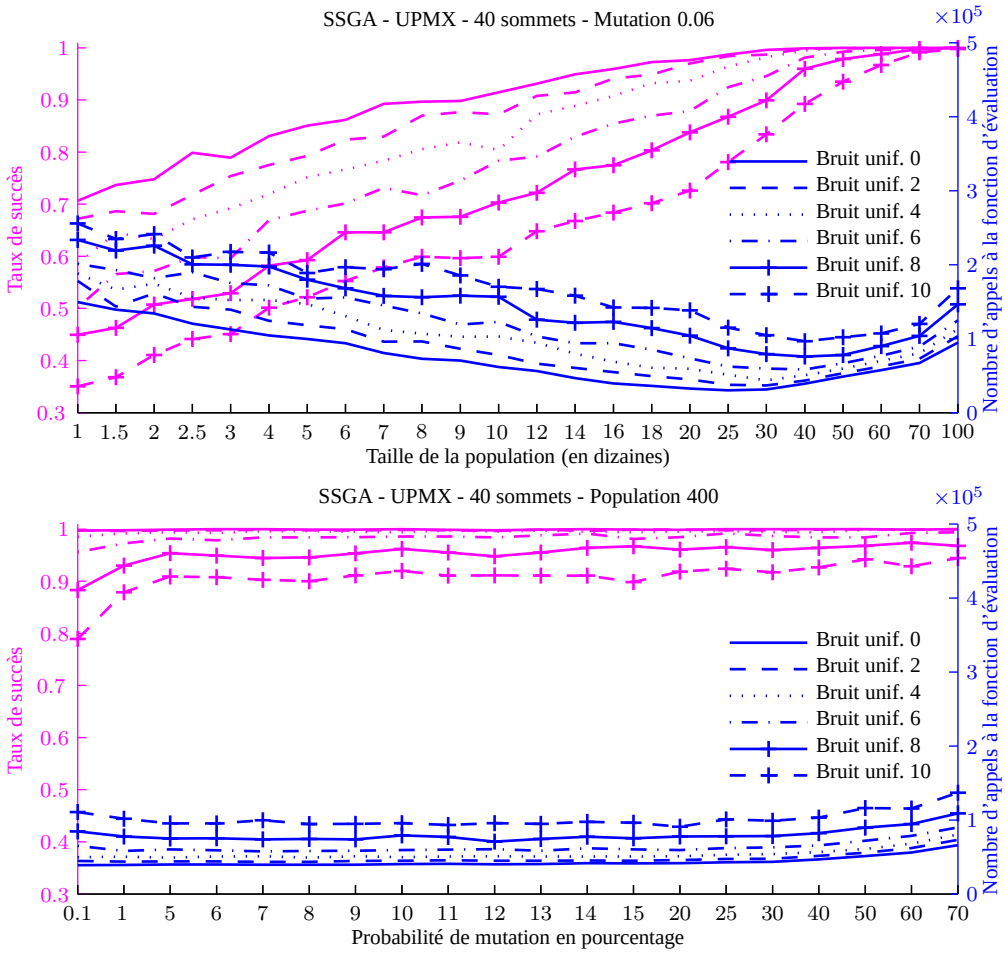

FIGURE A.21 – Sensibilité au bruit uniforme des paramètres du UPMX dans un moteur *steady-state*.

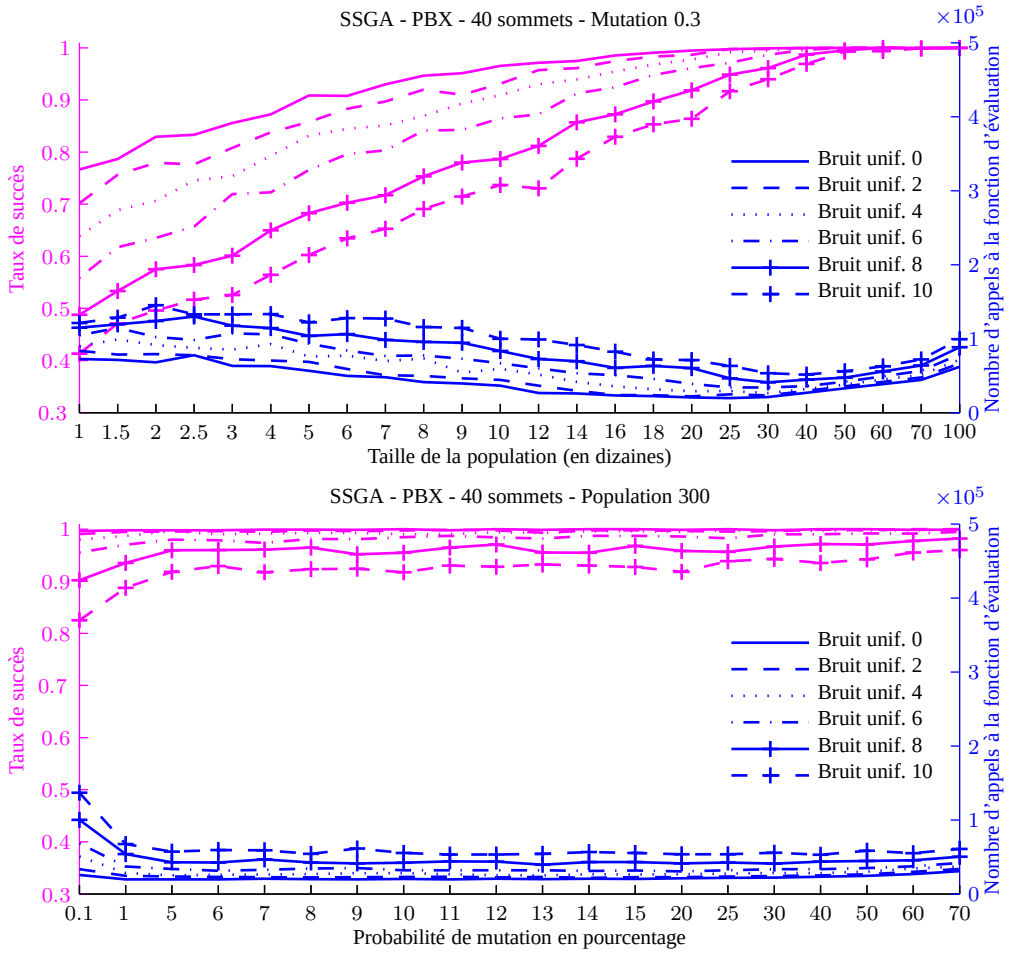

FIGURE A.22 – Sensibilité au bruit uniforme des paramètres du PBX dans un moteur *steady-state*.

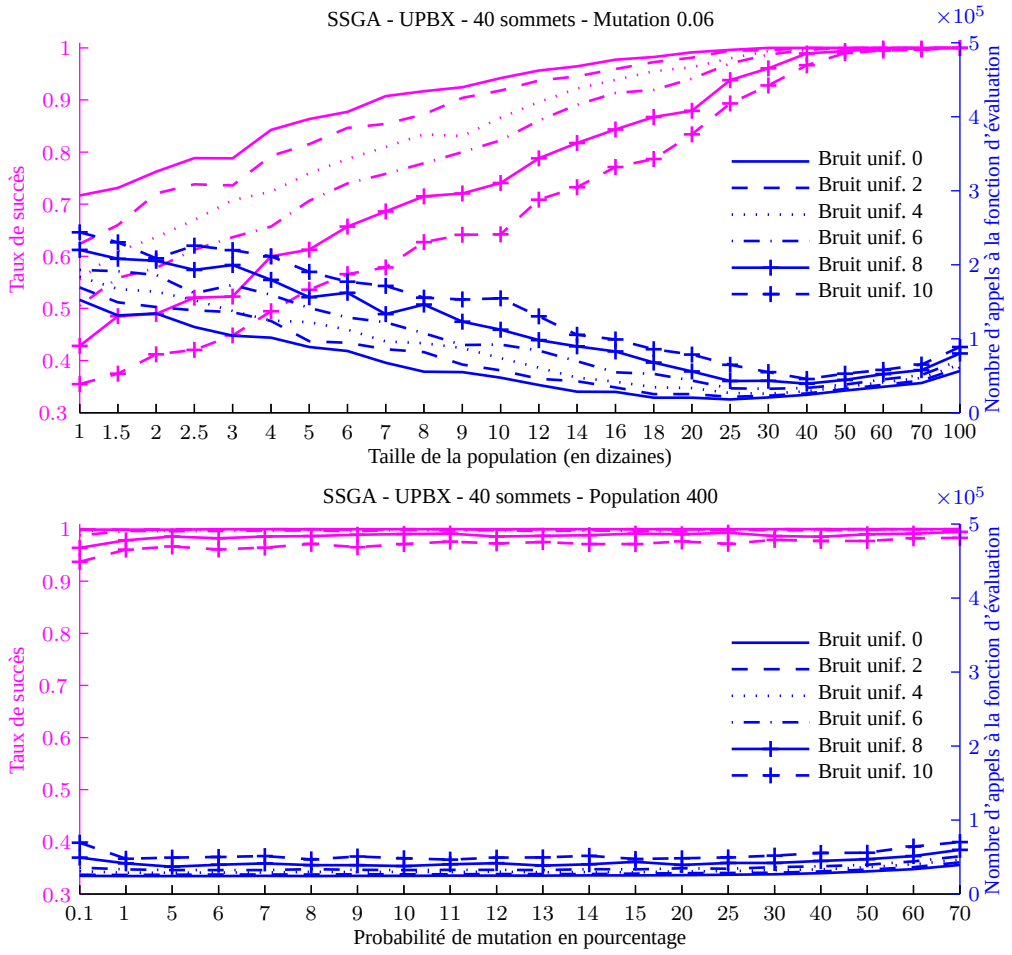

FIGURE A.23 – Sensibilité au bruit uniforme des paramètres du UPBX dans un moteur *steady-state*.

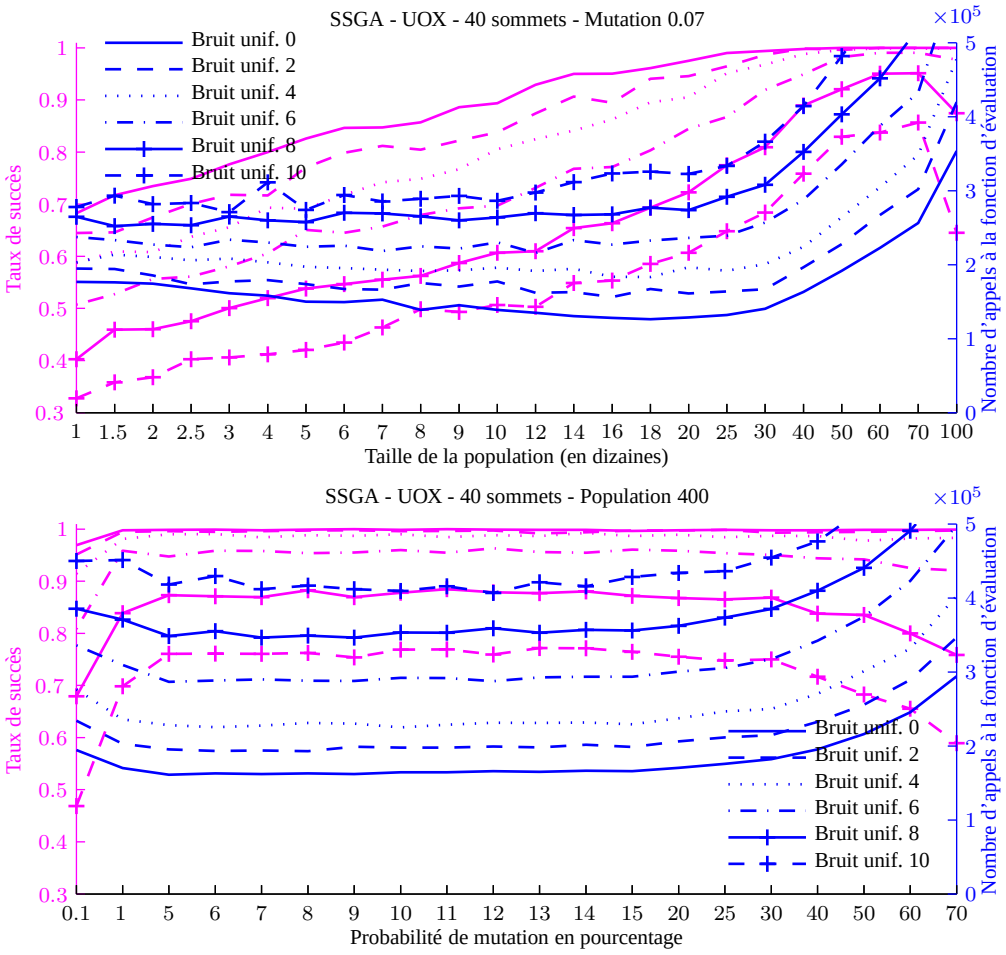

FIGURE A.24 – Sensibilité au bruit uniforme des paramètres du UOX dans un moteur *steady-state*.

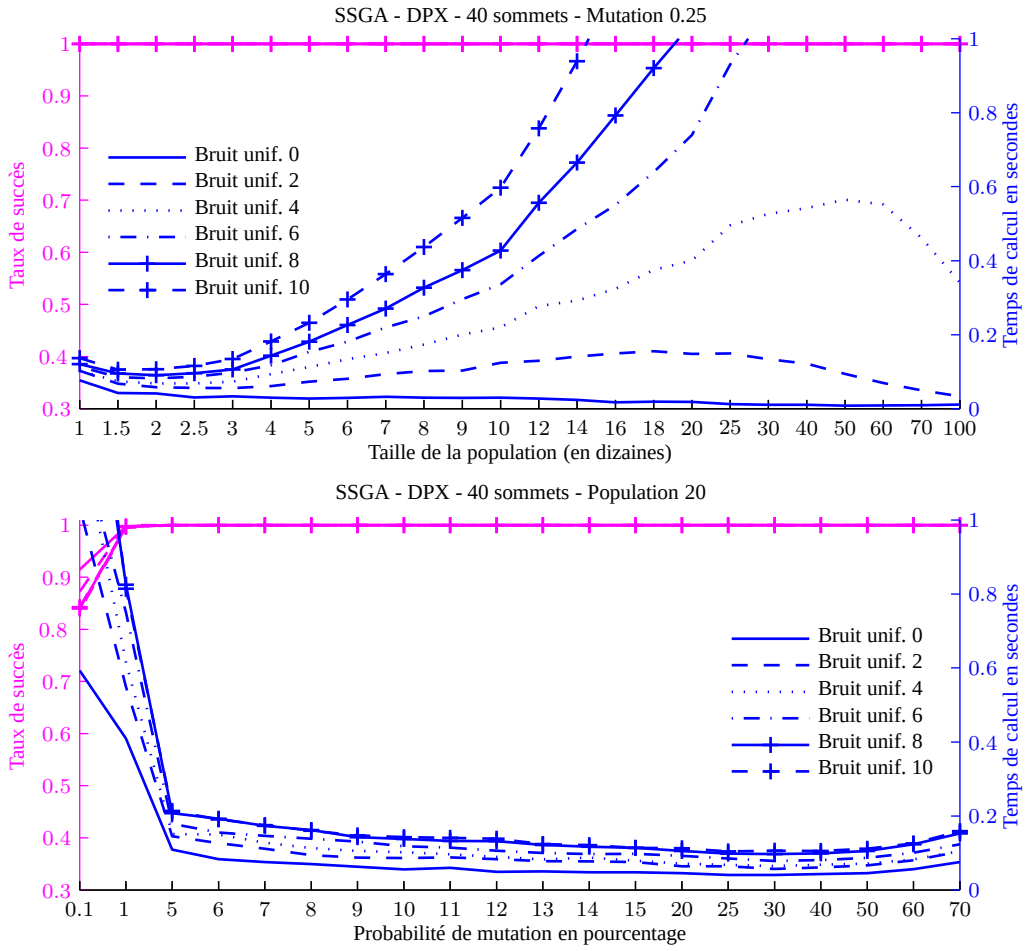

FIGURE A.25 – Sensibilité au bruit uniforme des paramètres du DPX dans un moteur *steady-state*.

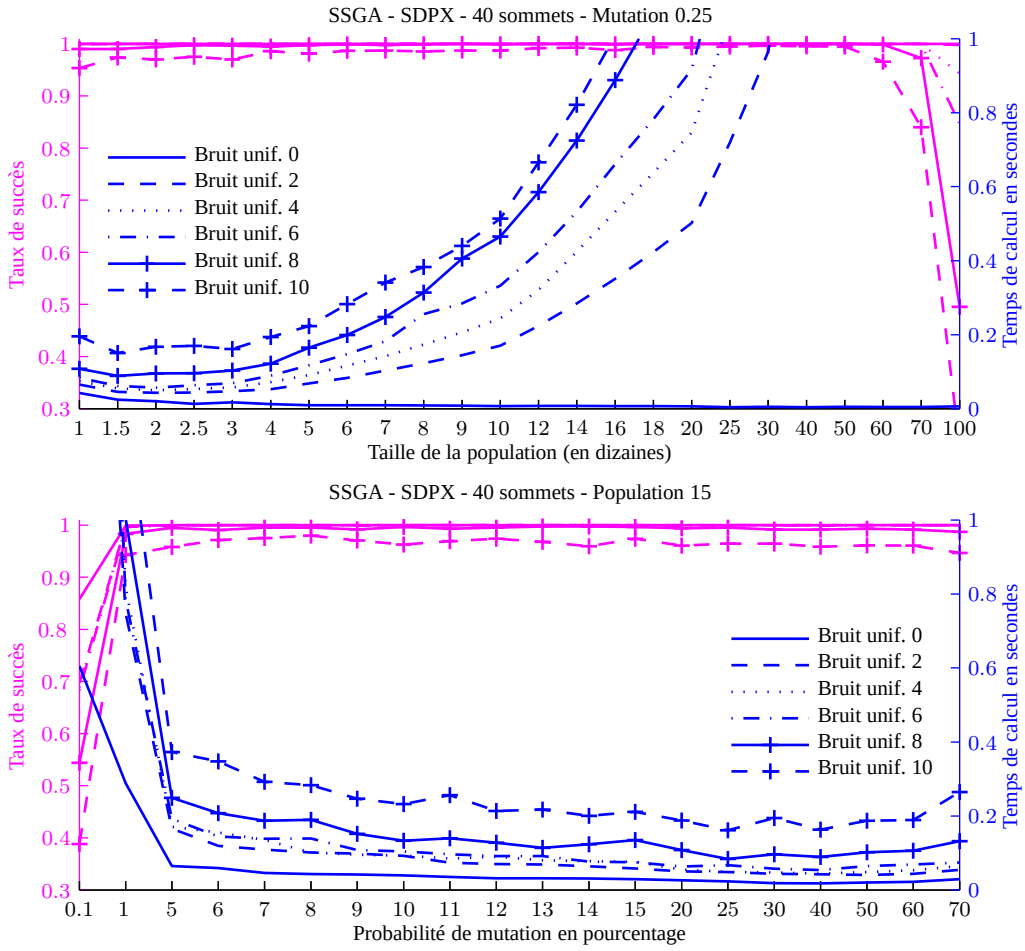

FIGURE A.26 – Sensibilité au bruit uniforme des paramètres du SDPX dans un moteur *steady-state*.

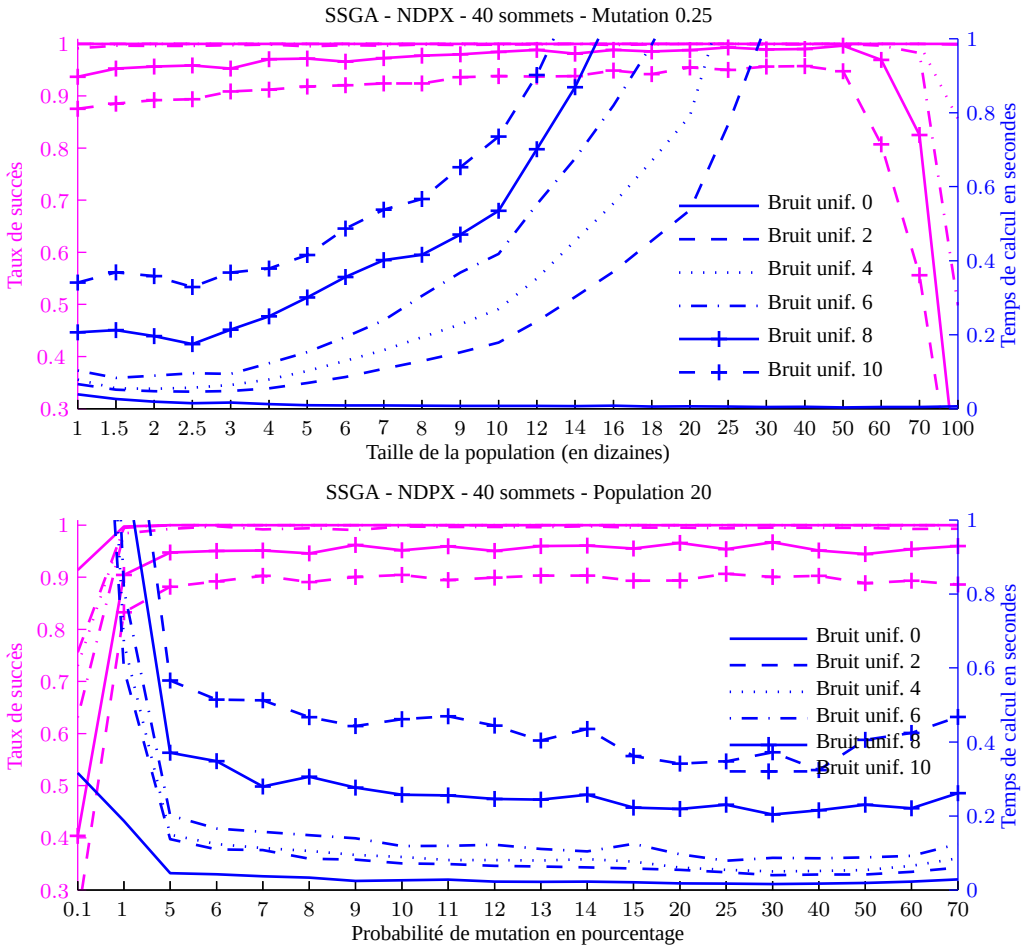

FIGURE A.27 – Sensibilité au bruit uniforme des paramètres du NDPX dans un moteur *steady-state*.

A.2.2 Bruit gaussien

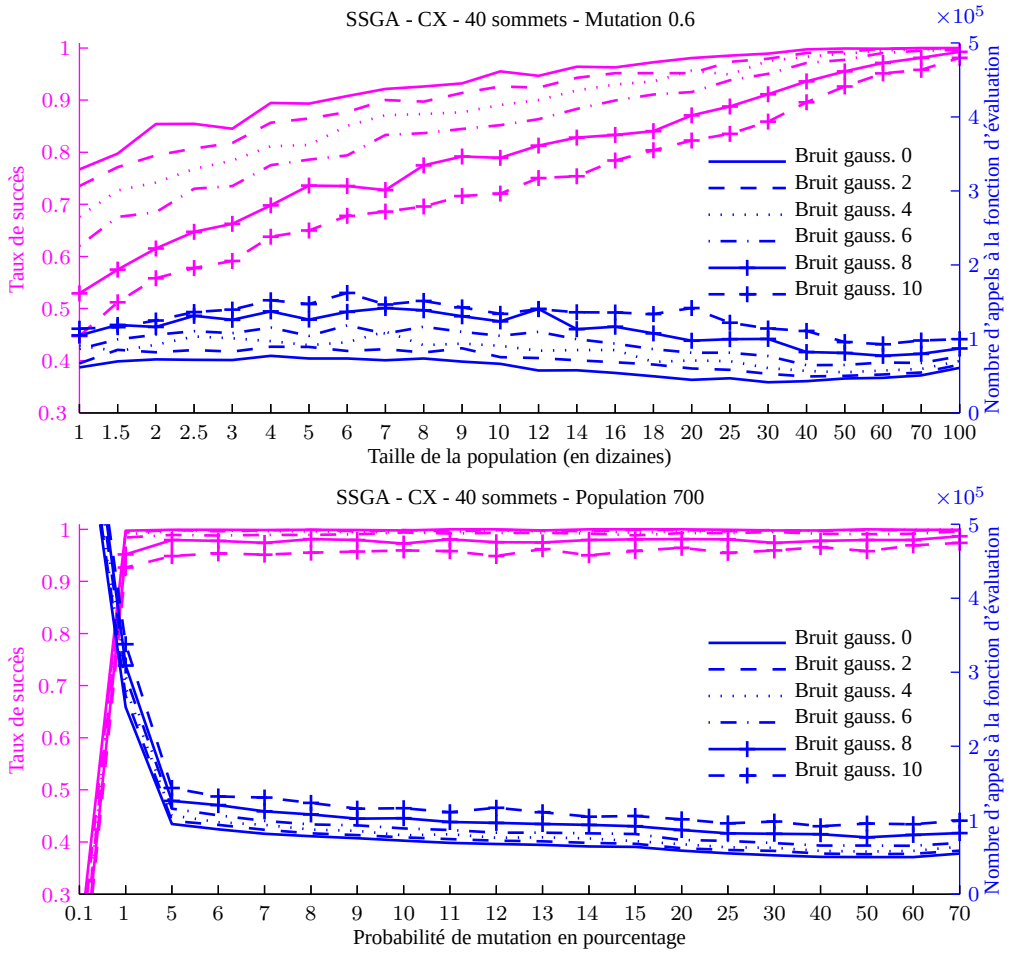

FIGURE A.28 – Sensibilité au bruit gaussien des paramètres du CX dans un moteur *steady-state*.

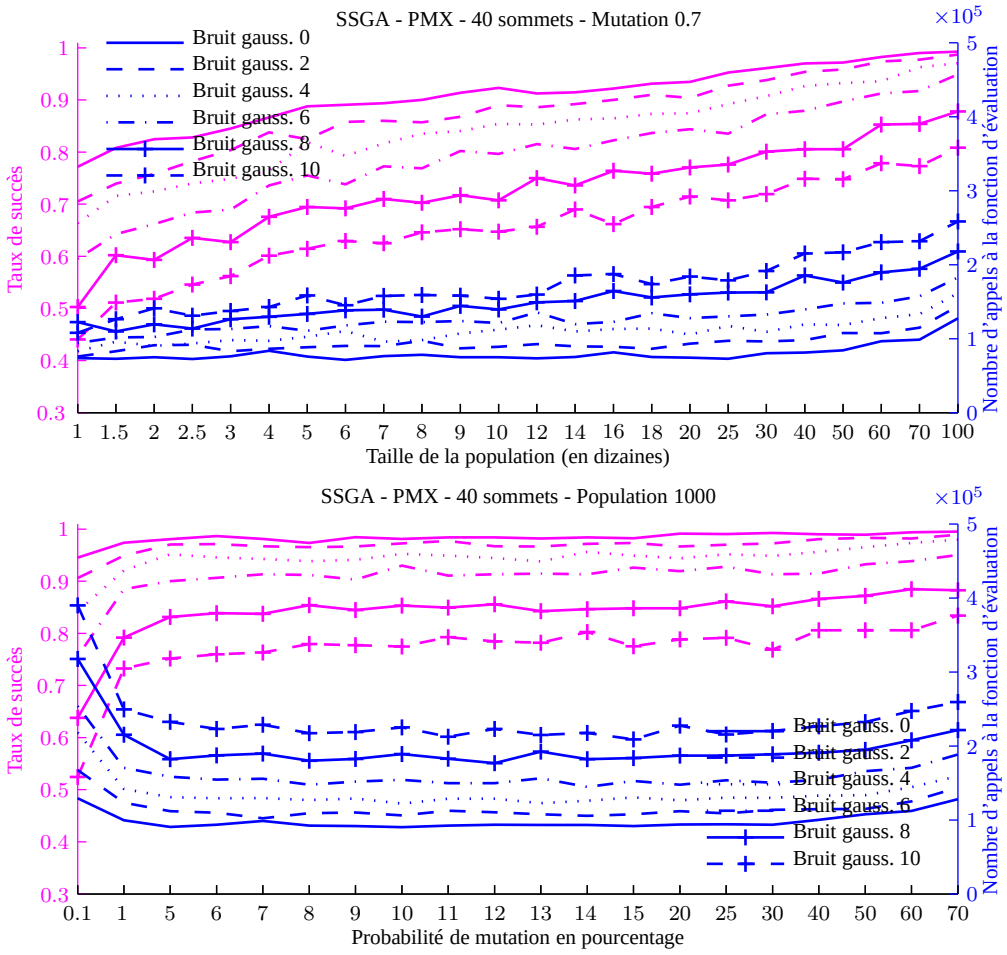

FIGURE A.29 – Sensibilité au bruit gaussien des paramètres du PMX dans un moteur *steady-state*.

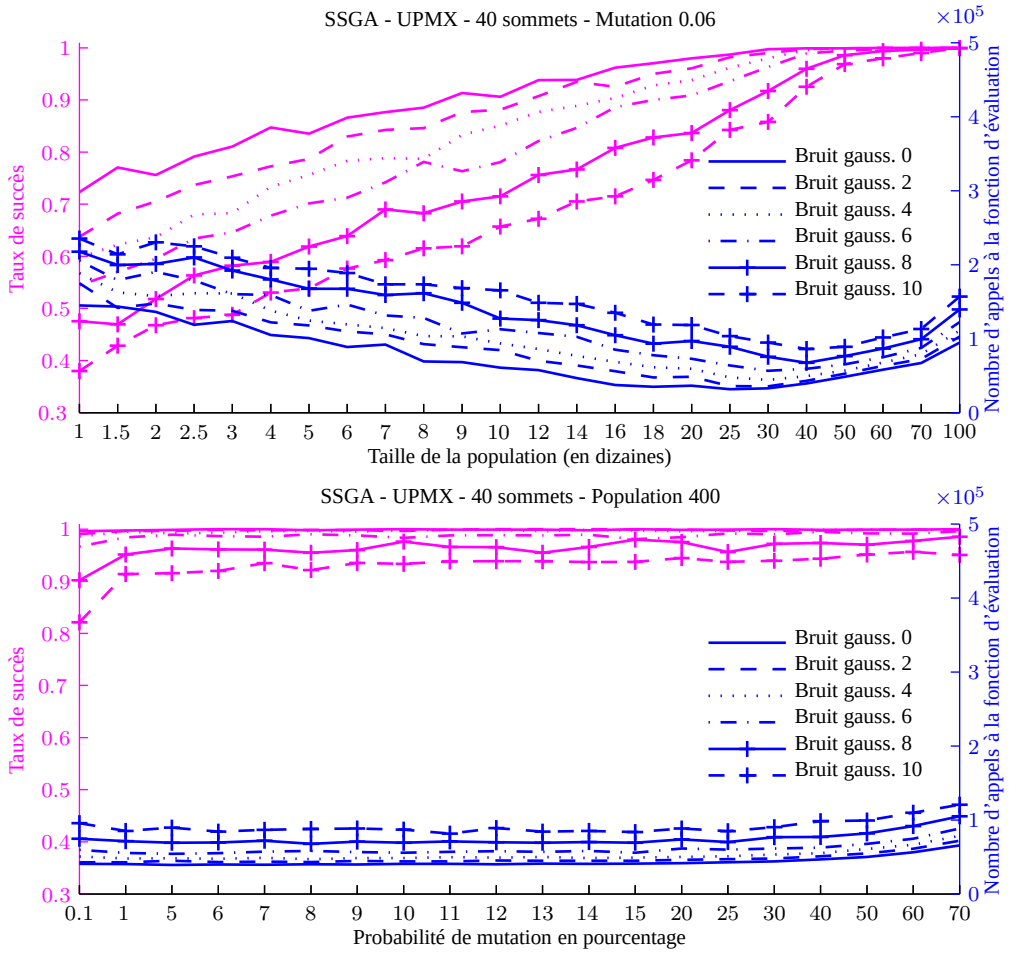

FIGURE A.30 – Sensibilité au bruit gaussien des paramètres du UPMX dans un moteur *steady-state*.

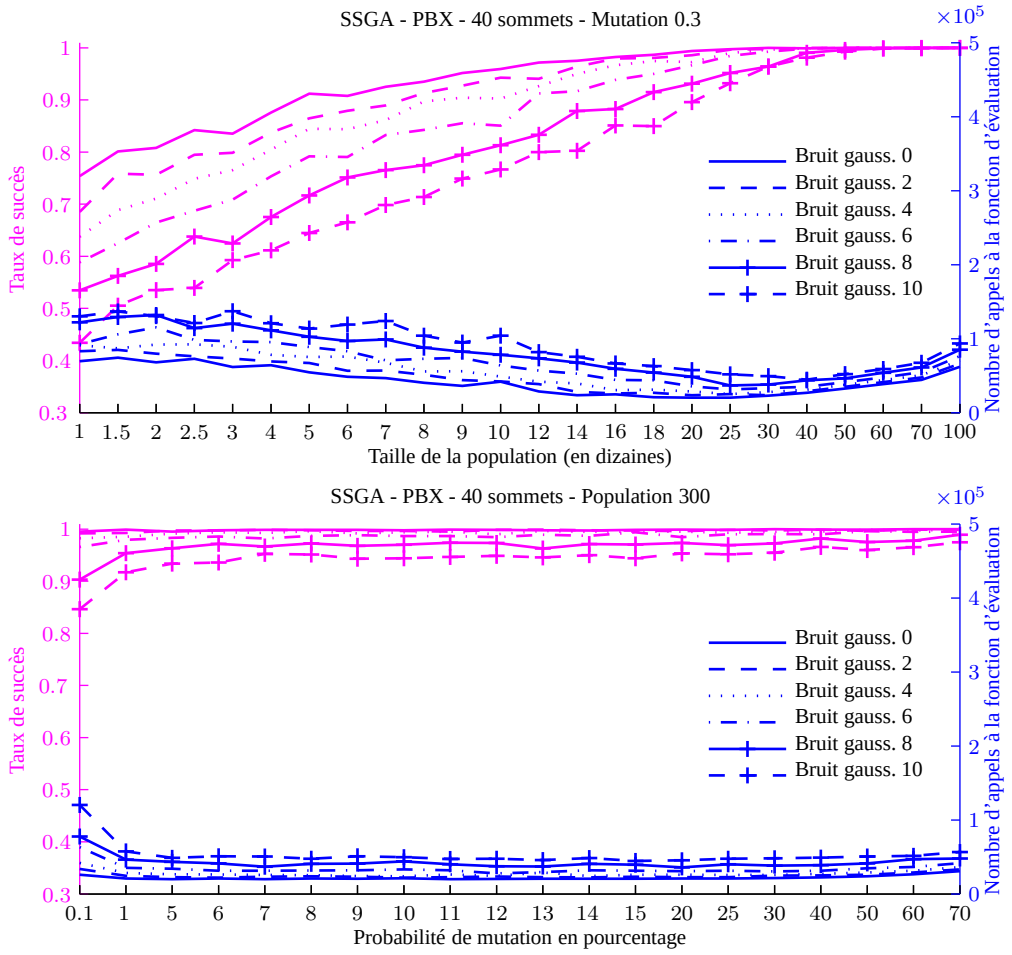

FIGURE A.31 – Sensibilité au bruit gaussien des paramètres du PBX dans un moteur *steady-state*.

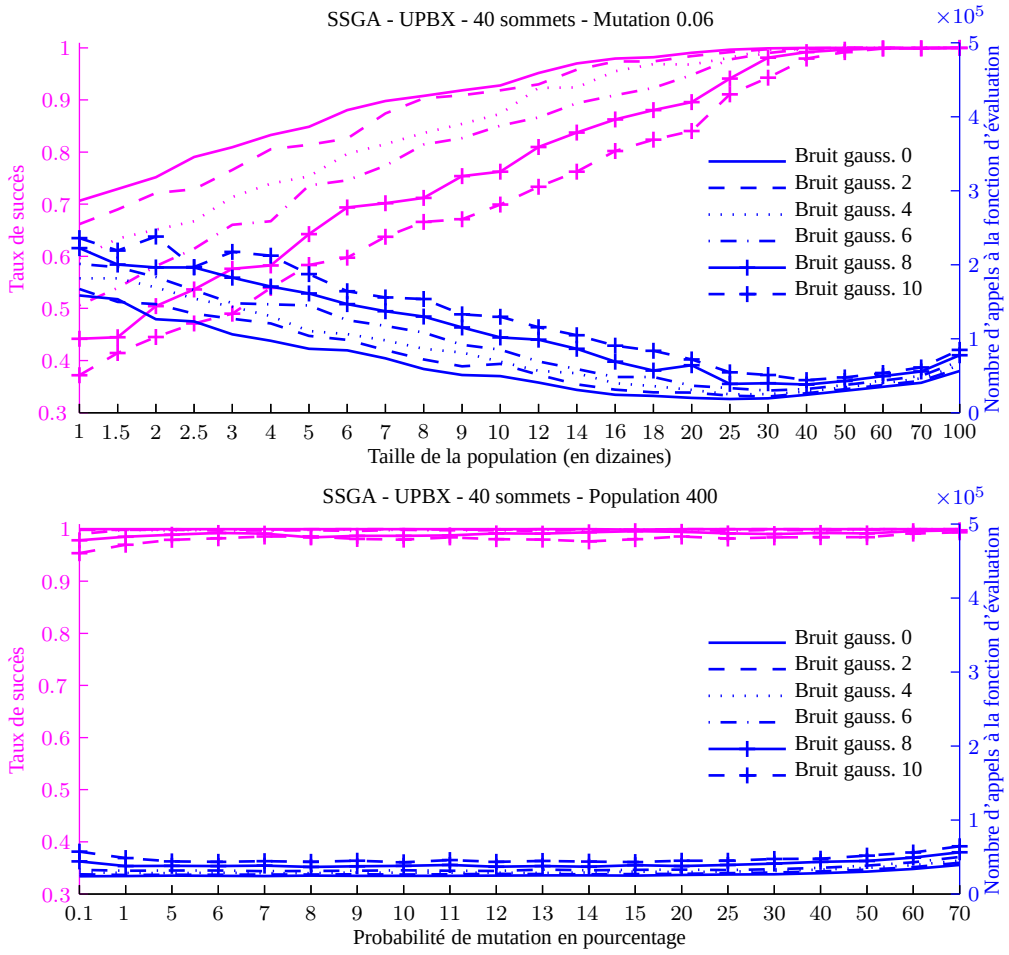

FIGURE A.32 – Sensibilité au bruit gaussien des paramètres du UPBX dans un moteur *steady-state*.

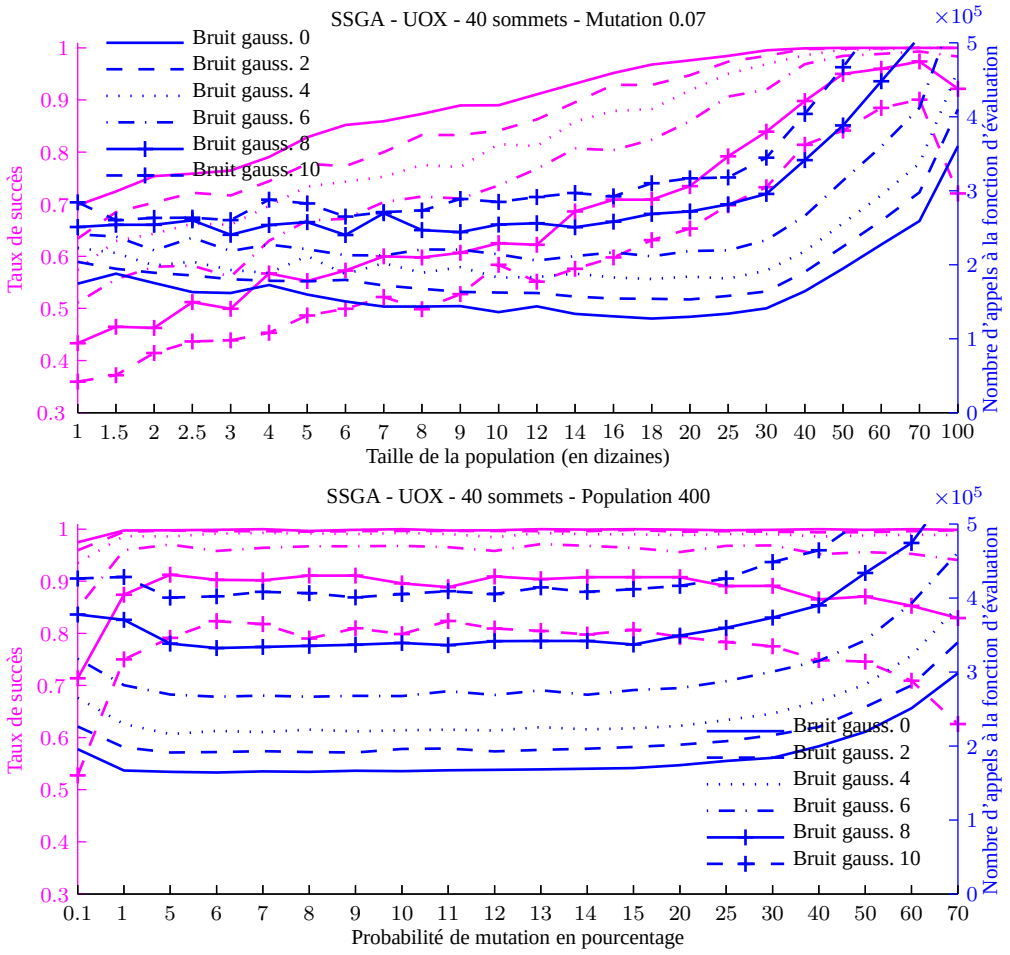

FIGURE A.33 – Sensibilité au bruit gaussien des paramètres du UOX dans un moteur *steady-state*.

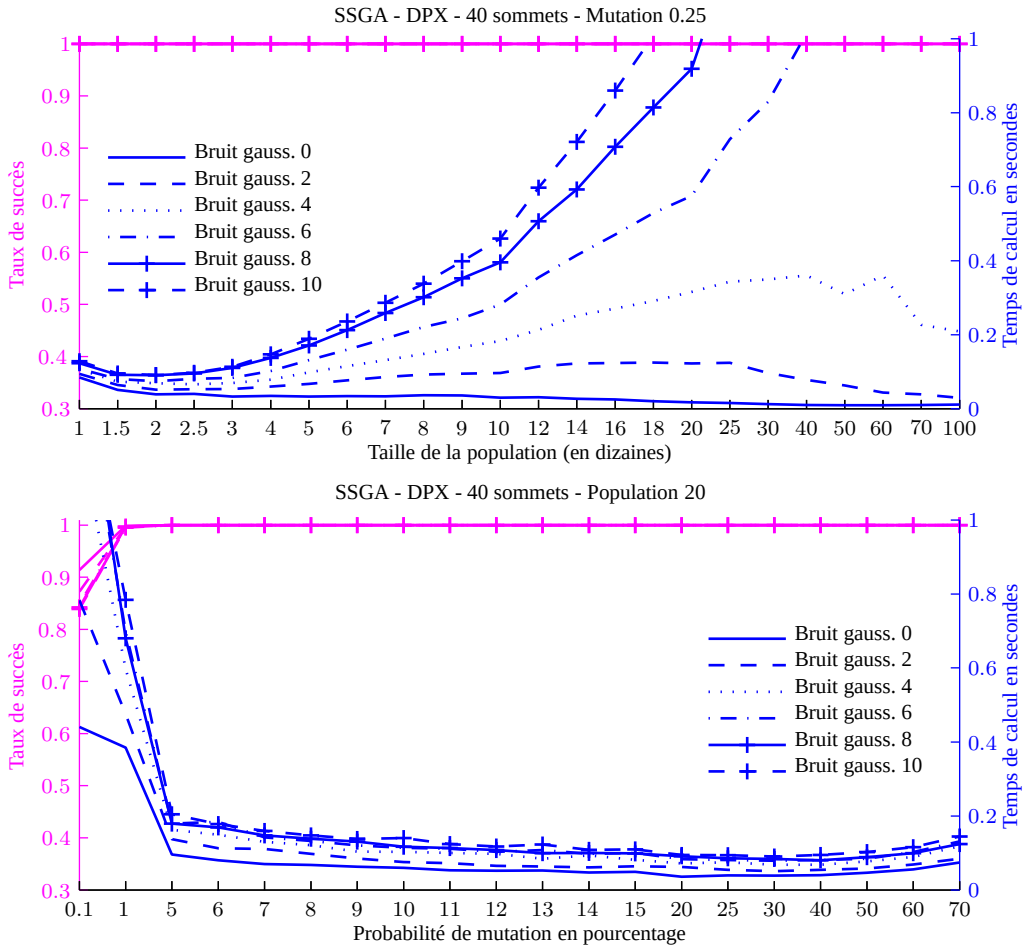

FIGURE A.34 – Sensibilité au bruit gaussien des paramètres du DPX dans un moteur *steady-state*.

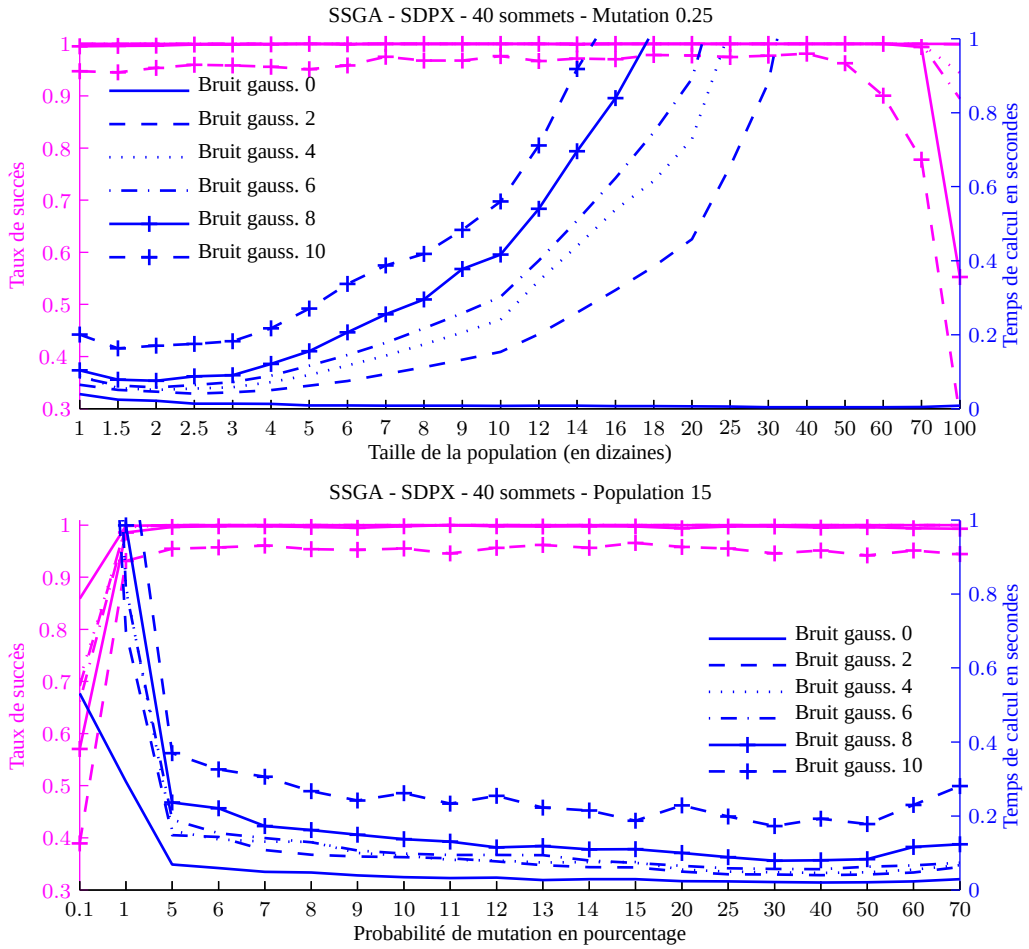

FIGURE A.35 – Sensibilité au bruit gaussien des paramètres du SDPX dans un moteur *steady-state*.

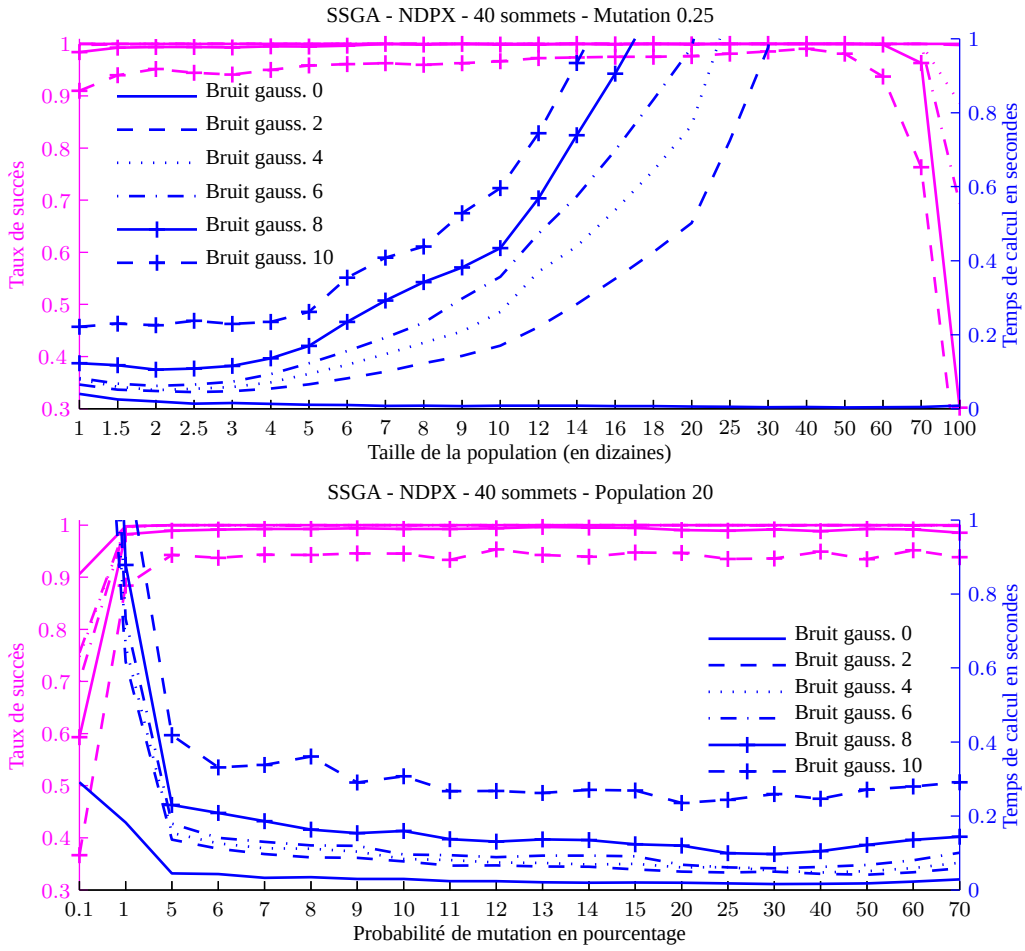

FIGURE A.36 – Sensibilité au bruit gaussien des paramètres du NDPX dans un moteur *steady-state*.

Annexe B

Comparaison d’opérateurs

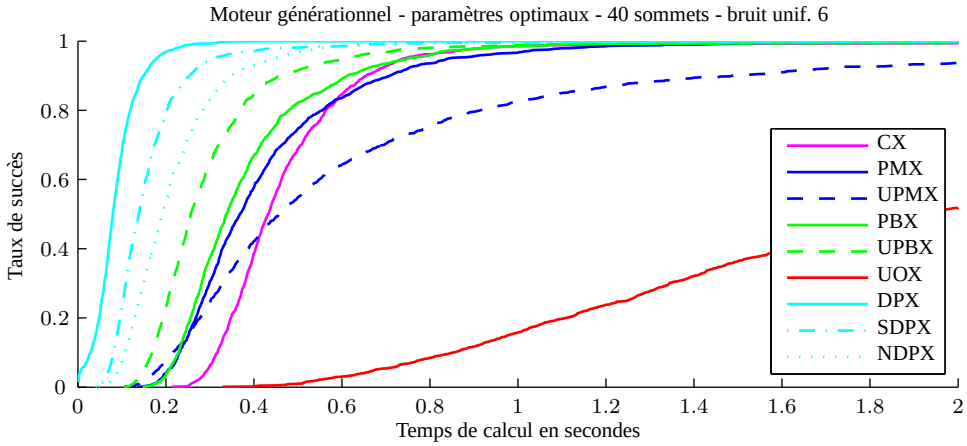

FIGURE B.1 – Comparaison des opérateurs avec un moteur générationnel en fonction du temps de calcul.

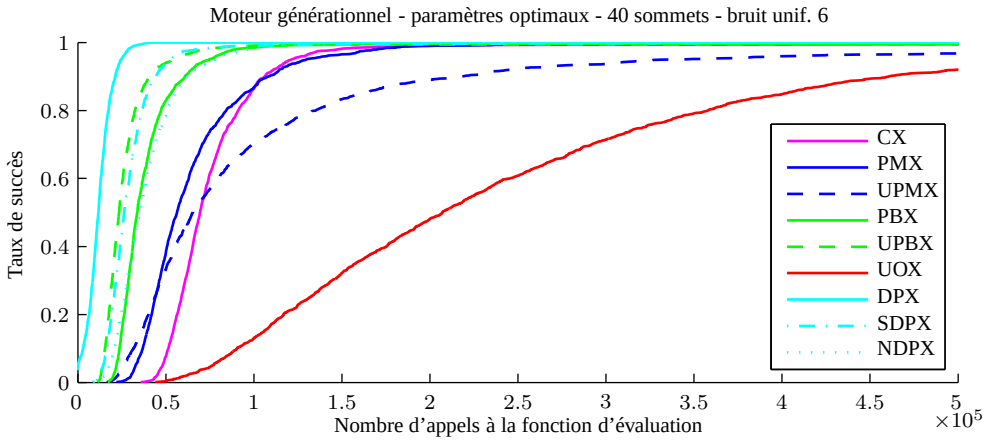

FIGURE B.2 – Comparaison des opérateurs avec un moteur générationnel en fonction du nombre d'appels à la fonction d'évaluation.

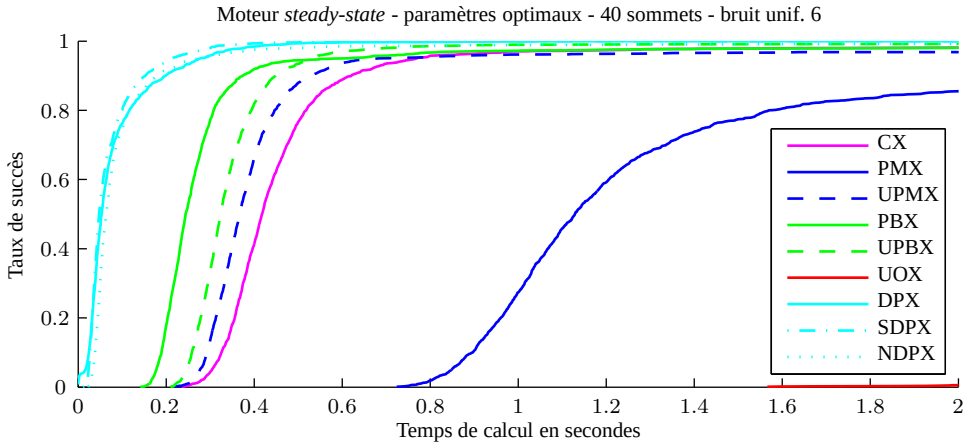

FIGURE B.3 – Comparaison des opérateurs avec un moteur *steady-state* en fonction du temps de calcul.

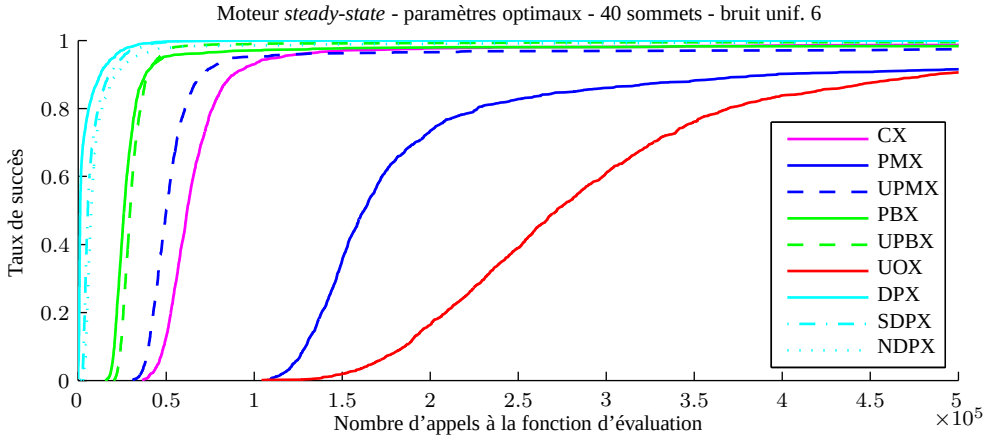

FIGURE B.4 – Comparaison des opérateurs avec un moteur *steady-state* en fonction du nombre d'appels à la fonction d'évaluation.

Annexe C

Mutation par brassage

C.1 Moteur générationnel

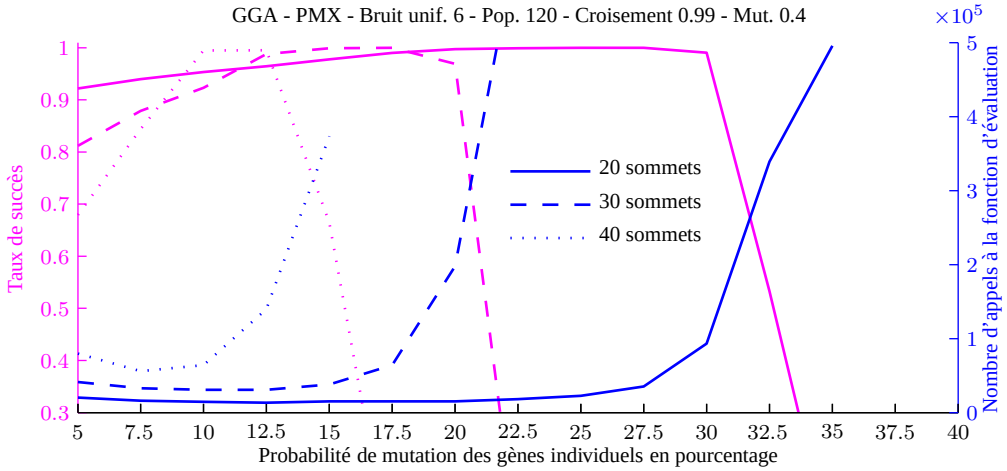

FIGURE C.1 – Performance du PMX en fonction du paramètre de la mutation par brassage.

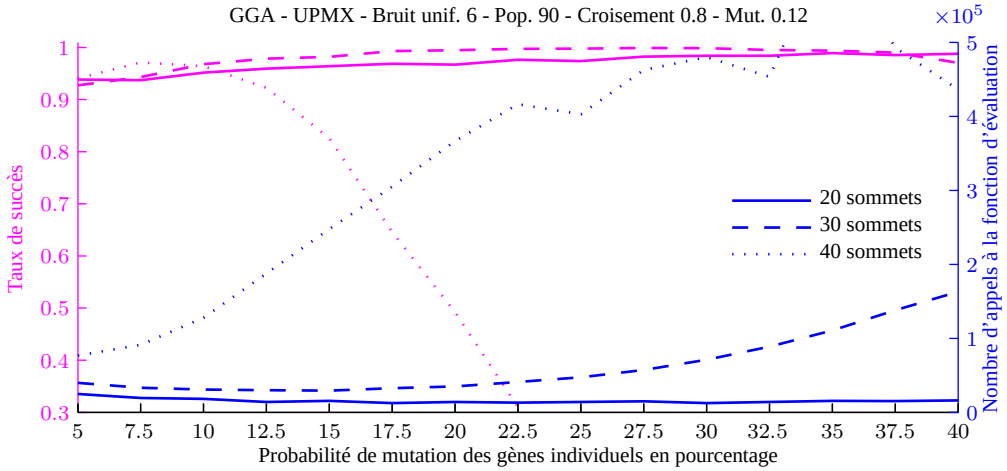

FIGURE C.2 – Performance du UPMX en fonction du paramètre de la mutation par brassage.

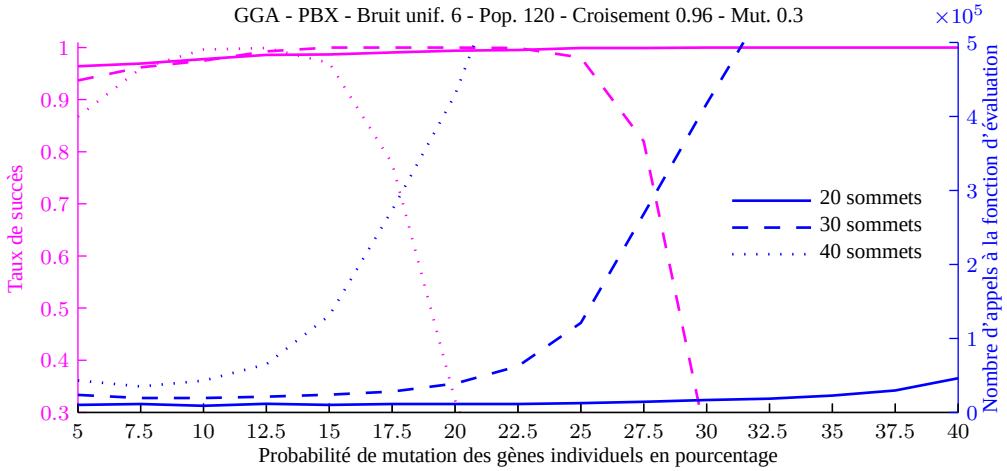

FIGURE C.3 – Performance du PBX en fonction du paramètre de la mutation par brassage.

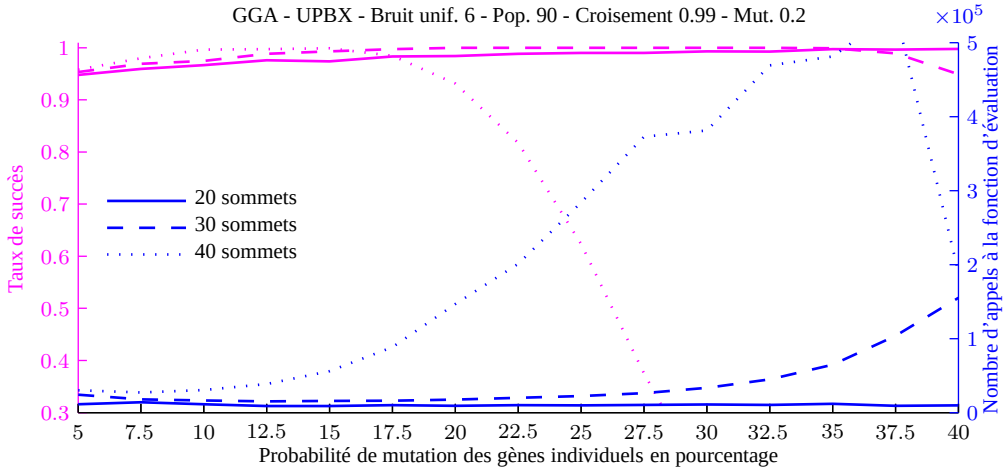

FIGURE C.4 – Performance du UPBX en fonction du paramètre de la mutation par brassage.

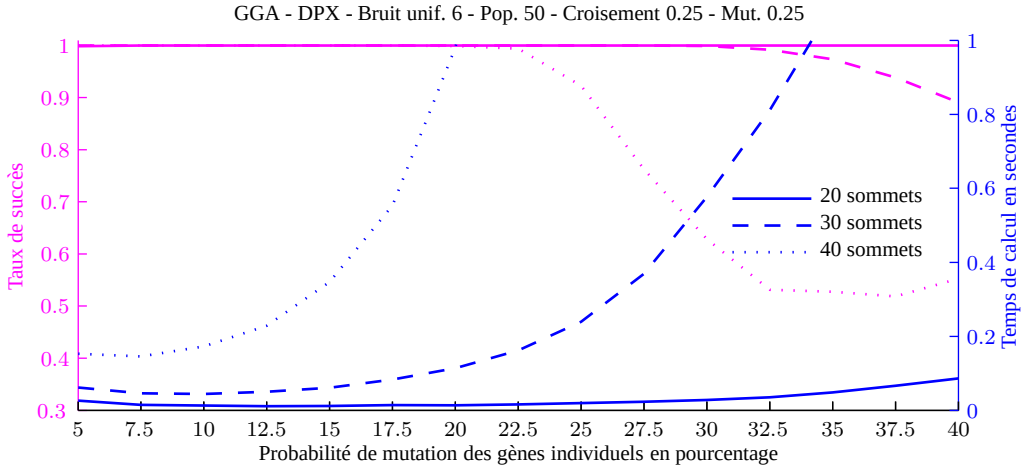

FIGURE C.5 – Performance du DPX en fonction du paramètre de la mutation par brassage.

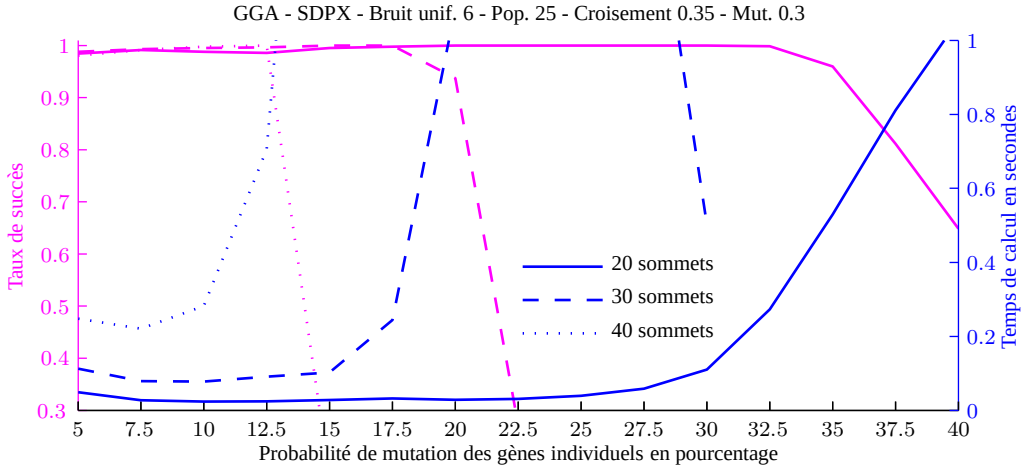

FIGURE C.6 – Performance du SDPX en fonction du paramètre de la mutation par brassage.

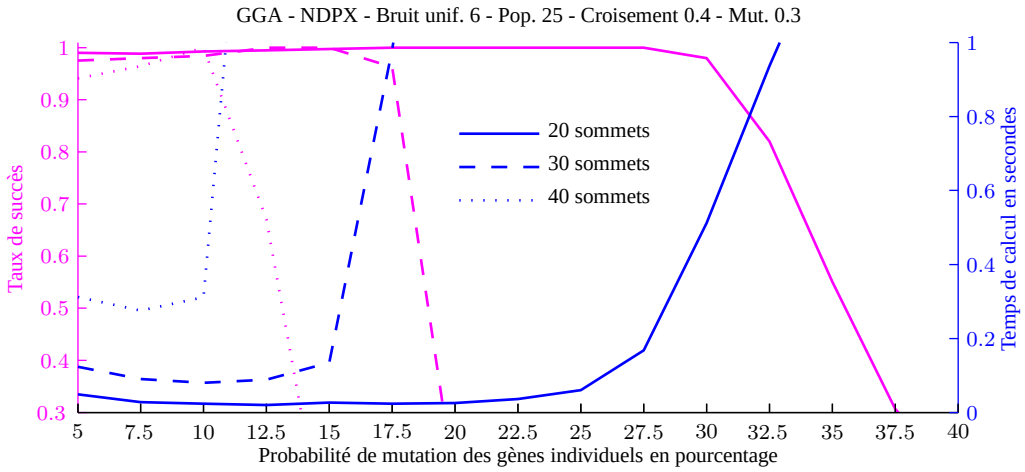

FIGURE C.7 – Performance du NDPX en fonction du paramètre de la mutation par brassage.

C.2 Moteur *steady-state*

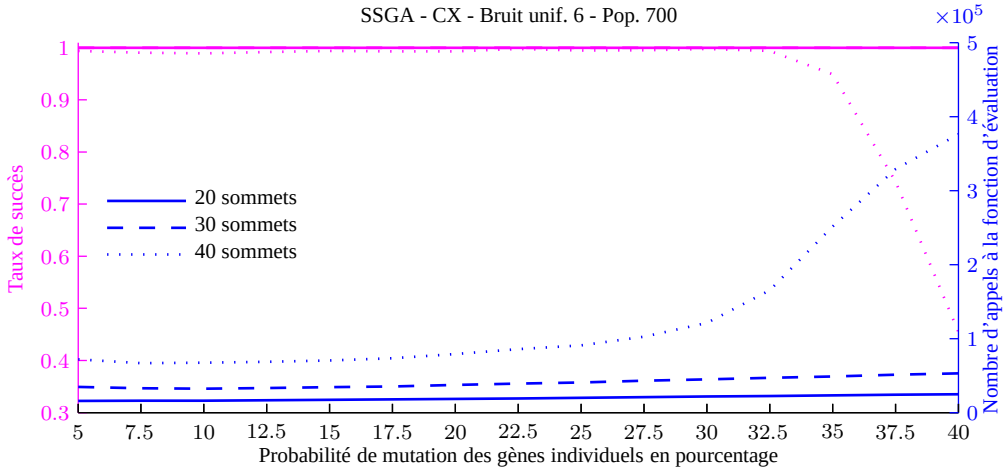

FIGURE C.8 – Performance du CX en fonction du paramètre de la mutation scramble.

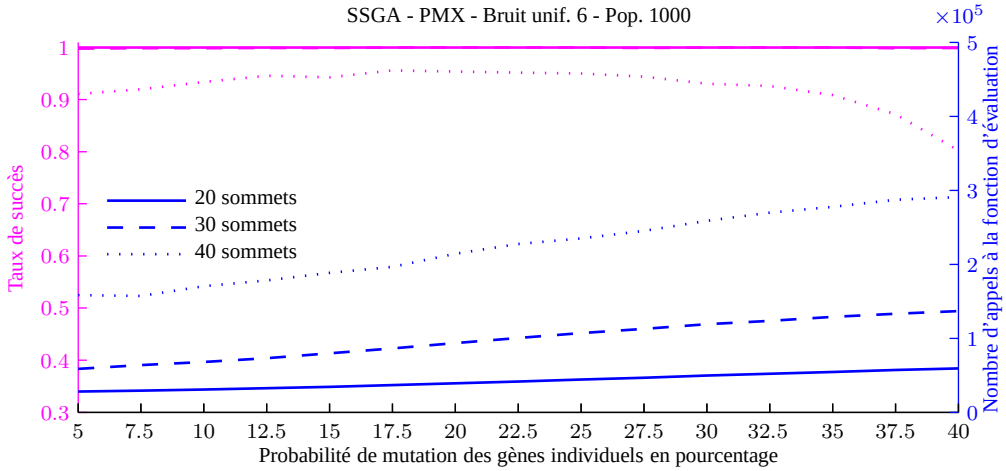

FIGURE C.9 – Performance du PMX en fonction du paramètre de la mutation scramble.

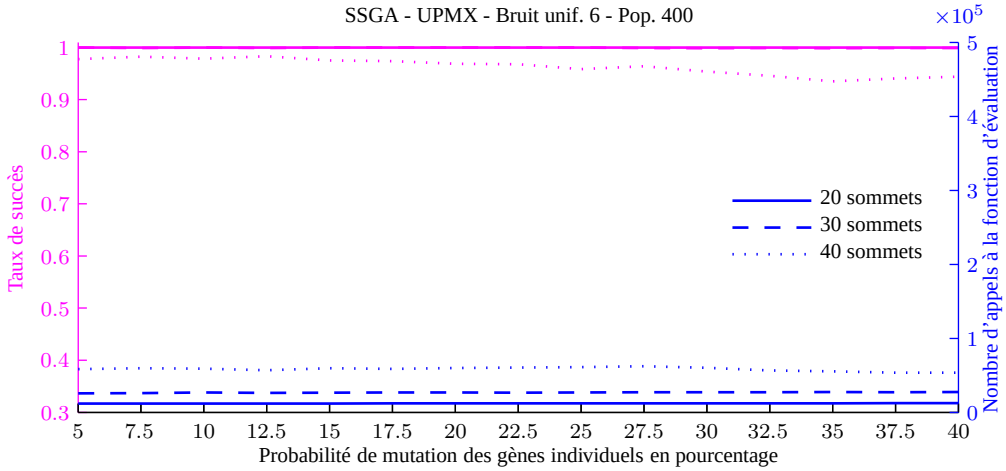

FIGURE C.10 – Performance du UPMX en fonction du paramètre de la mutation scramble.

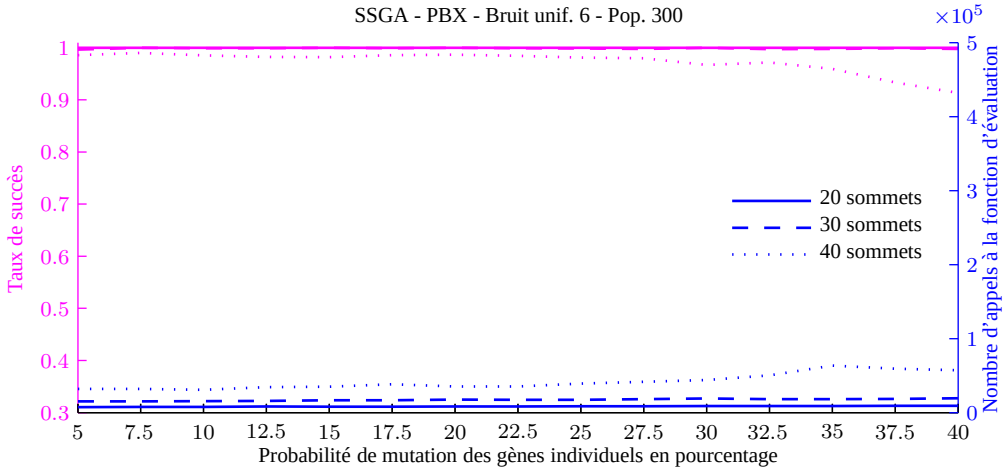

FIGURE C.11 – Performance du PBX en fonction du paramètre de la mutation scramble.

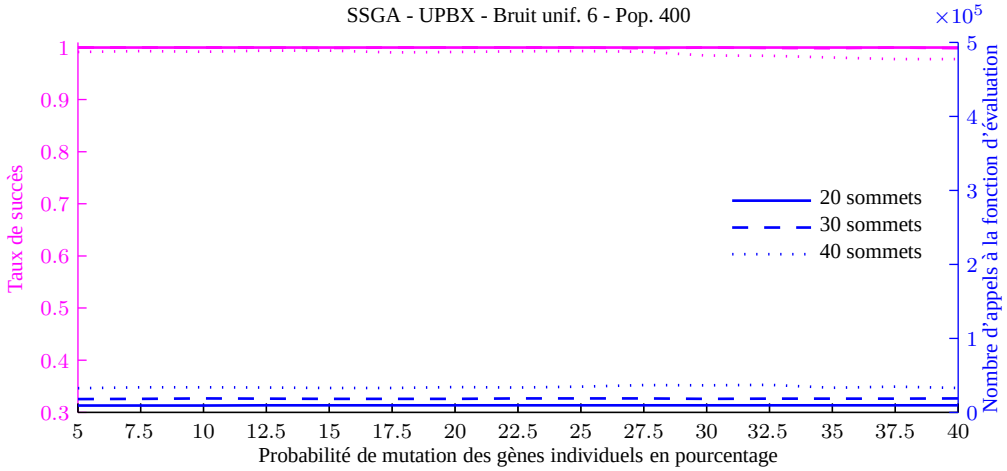

FIGURE C.12 – Performance du UPBX en fonction du paramètre de la mutation scramble.

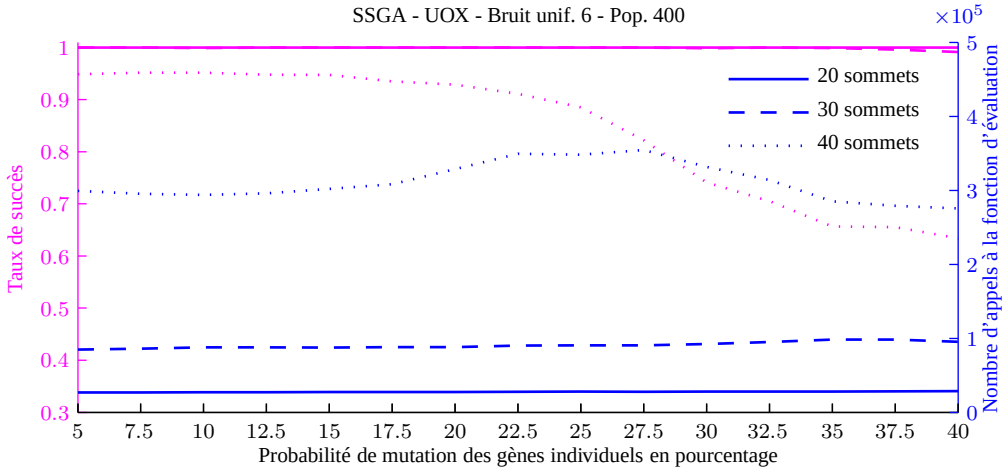

FIGURE C.13 – Performance du UOX en fonction du paramètre de la mutation scramble.

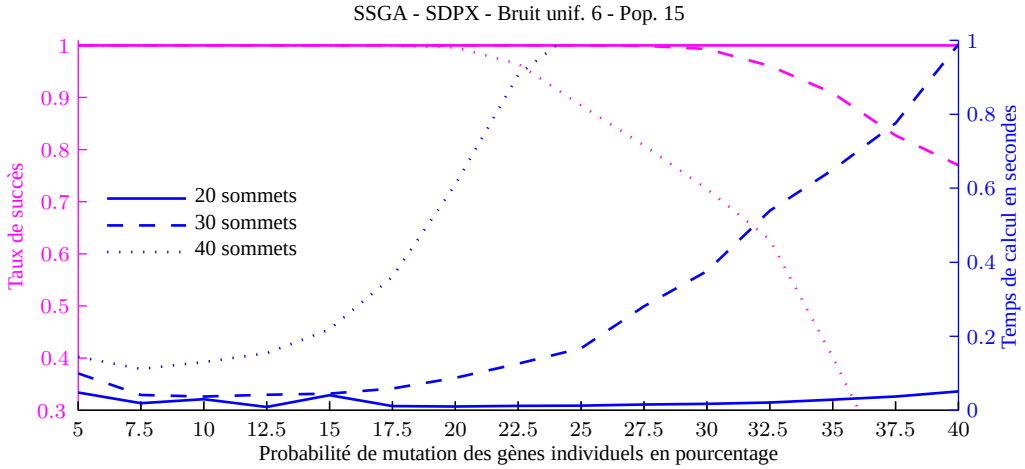

FIGURE C.14 – Performance du SDPX en fonction du paramètre de la mutation scramble.

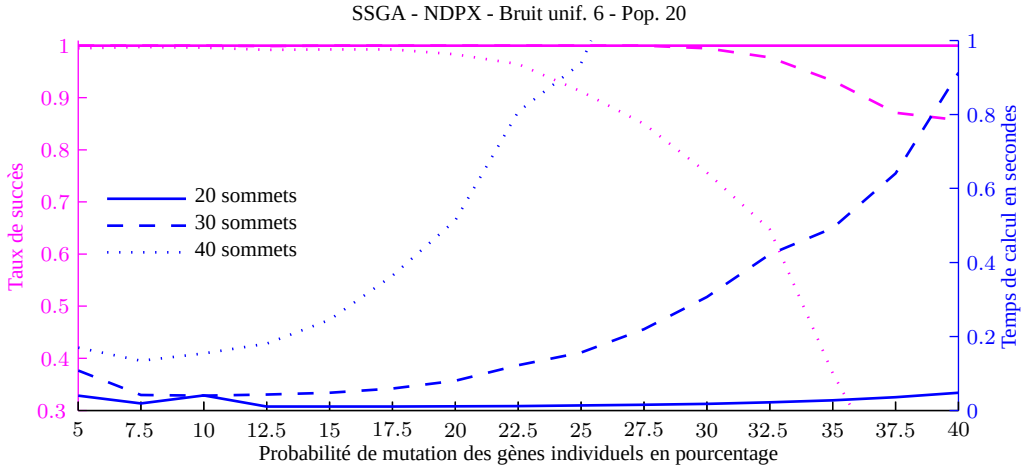

FIGURE C.15 – Performance du NDPX en fonction du paramètre de la mutation scramble.

Annexe D

Analyse de la variance (Kruskal-Wallis)

L'analyse de la variance permet d'identifier si les résultats observés d'un processus stochastique sont significatifs. Etant donné deux ou plusieurs échantillons, l'hypothèse H_0 est qu'ils sont identiques. On calcule la valeur p qui représente la probabilité que l'on obtienne un résultat au moins aussi extrême que la valeur observée sous l'hypothèse H_0. Si cette probabilité est faible, alors le résultat est statistiquement significatif. Il s'agit donc très probablement de différences réelles entre les échantillons et non des effets du hasard.

Le test de Kruskal-Wallis est une méthode classique sans paramètre et basée sur l'ordonnancement, qui permet de comparer plus de deux échantillons en même temps. De plus, il ne présuppose pas une distribution gaussienne des données. En général, on choisit 1% ou 5% comme seuil de signifiance. Ce choix est arbitraire mais largement répandu.

Nous effectuons le test de Kruskal-Wallis pour déterminer si les différences observées sont réelles. En général, nos échantillons sont décrits par deux valeurs et non par une seule. Il s'agit d'une part, du taux de succès et d'autre part du temps de calcul jusqu'à l'obtention de la solution. Nous utilisons alors deux p-valeurs, p_{SR} pour le taux de réussite (*success rate*) et p_{AES} (*average evaluations to success*) pour le nombre d'appels à la fonction d'évaluation nécessaires pour atteindre la solution, respectivement p_{ARS} (*average runtime to success*) si l'on mesure le temps de calcul (notamment pour les opérateurs qui incluent une heuristique gloutonne). Si la valeur de p n'est pas définie (ceci se produit notamment dans le cas où il n'y a pas de variance dans les échantillons), nous le signalons par un tiret dans les tableaux. Par exemple si l'algorithme ne converge jamais ou s'il converge toujours vers la bonne solution. Nous considérons les résultats comme significatifs, si l'une des deux valeurs p est inférieure au seuil. Si, par exemple, les différences du taux de succès sont significatives, alors les résultats sont évidemment différents dans le sens de la précision, même si la distribution du temps de calcul nécessaire ne montre pas de changement significatif. A l'inverse, s'il n'y a pas de changement significatif de la précision mais un changement significatif du temps de calcul, alors le résultat est aussi différent, au sens où les deux algorithmes à comparer ont la même

précision mais non la même complexité.

Nous utilisons des seuils de 1% et 5%. Les expériences qui dépassent ces seuils (et qui ne sont donc pas forcément statistiquement significatives) sont notées en italique ou en gras respectivement.

Il est important de noter que, dans de nombreux cas, nous évaluons le changement des paramètres en continu, par exemple le taux de croisement. Nous effectuons les tests entre deux valeurs consécutives de ce paramètre. Ce procédé est motivé par le fait que nous ne cherchons pas des optima globaux (qui sont d'ailleurs pratiquement impossibles à obtenir), mais un optimum local d'assez bonne qualité. Les intervalles sont choisis arbitrairement en prenant en compte la théorie existante. En général, la différence entre les résultats augmente avec la largeur de l'intervalle. Une insignifiance peut donc venir d'un intervalle très court. Cependant, il est possible qu'il y ait une tendance dans une série de valeurs. Les p-valeurs ne sont pas transitives. On ne peut donc pas conclure sur des séries. En général, la plupart des différences non significatives apparaissent pour des valeurs qui sont éloignées des paramètres optimaux. Dans le cas contraire, comme nous cherchons un bon paramétrage en sachant qu'il ne sera jamais optimal, nous choisissons la valeur qui donne le meilleur résultat en sachant qu'il est probablement possible d'ajuster le paramètre un peu sans trop changer la qualité des résultats. Une conclusion valide de la non-signifiance entre deux bornes d'un intervalle est qu'il n'est probablement pas nécessaire d'étudier des intervalles plus courts entre ces deux valeurs, si l'on suppose que la probabilité d'un pic à l'intérieur est assez faible.

Par la suite nous détaillons les résultats non significatifs observés, la totalité des études se trouvant dans les tableaux D.1 à D.19.

D.1 Taille de population

D.1.1 Algorithme générationnel

L'optimum observé pour CX est une population de 400 individus. Les différences entre 300 et 400 ($p_{SR} = 0,025$ et $p_{AES} = 0,026$) ne sont pas significatives avec le seuil à 1%, mais le sont pour le seuil à 5%. En ce qui concerne l'opérateur UOX, il n'y a pas de différence significative entre une population de 9 et 10 ($p_{SR} = 0,32$ et $p_{AES} = 0,77$). Ici, nous avons fait une expérience avec les intervalles les plus petits possibles. Pour les trois opérateurs qui incluent des heuristiques, les différences sont non significatives entre l'optimum et seulement une de ses deux valeurs voisines, pour DPX entre 40 et 50 individus (p_{SR} non défini, $p_{ARS} = 0,86$), pour SDPX entre 25 et 30 (p_{SR} non défini, $p_{ARS} = 0,12$) et pour NDPX entre 25 et 30 (p_{SR} non défini, $p_{ARS} = 0,24$).

D.1.2 Algorithme *steady-state*

Seul SDPX montre des différences non significatives entre son optimum et une valeur voisine : entre 10 et 15 ($p_{SR} = 0,083$, $p_{ARS} = 0,5$).

D.2 Probabilité de croisement

La probabilité de croisement s'applique uniquement dans le cadre des algorithmes générationnels. On peut noter que la granularité des observations entre 95% et 99% est très fine. Par conséquent, les différences ne sont pas nécessairement significatives. De plus, pour de nombreux opérateurs, nous avons trouvé l'optimum à 99% et donc au maximum. Notamment, pour PMX les différences entre 99 et 98 ne sont pas significatives ($p_{SR} = 0,65$ et $p_{AES} = 0,97$), tandis qu'entre 97 et 98 ($p_{SR} = 0,016$ et $p_{AES} = 0,013$) elles le sont. Pour UPBX toute la série entre 95% et 99% ne montre pas de différence significative si l'on utilise le seuil à 1%. Quant au seuil à 5%, la différence entre 98% et 99% est significative ($p_{SR} = 0,0672$ et $p_{AES} = 0,043$).

PBX présente un optimum à 96% mais les différences entre 95% et 96% ($p_{SR} = 0,46$ et $p_{AES} = 0,53$) et celles entre 96% et 97% ($p_{SR} = 0,059$ et $p_{AES} = 0,1$) ne sont pas significatives. Quand on regarde les valeurs voisines, elles sont significativement différentes : de 97% à 98%, $p_{SR} = 0,48$ et $p_{AES} = 0,0014$; de 90% à 95%, $p_{SR} = 0,023$ et $p_{AES} = 1,1E-15$. Pour UPMX nous effectuons une étude autour de 80% car nous observons un pic autour de cette valeur avec la granularité assez grossière initiale. Les valeurs sont significativement différentes à l'exception du passage de 80% à 81% dont les p-valeurs sont : $p_{SR} = 1$ et $p_{AES} = 0,12$.

Les croisements qui incluent une stratégie gloutonne ont des caractéristiques particulières : quand on augmente le taux de croisement, il faut en même temps diminuer ou augmenter la taille de la population selon l'opérateur. Pour DPX, on n'observe aucune différence significative pour les probabilités entre 20% et 45%

D.3 Probabilité de mutation

D.3.1 Algorithme générationnel

Si l'on utilise le seuil strict (1%), les différences entre 11% et 12% ne sont pas significatives pour l'opérateur UPMX ($p_{SR} = 0,056$ et $p_{AES} = 0,034$). De même pour UOX, la différence entre 14% et 15% n'est pas significative ($p_{SR} = 0,71$ et $p_{AES} = 0,013$). Quand on passe au seuil de 5%, les différences sont significatives dans les deux cas. En ce qui concerne DPX, les différences ne sont pas significatives entre 20% et 25% ($p_{SR} = 0,32$ et $p_{ARS} = 0,069$) indépendamment du seuil.

D.3.2 Algorithme *steady-state*

Pour les algorithmes *steady-state* la probabilité de mutation n'est pas aussi importante que pour le moteur générationnel. Par conséquent, des résultats non significatifs sont nombreux. Les p-valeurs qui dépassent 1% sont en particulier :

UPMX : entre 5% et 6% ($p_{SR} = 0,046$ et $p_{AES} = 0,81$) ; entre 6% et 7% ($p_{SR} = 0,86$ et $p_{AES} = 0,031$).

UPBX : entre 5% et 6% ($p_{SR} = 0,78$ et $p_{AES} = 0,6$) ; entre 6% et 7% ($p_{SR} = 1$ et $p_{AES} = 0,2$).

UOX : entre 6% et 7% ($p_{SR} = 0,27$ et $p_{AES} = 1$).

DPX : entre 20% et 25% (p_{SR} non définie et $p_{ARS} = 0,14$) ; entre 25% et 30% (p_{SR} non définie et $p_{ARS} = 0,16$).

SDPX : entre 25% et 30% ($p_{SR} = 0,32$ et $p_{ARS} = 0,83$).

NDPX : entre 20% et 25% ($p_{SR} = 0,83$ et $p_{ARS} = 0,62$) ; entre 25% et 30% ($p_{SR} = 0,83$ et $p_{ARS} = 0,092$)

D.4 Autres tests

Les tests qui concernent la comparaison des différents opérateurs (croisement ou mutation), des différents moteurs d'évolution (GGA-SSGA) sont tous significatifs, à la différence des tests d'optimisation de paramètres pour un opérateur.

D.5 Tableaux associés

	CX		PMX		UPMX		PBX		UPBX	
s_p	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}
10-15	-	-	0	1	6,3E-01	2,1E-04	0	0	0	0
15-20	-	-	-	0	**9,9E-02**	**6,7E-01**	0	0	7,3E-01	4,7E-05
20-25	-	-	0	1,0E-09	**6,5E-02**	**6,1E-01**	7,5E-03	6,4E-06	**9,4E-01**	**4,6E-01**
25-30	-	-	0	4,1E-01	9,1E-01	2,6E-03	2,3E-01	2,3E-03	3,1E-03	1,0E-03
30-40	-	-	0	7,8E-16	2,6E-03	1,6E-01	1,7E-11	5,1E-08	6,9E-13	2,0E-13
40-50	-	-	3,5E-01	9,1E-15	7,5E-11	5,3E-04	1,6E-12	3,2E-11	0	0
50-60	4,6E-03	1	1,0E-07	2,2E-06	3,9E-13	3,7E-11	4,7E-10	2,2E-05	5,8E-15	2,4E-11
60-70	0	9,2E-01	2,9E-12	8,3E-01	0	7,2E-04	9,2E-11	5,7E-01	0	4,8E-03
70-80	0	7,2E-01	2,2E-16	1,1E-04	0	5,3E-12	1,6E-09	7,3E-02	5,3E-10	4,1E-03
80-90	0	7,2E-13	1,1E-10	1,1E-10	5,6E-15	0	8,1E-06	1,9E-08	7,1E-04	9,8E-13
90-100	0	0	1,9E-04	1,6E-04	5,4E-04	0	1,6E-03	1,2E-12	6,5E-01	2,2E-16
100-120	1,9E-02	0	1,0E-03	0	0	0	4,2E-03	0	8,9E-04	0
120-140	3,2E-01	0	3,2E-01	0	0	6,0E-13	1,3E-01	0	-	0
140-160	1,5E-03	2,1E-10	3,2E-01	0	4,4E-11	3,6E-08	2,6E-01	0	-	1,1E-16
160-180	8,3E-01	1,9E-04	3,2E-01	3,0E-14	2,7E-08	1,1E-04	1,6E-01	0	3,2E-01	0
180-200	3,3E-01	5,7E-05	3,4E-02	6,3E-13	8,2E-07	3,1E-02	1,6E-01	0	3,8E-03	2,8E-15
200-250	5,0E-01	4,1E-12	8,8E-04	0	0	1,1E-01	6,5E-01	0	1,7E-05	0
250-300	4,0E-01	4,2E-07	5,3E-03	0	3,0E-07	2,5E-01	4,8E-01	0	2,5E-01	0
300-400	*2,5E-02*	*2,6E-02*	3,6E-09	0	1,5E-04	5,0E-05	8,0E-06	0	1,3E-08	0
400-500	1,0E-01	3,1E-05	2,9E-05	0	2,4E-02	2,2E-05	1,1E-01	0	1,5E-04	0
500-600	3,2E-01	1,3E-14	6,6E-03	0	*3,5E-02*	*3,5E-02*	2,3E-02	0	2,8E-02	0
600-700	-	0	7,5E-03	2,0E-15	5,7E-02	3,4E-03	6,4E-02	0	7,8E-02	0
700-1000	-	0	1,2E-06	0	1,6E-08	3,3E-01	5,0E-07	0	8,2E-06	0

Tableau D.1 – p-valeurs entre les résultats de deux expériences consécutives de la taille de population (GGA - opérateurs sans heuristique sauf UOX).

	CX		PMX		UPMX		PBX		UPBX		UOX	
s_p	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}
10-15	2,1E-03	1,5E-01	1,8E-03	2,3E-01	9,5E-03	7,0E-03	*2,9E-02*	*7,7E-01*	5,7E-02	5,6E-03	1,9E-04	4,0E-01
15-20	*1,2E-02*	*3,8E-01*	*4,8E-02*	*1,2E-01*	3,2E-03	1,1E-01	2,1E-03	4,6E-01	*2,3E-02*	*3,3E-02*	**4,2E-01**	**1,1E-01**
20-25	**1,8E-01**	**8,0E-01**	**7,3E-01**	**9,3E-01**	*6,6E-01*	*2,4E-02*	**9,1E-01**	**1,4E-01**	*1,4E-02*	*5,2E-01*	**2,5E-01**	**2,2E-01**
25-30	**2,2E-01**	**8,3E-02**	**7,3E-01**	**1,3E-01**	**8,4E-02**	**7,6E-01**	**4,1E-01**	**6,5E-01**	**8,5E-01**	**6,6E-02**	1,5E-03	9,2E-01
30-40	**3,4E-01**	**7,9E-01**	3,8E-03	5,7E-01	1,4E-02	9,8E-03	*5,7E-02*	*3,9E-02*	*1,2E-02*	*3,1E-02*	**7,1E-01**	**7,2E-01**
40-50	**1,7E-01**	**4,0E-01**	**2,9E-01**	**6,9E-02**	*9,7E-01*	*2,9E-02*	4,0E-04	5,2E-02	1,8E-02	5,6E-03	**6,8E-01**	**2,5E-01**
50-60	*2,2E-02*	*9,3E-01*	5,5E-04	6,4E-01	**8,2E-01**	**2,6E-01**	**8,5E-02**	**6,4E-01**	1,3E-01	1,2E-03	**6,4E-02**	**3,7E-01**
60-70	**8,2E-01**	**9,8E-01**	**7,7E-01**	**2,0E-01**	6,4E-02	7,5E-03	**8,7E-02**	**4,8E-01**	3,4E-01	8,2E-04	**7,3E-01**	**6,3E-01**
70-80	**6,1E-01**	**6,6E-02**	**2,8E-01**	**2,1E-01**	8,8E-04	1,2E-01	**4,4E-01**	**2,4E-01**	3,5E-02	2,3E-03	**3,2E-01**	**1,8E-01**
80-90	**2,0E-01**	**5,5E-01**	**7,6E-01**	**6,8E-01**	*1,5E-01*	*4,0E-02*	**5,6E-01**	**4,5E-01**	**1,8E-01**	**5,1E-01**	**9,1E-02**	**9,8E-02**
90-100	**1**	**8,6E-02**	*6,3E-01*	*3,5E-02*	**3,1E-01**	**3,9E-01**	9,6E-03	4,5E-01	3,3E-01	1,3E-03	**2,7E-01**	**7,2E-01**
100-120	**6,5E-02**	**4,5E-01**	*1,6E-02*	*1,3E-01*	3,1E-01	1,7E-03	**2,4E-01**	**9,5E-01**	8,2E-03	1,8E-03	*1,5E-02*	*2,1E-01*
120-140	*4,8E-02*	*8,2E-01*	**4,9E-01**	**4,7E-01**	**1,3E-01**	**3,8E-01**	1,4E-03	8,1E-02	1,8E-02	2,6E-05	1,4E-03	6,6E-03
140-160	**7,4E-01**	**8,4E-01**	**2,7E-01**	**6,1E-02**	**2,0E-01**	**6,8E-01**	*8,9E-01*	*1,3E-02*	**3,9E-01**	**1,7E-01**	*3,8E-01*	*4,1E-02*
160-180	**5,7E-01**	**1,3E-01**	**6,1E-01**	**2,3E-01**	**8,5E-02**	**2,9E-01**	5,3E-02	1,4E-04	1,8E-06	4,3E-01	**6,8E-02**	**1,4E-01**
180-200	*1,7E-01*	*2,8E-02*	*1,9E-02*	*9,9E-01*	**5,6E-02**	**1,1E-01**	7,4E-03	8,2E-03	**7,6E-01**	**3,5E-01**	**5,1E-01**	**1,4E-01**
200-250	*1,5E-02*	*2,3E-02*	5,6E-01	2,3E-07	1,3E-03	2,2E-13	5,2E-02	0	1,7E-03	0	1,1E-04	1,5E-04
250-300	**4,3E-01**	**1,3E-01**	*3,9E-02*	*2,0E-02*	1,1E-02	0	1,8E-04	0	1,0E-01	0	2,5E-04	4,1E-13
300-400	1,6E-04	3,0E-05	9,2E-01	0	1,1E-09	0	1,1E-01	0	1,1E-04	0	2,0E-09	0
400-500	9,6E-03	2,6E-10	1,0E-01	5,1E-13	9,3E-02	0	5,9E-02	0	1,1E-01	0	3,9E-03	0
500-600	1,5E-01	3,3E-15	1,9E-01	0	6,6E-03	0	3,2E-01	0	1,8E-01	0	2,2E-01	0
600-700	3,3E-02	0	4,0E-02	0	3,2E-01	0	-	0	3,2E-01	0	4,8E-01	0
700-1000	4,6E-03	0	7,0E-03	0	-	0	-	0	-	0	1,5E-02	0

Tableau D.2 – p-values entre les résultats de deux expériences consécutives de la taille de population (SSGA - opérateurs sans heuristique).

	DPX		SDPX		NDPX	
s_p	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}
10-15	0	0	0	0	0	0
15-20	1,6E-05	0	1	0	3,2E-01	0
20-25	8,0E-04	8,3E-01	3,2E-01	1,5E-08	-	5,6E-12
25-30	**2,5E-01**	**9,9E-02**	**-**	**1,2E-01**	**-**	**2,4E-01**
30-40	4,5E-02	2,7E-03	3,2E-01	0	-	0
40-50	**-**	**8,6E-01**	**5,6E-01**	**2,0E-01**	1,6E-01	0
50-60	3,2E-01	1,7E-05	5,6E-01	8,3E-11	1	0
60-70	**3,2E-01**	**6,8E-01**	**5,6E-01**	**1,2E-01**	5,6E-01	0
70-80	**-**	**9,0E-01**	6,5E-01	0	1	0
80-90	**-**	**7,2E-02**	*6,5E-01*	*1,0E-02*	3,2E-01	0
90-100	-	4,9E-05	1	2,0E-04	-	0
100-120	-	7,7E-03	5,6E-01	0	5,1E-15	0
120-140	**-**	**8,6E-01**	3,2E-01	0	0	0
140-160	-	1,4E-13	-	0	0	3,4E-12
160-180	**-**	**7,7E-01**	3,2E-01	0	0	8,6E-06
180-200	-	2,5E-04	3,2E-01	0	0	6,5E-01
200-250	-	1,7E-06	-	0	0	3,6E-01
250-300	-	2,1E-06	-	0	2,8E-03	5,8E-01
300-400	-	2,4E-06	3,2E-01	0	**8,3E-02**	**1**
400-500	-	1,0E-09	0	0	-	-
500-600	-	4,4E-04	0	0	-	-
600-700	-	6,1E-08	0	0	-	-
700-1000	-	1,1E-07	0	2,5E-07	-	-

Tableau D.3 – p-values entre les résultats de deux expériences consécutives de la taille de population (GGA - opérateurs avec heuristique gloutonne).

	DPX		SDPX		NDPX	
s_p	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}
10-15	-	5,0E-04	**8,3E-02**	**5,0E-01**	*6,0E-02*	*1,9E-02*
15-20	-	7,9E-03	-	7,1E-03	6,2E-01	2,4E-05
20-25	-	0	-	6,3E-13	6,2E-01	0
25-30	-	3,4E-14	3,2E-01	0	3,6E-01	0
30-40	-	0	3,2E-01	0	1	0
40-50	-	0	-	0	7,1E-01	0
50-60	-	0	-	0	6,5E-01	0
60-70	-	0	-	0	6,5E-01	0
70-80	-	0	-	0	7,1E-01	0
80-90	-	0	-	6,1E-05	7,1E-01	0
90-100	-	5,3E-14	-	0	6,5E-01	0
100-120	-	0	-	0	5,6E-01	0
120-140	-	0	-	0	1	0
140-160	-	0	-	0	5,6E-01	0
160-180	-	4,6E-12	-	0	1	0
180-200	-	1,7E-08	-	0	1,6E-01	0
200-250	-	0	-	0	1,6E-01	0
250-300	-	3,3E-14	-	0	5,6E-01	0
300-400	-	4,2E-06	-	0	1	0
400-500	-	*1,1E-02*	-	0	3,2E-01	0
500-600	-	*2,3E-02*	-	0	1,6E-01	0
600-700	-	**7,5E-01**	-	0	1,2E-04	0
700-1000	-	**7,5E-01**	0	0	0	0

Tableau D.4 – p-values entre les résultats de deux expériences consécutives de la taille de population (SSGA - opérateurs avec heuristique gloutonne).

	CX		PMX		UPMX		PBX		UPBX		UOX	
p_c	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}
0,1-0,2	0	6,1E-01	8,5E-01	1,1E-04	**4,5E-01**	**2,1E-01**	**4,4E-01**	**8,2E-01**	*6,6E-01*	*4,1E-02*	5,2E-05	1,0E-01
0,2-0,3	0	0	*4,2E-01*	*2,9E-02*	**3,5E-01**	**1,2E-01**	7,6E-01	6,3E-04	*3,2E-01*	*1,9E-02*	1,5E-09	5,8E-02
0,3-0,4	8,4E-01	0	**2,4E-01**	**1,1E-01**	**4,0E-01**	**3,3E-01**	3,6E-01	3,1E-04	**8,1E-01**	**9,8E-01**	7,7E-13	3,9E-01
0,4-0,5	4,8E-01	0	6,7E-02	1,4E-03	**1**	**2,2E-01**	1,9E-01	3,1E-06	1,0E-01	1,9E-04	0	4,1E-01
0,5-0,6	3,9E-02	0	2,7E-02	2,0E-07	1,1E-03	7,8E-04	1,6E-14	0	1,5E-05	1,0E-07	2,5E-11	7,9E-01
0,6-0,7	5,6E-02	0	1,1E-09	7,3E-10	0	0	0	0	3,0E-12	1,6E-13	0	0
0,7-0,8	5,1E-01	0	0	1,0E-12	0	0	0	7,6E-08	0	0	0	8,6E-01
0,8-0,85	1,5E-02	2,8E-13	4,6E-15	6,8E-02	0	0	1,5E-03	1,6E-01	1,1E-16	1,1E-16	**1,3E-01**	**6,8E-01**
0,85-0,9	6,4E-01	7,8E-05	3,3E-16	1,6E-07	0	0	1,5E-07	4,4E-06	2,5E-06	4,4E-03	4,6E-03	1
0,9-0,95	2,1E-02	2,8E-07	1,1E-14	7,4E-08	0	7,4E-02	2,3E-02	1,1E-15	1,2E-13	1,0E-02	-	-
0,95-0,96	9,5E-02	3,0E-03	**8,1E-01**	**3,1E-01**	*1,4E-02*	*3,0E-01*	**4,6E-01**	**5,3E-01**	**2,7E-01**	**3,9E-01**	-	-
0,96-0,97	**6,2E-01**	**8,2E-01**	**4,4E-01**	**1,6E-01**	3,7E-03	5,2E-01	**5,9E-02**	**1,0E-01**	*1,2E-02*	*8,5E-01*	-	-
0,97-0,98	**6,2E-01**	**5,8E-01**	*1,6E-01*	*1,3E-02*	3,5E-03	5,9E-01	4,8E-01	1,4E-03	**1**	**1,2E-01**	-	-
0,98-0,99	1	3,5E-03	**6,5E-01**	**9,7E-01**	2,2E-03	7,9E-01	*7,1E-01*	*2,2E-02*	*6,7E-02*	*4,3E-02*	-	-

Tableau D.5 – p-values entre les résultats de deux expériences consécutives de la probabilité de croisement (GGA - opérateurs sans heuristique).

	CX		PMX		UPMX		PBX		UPBX		UOX	
p_m	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}
0,001-0,01	0	1	0	1,3E-07	0	4,0E-01	0	6,0E-04	0	1,1E-01	0	1
0,01-0,05	7,2E-03	0	7,9E-02	0	1,8E-14	0	8,3E-02	0	2,7E-04	0	8,0E-11	0
0,05-0,06	9,7E-01	8,0E-11	5,1E-01	1,5E-07	**4,9E-01**	**1,3E-01**	5,2E-01	3,6E-08	1,7E-01	1,3E-03	*7,5E-01*	*1,2E-02*
0,06-0,07	2,4E-01	1,5E-04	6,1E-01	7,2E-07	**8,9E-01**	**5,4E-01**	4,8E-01	1,3E-05	**5,0E-01**	**1,8E-01**	4,0E-07	9,9E-01
0,07-0,08	5,8E-01	2,7E-05	2,0E-01	1,7E-04	2,5E-03	3,4E-01	6,3E-01	5,0E-04	*9,4E-01*	*2,5E-02*	*1,1E-02*	*5,3E-01*
0,08-0,09	2,0E-01	6,1E-04	*1,9E-01*	*1,3E-02*	*4,2E-01*	*3,0E-02*	**8,5E-01**	**5,4E-01**	**5,0E-01**	**1,6E-01**	0	7,4E-02
0,09-0,1	1,5E-03	3,8E-02	5,9E-02	6,7E-03	**6,3E-01**	**2,8E-01**	*5,6E-01*	*3,7E-02*	**2,3E-01**	**7,3E-01**	1,5E-10	7,7E-02
0,1-0,11	5,1E-02	6,2E-04	6,4E-02	7,5E-03	**5,5E-01**	**6,2E-02**	**6,9E-01**	**5,1E-02**	**6,8E-01**	**6,9E-01**	0	8,9E-01
0,11-0,12	**4,1E-01**	**2,1E-01**	3,3E-01	8,5E-03	*5,6E-02*	*3,4E-02*	*3,1E-01*	*3,2E-02*	**1,0E-01**	**3,4E-01**	2,9E-14	1,9E-01
0,12-0,13	8,6E-01	6,0E-04	**4,7E-01**	**2,7E-01**	9,0E-01	4,7E-03	*9,8E-02*	*2,7E-02*	**7,5E-02**	**6,8E-01**	5,8E-14	6,8E-05
0,13-0,14	**5,1E-01**	**7,8E-01**	**2,9E-01**	**8,3E-02**	5,5E-01	4,3E-03	**7,8E-01**	**2,3E-01**	**6,7E-01**	**4,6E-01**	1,7E-03	5,1E-01
0,14-0,15	**8,5E-02**	**1,3E-01**	**1,2E-01**	**8,9E-02**	8,2E-01	1,2E-04	**2,6E-01**	**2,7E-01**	*2,8E-02*	*6,1E-01*	*7,1E-01*	*1,3E-02*
0,15-0,2	3,6E-01	6,4E-08	9,8E-03	5,6E-06	0	0	7,1E-06	8,0E-01	2,2E-03	1,0E-09	0	1
0,2-0,25	1,6E-02	6,3E-04	6,2E-05	5,9E-04	0	0	3,5E-05	1,0E-14	1,5E-03	0	-	-
0,25-0,3	*8,0E-02*	*2,4E-02*	2,8E-09	1,1E-02	0	2,9E-12	9,6E-09	0	-	0	-	-
0,3-0,4	3,1E-05	6,5E-01	0	0	0	1	2,5E-01	0	0	0	-	-
0,4-0,5	6,5E-10	5,0E-06	0	0	-	-	0	0	0	1	-	-
0,5-0,6	1,6E-10	2,2E-02	0	1	-	-	1,5E-14	1	-	-	-	-
0,6-0,7	7,1E-12	0	-	-	-	-	-	-	-	-	-	-

Tableau D.6 – p-values entre les résultats de deux expériences consécutives de la probabilité de mutation (GGA - opérateurs sans heuristique).

	CX		PMX		UPMX		PBX		UPBX		UOX	
p_m	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}	p_{SR}	p_{AES}
0,001-0,01	0	0	1,4E-15	1,0E-03	*1,5E-02*	*4,4E-01*	6,8E-03	5,3E-01	3,8E-04	9,5E-01	0	7,3E-01
0,01-0,05	1,6E-01	0	1,1E-02	5,1E-04	*3,4E-02*	*8,8E-02*	**5,5E-01**	**5,6E-01**	7,8E-01	1,2E-03	*1,2E-02*	*2,2E-02*
0,05-0,06	8,8E-01	0	**6,5E-01**	**9,9E-01**	*4,6E-02*	*8,1E-01*	**5,2E-01**	**7,3E-01**	**7,8E-01**	**6,0E-01**	**7,8E-01**	**4,1E-01**
0,06-0,07	6,4E-01	1,6E-07	**6,8E-01**	**7,8E-01**	*8,6E-01*	*3,1E-02*	**8,9E-01**	**9,1E-01**	**1**	**2,0E-01**	**2,7E-01**	**1**
0,07-0,08	4,3E-01	1,8E-10	**1**	**6,8E-01**	**9,1E-02**	**8,0E-01**	**5,7E-01**	**7,1E-01**	**7,6E-01**	**9,8E-01**	4,3E-03	3,1E-01
0,08-0,09	8,7E-01	2,4E-04	**5,2E-01**	**9,8E-01**	**3,0E-01**	**4,6E-01**	**1**	**4,1E-01**	**5,6E-01**	**2,0E-01**	**7,7E-02**	**1,4E-01**
0,09-0,1	7,4E-01	1,0E-04	**9,4E-02**	**4,4E-01**	**5,4E-01**	**3,8E-01**	**4,8E-01**	**8,8E-01**	**5,6E-01**	**1**	**7,7E-01**	**8,4E-01**
0,1-0,11	*1*	*1,1E-02*	**7,1E-01**	**1**	**7,7E-01**	**2,1E-01**	**4,8E-01**	**1,0E-01**	**7,6E-01**	**8,4E-01**	**7,7E-01**	**2,4E-01**
0,11-0,12	2,1E-01	1,9E-03	**9,5E-01**	**6,0E-01**	**3,3E-01**	**4,9E-01**	**7,6E-01**	**6,4E-01**	**2,2E-01**	**6,6E-02**	**7,1E-01**	**4,6E-01**
0,12-0,13	**2,1E-01**	**8,2E-02**	**2,8E-01**	**9,8E-01**	**6,8E-02**	**3,9E-01**	**7,6E-01**	**3,6E-01**	**6,5E-01**	**9,8E-01**	**7,2E-01**	**1,1E-01**
0,13-0,14	7,2E-01	2,8E-05	*1,9E-02*	*1,3E-02*	**6,8E-01**	**4,4E-01**	**6,7E-01**	**8,3E-01**	**8,3E-02**	**3,1E-01**	**3,0E-01**	**3,5E-01**
0,14-0,15	*2,5E-01*	*2,9E-02*	*2,2E-02*	*2,7E-01*	**6,6E-01**	**3,2E-01**	**6,7E-01**	**5,6E-02**	7,1E-01	6,0E-03	**3,9E-01**	**7,3E-01**
0,15-0,2	4,2E-01	0	**2,1E-01**	**4,4E-01**	7,6E-01	1,5E-03	**1**	**5,1E-01**	**4,1E-01**	**5,5E-02**	*6,6E-01*	*2,4E-02*
0,2-0,25	6,2E-01	5,9E-11	**9,5E-01**	**1,5E-01**	8,7E-01	6,8E-03	8,8E-01	5,8E-04	9,5E-02	5,9E-06	*8,0E-01*	*4,8E-02*
0,25-0,3	5,0E-01	1,3E-03	5,3E-01	9,8E-04	9,3E-02	1,1E-04	2,7E-01	1,8E-03	**7,8E-01**	**1,4E-01**	*5,5E-01*	*2,1E-02*
0,3-0,4	1	1,8E-06	8,5E-01	4,5E-04	3,2E-01	3,9E-14	1	3,0E-08	5,3E-01	0	2,7E-01	1,6E-14
0,4-0,5	**6,1E-01**	**2,7E-01**	6,0E-01	3,8E-11	8,5E-01	0	8,7E-01	6,2E-15	4,5E-02	0	1,5E-01	0
0,5-0,6	4,9E-01	7,2E-06	2,4E-01	0	8,5E-01	0	3,7E-01	0	1,6E-01	0	9,4E-01	0
0,6-0,7	1,4E-01	0	1,8E-01	0	6,8E-01	0	2,9E-02	0	1	0	4,3E-01	0

Tableau D.7 – p-values entre les résultats de deux expériences consécutives de la probabilité de mutation (SSGA - opérateurs sans heuristique).

	DPX		SDPX		NDPX	
p_m	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}
0,001-0,01	0	0	0	1	0	1
0,01-0,05	-	0	2,4E-01	0	4,1E-04	0
0,05-0,06	-	1,5E-07	5,1E-01	0	1	0
0,06-0,07	3,2E-01	4,8E-11	1,0E-01	0	3,5E-01	0
0,07-0,08	3,2E-01	6,0E-03	5,0E-01	0	2,3E-01	3,7E-13
0,08-0,09	-	*1,6E-02*	7,8E-01	4,1E-15	7,4E-01	6,3E-11
0,09-0,1	-	4,6E-06	7,9E-01	5,0E-13	8,9E-01	1,2E-08
0,1-0,11	-	**9,4E-01**	2,7E-01	4,2E-08	8,4E-01	5,3E-06
0,11-0,12	**3,2E-01**	**2,6E-01**	1,0E-01	9,3E-06	9,8E-02	6,4E-07
0,12-0,13	3,2E-01	2,7E-05	8,9E-01	7,6E-07	4,0E-01	8,8E-04
0,13-0,14	**3,2E-01**	**3,2E-01**	7,8E-01	1,6E-04	3,6E-01	2,8E-05
0,14-0,15	**3,2E-01**	**5,9E-01**	6,9E-01	5,3E-04	6,2E-01	2,5E-03
0,15-0,2	-	5,0E-06	5,6E-02	0	4,3E-02	0
0,2-0,25	**3,2E-01**	**6,9E-02**	1,8E-01	1,6E-05	2,0E-11	1,4E-03
0,25-0,3	3,2E-01	1,7E-04	1,5E-03	4,4E-16	6,1E-05	0
0,3-0,4	-	0	0	1	0	1
0,4-0,5	0	1,4E-14	-	-	-	-
0,5-0,6	**2,1E-01**	**6,3E-01**	-	-	-	-
0,6-0,7	0	0	-	-	-	-

Tableau D.8 – p-values entre les résultats de deux expériences consécutives de la probabilité de mutation (GGA - opérateurs avec heuristique gloutonne).

	DPX		SDPX		NDPX	
p_m	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}	p_{SR}	p_{ARS}
0,001-0,01	0	5,4E-02	0	0	0	0
0,01-0,05	4,5E-02	4,7E-03	1	0	6,8E-03	0
0,05-0,06	-	*1,6E-02*	3,2E-01	5,5E-11	4,9E-01	1,7E-06
0,06-0,07	-	**5,0E-01**	-	4,4E-04	8,3E-01	6,3E-04
0,07-0,08	-	**6,6E-02**	-	2,8E-03	8,3E-01	9,0E-06
0,08-0,09	-	**7,9E-01**	-	1,5E-03	**1,3E-01**	**7,7E-01**
0,09-0,1	-	**2,4E-01**	-	3,0E-05	*2,8E-01*	*3,2E-02*
0,1-0,11	-	**5,7E-01**	**3,2E-01**	**5,6E-02**	6,2E-01	9,4E-03
0,11-0,12	-	**5,6E-01**	*3,2E-01*	*4,2E-02*	**3,6E-01**	**1,8E-01**
0,12-0,13	-	**6,6E-01**	-	**8,4E-01**	**5,3E-01**	**1,7E-01**
0,13-0,14	-	**5,6E-02**	-	**4,1E-01**	**1**	**2,1E-01**
0,14-0,15	-	**1,2E-01**	-	**2,4E-01**	*5,9E-01*	*1,2E-02*
0,15-0,2	-	**1,1E-01**	-	4,5E-09	4,9E-01	1,0E-07
0,2-0,25	-	**1,4E-01**	-	4,4E-07	**8,3E-01**	**6,2E-01**
0,25-0,3	-	**1,6E-01**	**3,2E-01**	**8,3E-01**	**8,3E-01**	**9,2E-02**
0,3-0,4	-	9,1E-04	**3,2E-01**	**6,4E-01**	*3,4E-01*	*2,4E-02*
0,4-0,5	-	1,4E-03	3,2E-01	4,0E-04	2,5E-01	3,5E-08
0,5-0,6	-	1,4E-10	1	1,2E-15	3,7E-01	6,0E-12
0,6-0,7	-	0	1	0	5,9E-01	8,5E-14

Tableau D.9 – p-values entre les résultats de deux expériences consécutives de la probabilité de mutation (SSGA - opérateurs avec heuristique gloutonne).

	PMX	
p_m	p_{SR}	p_{AES}
0,35-0,36	6,3E-01	9,2E-04
0,36-0,37	1,9E-04	1,3E-05
0,37-0,38	*1,4E-02*	*1,3E-02*
0,38-0,39	7,6E-02	4,8E-06
0,39-0,40	7,0E-02	1,9E-03
0,40-0,41	4,1E-01	4,2E-06
0,41-0,42	1,6E-01	4,0E-07
0,42-0,43	-	8,7E-07
0,43-0,44	-	2,1E-08
0,44-0,45	4,5E-02	3,8E-12

Tableau D.10 – p-valeurs entre les expériences détaillées de la probabilité de mutation (GGA-PMX, Fig. 3.3).

	UPMX	
p_c	p_{SR}	p_{AES}
0,75-0,76	3,2E-03	9,9E-01
0,76-0,77	7,7E-06	1,4E-03
0,77-0,78	2,6E-05	4,8E-02
0,78-0,79	6,4E-04	4,7E-08
0,79-0,8	1,3E-04	0
0,8-0,81	**1**	**1,2E-01**
0,81-0,82	3,4E-01	9,5E-15
0,82-0,83	5,5E-05	6,4E-10
0,83-0,84	8,3E-08	3,0E-11
0,84-0,85	1,5E-12	3,5E-05

Tableau D.11 – p-valeurs entre les expériences détaillées de la probabilité de croisement (GGA-UPMX, Fig. 3.4).

	UOX $p_c = 0,6$		UOX 4 individus sans croisement	
s_p	p_{SR}	p_{AES}	p_{SR}	p_{AES}
8-9	6,5E-01	1,2E-06	6,5E-01	1,2E-06
9-10	**3,2E-01**	**7,7E-01**	**3,2E-01**	**7,7E-01**
10-11	0	7,7E-02	**1,1E-01**	**2,0E-01**
11-12	3,0E-04	7,3E-01	6,9E-02	4,5E-03
12-13	0	9,3E-01	2,2E-03	1,3E-03
13-14	**2,9E-01**	**5,3E-01**	1,1E-05	4,5E-03
14-15	**1,7E-01**	**1,6E-01**	7,6E-06	2,9E-04
15-16	0	1,9E-06	4,0E-04	4,1E-01
16-17	**7,1E-01**	**4,1E-01**	4,3E-11	1,9E-02
17-18	6,9E-05	1,3E-04	5,2E-04	2,2E-01
18-19	**6,7E-01**	**5,5E-01**	5,8E-08	7,9E-01
19-20	**7,6E-01**	**8,8E-02**	1,3E-07	9,2E-02

Tableau D.12 – p-valeurs entre les expériences détaillées de la taille de population (GGA-UOX, Figs. 3.8, 3.9).

	UOX	
p_m	p_{SR}	p_{AES}
0,1-0,11	0	4,1E-02
0,11-0,12	3,2E-12	8,6E-01
0,12-0,13	2,1E-13	4,7E-07
0,13-0,14	1,5E-03	5,2E-01
0,14-0,15	7,8E-01	1,2E-04
0,15-0,16	9,7E-11	0
0,16-0,17	0	0
0,17-0,18	0	2,2E-01
0,18-0,19	5,6E-16	8,1E-02
0,19-0,2	**1,6E-01**	**1**

Tableau D.13 – p-valeurs entre les expériences détaillées de la probabilité de mutation (GGA-UOX, Fig. 3.8).

	DPX	SDPX	NDPX
p_c	p_{ARS}	p_{ARS}	p_{ARS}
0,15-0,20	8,98E-05	0	*2,23E-02*
0,20-0,25	**1,94E-01**	**5,98E-01**	0
0,25-0,30	**5,48E-01**	5,46E-03	**1,62E-01**
0,30-0,35	**4,54E-01**	9,30E-07	7,35E-03
0,35-0,40	**8,34E-01**	0	0
0,40-0,45	**9,76E-01**	0	0

Tableau D.14 – p-valeurs entre les valeurs consécutives de croisement pour les opérateurs *DPX (Figs. 3.10, 3.12, 3.13).

p_c	s_p	p_{ARS}
0,15	70	6,96E-03
0,20	60	**1,94E-01**
0,25	50	**1**
0,30	50	**5,48E-01**
0,35	40	**8,81E-01**
0,40	30	**9,53E-01**
0,45	25	**9,40E-01**

Tableau D.15 – p-valeurs par rapport à la troisième ligne - pas de signifiance (GGA-DPX, Fig. 3.10).

Opérateur	p_{SR}	p_{AES}	p_{SR}	p_{ARS}
CX	6,0E-02	0	6,0E-02	3,6E-03
PMX	0	0	0	0
UPMX	8,0E-01	0	8,0E-01	0
PBX	9,0E-03	0	9,0E-03	0
UPBX	4,4E-01	0	4,4E-01	0
UOX	2,9E-13	0	2,9E-13	0
DPX	-	0	-	0
SDPX	-	0	-	0
NDPX	4,6E-03	0	4,6E-03	0

Tableau D.16 – p-valeurs entre les résultats obtenus par le GGA par rapport au SSGA pour chaque opérateur.

Paramètre	Taille 20		Taille 30		Taille 40	
5-7,5	3,2E-01	9,9E-12	9,7E-02	0	1,2E-06	4,5E-01
7,5-10	**2,0E-01**	**6,3E-01**	3,2E-03	7,0E-02	1,7E-06	3,1E-05
10-12,5	**4,8E-01**	**1,0E-01**	*3,3E-02*	*1,6E-02*	1,5E-09	7,8E-07
12,5-15	8,3E-02	1,2E-12	6,6E-03	1,5E-06	3,0E-04	9,0E-01
15-17,5	-	1,9E-15	3,2E-01	0	-	0
17,5-20	-	0	-	0	-	0
20-22,5	-	0	-	0	0	0
22,5-25	-	0	-	0	0	1
25-27,5	-	0	0	0	-	-
27,5-30	-	0	0	1	-	-
30-32,5	-	0	-	-	-	-
32,5-35	1,1E-16	0	-	-	-	-
35-37,5	0	3,4E-10	-	-	-	-
37,5-40	1,1E-10	1	-	-	-	-

Tableau D.17 – p-valeurs entre les valeurs consécutives du paramètre de la mutation *scramble* (uniforme, GGA-CX).

Paramètre	Taille 20		Taille 30		Taille 40	
5-7,5	-	1,7E-06	-	**3,0E-01**	-	**5,8E-02**
7,5-10	-	**7,7E-01**	-	1,0E-03	-	2,1E-04
10-12,5	-	5,4E-08	-	*1,4E-02*	-	*1,2E-02*
12,5-15	-	**3,3E-01**	-	**2,3E-01**	-	**6,0E-02**
15-17,5	-	7,6E-06	-	**8,8E-01**	-	8,6E-03
17,5-20	-	**1,9E-01**	-	*2,3E-02*	2,7E-03	2,4E-02
20-22,5	-	**1,2E-01**	-	**2,4E-01**	8,7E-10	3,6E-02
22,5-25	-	**1,4E-01**	-	1,3E-03	4,8E-08	2,9E-02
25-27,5	-	**1,5E-01**	**8,3E-02**	**4,8E-01**	9,1E-08	7,1E-01
27,5-30	-	7,3E-03	7,5E-03	1,1E-03	2,4E-05	4,1E-04
30-32,5	-	**4,8E-01**	1,6E-04	6,3E-02	**6,0E-02**	**4,6E-01**
32,5-35	-	**3,0E-01**	3,0E-10	2,0E-01	2,5E-05	1,5E-02
35-37,5	-	9,1E-04	1,4E-03	2,8E-02	4,5E-04	1,6E-02
37,5-40	-	**1,2E-01**	1,2E-05	2,4E-02	**4,1E-01**	**7,0E-02**

Tableau D.18 – p-valeurs entre les valeurs consécutives du paramètre de la mutation *scramble* (uniforme, SSGA-DPX).

	GGA-CX		SSGA-DPX	
Paramètre	p_{SR}	p_{AES}	p_{SR}	p_{ARS}
5	0	0	-	**6,4E-02**
7,5	0	1,4E-15	-	4,2E-05
10	2,8E-12	2,9E-03	-	8,9E-16
12,5	1,8E-01	6,0E-03	-	0
15	8,1E-03	1,3E-03	-	0
17,5	8,1E-03	0	-	0
20	8,1E-03	0	2,7E-03	0
22,5	0	0	8,8E-15	0
25	0	1	0	0
27,5	0	1	0	0
30	0	1	0	7,8E-16
32,5	0	1	0	4,1E-12
35	0	1	0	1,1E-04
37,5	0	1	0	4,2E-01
40	0	1	0	1,3E-01

Tableau D.19 – Comparaison entre la mutation d'échange et la mutation *scramble* avec tous les paramètres.

Annexe E

Note sur la représentation binaire des nombres réels

Le problème de l'appariement inexact de graphes nécessite la minimisation d'une distance. En raison du bruit inhérent, nous avons besoin d'utiliser des nombres réels. La représentation en binaire de ceux-ci pour la grande majorité d'architectures d'aujourd'hui suit le standard IEEE 754 pour l'arithmétique à virgule flottante (IEEE754, 2008). Bien que la question de l'arithmétique des nombres réels soit rarement posée, elle présente des particularités inhérentes qui méritent et nécessitent leur examen (Goldberg, 1991a). En effet, il se pourrait que l'exactitude du calcul soit affectée si l'on traite des nombres réels comme des entiers. Les erreurs sont souvent dues à l'étape d'arrondi, ou à des valeurs particulières comme l'infini, +/- 0, ou le concept *Not-a-Number, NaN*. Une représentation binaire de longueur fixe distincte pour chaque nombre est impossible, même en appliquant des bornes comme pour les entiers, car les nombres réels sont un corps archimédien. Il existe toujours une valeur intermédiaire entre deux nombres donnés. Par conséquence, les valeurs binaires sont arrondies. Ainsi pour la valeur 0,2, la représentation en 32 bit en arrondissant avec la méthode *round-to-nearest-value* de l'IEEE-754 a 0 pour signe, 01111100 pour exposant et 10011001100110011001101 pour mantisse [16].

Ceci représente une valeur proche de 0,2 mais non la valeur 0,2 exacte qui nécessiterait une longueur infinie de successions de 1100. La différence est assez petite (dans ce cas elle vaut environ 3E-9), et dans un même logiciel, sur la même machine, les valeurs sont arrondies exactement de la même façon. Par conséquent, cela ne devrait pas poser problème.

Cependant, ces différences peuvent se renforcer au fur et à mesure après l'application des opérations arithmétiques. Ainsi, considérons la fonction d'évaluation $f(\vec{x})$ qui évalue un chromosome. Quand nous appliquons la recherche locale nous obtenons un individu $\vec{y}$ qui est très proche de $\vec{x}$.

16. Ce calcul a été effectué en utilisant la page web intitulée IEEE-754 Floating-Point Conversion http://babbage.cs.qc.edu/IEEE-754/Decimal.html).

Nous connaissons aussi la valeur d'amélioration de la *fitness* ($\Delta_f(\vec{x},\vec{y})$) par la recherche locale. En théorie, nous pouvons utiliser l'égalité $f(\vec{y}) = f(\vec{x}) + \Delta_f(\vec{x},\vec{y})$ pour remplacer l'évaluation de l'individu $\vec{y}$ qui est très coûteuse, par l'addition de deux valeurs connues. Néanmoins il s'avère que cette égalité n'est pas respectée dans la représentation binaire. Sur des graphes de taille 40, nous observons par exemple des différences de l'ordre de 3×10^{-4} en *single* précision et 5×10^{-13} en *double* précision. De fait, tous les opérateurs de comparaison (==,>,<,etc.) ne peuvent pas fonctionner de manière habituelle. Une approche simple à utiliser et efficace que nous avons adoptée est de fixer un seuil de tolérance. Elle se montre assez robuste durant nos expérimentations. Toutefois, il faut être conscient que cette méthode présente certains inconvénients quand elle est appliquée à des valeurs particulières ou de grande variabilité.

Une discussion de méthodes plus robustes encore et incluant les nouvelles valeurs particulières de la norme IEEE754-2008 est proposée par Dawson (2008).

La norme IEEE754 définit des représentations classiques de nombres réels par trois entiers significatifs : le signe, la mantisse et l'exposant. Il existe plusieurs niveaux de précision, avec des longueurs différentes de la séquence de bits. En général, sont observés deux effets : plus la séquence utilisée est longue, plus les calculs sont fiables, c'est-à-dire représentent une faible déviation de la valeur théorique. Toutefois, une séquence plus longue nécessite plus de ressources et mène ainsi à des temps d'exécution plus longs. La question est alors de savoir s'il existe un choix optimal pour toutes les applications ?

Selon la norme mentionnée ci-dessus, la longueur peut varier entre 32 bits (*single* précision) et 80 bits (*double* précision étendue), voire 128 bits (*quadruple* précision) depuis la mise à jour en 2008. La version initiale de la norme prévoit la base 2 uniquement, dans la version actuelle on trouve aussi des représentations utilisant la base 10. *Single* et *double* (64 bits) précisions sont les formats classiques dans des langages de programmation de haut niveau. En Java, ils correspondent aux types primitifs *float* et *double* respectivement. De nos jours, l'architecture des ordinateurs est en train de changer vers des système 64 bits, notamment le x86-64 : les registres, le bus d'adressage et les bus mémoire peuvent tous contenir 64 bits au lieu de 32 auparavant. Une valeur double rentre donc aussi facilement qu'un single sur des systèmes 32 bits. Quant aux unités d'arithmétique à virgule flottante (FPU), elles utilisent déjà des registres d'au moins 80 bits depuis le Pentium Pro en 1995. Une perte de performance est néanmoins possible due aux accès aux données (transferts cache par exemple) sur les machines 32 bits. Sur les architectures à 64 bits, les *doubles* précisions sont naturelles et plus précises. Nous n'avons pas observé de perte de vitesse en utilisant la double précision.

Néanmoins, l'utilisation du *single* peut offrir quelques avantages dans des cas spécifiques. Lorsque l'on est confronté à une grande quantité de valeurs à stocker en mémoire/cache, on peut en stocker deux fois plus. Avec deux fois plus de valeurs dans le cache, le temps d'accès à la mémoire est réduit. Les processeurs actuels peuvent exécuter des opérations SIMD (*single instruction multiple data*) sur des registres de 128 bits qui peuvent contenir par exemple soit deux valeurs en *double* précision soit

quatre en *single* précision. Afin d'exploiter ces fonctions il est nécessaire d'optimiser le code pour avoir plusieurs opérations du même genre à effectuer en même temps et choisir un compilateur/une machine virtuelle qui les transmette sous la forme d'une commande SIMD.