Homologie en Programmation GénétiqueApplication à la résolution d'un problème inverse

Michael Defoin Platel

▶ **To cite this version:**

Michael Defoin Platel. Homologie en Programmation GénétiqueApplication à la résolution d'un problème inverse. Autre [cs.OH]. Université Nice Sophia Antipolis, 2004. Français. NNT : . tel-00131993

HAL Id: tel-00131993

https://theses.hal.science/tel-00131993v1

Submitted on 20 Feb 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE NICE-SOPHIA ANTIPOLIS - UFR Sciences
Ecole Doctorale de Sciences et Technologies de l'Information et de la Communication

T H E S E

pour obtenir le titre de
Docteur en Sciences
de l'UNIVERSITÉ de Nice-Sophia Antipolis

Discipline : Informatique

présentée et soutenue par
Michaël DEFOIN PLATEL

Homologie en Programmation Génétique

Application à la résolution d'un problème inverse

Thèse dirigée par ***Philippe COLLARD***
soutenue le *19 Novembre 2004*

Jury :

M. Jean-Louis CAVARERO	Président
M. Marc SCHOENAUER	Rapporteur
M. Marco TOMASSINI	Rapporteur
M. Philippe COLLARD	Directeur
M. Yves DUTHEN	Examinateur
M. Manuel CLERGUE	Co-Directeur
M. Antoine MANGIN	Invité

Remerciements

A l'origine de cette thèse, il y a tout d'abord ma rencontre avec la société ACRI-ST qui a co-financé ces travaux. En particulier, je me dois de remercier Odile Hembise et Antoine Mangin qui m'ont fait confiance et qui m'ont assuré des conditions de travail exceptionnelles. Quelques regrets demeureront néanmoins ...

Je tiens à remercier sincèrement Philippe Collard, mon directeur de thèse, pour s'être engagé dans un projet qui pouvait paraître, à l'origine, un peu vague et qui aborde des thématiques nouvelles pour notre équipe. La disponibilité de Philippe, sa sérénité et son à-propos ont été pour moi décisifs. Mais c'est Manuel Clergue qui est descendu dans l'arène et qui a su canaliser mon tempérament, car entre excitations excessives et pertes de confiances définitives, j'ai bien conscience que la tâche a parfois été rude. De cette collaboration avec Manu, je retiendrai avant tout une grande complicité dans de nombreux domaines qui dépasse, et de loin, le cadre académique.

Je remercie Marc Schoenauer et Marco Tomassini qui ont bien voulu émettre un avis favorable sur ces travaux. Leurs rapports et leurs remarques m'ont aidé à finaliser ce mémoire. Mes remerciements vont également à Jean-Louis Cavarero, le président du Jury, ainsi qu'à Yves Duthen pour sa participation.

A l'heure de dresser un bilan, je me rend compte que durant cette aventure, qui m'était dans l'instant apparu solitaire, j'ai toujours été très bien accompagné. Je pense avant tout à Sébastien avec qui j'ai beaucoup partagé tant sur le plan humain que scientifique. Je pense également aux autres thésards, trop nombreux pour les remercier tous, mais je tiens néanmoins à en citer quelques uns : Eric, Stéphane, Karim, Gilles et plus récemment William.

Et puis, il y a ceux à qui l'on ne s'adresse qu'en cas de problèmes et qui agissent en coulisses pour que tout fonctionne bien, c'est à dire les équipes administratives et techniques : Micheline, Lionel, Guy, ... Je vous remercie.

Enfin et surtout, il y a mes proches et la première d'entre eux, Claire, à qui je ne pourrais en quelques lignes exprimer toute ma reconnaissance et puis bien d'autres choses qui n'ont pas leurs places ici. A dire vrai, sans toi, rien de tout cela n'aurait été possible et j'espère être à ta hauteur quand le moment viendra. Arrivé au terme de cette histoire, je pense à mes parents : à ton affection ou bien à ta mémoire, je vous dédie cette thèse, bien conscient de tout ce que je vous dois.

Table des matières

Table des figures

Liste des tableaux

Introduction

Alan Turing a marqué de façon profonde la naissance et le développement d'une science que nous appelons aujourd'hui l'Intelligence Artificielle. Un des objectifs de l'Intelligence Artificielle est de définir et de mettre en œuvre les moyens permettant de construire des machines intelligentes, c'est-à-dire capables d'effectuer des tâches complexes qui, quand elles sont effectuées par des humains, sont supposées nécessiter une forme d'intelligence. Dès le milieu du 20$^{\text{ième}}$ siècle dans "Computing Machinery and Intelligence ", Alan Turing défend l'idée qu'il serait possible de produire des machines intelligentes grâce à une forme d'apprentissage par essais et erreurs, qu'il compare au processus de l'évolution biologique.

Depuis une vingtaine d'années, une méthode de recherche automatique de programmes informatiques, nommée Programmation Génétique, est en passe de concrétiser le projet de Turing. En effet, la Programmation Génétique permet, à partir d'une description d'un problème à résoudre, de fournir des solutions, sous forme de programmes, qui répondent au moins partiellement à ce problème. La recherche de solutions est réalisée de manière itérative par l'évolution d'une population de descriptions de programmes qui subissent des modifications progressives et qui sont sélectionnées en fonction de leur adéquation au problème à résoudre.

Dans ce mémoire, nous nous intéresserons particulièrement à un de ces mécanismes de modification, appelé opérateur de recombinaison ou de croisement, dont le but est de permettre l'exploitation des qualités des différents programmes de la population durant l'évolution. L'objectif du premier chapitre est d'examiner en détail ce qu'est la Programmation Génétique et de la situer dans le cadre plus général de l'Évolution Artificielle. Nous présente-

rons les différents types de représentations de programme et nous justifierons notre intérêt pour la Programmation Génétique Linéaire. Nous verrons également qu'elles sont les limites et les améliorations possibles déjà identifiées de l'opérateur de croisement. Notamment, nous évoquerons ce qui est appelé le caractère "brutal" de ce type d'opérateur ainsi que son rôle dans les phénomènes de croissance incontrôlée, mais surtout injustifiée, de la taille des programmes durant l'évolution. Nous présenterons enfin le système de Programmation Génétique par Pile qui nous permettra, tout au long de ce mémoire, de confronter nos hypothèses à l'expérience.

Dans le chapitre 2, nous reviendrons sur les critiques formulées à l'encontre de l'opérateur de recombinaison utilisé de façon standard en Programmation Génétique et nous verrons comment la brutalité de l'opérateur est liée au concept d'*homologie*. Puis, après avoir résumé l'évolution historique du concept d'homologie, ainsi que ces relations avec les mécanismes de recombinaison dans la nature, nous définirons un nouvel opérateur pour la Programmation Génétique Linéaire appelé Croisement par Maximum d'Homologie (CMH). Le nouvel opérateur sera par la suite étudié de manière théorique et sa capacité à préserver l'homologie sera démontrée, puis illustrée.

Pour bien comprendre le fonctionnement d'un opérateur génétique, ainsi que ses implications sur l'évolution, une démarche classique consiste à utiliser un cadre expérimental dont les propriétés sont particulièrement bien connues, par exemple en ayant recours à des problèmes dits synthétiques dont les caractéristiques peuvent être ajustées. Le chapitre 3 sera dans un premier temps consacré à la définition et à l'étude de trois problèmes synthétiques pour la Programmation Génétique Linéaire. A l'aide de ces trois problèmes, le comportement du nouvel opérateur CMH sera dans un second temps comparé à celui de l'opérateur de croisement standard et nous verrons pour ces deux opérateurs, si les hypothèses faites sur la relation entre l'homologie et la brutalité de la recombinaison peuvent être confirmées.

Dans le chapitre 4, nous nous intéresserons plus précisément aux propriétés et à l'évolution des populations de programmes en fonction de l'opérateur de recombinaison utilisé. Nous débuterons ce chapitre par la poursuite de

l'étude expérimentale du CMH sur un problème booléen qui fait référence en Programmation Génétique. Les liens entre les phénomènes de croissance de la taille des programmes et les opérateurs de croisement seront étudiés en détail, puis nous verrons comment la conservation de l'homologie permet de concevoir des mécanismes de contrôle de la taille durant le processus évolutionnaire. Enfin, au cours d'une discussion sur la recherche réalisée, par les deux opérateurs de croisement, dans l'espace des programmes possibles, nous distinguerons trois types de recherches possibles : le mélange, la diffusion et l'exploration de la taille des programmes. Nous verrons que le CMH est quasi exclusivement un "mélangeur" pour la Programmation Génétique Linéaire.

La société ACRI-ST , qui a co-financé ce travail, est considérablement impliquée dans les Sciences de la Terre et dans la surveillance de l'Environnement en particulier. L'objectif du chapitre 5 est de mettre en évidence l'intérêt de méthodes issues de l'Intelligence Artificielle, comme la Programmation Génétique, pour les problématiques environnementales. La résolution de problèmes inverses constituant une des applications les plus intéressantes de la Programmation Génétique, nous utiliserons le Croisement par Maximum d'Homologie pour rechercher un modèle d'inversion des propriétés des aérosols atmosphériques. Le problème de la recherche d'un modèle d'inversion sera vu comme une modélisation statistique par apprentissage. Dans le contexte de la modélisation statistique, nous proposerons et nous étudierons les performances d'une amélioration du système de Programmation Génétique par Pile, appelée approche par *équipes de prédicteurs*. Enfin, nous verrons si les mécanismes de contrôle de taille envisagés au chapitre 4 sont fonctionnels et permettent de rechercher de manière plus efficace des modèles d'inversion utilisables de façon opérationnelle dans l'industrie.

Chapitre 1

Programmation Génétique

1.1 Évolution Artificielle

C'est dans "Origin of Species by Means of Natural Selection" [24], que Charles Darwin introduit le principe de la sélection naturelle qui unifie origine et évolution des espèces vivantes. Les critères de bases définissant l'évolution *in vivo* sont [67] :

la fécondité : les chances de reproduction d'un organisme dépendent de son adaptation à l'environnement ;

l'hérédité : les descendants sont semblables à leurs parents, *i.e.* la reproduction est fidèle ;

la variabilité : les descendants ne sont pas identiques à leurs parents, *i.e.* la reproduction n'est pas une copie parfaite.

En empruntant à la biologie ces critères, les Algorithmes Évolutionnaires (AE) permettent de résoudre des problèmes réputés difficiles par évolution *in silico* de solutions partielles ; ce domaine de recherche est appelé Évolution Artificielle (EA). Cependant l'évolution biologique ne constitue qu'une inspiration pour l'EA, ce n'est ni une justification ni une limite.

A partir d'une population initiale P_0 de solutions potentielles créées aléatoirement, un AE, grâce à des variations et des sélections successives, améliore la qualité globale des solutions. On définit une population au temps $t+1$ comme :

$$P_{t+1} = V(S(P_t))$$

où V et S sont des fonctions dites de variation et de sélection. Ce processus est itéré jusqu'à ce qu'un critère d'arrêt soit atteint et une itération de l'algorithme est appelée une *génération*. De façon générale, la taille de la population est fixée *a priori* et n'évolue pas.

Une solution potentielle est à la fois : *un génotype*, *un phénotype* et *une adaptation*. Le génotype correspond au codage de la solution, c'est le support de l'information et n'a pas d'autre signification. Celui-ci peut-être décodé, par une opération dite de morphogénèse, en une forme ayant un sens dans l'environnement, le phénotype. C'est le phénotype qui interagit avec l'environnement, *i.e.* le problème à résoudre, afin de déterminer la qualité d'une solution potentielle, c'est-à-dire son adaptation. La performance d'une solution est appelée valeur adaptative ou fitness.

C'est la fonction de sélection S qui, appliquée dans l'espace phénotypique, réalise le critère de fécondité en garantissant, à la génération suivante, que les solutions les plus adaptées seront plus représentées dans la population. De nombreux opérateurs de sélection existent, on citera par exemple la *sélection proportionnelle* où le nombre de copies d'une solution dans la prochaine génération est directement proportionnel à sa valeur de fitness et la *sélection par tournoi* où des sous-ensembles de la population, appelés tournois, sont choisis aléatoirement et où seule la plus performante des solutions de chaque tournoi est retenue pour la génération suivante. Des tournois étant organisés jusqu'à ce que la nouvelle population soit entièrement constituée.

Les solutions sélectionnées sont modifiées par la fonction de variation V qui est appliquée dans l'espace génotypique. Elle assure un compromis entre les critères d'hérédité et de variabilité. En effet, trop de variations ne permettent pas à la population de converger et la recherche est alors trop aléatoire, tandis qu'avec trop peu de variations la population converge trop rapidement et la recherche est susceptible de s'arrêter de façon prématurée. Deux classes d'opérateurs de variations ou opérateurs génétiques sont couramment utilisées : les opérateurs de *mutation* qui réalisent de petites variations aléatoires des solutions et permettent une recherche locale ; les opérateurs de *croisement* qui recombinent aléatoirement des informations extraites de deux solutions données pour en créer de nouvelles ce qui permet

d'exploiter les qualités des solutions présentes dans la population. On notera que des solutions peuvent être également recopiées tel quel d'une génération à l'autre par une opération que l'on appelle la *reproduction*.

D'un point de vue technique, la production d'une nouvelle population est réalisée soit par une méthode de *remplacement générationel*, où les nouveaux individus remplacent les anciens quand une population entière a été conçue, soit par une méthode de *remplacement par états fixes* ou *steady-state*, où les nouveaux individus remplacent les anciens au fur et à mesure de leurs conceptions, *i.e.* les parents et les enfants coexistent au sein de la population et peuvent donc être amenés à se reproduire par croisement.

Quatre grandes familles d'AE ont été définies à partir du type de codage, de sélection, de variation et de remplacement choisi. Elles ont des propriétés spécifiques qui leur permettent de traiter des objectifs différents :

1. Les Algorithmes Génétiques (AG) ont été introduits par Holland [42] comme modèles de systèmes adaptatifs. Historiquement, le codage est effectué grâce à des chaînes binaires de taille fixe. Le croisement ainsi que la mutation sont utilisés.

2. Les Stratégies d'Évolution (SE) ont été développées par Rechenberg et Schwefel [86][94], dans le but de résoudre des problèmes d'optimisation de paramètres discrets ou continus, principalement expérimentaux. Les génotypes sont des vecteurs de réels. La mutation, variation gaussienne des composantes du vecteur solution, est l'opérateur principal mais le croisement peut-être utilisé. Le cycle évolutif utilisé est un peu différent de ce que nous avons déjà décrit, notamment en ce qui concerne la sélection.

3. La Programmation Évolutive (PE) a été initialement proposée par Fogel [31] pour créer une nouvelle forme d'intelligence artificielle. Les solutions sont représentées sous forme d'automates à États Finis. Seule la mutation est utilisée.

4. La Programmation Génétique (PG), dont l'idée originale est souvent disputée [21][54], a pour objectif la recherche automatique de programmes. Ces programmes sont constitués par un ensemble de taille

variable de symboles représentant des instructions. Le croisement est l'opérateur principal mais la mutation peut-être utilisée.

1.2 Programmation Génétique

La PG est une méthode de recherche universelle qui, à partir d'une description d'un problème donné, permet de façon automatique de trouver des programmes qui résolvent ce problème. Ainsi, contrairement à d'autres méthodes de résolution qui utilisent des représentations spécialisées, comme les Arbres de Décisions ou les Réseaux de Neurones Artificiels, la PG utilise le cadre représentatif très général des programmes informatiques. En PG, un programme est une expression structurée composée d'un nombre variable de symboles, qui doit pouvoir être interprétée ou compilée puis exécutée par une machine.

1.2.1 PG Standard

Bien que la question de la paternité de l'invention de la PG puisse encore être débattue [7], le travail réalisé par J. R. Koza à partir de 1989 [53] est véritablement le point de départ de ce domaine de recherche. Dans plusieurs ouvrages de référence [54][55][56], il jette les bases de la PG puis réalise un vaste tour d'horizon de ses problématiques, tant théoriques que pratiques, il montre également comment des problèmes classiques de l'Intelligence Artificielle (IA) peuvent être résolus.

Représentation arborescente

Koza propose d'utiliser des notations préfixées en langage Lisp[1] pour coder les solutions d'un problème, ce qui permet de les évaluer directement avec un interprète adéquat. En effet, un programme peut être vu comme une structure hiérarchique arborescente [21]. A titre d'exemple, l'expression $1+(5-2)$ est équivalente à l'arbre syntaxique présenté Figure 1.1. Les nœuds internes de l'arbre sont appelés *fonctions* alors que les feuilles sont appelées

[1]Lisp est un langage de programmation couramment utilisé en IA classique dans le cadre du calcul symbolique.

terminaux. La notation $\big(+\ 1\ (-\ 5\ 2)\big)$ correspond à un parcours préfixé de cet arbre et constitue une S-expression en langage Lisp. Dans cette étude, on appellera Programmation Génétique Arborescente (PGA), un système PG utilisant ce type de représentation.

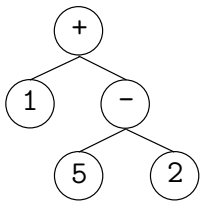

FIG. 1.1 – Exemple de représentation arborescente d'un programme

Préparation d'une expérience

Avant d'entreprendre la résolution d'un problème donné, Koza préconise quelques réflexions préparatoires, les principales étant :

1. choix de l'ensemble des terminaux, comme par exemple des variables ou des constantes ;
2. choix de l'ensemble des fonctions, comme par exemple des opérations mathématiques, des instructions de contrôle et de branchement ou des fonctions spécifiques ;
3. choix de la mesure de fitness ;
4. choix des paramètres de l'expérience, comme la taille de la population, la taille de programme maximum autorisée ou la probabilité d'application des opérateurs génétiques ;
5. choix du critère d'arrêt.

Ces différents choix sont évidement dépendants du problème à résoudre, ils sont souvent complexes et peuvent également s'avérer critiques. C'est pourquoi une bonne connaissance, à la fois du problème posé et de la PG est nécessaire.

Initialisation de la population

En PG, comme pour tout AE, une expérience débute par la création d'une population initiale qui constitue toujours une étape prépondérante car la population initiale doit être suffisamment diverse pour que la recherche puisse correctement explorer l'espace des solutions possibles. Dans le cas de la PG, cette diversité comprend, hormis l'influence de la taille de la population, la répartition des fonctions et des terminaux dans les arbres, la variabilité en profondeur (nombre de nœuds entre la racine et les feuilles des arbres) et en taille (nombre de nœuds total des arbres) ainsi que la variabilité en forme (arbres plus ou moins équilibrés).

Différentes stratégies ont été proposées pour initialiser la population. La procédure est la plupart du temps récursive et tient compte de la proportion de fonctions et de terminaux dans l'ensemble des primitives ainsi que de la taille et de la hauteur maximum des arbres autorisée, voir [64] pour une comparaison empirique de ces stratégies.

Opérateurs Génétiques

Pour Koza, les variations dans l'espace génotypique sont assurées par l'opérateur de croisement, qui réalise un échange de sous-arbres entre les individus, c'est pourquoi il préconise l'usage de grandes populations (plus de 2000 individus) assurant une forte diversité des sous-arbres. Un exemple de croisement produisant deux enfants est présenté Figure 1.2. Les deux parents sont les expressions $1+(5-2)$ et $ln(x)*4$. L'opération de croisement consiste à choisir un nœud, *i.e.* un sous-arbre, dans chacun des deux parents, en l'occurrence $(5-2)$ et x, puis à réaliser l'échange de ces deux sous-arbres, ce qui permet d'obtenir deux nouveaux programmes, les enfants $1+x$ et $ln(5-2)*4$.

Koza montre que cette opération n'est pas satisfaisante car la quantité de matériel génétique échangé est souvent trop faible. En effet, prenons l'exemple d'un arbre binaire, c'est-à-dire que les fonctions utilisées sont toutes d'arité 2, dans ce cas plus de 50% des sous-arbres sont des terminaux et donc 50% des sous-arbres échangés ne contiennent qu'un seul nœud. Pour éviter ce phé-

nomène [54], Koza sélectionne de façon prioritaire (90%) les nœuds internes des arbres et plus rarement des terminaux (10%). D'autres approches ont également été proposées pour obtenir différentes distributions de sélection des sous-arbres [58][39].

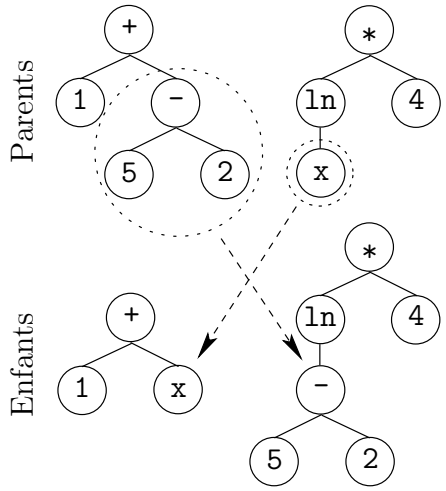

FIG. 1.2 – Exemple de croisement en PGA

Des opérateurs de mutations ont également été définis en PG. La mutation classique consiste à substituer à un des sous-arbres du programme à modifier, un sous-arbre créé aléatoirement. La Figure 1.3 montre comment le programme $ln(x) * 4$ peut être muté en $ln(1/y) * 4$. L'opérateur de mutation est parfois implémenté comme un croisement entre l'arbre devant être muté et un arbre spécifiquement créé à cet effet. Des opérateurs spécialisés dans la mutation des constantes sont fréquemment utilisés. Ils permettent par de petites variations de faciliter l'optimisation de ces constantes.

1.2.2 Domaines d'applications

On considère que la PG est une méthode très efficace pour la résolution de problèmes ayant les caractéristiques suivantes :

– les relations entre les variables explicatives du problème à résoudre sont inconnues (ou mal connues) ;

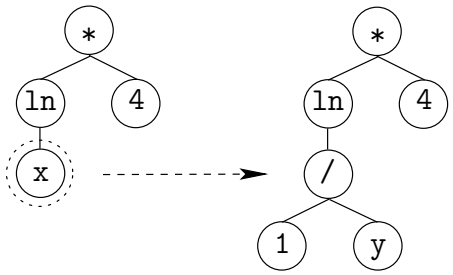

FIG. 1.3 – Exemple de mutation en PGA

- l'identification de la taille et la forme des solutions optimales du problème à résoudre est une question à part entière ;
- une quantité de données suffisante pour permettre de décrire le problème à résoudre est disponible ;
- il est possible d'évaluer la performance d'une solution candidate ;
- les méthodes classiques de résolution mathématiques n'ont pas permis de découvrir de solution analytique au problème ;
- une solution partielle, c'est-à-dire complète mais non optimale, du problème à résoudre est acceptable.

PG s'est d'ores et déjà illustrée comme une méthode adaptée à des tâches de classification [10], d'approximation de fonctions [48], de prédiction de séries temporelles [30], de résolution de problèmes inverses [20], de pilotage de robots [4], de conception de circuits électroniques [56], de recherche de stratégies multi-agents [66], de recherche de formules scientifiques [47], de recherche d'architecture de Réseaux de Neurones Artificiels [108], de création d'algorithmes en informatique quantique [101].

1.2.3 Axes de recherches

Représentation des programmes

La question de la représentation des solutions est toujours centrale en EA et c'est également le cas en PG, en particulier en ce qui concerne le principe de causalité qui veut que de petites variations dans l'espace génotypique induisent de petites variations dans l'espace phénotypique [88]. En PG,

cette question revêt aussi d'autres aspects spécifiques, comme par exemple la vitesse d'évaluation, la quantité de mémoire utilisée, la complexité des opérateurs génétiques associés.

D'un point de vue pratique, la représentation arborescente des individus peut être réalisée grâce à l'utilisation soit directement d'une structure d'arbre avec pointeurs soit de chaînes de symboles représentant un parcours préfixé ou postfixé d'un arbre, voir [50]. Dans le premier cas, l'évaluation est effectuée de façon récursive ce qui peut poser des problèmes de contrôle ainsi que d'allocations et de libérations répétées de la mémoire. Dans le deuxième, l'évaluation est itérative, les symboles étant traités séquentiellement, mais l'identification des sous-arbres dans le but d'appliquer les opérateurs génétiques est nécessaire et parfois coûteuse.

Des systèmes PG manipulant des structures linéaires ont également été testés et ont démontré des performances équivalentes aux systèmes PGA, comme c'est le cas dans [10]. Contrairement à la PGA, en Programmation Génétique Linéaire (PGL), les programmes sont représentés par des séquences d'instructions d'un langage impératif, comme le langage C ou l'assembleur. L'évaluation d'un programme ne peut se faire de manière récursive et implique l'utilisation de mécanismes supplémentaires pour le stockage des calculs intermédiaires. Il existe au moins deux types de mécanismes et donc de systèmes PGL où l'évaluation d'un programme requiert soit un nombre fini de registres dont un est choisi pour contenir la sortie du programme [10], soit une pile d'opérandes dont le sommet contient la sortie d'un programme [77][102][14][100]. Dans cette étude, nous utilisons un système de Programmation Génétique par Pile, appelé PGP. Cependant, la majeure partie des résultats présentés ne dépend pas du type de PGL utilisé.

Des représentations plus complexes ont été envisagées, comme la représentation de toute la population à l'aide d'un graphe acyclique direct [38] permettant d'économiser du temps de calcul et de la mémoire ou bien encore une représentation mixte des programmes [45], à la fois linéaire et arborescente, qui semble posséder les avantages des deux types de représentations classiques.

Nombre d'innovations dans le domaine de la représentation visent à faci-

liter la recherche de programmes particulièrement complexes :

- les contraintes syntaxiques et le typage [70][1][100] [91][92][73]
- les fonctions et les macros [54][55][99][89][100] ;
- les itérations et la récursivité [99][89][56][100].

Neutralité

Dans la théorie darwinienne, la pression de sélection est la principale force responsable de l'évolution. Cette vision, bien que largement acceptée, pose néanmoins certains problèmes et ne permet pas d'expliquer toutes les dynamiques évolutives observées dans la nature. La théorie de l'évolution neutre, proposée par Kimura [51], permet d'expliquer le haut niveau de diversité génétique des populations naturelles qui est difficilement compatible avec l'hypothèse sélectionniste. Pour Kimura, la majorité des changements évolutifs sont dus à la fixation dans les populations, par dérive génétique, de mutants dits neutres, c'est-à-dire des variations fréquentes qui n'ont pas d'effets sur la fitness des individus. Ceci implique que la plupart du temps les variations génétiques observées correspondent à des mutations neutres, *i.e.* des déplacements aléatoires entre des génotypes de fitness équivalentes et que plus rarement des variations avantageuses puissent apparaître. Des auteurs [43] ont ainsi pu envisager des dynamiques évolutives appelées marches neutres où la population se diffuse dans des réseaux neutres (ensembles de génotypes reliés par des mutations neutres) jusqu'à ce qu'une variation permette de découvrir d'autres réseaux neutres de meilleure fitness, et ainsi de suite.

La question de la neutralité est devenue également très importante en EA et peut-être plus encore en PG où le degré de neutralité des problèmes traités est souvent très élevé. Les programmes obtenus avec PG contiennent de grande quantité de code qui ne contribue pas ou très peu à la performance, ces parties des programmes sont appelés *les introns*. Les introns ont fait l'objet de nombreuses études en PG par exemple [72][40][63] et leur diversité a même amené certains auteurs à en dresser une taxonomie [3][95]. La présence d'introns dans les programmes et plus généralement la neutralité dans l'espace de recherche n'est pas nécessairement pénalisante pour le système et l'on

cherchera, le plus souvent, plutôt à exploiter la neutralité qu'à la supprimer.

Théorème des schémas

L'espace dans lequel s'effectue la recherche des programmes est trop complexe et de trop grande dimension pour que l'on puisse le décrire totalement et l'on doit souvent, sauf à multiplier les observations empiriques, avoir recours à l'intuition pour prédire le comportement d'un système PG.

La théorie des schémas est sans doute la plus aboutie des classes de modèles d'AE. Un schéma est une partition de l'espace de recherche composée de génotypes qui partagent une caractéristique particulière. L'étude théorique de l'évolution des schémas permet de décrire comment varie la proportion, dans une population, des individus appartenant à un schéma donné au cours du temps. Les travaux de Holland sur les schémas [42], repris et étendus par Goldberg[34] et par Radcliffe[84], ont permis de mieux comprendre et d'expliquer le fonctionnement des AG, notamment avec l'étude du parallélisme implicite, de l'hypothèse des blocs de construction et de propriétés fondamentales de l'opérateur recombinaison comme par exemple le respect. On appelle parallélisme implicite le fait que l'évaluation de la performance d'une solution donne des informations sur la performance de tous les schémas qu'elle contient et donc que l'évaluation d'une population nous renseigne sur un nombre de schémas beaucoup plus important que le nombre d'individus qui la composent. L'hypothèse des blocs de construction précise que les AG sont efficaces quand des schémas relativement courts ayant une fitness supérieure à la moyenne sont combinés, grâce à l'opérateur de croisement, pour former des solutions optimales ou proches d'un optimum. Le rôle d'un opérateur de croisement est donc fondamental. Pour Radcliffe qui propose des extensions des schémas de Holland nommées *formae*, il semble préférable qu'un opérateur de croisement soit construit sur le principe du respect, c'est-à-dire que le croisement de deux instances d'une forma donnée doit toujours produire une nouvelle instance de cette même forma. Il illustre son propos avec cet exemple :

> "Si deux parents ont les yeux bleus alors tous leurs enfants obtenus

par croisement devront avoir les yeux bleus." (traduit de l'anglais)

En PG, c'est la représentation par des structures de tailles variables des individus qui a rendu complexe la définition des schémas et parallèlement la compréhension du fonctionnement de l'opérateur de recombinaison. La notion de schéma a largement évoluée depuis les premières tentatives de définitions pour la PG [54] et n'a vraisemblablement atteint un niveau satisfaisant que récemment avec les travaux de Poli [60][82]. Pour Poli, cette théorie est un outil opérationnel qui permet de décrire et de prédire le comportement des AG et de la PG, d'expliquer pourquoi des individus sont conservés, ajoutés ou supprimés dans la population durant l'évolution, de définir exactement ce que sont les blocs de construction, de décrire et de prédire le comportement des opérateurs génétiques, notamment d'en prédire le réglage optimal. A titre d'exemples, on citera [80] où le théorème exact des schémas a permis de réaliser une étude théorique, confirmée expérimentalement, des propriétés de l'opérateur de recombinaison et de l'évolution de la taille des individus en PGL et également [78] où une amélioration de la PG, appelée méthode Tarpéienne, est directement issue de travaux théoriques sur les rapports entre la taille et la fitness des programmes.

Opérateurs de croisement

Le rôle de l'opérateur de croisement est un des sujets les plus débattus en PG. Ceci est dû, en partie, au fait que les connaissances et la compréhension que nous avons du comportement, ainsi que de la fonction, de la recombinaison dans les AG ne peuvent être directement reprises en PG. Entre autre l'hypothèse des blocs de construction s'est avérée difficile à transposer en PG et la notion de schéma, des plus ardues à définir.

Aux origines de la PG, avec les travaux de Koza, les individus sont représentés par des structures arborescentes en langage Lisp et le croisement consiste en un échange de sous-arbres entre les parents. Les blocs de construction sont des sous-arbres qui contribuent de façon très positive à la performance des individus et ont donc tendance à se répandre lorsqu'ils sont échangés. Pour Koza, l'opérateur de recombinaison est central dans le processus

de recherche.

Cependant dans [103], les auteurs s'interrogent sur la fonction de l'opérateur de croisement et mettent en évidence son comportement proche de celui d'un opérateur de macro mutation. D'après eux, PG devrait être vu comme un système de recherche par exploration locale avec population. Un constat similaire est dressé dans [6], où les résultats expérimentaux obtenus en utilisant le croisement sont moins bons que lorsque la mutation est utilisée seule. De plus, l'opérateur de croisement réalise un échange des sous-arbres sans tenir compte du contexte ni dans lequel ils sont prélevés ni dans lequel ils viennent s'insérer. Cette opération est considérée comme "brutale", dans [29], et la possibilité pour le processus évolutionnaire de faire émerger des solutions structurées est remise en cause, ce qui est difficilement compatible avec l'hypothèse des blocs de construction. On peut définir la brutalité de l'opérateur comme le fait que les enfants obtenus par recombinaison standard ont une fitness majoritairement moins bonne que leurs parents et l'on parle alors des effets délétère du croisement. Enfin, dans [79], les auteurs montrent que l'opérateur standard est un opérateur biaisé, qui ne permet pas de réaliser une exploration optimale de l'espace de recherche, en particulier en ce qui concerne la taille des programmes. De façon générale, les remarques précédentes valent pour toutes les représentations en tailles variables, arborescentes ou linéaires.

Différentes propositions ont été faites pour définir des opérateurs ne souffrant pas des défauts de l'opérateur standard. On peut citer, pour la PGA :

- tenir compte du contexte (identifié par la position, la forme, la taille, l'arité) dans lequel sont prélevés les sous-arbres [29][79] ;
- garantir que la quantité de matériel génétique échangé est identique dans les deux parents [58] ;
- modifier la probabilité de sélection des sous-arbres par auto-adaptation [5] ou bien en fonction de la profondeur [39].

En ce qui concerne la PGL, les propositions ont été relativement similaires :

- tenir compte du contexte (identifié par la position) dans lequel sont prélevées les instructions [71] ;
- garantir que la quantité de matériel génétique échangé est identique

dans les deux parents [41].

La démarche sous-jacente, commune à ces différentes tentatives de définition d'un nouvel opérateur, est de favoriser la transmission aux enfants de certains critères partagés par les parents, autrement dit la conservation de l'homologie parentale. L'homologie est une notion complexe à définir qui rend compte de la similarité entre des structures pourtant différentes. Dans cette étude, nous définissons puis nous étudions les propriétés d'un opérateur de croisement pour la PGL qui conserve l'homologie des séquences d'instructions des programmes.

Croissance de la taille des programmes

Une des problématiques principales en PG est la gestion de la taille des programmes. En effet, la taille des programmes a tendance, au cours de l'évolution, à croître de façon exagérée, c'est-à-dire sans rapport avec les gains en fitness, et cette croissance n'est souvent stoppée que par la limite dure que constitue la taille maximum autorisée. Ce phénomène est appelé *bloat* en anglais, c'est ce terme que nous utiliserons dans cette étude.

Le bloat constitue une réelle faiblesse pour la PG et il rend délicate son utilisation opérationnelle, car les ressources utilisées par le système pour résoudre un problème donné ne sont pas adaptées à sa difficulté, en particulier en ce qui concerne le temps de calcul et la mémoire allouée. De plus, ce comportement peut largement dégrader les performances en terme de fitness des solutions découvertes. En effet, les génotypes obtenus dans un contexte de bloat sont souvent constitués de code inutile, les introns, ce qui diminue les effets des opérateurs génétiques et donc ralentit la recherche. Enfin, le bloat est contradictoire avec le critère de parcimonie, couramment retenu [80], qui veut que si des individus ont une valeur adaptative équivalente, il est préférable de choisir les plus petits qui sont souvent plus génériques.

Plusieurs hypothèses ont été avancées pour expliquer le bloat :

1. L'*hypothèse de protection* [2][9][68] considère que le bloat apparaît au cours de l'évolution quand la population "tente" de protéger les sous-arbres importants, *i.e.* ceux qui contribuent de façon très positive à la

fitness générale, des effets destructeurs des opérateurs génétiques, en particulier de l'opérateur de croisement.

2. Pour l'*hypothèse de dérive* [59][8], c'est la nature de l'espace de recherche qui est en cause. En effet, durant l'évolution, la recherche débute avec des solutions ayant des génotypes courts et possédant une valeur de fitness donnée et celle-ci s'améliore grâce à l'action conjuguée des mécanismes de sélection et variation. Cependant, rapidement les chances de découvrir des meilleures solutions diminuent et l'évolution devient de plus en plus neutre car seuls des individus possédant une valeur de fitness comparable aux individus de la population sont découverts. Comme par ailleurs, les génotypes longs correspondant à une fitness donnée sont, dans l'espace de recherche, beaucoup plus nombreux que les courts, la taille moyenne des programmes dans la population augmente systématiquement.
3. L'*hypothèse du biais* [96][98] met en avant le caractère dissymétrique des effets sur la fitness des opérations de suppression et d'insertion de sous-arbres. Les effets de la suppression d'un sous-arbre dépendent de sa taille : peu d'effets si le sous-arbre est petit alors que pour l'insertion aucune relation comparable n'a pu être constatée. Ainsi, les individus dans lesquels de petits sous-arbres sont supprimés et de grands sous-arbres ajoutés sont favorisés, c'est pourquoi, les programmes dans la population ont tendance à croître. De plus ce biais est proportionnel à la taille des programmes ce qui peut expliquer une croissance exponentielle de la taille.
4. L'*hypothèse de difficulté* [37] suppose que les rapports entre la pression de sélection et la difficulté des problèmes à résoudre puissent influencer la diversité dans la population ainsi que l'évolution de la taille des programmes.
5. L'*hypothèse des variations neutres* [12] prétend que la croissance de la quantité d'introns dans les programmes est directement et quasi exclusivement causée par des variations neutres dans l'espace de recherche.

Toutes ces hypothèses ne sont pas nécessairement contradictoires, certaines

étant spécialisées pour une représentation ou un opérateur particulier. Plus généralement, nous pensons que cette multiplicité d'explications est liée à une différence de point de vue adopté pour étudier le phénomène. Comme souvent en science, c'est la théorie qui unifiera ces hypothèses et en l'occurrence le théorème exact des schémas est un candidat sérieux.

La littérature en PG a produit encore plus de méthodes "anti-bloat" que d'explications au phénomène. Pour atteindre leur objectif, ces méthodes proposent de modifier soit l'implémentation du système PG, soit la fonction fitness et la fonction de sélection, soit les opérateurs génétiques. Dès les origines [54], Koza propose de limiter dans l'implémentation la taille des programmes à l'aide d'une taille maximum autorisée. Cependant, l'idée la plus répandue est d'intervenir directement sur la fitness et donc sur la sélection des individus pour limiter le nombre de programmes trop grands. Ainsi, l'utilisation d'une pression de parcimonie, mécanisme permettant de tenir compte de la taille lors de l'évaluation d'un programme, constante [97] ou adaptative [98], pénalise les individus les plus complexes et limite leur développement dans la population. On peut aussi réduire le bloat en réalisant une sélection en deux temps des individus, avec deux tournois successifs par exemple [65], les premiers basés sur la fitness et les seconds sur la taille. La recherche d'un programme est donc vue comme l'optimisation des deux critères fitness et taille, c'est pourquoi l'idée de recourir à des méthodes multi-objectifs a également été envisagée [25]. Une autre méthode consiste à rendre dynamique la fonction de fitness [103], par exemple en n'évaluant les programmes que sur des sous-parties du problème à résoudre, différentes à chaque générations, on favorise ainsi les programmes les plus génériques qui sont souvent plus courts. Les modifications de la fitness et de la sélection peuvent être encore plus définitives, comme avec la méthode Tarpéienne [78] qui possède de sérieuses justifications théoriques et qui propose d'éliminer, avec une certaine probabilité, les individus ayant une taille supérieure à la moyenne dans la population. Certaines méthodes ont abordé un autre aspect du problème puisqu'elles visent à limiter la brutalité des variations génétiques, en particulier celles liées à l'opérateur de croisement, qui sont susceptibles de favoriser la croissance en taille des programmes. On pourra citer les opérateurs de

croisement qui préservent le contexte [29], les opérateurs *size fair* [58] qui respectent la taille et les opérateurs dits homologues qui préservent la forme [79] ou la position dans les programmes [71] et qui ont tous démontrer une tendance significative dans la réduction du phénomène de bloat.

1.3 Le système PGP

Dans cette étude, nous définirons un nouvel opérateur de croisement pour la PGL et nous aurons à évaluer les performances de programmes en représentation linéaire. Nous avons choisi d'utiliser un système PGP notamment en raison de sa simplicité d'implémentation. Le système PGP utilisé est largement inspiré des travaux de Perkis [77] et ne contient pas d'innovations particulières ; il convient toutefois de préciser les choix techniques que nous avons effectués.

Représentation et Évaluation des programmes

En PGP, les programmes sont codés sous formes de chaînes de tailles variables composées de symboles représentant des instructions du langage. Une chaîne est interprétée comme la notation polonaise inversée d'un programme, ce qui nécessite l'utilisation d'une pile d'opérandes. A chaque instruction PGP correspond un ensemble d'actions prédéfinies sur la pile. Par exemple quand l'instruction `ADD` est reconnue trois actions sont effectuées : deux opérandes sont dépilées puis leur somme est empilée. Une chaîne est donc interprétée de façon itérative et correspond à une séquence d'actions devant être réalisées sur la pile.

Avec une représentation linéaire, on ne peut distinguer directement dans la structure les fonctions des terminaux. Cependant, on peut différencier les instructions représentant des fonctions des instructions représentant des terminaux à partir de leurs actions sur la pile : au moins un dépilement pour les fonctions, pas de dépilement pour les terminaux. Pour une fonction, le nombre d'opérations de dépilement est équivalent à l'arité de cette fonction, c'est-à-dire son nombre d'arguments.

En PGP, toutes les séquences d'instructions sont des programmes valides

et aucune contrainte syntaxique n'est imposée. En effet, si une fonction est détectée alors que la pile d'opérande ne contient pas suffisamment d'éléments pour que celle-ci puisse être exécutée, l'évaluation de la chaîne reprend à l'instruction suivante, *i.e.* la fonction est ignorée. A titre d'exemple, la Figure 1.4 représente, en six étapes, l'évaluation du programme `1 ADD 5 2 SUB`.

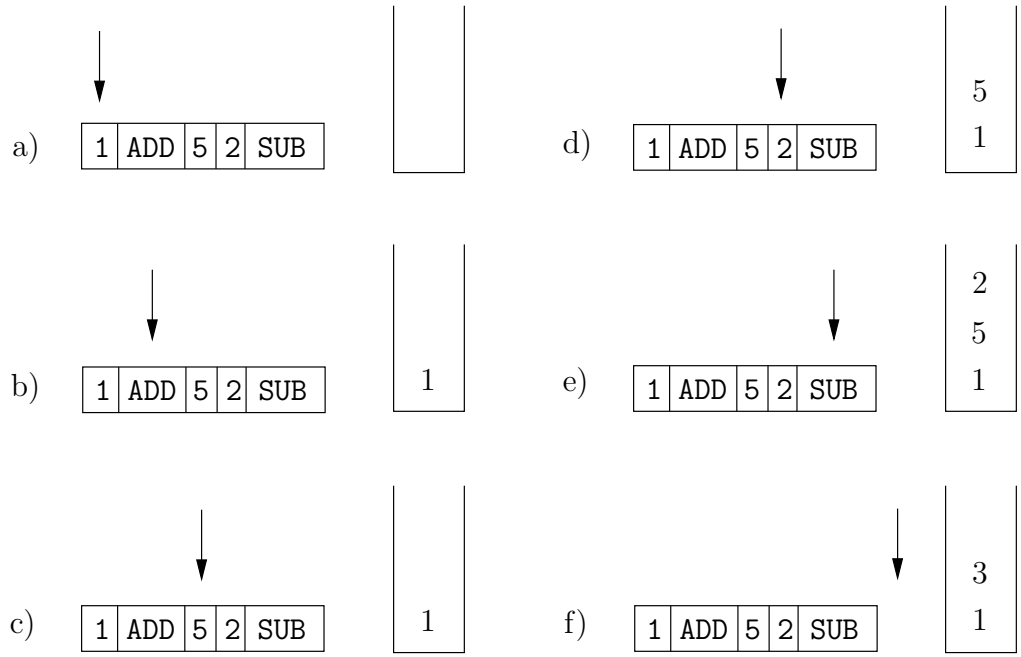

FIG. 1.4 – Évaluation d'un programme en PGP.

La première étape de l'évaluation, notée *a* sur la figure, correspond à une phase préparatoire durant laquelle la pile est vidée et un pointeur est positionné sur la première instruction à interpréter. A la fin de chaque étape, l'instruction suivante sera pointée. Durant les étapes *b*, *d* et *e*, des constantes numériques, respectivement 1, 5 et 2 sont empilées. Durant l'étape *f*, la fonction `SUB` est reconnue et puisque la pile contient suffisamment d'éléments (au moins deux), la fonction est exécutée et le résultat $(3 = 5 - 2)$ empilé. Par contre, durant l'étape *c*, la pile ne contient qu'une valeur alors que la fonction `ADD` en requiert deux. Comme nous l'avons déjà précisé, dans ce cas l'instruction ne peut être exécutée et la pile est laissée inchangée.

D'un point de vue pratique, les problèmes traités en PG nécessitent sou-

vent de multiples interprétations des programmes pour le calcul de la fitness. Dans ce cas, la présence des instructions inutiles décrites précédemment pourrait devenir très coûteuse en temps de calcul, c'est pourquoi une phase dite de compilation permet de les identifier et de les ignorer durant chaque calcul de fitness. Dans cette étude, on parlera de *taille effective* pour désigner la taille des programmes après la phase de compilation.

Mutation et Croisement des programmes

Nous avons vu que dans le système PGP, toutes les séquences d'instructions constituent des programmes valides, c'est pourquoi il existe une grande liberté dans la conception des opérateurs génétiques.

L'opérateur de mutation de notre système réalise trois types de variations différentes :

- des insertions d'instructions, où une instruction du langage choisie aléatoirement est ajoutée à une position choisie aléatoirement dans le programme ;
- des délétions d'instructions, où une instruction du programme choisie aléatoirement est supprimée ;
- des substitutions d'instructions, où une instruction du programme choisie aléatoirement est remplacée par une instruction du langage choisie aléatoirement.

Seules les insertions et délétions d'instructions sont des opérations génétiques élémentaires car la substitution peut être vue comme une composée des deux, néanmoins l'expérience a montré qu'il est préférable que le système dispose de cette opération.

On notera qu'un paramètre spécifique permet, indépendemment pour chaque type de mutation (insertion, délétion et substitution), de contrôler son taux d'application durant une expérience. Cependant, on utilisera généralement un réglage équivalent pour les trois types de mutation qui pourra être contrôlé par un unique paramètre noté $\mathcal{T}_m$. Une valeur de $\mathcal{T}_m$ de 1.0 signifie que les programmes sélectionnés subiront, en moyenne, une mutation de chaque type.

L'opérateur de croisement utilisé de façon standard en PGP réalise un

échange de suffixes quelconques entre deux chaînes. La taille de ces suffixes est la plupart du temps différente dans les deux chaînes. Le taux d'application de cet opérateur est contrôlé par un paramètre appelé $\mathcal{T}_c$. Une valeur de $\mathcal{T}_c$ de 1.0 signifie que l'opérateur de croisement sera appliqué sur tous les programmes sélectionnés. Un exemple de croisement produisant deux enfants est présenté Figure 1.5. Les deux parents sont les chaînes `1 ADD 5 2 SUB` et `X LOG 4 MUL`. La première étape du croisement consiste à identifier un suffixe (et donc un préfixe) dans chacun des parents, dans notre cas `5 2 SUB` et `4 MUL`. Durant la deuxième étape, les deux suffixes sont échangés pour obtenir deux nouveaux programmes `1 ADD 4 MUL` et `X LOG 5 2 SUB`.

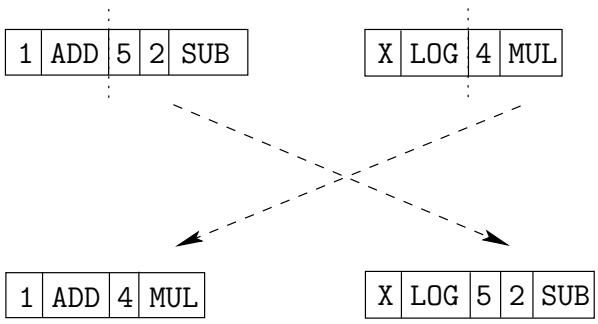

FIG. 1.5 – Exemple de croisement en PGP

L'utilisation d'un paramètre contrôlant la taille maximum autorisée est courante en PG [54], c'est également le cas avec PGP où nous l'appelons λ_{max}. Il peut arriver que l'opérateur de croisement, et plus rarement l'opérateur de mutation, génère des programmes dont la taille est supérieure à λ_{max}. En ce qui concerne la mutation, le choix le plus naturel est d'interdire l'insertion d'instruction dans les programmes de taille λ_{max} (ainsi que la suppression d'instruction dans des programmes de taille 1). Les choses sont plus complexes en ce qui concerne le croisement où au moins trois possibilités existent :

1. n'insérer dans la population que les programmes ayant une taille valide ;
2. s'assurer que les tailles des suffixes échangés permettent la création de deux programmes ayant une taille valide ;

3. supprimer les instructions excédentaires des programmes trop longs.

Avec la première solution, le nombre de programmes recombinés insérés dans la population à chaque génération diminue quand les tailles des programmes deviennent trop grandes et la valeur du paramètre $\mathcal{T}_c$ n'est donc pas toujours respectée. La deuxième possibilité garantit que la valeur de $\mathcal{T}_c$ est toujours respectée mais un biais est introduit dans le choix des suffixes. Prenons l'exemple d'une recombinaison impliquant deux programmes de taille λ_{max}. Dans ce cas, l'opérateur de croisement se comporte comme l'opérateur utilisé dans les AG avec des chaînes binaires de taille fixe et n'assure plus la redistribution des instructions entre les différentes positions des génotypes. Enfin, avec la troisième possibilité, les deux défauts décrits précédemment sont évités mais la recombinaison est certainement plus brutale. Dans cette étude, nous choisissons d'utiliser cette dernière méthode et nous supprimons les instructions excédant λ_{max}.

Chapitre 2

Croisement par Maximum d'Homologie

De nombreuses critiques ont été formulées à l'encontre de l'opérateur de croisement standard (CS) utilisé en PG, que ce soit sa version en PGA, où il consiste à échanger des sous-arbres entre les parents ou bien en PGL, où ce sont des séquences d'instructions de tailles variables qui sont échangées. Ces critiques ont été détaillées dans le chapitre précédent (cf Section 1.2.3). Nous rappellerons qu'entre autre, une forme de brutalité de la recombinaison a été mise en évidence [2], en particulier ses effets délétères sur la fitness, et que son comportement biaisé vis-à-vis de la taille des programmes a été clairement démontré [79]. Nous avons déjà cité quelques tentatives intéressantes pour l'amélioration des performances de la recombinaison en PG et la réduction de la brutalité, qui, soit implicitement soit explicitement, consistent à préserver l'*homologie* durant une opération de croisement, voir par exemple [29][58][71][79].

Dans ce chapitre, nous nous intéressons à la définition du concept d'homologie, en particulier à ses origines en biologie. Puis, suivant [103] où la nécessité de définir des opérateurs de croisement "plus homologues" avait déjà été évoquée, nous introduirons le Croisement par Maximum d'Homologie pour la PGL. Largement inspiré par la recombinaison homologue de l'ADN, cet opérateur garantit qu'une des plus longues sous-séquences d'instructions communes aux parents sera conservée et transmise à leur descendants. Nous utiliserons le concept d'ensemble de recombinaison pour démontrer certaines

propriétés théoriques du nouvel opérateur et, en dehors de toute notion de performance, nous étudierons son comportement dynamique et son influence sur la taille des programmes recombinés. Enfin, nous montrerons ses effets sur le choix du site de croisement et comment la recherche est orientée vers les régions les moins homologues des génotypes recombinés.

2.1 Homologie

2.1.1 Le concept biologique

Le concept d'homologie a été introduit en biologie et, plus précisément, en anatomie comparée, par R. Owen qui, en 1843, caractérise pour la première fois un organe homologue comme "le même organe quelque soit sa forme et sa fonction" (traduit de l'anglais) [75]. C'est cette notion de "même organe" qui dès les origines pose problème car elle est particulièrement complexe à définir. Pour Owen, seules la structure et la localisation comptent, c'est-à-dire que l'homologie correspond à la similarité structurelle entre des organismes pourtant différents. Il identifie tout particulièrement des organes homologues d'après les relations qu'ils entretiennent avec les organes voisins et il pense que cette homologie de structure indique que des organismes distincts ont été conçus de façon indépendante mais selon un schéma commun, un archétype.

La controverse, à l'époque, porte sur l'existence, dans ces archétypes, d'organes homologues n'ayant aucune fonction, les structures vestigiales, comme les ailes inutiles d'une autruche par exemple. C. Darwin apporte une réponse évolutionniste à cette question :

> "Une structure est similaire parmi certains organismes parce que ceux-ci descendent tous d'un ancêtre commun possédant lui-même cette structure." (traduit de l'anglais)

Ce qui permet de définir l'homologie comme la relation entre deux caractères, comme par exemple des organes, des cellules ou des tissus, qui sont hérités, souvent avec des divergences, d'un ancêtre commun. Cette définition de l'homologie, par filiation, ne prend tout son sens que dans un cadre taxonomique donné. Sans un cadre taxonomique de référence, dans la mesure où l'arbre

de la vie est unique, tout serait finalement homologue à tout.

Alors que l'homologie de structure est la première que l'on détecte et qu'elle constitue une sorte de pari initial, l'homologie de filiation est validée par une analyse phylogénétique. La phylogénie permet de reconstituer les liens de parenté des organismes actuels et fossiles en les comparant les uns avec les autres. Cette comparaison ne permet pas de reconstituer des généalogies (qui descend de qui ?) mais des phylogénies (qui est plus proche de qui ?). Toutes les homologies de structure réfutées par l'analyse phylogénétique sont des homoplasies [61], c'est-à-dire des ressemblances non héritées d'un ancêtre commun. On distingue également l'homologie, processus divergeant, de l'analogie, processus convergeant, que l'on peut définir comme la relation entre deux caractères hérités d'ancêtres différents mais qui ont convergé.

En biologie moléculaire, quand deux caractères, deux séquences d'ADN par exemple, sont significativement similaires, on considère qu'elles sont homologues car la probabilité qu'elles aient été obtenues par convergence est beaucoup trop faible. De plus, l'homologie de séquences d'ADN implique, dans la majorité des cas, des similarités dans les structures protéiques codées par ces séquences mais également dans leurs fonctions, et donc des similarités depuis les cellules, les tissus, jusqu'aux organes correspondants. C'est pourquoi, une des définitions moderne de l'homologie est simplement la similarité du génome qui implicitement implique une relation de filiation. En outre, cette définition présente un intérêt opérationnel décisif : l'homologie de séquence est mesurable.

La recherche de similitude entre séquences permet de révéler des régions homologues en se basant sur le principe de parcimonie, c'est-à-dire en considérant le minimum de changements en insertion, suppression, ou substitution qui séparent deux séquences. On peut apprendre ainsi, par association, des informations importantes sur la structure, la fonction ou l'évolution des biomolécules. Pour qualifier et quantifier la similitude entre séquences, un score est calculé, comme par exemple la *distance d'édition* qui sera définie plus loin dans ce chapitre. On utilisera dans ce mémoire, alternativement et de façon équivalente, les notions de similitude de séquences, d'homologie ou de

distance d'édition[1].

2.1.2 Recombinaison homologue de l'ADN

L'ADN est à la base de tous les processus et de toutes les caractéristiques des êtres vivants. C'est dans la structure de cette molécule que réside deux aspects majeurs de la vie, la réplication et la genèse de la forme. Le processus de réplication de l'ADN permet d'effectuer des copies des cellules ainsi que des organismes et de les perpétuer au cours du temps, c'est donc le fil qui nous rattache à nos ancêtres à travers l'évolution. Cependant cette copie n'est pas parfaite, ce qui constitue une des causes de l'immense diversité génétique que nous connaissons.

Si la copie n'est pas réalisée à l'identique, cela est en partie du à la diploïdie, c'est-à-dire au fait que certaines espèces possèdent deux jeux de chromosomes dits homologues que l'on peut associer par paires. Chez ces espèces, deux parents sont impliqués dans le processus de reproduction et ils contribuent chacun pour moitié au matériel génétique de leurs descendants en transmettant un seul jeu de chromosomes, c'est-à-dire un représentant de chaque paire. Ce sont les cellules sexuelles des parents, les gamètes, qui en fusionnant reforment le génome complet d'un nouvel individu. Les gamètes sont obtenues par des divisions cellulaires spécifiques, les méioses, durant lesquelles l'ADN est répliqué d'une façon particulière. Ces cellules sont toutes haploïdes, c'est-à-dire possédant un seul jeu de chromosomes, et génétiquement différentes car leur génome est issu d'un brassage aléatoire, avec un représentant de chaque paire, des chromosomes parentaux. De plus, grâce à un processus appelé recombinaison homologue de l'ADN, les chromosomes contenus dans les gamètes sont constitués d'une redistribution de l'ADN des chromosomes homologues parentaux. Notons que cette opération n'est pas complètement laissée au hasard car ce processus doit garantir que les chromosomes obtenus soient valides, en particulier qu'ils puissent permettre la formation d'un nouvel individu après fusion des gamètes. Ainsi, les enfants issus de mêmes parents sont toujours uniques (à l'exception de la gémellité)

[1]En fait, nous verrons qu'une très faible distance d'édition correspond à une très forte homologie.

ce qui permet d'assurer une importante diversité génétique intra spécifique.

La méiose est un mécanisme complexe que les biologistes ont divisé en plusieurs phases et qui, à partir d'une cellule diploïde produit quatre gamètes toutes différentes. La recombinaison homologue de l'ADN a lieu durant le deuxième stade, appelé *zygotène*, de la première prophase de la méiose. Durant cette étape les chromosomes entrent tout d'abord en *synapsis*, ce qui signifie qu'ils s'assemblent par paires d'homologues dans la cellule par un lent processus d'alignement, encore mal connu, qui débute à une de leurs extrémités et qui rappelle le glissement d'une fermeture-éclair. Puis, des échanges de matériel génétique se font, par la rupture et la réunion de fragments d'ADN ; ces évènements sont appelés crossing-over en anglais, ce que l'on peut traduire par enjambement ou croisement. Le crossing-over réalise ainsi une redistribution des gènes dans les chromosomes. C'est le processus d'alignement qui garantit que les gènes ou les groupes de gènes échangés sont similaires et donc qu'ils codent pour la même fonction. Bien que l'alignement tende à rapprocher les régions les plus homologues des chromosomes, il n'est pas toujours parfaitement réalisé et il arrive que le crossing-over produise ainsi un gène nouveau, potentiellement défectueux. Par la suite, ces nouveaux chromosomes sont répartis, par ségrégation, dans des cellules filles qui constitueront les quatre gamètes produits de la division cellulaire méiotique. Il semble que les crossing-over jouent un rôle déterminant dans la disjonction des chromosomes homologues appariés et que la formation d'au moins un crossing-over par paire soit nécessaire pour que la ségrégation se déroule correctement.

2.2 Définition d'un nouvel opérateur

Dans cette section, nous introduisons un nouvel opérateur de recombinaison pour la PGL, le Croisement par Maximum d'Homologie (CMH). Le CMH est un opérateur dont le fonctionnement est inspiré de la recombinaison homologue de l'ADN. Les programmes devant être recombinés sont tout d'abord alignés, de façon à ce qu'un appariement de leurs instructions soit réalisé en fonction de leur homologie, puis la recombinaison proprement dite a lieu. Pour mesurer l'homologie pendant le processus d'alignement du CMH,

la distance d'édition est utilisée.

2.2.1 Distance d'édition

La distance d'édition est une mesure de similarité (ou de dis-similarité) entre chaînes de taille variable. Pour effectuer un CMH, nous utilisons une distance d'édition comparable à la *distance de Levenshtein*, issue de travaux en théorie de l'information [62]. La distance de Levenshtein a été utilisée à plusieurs reprises en PG, par exemple pour calculer ou contrôler la diversité génotypique [13] ou bien pour étudier l'influence de différents opérateurs génétiques [74]. Par définition, la distance d'édition entre deux programmes correspond au nombre minimal d'opérations élémentaires (insertion, délétion ou substitution) nécessaires pour transformer un programme en un autre.

Plus formellement, on définit x un programme de taille m comme une séquence d'instructions telle que :

$$x = x_0 x_1 \dots x_{m-1}$$

avec $\forall i \in [0, m-1]\ x_i \in \Sigma$, où Σ est un alphabet composé de N symboles correspondant à l'ensemble fini des N instructions disponibles. On appelle P l'ensemble infini des programmes définis sur Σ. Soient x et y deux programmes de tailles respectives m et n. Soit ε une instruction supplémentaire ne réalisant aucune fonction. Alors un alignement $(\overline{x}, \overline{y})$ de longueur p avec $\overline{x}$ et $\overline{y} \in \overline{P}$ et $\overline{P}$ l'ensemble infini des séquences définies sur $\Sigma \cup \{\varepsilon\}$, peut s'écrire comme suit :

$$(\overline{x}, \overline{y}) = \begin{pmatrix} \overline{x}_0 & \overline{x}_1 \dots & \overline{x}_{p-1} \\ \overline{y}_0 & \overline{y}_1 \dots & \overline{y}_{p-1} \end{pmatrix}$$

où :

- $p \in [max(m, n), m + n]$;
- $\overline{x}_i = x_j$ ou $\overline{x}_i = \varepsilon$ pour $i \in [0, p-1]$ et $j \in [0, m-1]$;
- $\overline{y}_i = y_j$ ou $\overline{y}_i = \varepsilon$ pour $i \in [0, p-1]$ et $j \in [0, n-1]$;
- $\nexists\ i \in [0, p-1]$ tel que $\overline{x}_i = \overline{y}_i = \varepsilon$;
- la suppression des instructions ε dans $\overline{x}$ donne toujours x (de même pour $\overline{y}$ et y).

Dans un alignement, un appariement d'instructions $\binom{\overline{x}_i}{\overline{y}_i}$ peut indiquer soit une substitution de x_j par y_j, soit une délétion de x_j (si $\overline{y}_i = \varepsilon$), soit une insertion de y_j (si $\overline{x}_i = \varepsilon$). On remarquera qu'un alignement $(\overline{x}, \overline{y})$ peut également être vu comme une séquence d'opérations (insertion, délétion ou substitution) qui permettent de transformer x en y.

On définit le coût χ d'alignement par :

$$\chi(\overline{x}, \overline{y}) = \sum_{i=0}^{p-1} \mathcal{C}(\overline{x}_i, \overline{y}_i)$$

avec

$$\mathcal{C}(a,b) = \begin{cases} C_1 \text{ (insertion ou délétion)} & \text{si} & a = \varepsilon \text{ ou bien } b = \varepsilon \\ C_2 \text{ (substitution)} & \text{sinon si} & a \neq b \\ 0 & \text{sinon} & \end{cases}$$

Soit $A(x, y)$ l'ensemble de tous les alignements de x et y, alors la distance d'édition entre x and y est :

$$\mathcal{D}(x, y) = min\big\{\chi(\overline{x}, \overline{y}) | (\overline{x}, \overline{y}) \in A(x, y)\big\}$$

La distance d'édition est connue pour être une véritable distance au sens classique, elle vérifie donc $\forall x, y, z \in P$:

$$\mathcal{D}(x, y) = 0 \quad \Leftrightarrow \quad x = y \tag{2.1}$$

$$\mathcal{D}(x, y) > 0 \quad \Leftrightarrow \quad x \neq y \tag{2.2}$$

$$\mathcal{D}(x, y) \quad = \quad \mathcal{D}(y, x) \tag{2.3}$$

$$\mathcal{D}(x, z) + \mathcal{D}(z, y) \quad \geq \quad \mathcal{D}(x, y) \tag{2.4}$$

En remarquant qu'une substitution est toujours équivalente à un couple (insertion, délétion), on peut définir différents types de distance d'édition grâce aux coûts des opérations élémentaires, C_1 (insertion ou délétion) et C_2 (substitution) :

– avec C_1=C_2=1, où le coût d'une substitution est inférieur au coût du couple (insertion, délétion). Ce réglage est celui qui est utilisé pour le calcul de la *distance de Levenshtein*. Par exemple, l'alignement $\binom{a}{b}$ est moins coûteux que $\left(\begin{smallmatrix} a & \varepsilon \\ \varepsilon & b \end{smallmatrix}\right)$.

- avec C_1=1 et C_2=2, où le coût d'une substitution est équivalent au coût du couple d'une insertion et d'une délétion. Par exemple, le coût de alignement $\binom{a}{b}$ est identique à celui de $\left(\begin{smallmatrix} a & \varepsilon \\ \varepsilon & b \end{smallmatrix}\right)$.
- avec C_1=1 et C_2=3, où les substitutions sont plus coûteuses que les couples d'insertions et de délétions. Par exemple, le coût de alignement $\binom{a}{b}$ est supérieur à celui de $\left(\begin{smallmatrix} a & \varepsilon \\ \varepsilon & b \end{smallmatrix}\right)$. C'est le réglage qui sera utilisé dans cette étude, ainsi la distance d'édition sera basée sur des alignements ne contenant aucun mésappariement.

Comme nous l'avons vu précédemment, Section 2.1.1, la distance d'édition $\mathcal{D}(x, y)$ représente également l'homologie entre x et y, on utilisera donc indifféremment les termes homologie et distance pour refléter la similarité entre deux programmes[2].

Cardinalité de l'alphabet

La longueur des programmes ainsi que la taille N de l'alphabet utilisé pour les définir, sont les principales variables influant sur l'homologie. Figure 2.1, nous avons représenté la distance $\mathcal{D}(x, y)$ moyenne en fonction de N, entre des programmes x et y choisis aléatoirement mais dont la taille est identique, variant de 10 à 40 instructions. On constate qu'évidemment la distance moyenne est nulle pour N=1 et qu'elle converge asymptotiquement avec N, vers le double de la taille des programmes (20, 40 , 60 et 80 instructions) ; le cas limite correspond à deux programmes n'ayant aucune instruction en commun et donc pour transformer un programme en un autre, il faut supprimer toutes les instructions de l'un puis insérer toutes les instructions de l'autre.

2.2.2 Meilleurs Alignements

Un alignement $(\overline{x}, \overline{y})$ est appelé meilleur alignement de x et y si et seulement si $\chi(\overline{x}, \overline{y}) = \mathcal{D}(x, y)$. Les meilleurs alignements dépendent de la définition de $\mathcal{D}$ et donc du choix de C_1 et C_2. La Figure 2.2 montre un exemple de meilleur alignement entre x =`CATAGA` et y =`CCAACTGAC`. Chaque colonne de

[2]En fait, une très faible distance d'édition correspond à une très forte homologie.

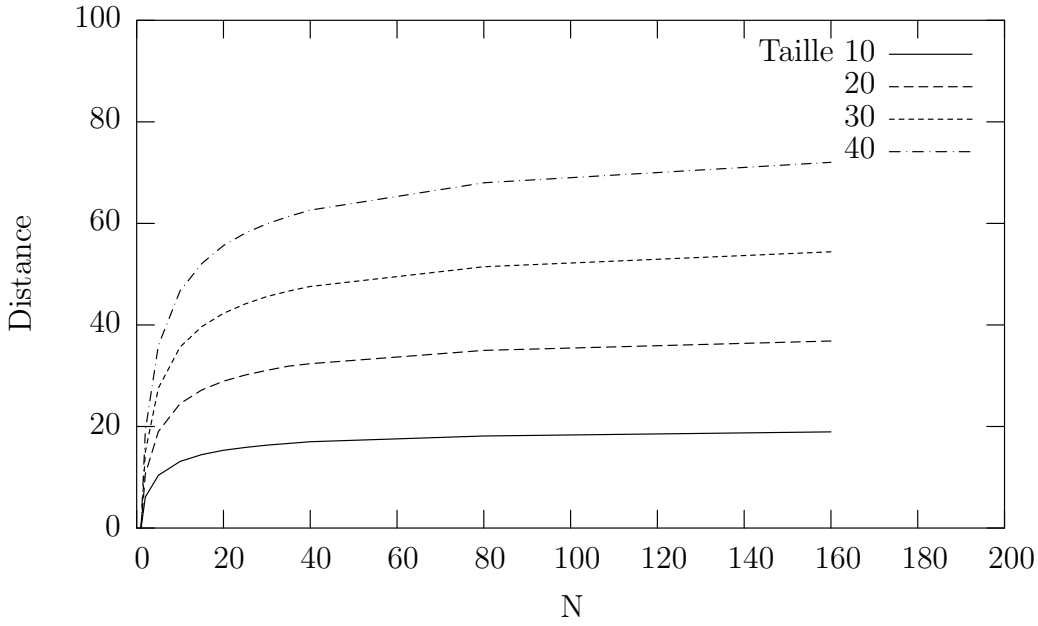

FIG. 2.1 – Distance d'édition en fonction de la taille de l'alphabet pour différentes tailles de programmes

l'alignement $(\overline{x}, \overline{y})$ correspond à un des deux programmes : plus précisément, à sa séquence d'instructions où des instructions ε ont été éventuellement insérées. On peut voir que la distance entre x et y est 5, *i.e.* 5 opérations sont nécessaires pour transformer x en y (4 insertions, 1 délétion).

Calcul de l'ensemble des meilleurs alignements

On note $A^*(x, y)$ l'ensemble des meilleurs alignements de x et y. Le calcul de $A^*(x, y)$ peut être effectué en temps $O(nm)$ par programmation dynamique [36]. Cette méthode est classiquement utilisée en Bio-Informatique pour aligner des séquences d'ADN.

On note $\emptyset$ la chaîne de taille nulle. Soit $p(x, i)$, un préfixe de x tel que $p(x, 0) = \emptyset$ et $\forall i \in [1, m], p(x, i) = x_0 x_1 \dots x_{i-1}$. La formule de récursivité

Opération	$(\overline{x}, \overline{y})$
	C C
Insertion	ε C
	A A
Insertion	ε A
Insertion	ε C
	T T
Délétion	A ε
	G G
	A A
Insertion	ε C

FIG. 2.2 – Exemple de meilleur alignement

suivante[3] :

$$\begin{aligned}
\mathcal{D}\big(p(x,0),p(y,0)\big) &= 0 \\
\mathcal{D}\big(p(x,i),p(y,j)\big) &= min \left\{ \begin{array}{lcl} \mathcal{D}\big(p(x,i-1),p(y,j)\big) & + & C_1 \\ \mathcal{D}\big(p(x,i),p(y,j-1)\big) & + & C_1 \\ \mathcal{D}\big(p(x,i-1),p(y,j-1)\big) & + & \mathcal{C}(x_i,y_j) \end{array} \right\}
\end{aligned}$$

permet de construire la matrice d'alignement T telle que :

$$\forall i \in [0,m], j \in [0,n] \quad T[i,j] = \mathcal{D}\big(p(x,i),p(y,j)\big)$$

Nous avons vu qu'un alignement peut être vu comme une séquence de transformations élémentaires. Se déplacer d'une cellule dans la matrice T revient à effectuer une des transformations élémentaires :

- un déplacement horizontal correspond à une insertion : $\binom{\varepsilon}{y_j}$
- un déplacement vertical correspond à une délétion : $\binom{x_i}{\varepsilon}$
- un déplacement diagonal correspond à une substitution : $\binom{x_i}{y_j}$

Pour trouver $A^*(x,y)$ il suffit de trouver toutes les séquences de transformations de $T[0,0]$ à $T[m,n]$ qui donne $T[m,n] = \mathcal{D}(x,y)$, ce qui peut être fait par un appel à la Procédure 1 avec ALL(x, $m-1$, y, $n-1$, $\binom{\emptyset}{\emptyset}$, T).

La Figure 2.3 représente la matrice d'alignement T entre x =CATAGA et y =CCAACTGAC. Les flèches montrent quelles sont les transformations qui

[3]On rappelle que dans cette étude C_1=1 et C_2=3.

Algorithm 1 ALL(x, i, y, j, z, T)

if $i \neq -1$ **then**
  if $j \neq -1$ **then**
    if $T[i+1, j+1] = T[i, j] + \mathcal{C}(x_i, y_j)$ **then**
      ALL(x, $i-1$, y, $j-1$, $\binom{x_i}{y_j} \cdot z$, T)
    end if
    if $T[i+1, j+1] = T[i, j+1] + C_1$ **then**
      ALL(x, $i-1$, y, j, $\binom{x_i}{\varepsilon} \cdot z$, T)
    end if
    if $T[i+1, j+1] = T[i+1, j] + C_1$ **then**
      ALL(x, i, y, $j-1$, $\binom{\varepsilon}{y_i} \cdot z$, T)
    end if
  else
    ALL(x, $i-1$, y, -1, $\binom{x_i}{\varepsilon} \cdot z$, T)
  end if
else if $j \neq -1$ **then**
  ALL(x, -1, y, $j-1$, $\binom{\varepsilon}{y_i} \cdot z$, T)
else
  Afficher(z)
end if

			C	C	A	A	C	T	G	A	C
	-1	0	1	2	3	4	5	6	7	8	9
	0	0	1	2	3	4	5	6	7	8	9
C	1	1	0	1	2	3	4	5	6	7	8
A	2	2	1	3	1	2	3	4	5	6	7
T	3	3	2	3	2	3	4	3	4	5	6
A	4	4	3	4	3	2	3	4	5	4	5
G	5	5	4	5	4	3	4	5	4	5	6
A	6	6	5	6	5	4	5	6	5	4	5

FIG. 2.3 – Exemple de matrice d'alignement

permettent d'obtenir $T[m,n] = 5$ et définissent 4 meilleurs alignements :

1. $(\overline{x},\overline{y}) = \begin{pmatrix} \mathtt{C} & \varepsilon & \mathtt{A} & \varepsilon & \varepsilon & \mathtt{T} & \mathtt{A} & \mathtt{G} & \mathtt{A} & \varepsilon \\ \mathtt{C} & \mathtt{C} & \mathtt{A} & \mathtt{A} & \mathtt{C} & \mathtt{T} & \varepsilon & \mathtt{G} & \mathtt{A} & \mathtt{C} \end{pmatrix}$
2. $(\overline{x},\overline{y}) = \begin{pmatrix} \varepsilon & \mathtt{C} & \mathtt{A} & \varepsilon & \varepsilon & \mathtt{T} & \mathtt{A} & \mathtt{G} & \mathtt{A} & \varepsilon \\ \mathtt{C} & \mathtt{C} & \mathtt{A} & \mathtt{A} & \mathtt{C} & \mathtt{T} & \varepsilon & \mathtt{G} & \mathtt{A} & \mathtt{C} \end{pmatrix}$
3. $(\overline{x},\overline{y}) = \begin{pmatrix} \mathtt{C} & \varepsilon & \varepsilon & \mathtt{A} & \varepsilon & \mathtt{T} & \mathtt{A} & \mathtt{G} & \mathtt{A} & \varepsilon \\ \mathtt{C} & \mathtt{C} & \mathtt{A} & \mathtt{A} & \mathtt{C} & \mathtt{T} & \varepsilon & \mathtt{G} & \mathtt{A} & \mathtt{C} \end{pmatrix}$
4. $(\overline{x},\overline{y}) = \begin{pmatrix} \varepsilon & \mathtt{C} & \varepsilon & \mathtt{A} & \varepsilon & \mathtt{T} & \mathtt{A} & \mathtt{G} & \mathtt{A} & \varepsilon \\ \mathtt{C} & \mathtt{C} & \mathtt{A} & \mathtt{A} & \mathtt{C} & \mathtt{T} & \varepsilon & \mathtt{G} & \mathtt{A} & \mathtt{C} \end{pmatrix}$

Remarque Soit $s(x,i)$, un suffixe de x tel que $s(x,m) = \emptyset$ et $\forall i \in [0, m-1], s(x,i) = x_i \dots x_{m-1}$. On peut réécrire x, $\forall i \in [0,m]$, comme la concaténation d'un préfixe et d'un suffixe de x, notée $p(x,i) \centerdot s(x,i)$. Pour tous les couples (i,j) tels que $T[i,j]$ appartient à un meilleur alignement de x et y, on a :

$$\begin{aligned} \mathcal{D}(x,y) &= \mathcal{D}\big(p(x,i) \centerdot s(x,i), p(y,j) \centerdot s(y,j)\big) \\ &= \mathcal{D}\big(p(x,i), p(y,j)\big) + \mathcal{D}\big(s(x,i), s(y,j)\big) \end{aligned} \tag{2.5}$$

Longueur d'un meilleur alignement

Soit $(\overline{x},\overline{y}) \in A^*(x,y)$, un meilleur alignement de longueur p entre deux programmes x et y de longueur respective m et n. On a :

$$p = \tfrac{1}{2}\big(n + m + \mathcal{D}(x,y)\big) \tag{2.6}$$

Preuve On note $\#()$, la fonction qui donne le nombre d'ε contenus dans une séquence d'instructions quelconque. Nous avons défini $\mathcal{D}(x,y)$ comme le nombre d'insertions et de délétions nécessaires pour transformer x en y, on a donc :

$$\mathcal{D}(x,y) = \#(\overline{x}) + \#(\overline{y})$$

La longueur p d'un meilleur alignement correspond au nombre d'instructions de $\overline{x}$ et de $\overline{y}$, c'est-à-dire que :

$$\begin{aligned} p &= n + \#(\overline{x}) \\ &= m + \#(\overline{y}) \end{aligned}$$

d'où

$$\begin{aligned} p &= \tfrac{1}{2}\big(n + \#(\overline{x}) + m + \#(\overline{y})\big) \\ &= \tfrac{1}{2}\big(n + m + \mathcal{D}(x,y)\big) \end{aligned}$$

Cardinalité de l'ensemble des meilleurs alignements

On note $||A^*(x,y)||$, le nombre de meilleurs alignements de x et y. Sachant m, n et $\mathcal{D}(x,y)$, on ne connaît pas *a priori*, sans faire le calcul de $A^*(x,y)$, la valeur de $||A^*(x,y)||$ mais on peut en déterminer une borne supérieure.

En effet, dans le cas limite où toutes les instructions de x et y diffèrent, c'est-à-dire que $p = \mathcal{D}(x,y) = m+n$, il n'y a pas de contraintes d'appariement entre les instructions de $\overline{x}$ et $\overline{y}$ et toutes les combinaisons des instructions de x avec $p-m$ instructions ε forment un $\overline{x}$ valide. De plus, on remarque que dans ce cas, à un $\overline{x}$ donné correspond toujours un $\overline{y}$ et un seul. C'est pourquoi, trouver $||A^*(x,y)||$ quand toute les instructions de x et y sont différentes revient, de façons équivalentes, à :

- soit placer toutes les instructions de x dans $\overline{x}$: C_p^m
- soit placer toutes les instructions de y dans $\overline{y}$: C_p^n
- soit placer tous les ε dans $\overline{x}$: C_p^{p-m}
- soit placer tous les ε dans $\overline{y}$: C_p^{p-n}

Dans l'autre cas limite, où toutes les instructions sont identiques, c'est-à-dire x=y, il n'existe qu'un seul meilleur alignement possible.

Pour le cas général, où $1 \leq \mathcal{D}(x,y) \leq m+n$, on aura :

$$1 \leq ||A^*(x,y)|| \leq max(\mathrm{C}_p^m, \mathrm{C}_p^n) \tag{2.7}$$

2.2.3 Recombinaison

Pour réaliser un CMH entre deux programmes x et y, un alignement $(\overline{x}, \overline{y})$ est choisi aléatoirement dans A^*. Une fois le choix d'un meilleur alignement effectué, la recombinaison proprement dite peut avoir lieu. Puisque $\overline{x}$ et $\overline{y}$ sont de même longueur, les opérateurs de croisement classiquement utilisés pour recombiner des structures de tailles fixes, comme dans le cas des AG, peuvent être appliqués. On notera $X_i(\overline{x}, \overline{y})$ le croisement 1-point classique en position i d'un alignement. Des variantes, 1-point, 2-points ou uniforme, du

CMH peuvent être définies, cependant nous nous limiterons dans cette étude au croisement 1-point, en particulier on notera CMH la version 1-point de l'opérateur. Enfin, pour obtenir des programmes valides, les instructions ε sont supprimées de l'alignement, on notera $\varphi(\overline{x}, \overline{y}) = (x, y)$ cette opération.

Dans la Figure 2.4, un exemple de recombinaison entre x et y par CMH est présenté en trois étapes :

1. calcul d'un meilleur alignement $(\overline{x}, \overline{y}) \in A^*(x, y)$ et choix du site de croisement (ici en position 4) ;
2. échange des séquences d'instructions comprises entre le site de croisement et la fin de l'alignement, c'est-à-dire création de $(\overline{x}', \overline{y}') = X_4(\overline{x}, \overline{y})$;
3. suppression des instructions ε et obtention des enfants $(x', y') = \varphi(\overline{x}', \overline{y}')$.

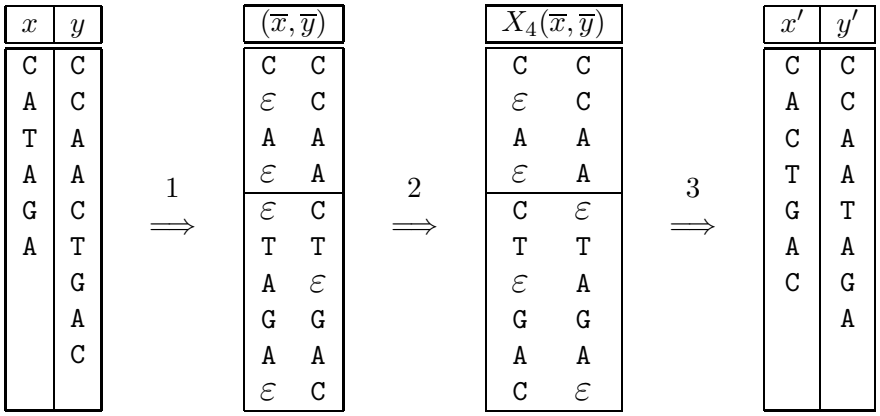

FIG. 2.4 – Exemple de CMH entre x et y

2.3 Propriétés

Dans cette section, nous nous intéressons aux propriétés théoriques du CMH qui peuvent être mises en évidence à partir de sa définition.

2.3.1 Conservation de l'homologie

Pour définir l'homologie entre deux programmes, nous avons choisi d'utiliser la distance d'édition qui est une mesure de similarité entre des séquences

de symboles de taille variable. On peut montrer, qu'ainsi définie, l'homologie est conservée par la recombinaison CMH, c'est-à-dire que les enfants obtenus avec l'opérateur CMH sont au moins aussi homologues entre eux que le sont leurs parents.

On appelle *sous-séquence* d'un programme x la séquence d'instructions pouvant être obtenue en supprimant un certain nombre d'instructions à x (éventuellement aucune). Plus formellement, $w = w_0 w_1 \dots w_{i-1}$ est une sous-séquence de $x = x_0 x_1 \dots x_{m-1}$ si il existe une suite $(k_0, k_1, \dots, k_{i-1})$ composée d'entiers et strictement croissante telle que $\forall j \in [0, i-1]$, $w_j = x_{k_j}$. On remarque qu'un programme x de taille m possède au plus 2^m sous-séquences différentes. Soient x et y, deux programmes appartenant à P, on notera $lcs(x, y)$ une *plus longue sous-séquence commune* de x et de y.

En parcourant un meilleur alignement $(\overline{x}, \overline{y})$ de longueur p entre x et y, la sous-séquence w constituée des $\overline{x}_i$ (ou $\overline{y}_i$) telle que $\overline{x}_i = \overline{y}_i$ est toujours une $lcs(x, y)$[4]. En effet, en appariant le plus possible d'instructions d'une $lcs(x, y)$ dans un alignement, on garantit que le coût d'alignement sera minimal et donc qu'il s'agit d'un meilleur alignement. Il existe donc au moins autant de meilleurs alignements entre deux programmes que de plus longues sous-séquences communes.

Soit q, la longueur d'une $lcs(x, y)$ avec x et y deux programmes de tailles respectives m et n, et p la longueur d'un meilleur alignement $(\overline{x}, \overline{y})$ de x et de y. La longueur q correspond aux nombres de positions de $(\overline{x}, \overline{y})$ où aucun ε n'a été inséré, on a donc :

$$\begin{aligned} q &= p - \#(x) - \#(y) \\ &= p - \mathcal{D}(x, y) \\ &= \tfrac{1}{2}\big(m + n - \mathcal{D}(x, y)\big) \end{aligned}$$

Comme les q instructions d'une $lcs(x, y)$ sont présentes aux mêmes positions dans un meilleur alignement, la fonction $X_i(\overline{x}, \overline{y})$ qui échange les suffixes $s(\overline{x}, i)$ et $s(\overline{y}, i)$ lors d'une recombinaison CMH ne peut modifier leurs

[4]Cette assertion n'est valide que pour $C_2 > 2 \times C_1$, comme c'est le cas dans notre étude, car pour $C_2 \leq 2 \times C_1$, des mésappariements pourraient exister dans les meilleurs alignements.

positions. On rappelle que la fonction $\varphi(x, y)$ permet d'extraire les sous-séquences de x et de y qui ne contiennent pas d'instruction ε . On est donc assuré que $lcs(x, y)$ est également une sous-séquence commune aux enfants $(x', y') = \varphi\big(X_i(\overline{x}, \overline{y})\big)$.

En reprenant, l'exemple de la Figure 2.4, on peut identifier une $lcs(x, y)$ dans $(\overline{x}, \overline{y})$ aux 5 positions où aucun ε n'a été inséré, c'est-à-dire `CATGA`. On peut vérifier que pour toutes les positions de l'alignement, $X_i(\overline{x}, \overline{y})$ ne modifie pas la position des instructions de la séquence `CATGA` et que celle-ci est donc une sous-séquence commune à tous les produits x' et y' de la recombinaison.

2.3.2 Ensemble de recombinaison en PGL

Dans [33], les auteurs ont défini le concept d'ensemble de recombinaison dans le but d'étudier l'espace de recherche des opérateurs de croisement pour les structures de tailles fixes. Nous proposons d'adapter cette définition dans le cadre de la PGL.

Définition

Soit P, l'ensemble des programmes définis sur l'alphabet Σ. On définit $\mathfrak{R}$, un opérateur de recombinaison en PGL, comme une fonction

$$\mathfrak{R} : P^2 \to \mathcal{P}(P)$$

qui à chaque couple de programmes de P fait correspondre une partie de P.

On appellera ensemble de recombinaison, noté $\mathfrak{R}(x, y)$, l'ensemble des programmes pouvant être obtenus en appliquant $\mathfrak{R}$ au couple de parents x et y[5]. De même, on notera les ensembles de recombinaison $\mathrm{CMH}(x, y)$ et $\mathrm{CS}(x, y)$, les ensembles des programmes pouvant être obtenus en appliquant, respectivement, les opérateurs CMH et CS au couple de parents x et y.

À partir de cette définition d'un ensemble de recombinaison, on peut mettre en évidence quelques propriétés remarquables :

[5]Il est important de noter que, par convention, x et y appartiennent à $\mathfrak{R}(x, y)$.

La symétrie : Cette propriété signifie que l'ensemble de recombinaison de deux programmes est identique quelque soit l'ordre dans lequel ceux-ci sont choisis.

$$\mathfrak{R}(x, y) = \mathfrak{R}(y, x) \tag{2.8}$$

L'idempotence : Cette propriété signifie qu'aucun programme nouveau ne peut-être obtenu en recombinant un programme unique.

$$\mathfrak{R}(x, x) = \{x\} \tag{2.9}$$

Le respect : Cette propriété signifie que des programmes appartenant à un même ensemble de recombinaison ne sont pas plus différents, *i.e.* pas plus distants[6], de leurs parents que les parents entre eux. Elle traite donc des implications topologiques de la recombinaison. Nous avons choisi d'utiliser le terme "respect" car le concept développé ici est comparable à celui de Radcliffe [84], voir Section 1.2.3.

$$\forall z \in \mathfrak{R}(x, y), \quad \mathcal{D}(x, z) \leq \mathcal{D}(x, y) \text{ et } \mathcal{D}(y, z) \leq \mathcal{D}(x, y) \tag{2.10}$$

On notera que ces trois propriétés sont vérifiées par l'opérateur de croisement utilisé dans les AG.

Ensemble de recombinaison du CMH

On peut définir formellement l'ensemble de recombinaison du CMH, comme :

$$\begin{aligned} \forall x, y \in P, \mathrm{CMH}(x, y) \quad = \quad & \Big\{ (x', y') \quad \Big| \quad \exists i \in [0, p], \\ & (x', y') = \varphi\left(X_i(\overline{x}, \overline{y})\right) \text{ et } (\overline{x}, \overline{y}) \in A^*(x, y) \\ & \text{avec } p \text{ la longueur de } (\overline{x}, \overline{y}) \Big\} \end{aligned}$$

[6] au sens de la distance d'édition.

La symétrie : Nous avons vu que le processus d'alignement du CMH est réalisé grâce à la distance d'édition. Comme la distance d'édition rend compte du nombre de transformations nécessaires pour transformer un programme en un autre, c'est une fonction symétrique (cf Équation 2.3). On peut donc en déduire que $A^*(x, y) = A^*(y, x)$, *i.e.* l'ensemble des meilleurs alignements est également symétrique. C'est pourquoi, la recombinaison par CMH est symétrique et donc $\forall x, y \in P$, $\text{CMH}(x, y) = \text{CMH}(y, x)$.

L'idempotence : Dans le cas d'une recombinaison à partir d'un parent unique, l'ensemble $A^*(x, x)$ ne contient qu'un seul alignement (avec $x = \overline{x}$) et son coût est nul. Ceci implique que les séquences échangées pendant la recombinaison sont identiques, puisque $\forall i \in \mathbb{N}, X_i(\overline{x}, \overline{x}) = (\overline{x}, \overline{x})$, et qu'aucune variation ne peut donc être attendue dans ce cas. C'est pourquoi, $\forall x \in P$, $\text{CMH}(x, x) = \{x\}$.

Le respect : On peut montrer que $\forall x, y \in P$, et $z \in \text{CMH}(x, y)$, on a $\mathcal{D}(x, z) + \mathcal{D}(y, z) = \mathcal{D}(x, y)$. En effet, on peut réécrire z comme la concaténation d'un préfixe d'un des parents et d'un suffixe de l'autre, par exemple $\exists i \in [0, m]$ et $\exists j \in [0, n]$, tels que $z = p(x, i) \centerdot s(y, j)$. La distance d'un programme recombiné à un de ces parents est donc :

$$\mathcal{D}(x, z) = \mathcal{D}\big(p(x, i) \centerdot s(x, i), p(x, i) \centerdot s(y, j)\big)$$

et

$$\mathcal{D}(z, y) = \mathcal{D}\big(p(x, i) \centerdot s(y, j), p(y, j) \centerdot s(y, j)\big)$$

Par définition, on sait que le couple (i, j) correspond à une position identique dans un meilleur alignement $(\overline{x}, \overline{y})$, l'équation 2.5 peut donc s'appliquer :

$$\mathcal{D}(x, z) = 0 + \mathcal{D}\big(s(x, i), s(y, j)\big)$$

et

$$\mathcal{D}(z, y) = \mathcal{D}\big(p(x, i), p(y, j)\big) + 0$$

On a donc :

$$\mathcal{D}(x, z) + \mathcal{D}(y, z) = \mathcal{D}\big(s(x, i), s(y, j)\big) + \mathcal{D}\big(p(x, i), p(y, j)\big)$$

En appliquant encore l'équation 2.5, on obtient :

$$\begin{aligned}\mathcal{D}(x,z)+\mathcal{D}(y,z) &= \mathcal{D}\big(p(x,i)\centerdot s(x,i), p(y,j)\centerdot s(y,j)\big)\\ &= \mathcal{D}(x,y)\end{aligned}$$

C'est pourquoi :

$$\forall z \in \mathrm{CMH}(x,y), \quad \mathcal{D}(x,z) \leq \mathcal{D}(x,y) \text{ et } \mathcal{D}(y,z) \leq \mathcal{D}(x,y) \tag{2.11}$$

Ensemble de recombinaison du CS

On peut définir formellement l'ensemble de recombinaison du CS, comme :

$$\begin{aligned}\forall x,y \in P, \mathrm{CS}(x,y) = \Big\{(x',y') \;\Big|\; &\exists i \in [0,m], \exists j \in [0,n],\\ &x' = p(x,i)\centerdot s(y,j) \text{ et } y' = p(y,j)\centerdot s(x,i)\\ &\text{avec } m \text{ la taille de } x \text{ et } n \text{ la taille de } y\Big\}\end{aligned}$$

On notera que $\forall x,y \in P$, $\mathrm{CMH}(x,y) \subset \mathrm{CS}(x,y)$.

La symétrie : L'opérateur CS qui échange aléatoirement des séquences d'instructions de tailles quelconques entre les parents est évidement symétrique. Ainsi, $\mathrm{CS}(x,y) = \mathrm{CS}(y,x)$.

L'idempotence : Pour l'opérateur CS, cette propriété n'est pas vérifiée puisque, même dans le cas d'un parent unique, des séquences de tailles différentes peuvent être échangées. En effet, prenons l'exemple d'un programme x de taille m composé d'une seule instruction, alors son ensemble de recombinaison $\mathrm{CS}(x,x)$ contient $2m+1$ programmes dont la taille varie entre 0 et $2m$.

Le respect : On peut déduire de la propriété précédente que celle-ci n'est pas non plus vérifiée pour l'opérateur CS puisqu'à partir d'un programme unique x de taille m on peut obtenir des programmes z de taille $2m$ avec $\mathcal{D}(x,z) = m$.

On note $\mathcal{D}_{xy}(x,z) = \mathcal{D}(x,z)/\mathcal{D}(x,y)$, la distance entre x et z relative à x et y. Pour visualiser la distance dans un ensemble de recombinaison,

nous avons tracé, Figure 2.6, le contour du nuage de points de coordonnées $\mathcal{D}_{xy}(x,z)$ en abscisse et $\mathcal{D}_{xy}(y,z)$ en ordonnée, avec $z \in \mathrm{CS}(x,y)$ et $x,y \in P$ deux parents composés de 50 instructions et P l'ensemble des programmes définit sur Σ un alphabet de taille N. Il s'agit donc d'un nuage symétrique par rapport à la première bissectrice. Tout d'abord, on peut vérifier que $\mathcal{D}(x,z)$ et $\mathcal{D}(y,z)$ peuvent être au moins deux fois plus grande que $\mathcal{D}(x,y)$, en particulier avec N=4. On note également que la surface du nuage décroît avec N et on peut supposer que la propriété étudiée précédemment pourrait être vérifiée pour des valeurs de N très grandes.

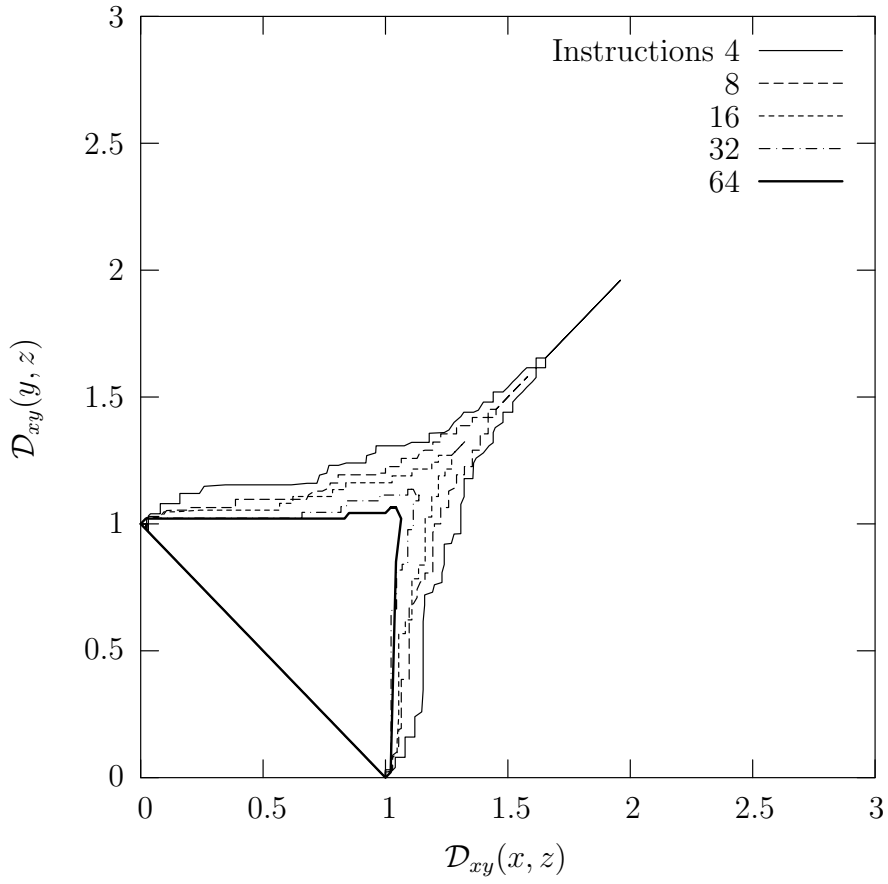

FIG. 2.5 – Opérateur CS : Distance relative aux parents en fonction de la taille de l'alphabet dans un ensemble de recombinaison.

Nous avons tracé le même type de contour, Figure 2.5, pour des parents

de tailles différentes et définis sur un alphabet de 16 instructions. On constate que la surface du nuage est maximale pour des parents de tailles identiques et qu'elle diminue quand la différence de taille augmente.

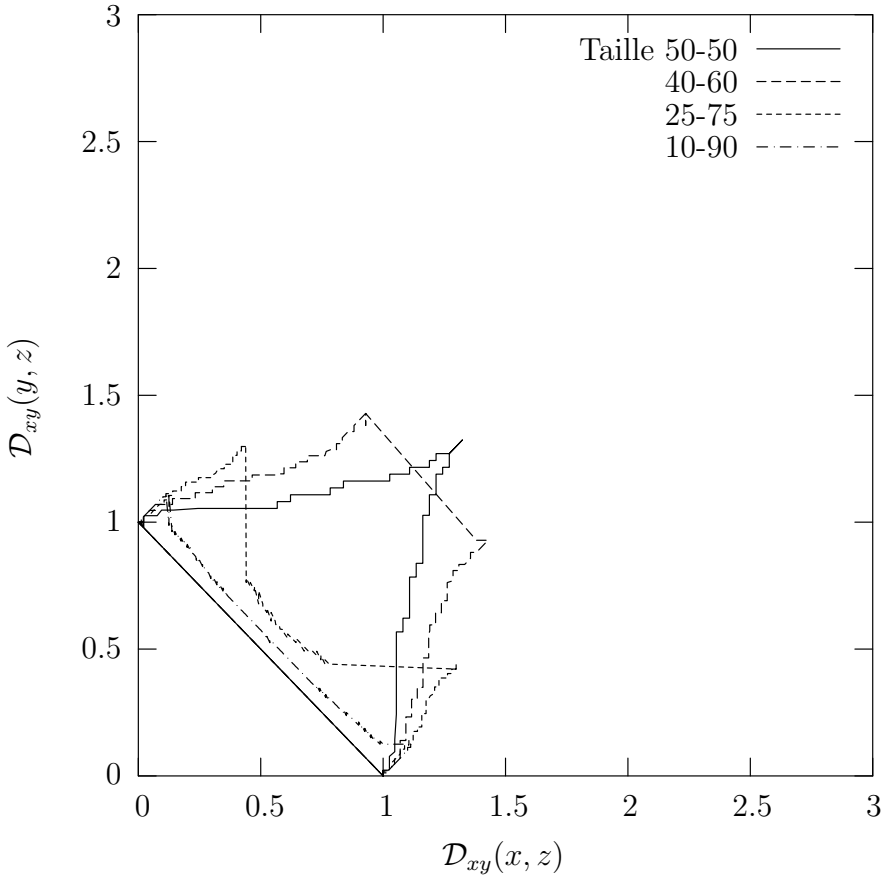

FIG. 2.6 – Opérateur CS : Distance relative aux parents en fonction de la taille des programmes dans un ensemble de recombinaison.

On notera que de telles figures n'auraient aucun intérêt dans le cas de l'opérateur CMH, car comme nous l'indique l'équation 2.11, tous les points de la figure se situeraient sur le segment de coordonnées $(0,1)$ $(1,0)$.

2.3.3 Lignée

Bien que le concept d'ensemble de recombinaison soit un outil pertinent pour décrire certaines propriétés d'un opérateur de croisement, il ne permet

pas de rendre compte des effets complexes de la recombinaison dans un système PG, où l'opérateur de croisement est appliqué sur les meilleurs individus qui composent la population. Ces individus sont plus ou moins divers et sont souvent eux aussi le produit de multiples recombinaisons. Nous proposons d'aller plus loin grâce au concept de lignée.

Définition

On appellera lignée, notée $\mathfrak{R}^n(x, y)$, l'ensemble des programmes pouvant être obtenus par n applications successives de $\mathfrak{R}$ au couple de géniteurs x, y et à tous les couples de leurs descendants tel que :

$$\begin{aligned} \mathfrak{R}^0(x, y) &= \{x, y\} \\ \mathfrak{R}^1(x, y) &= \mathfrak{R}(x, y) \\ \mathfrak{R}^{n+1}(x, y) &= \{\mathfrak{R}(a, b) | a, b \in \mathfrak{R}^n(x, y)\} \end{aligned}$$

De même, on notera les lignées $\mathrm{CMH}^n(x, y)$ et $\mathrm{CS}^n(x, y)$.

Distance dans une lignée

Nous avons tracé, Figure 2.7 et 2.8, la distance relative aux géniteurs des lignées, respectivement $\mathrm{CS}^{100}(x, y)$ et $\mathrm{CMH}^{100}(x, y)$, à partir d'un échantillon de 10^5 programmes, avec $x, y \in P$ deux géniteurs composés de 50 instructions et P l'ensemble des programmes définit sur Σ un alphabet de taille 20. Chaque point correspond à un programme z d'une lignée et a pour coordonnées $\mathcal{D}_{xy}(x, z)$ en abscisse et $\mathcal{D}_{xy}(y, z)$ en ordonnée. Il s'agit donc d'un nuage symétrique par rapport à la première bissectrice.

Nous avons également indiqué, par une ligne pointillée, une zone rectangulaire délimitant les points de coordonnées valides. En effet, conformément à l'inégalité triangulaire[7], les parties inférieure gauche, inférieure droite et supérieure gauche des figures ne peuvent contenir aucun point puisque :

– dans la partie inférieure droite, on aurait $\mathcal{D}_{xy}(z, y) \leq \mathcal{D}_{xy}(z, x) - 1$, d'où $\mathcal{D}(z, y) + \mathcal{D}(y, x) \leq \mathcal{D}(z, x)$, ce qui est impossible ;

[7]La distance d'édition est une véritable mesure de distance, qui respecte donc l'inégalité triangulaire (cf Équation 2.4).

– dans la partie supérieure gauche, on aurait $\mathcal{D}_{xy}(z,x) \leq \mathcal{D}_{xy}(z,y) - 1$, d'où $\mathcal{D}(z,x) + \mathcal{D}(y,x) \leq \mathcal{D}(z,y)$, ce qui est impossible ;
– dans la partie inférieure gauche, on aurait $\mathcal{D}_{xy}(z,x) + \mathcal{D}_{xy}(z,y) \leq 1$, d'où $\mathcal{D}(x,z) + \mathcal{D}(z,y) \leq \mathcal{D}(x,y)$, ce qui est impossible.

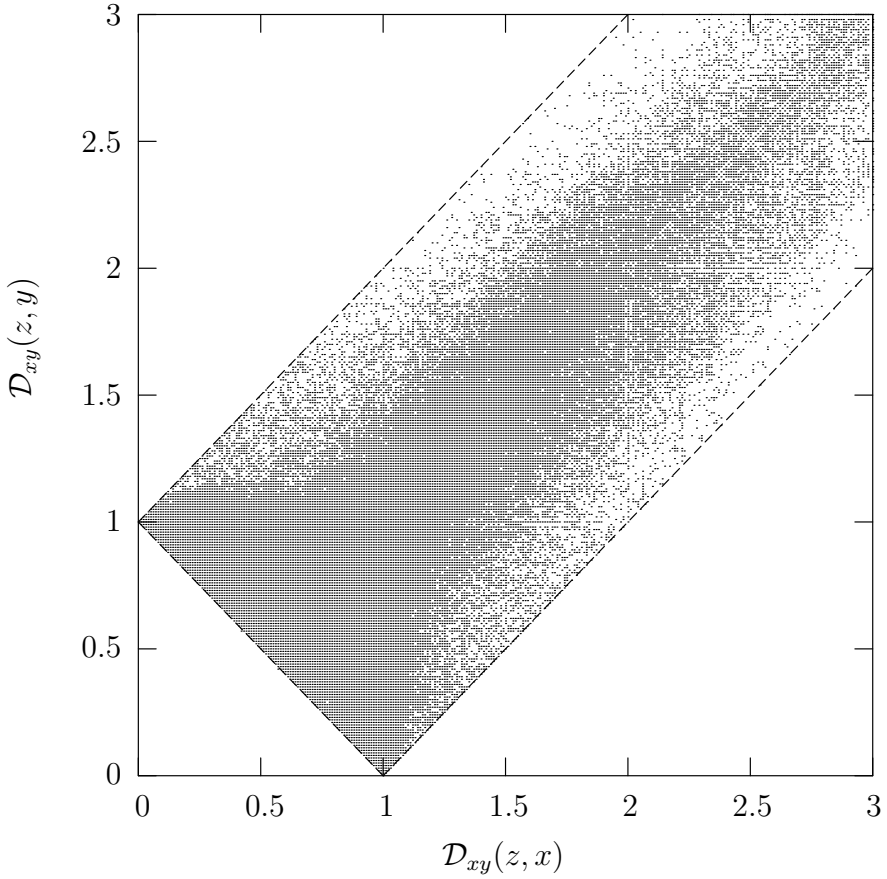

FIG. 2.7 – Opérateur CS : Distance relative aux géniteurs d'une lignée

Pour permettre une meilleure comparaison entre les deux figures, nous avons uniquement représenté les points de coordonnées inférieures à $(3,3)$. On constate que les deux figures diffèrent nettement. Presque toute la zone de coordonnées possibles est couverte dans le cas de l'opérateur CS et la distance aux géniteurs a atteint des valeurs très élevées, jusqu'à 8. Avec le CMH, le nuage de points est beaucoup plus compact et la distance relative aux géniteurs n'est jamais supérieure à 2.

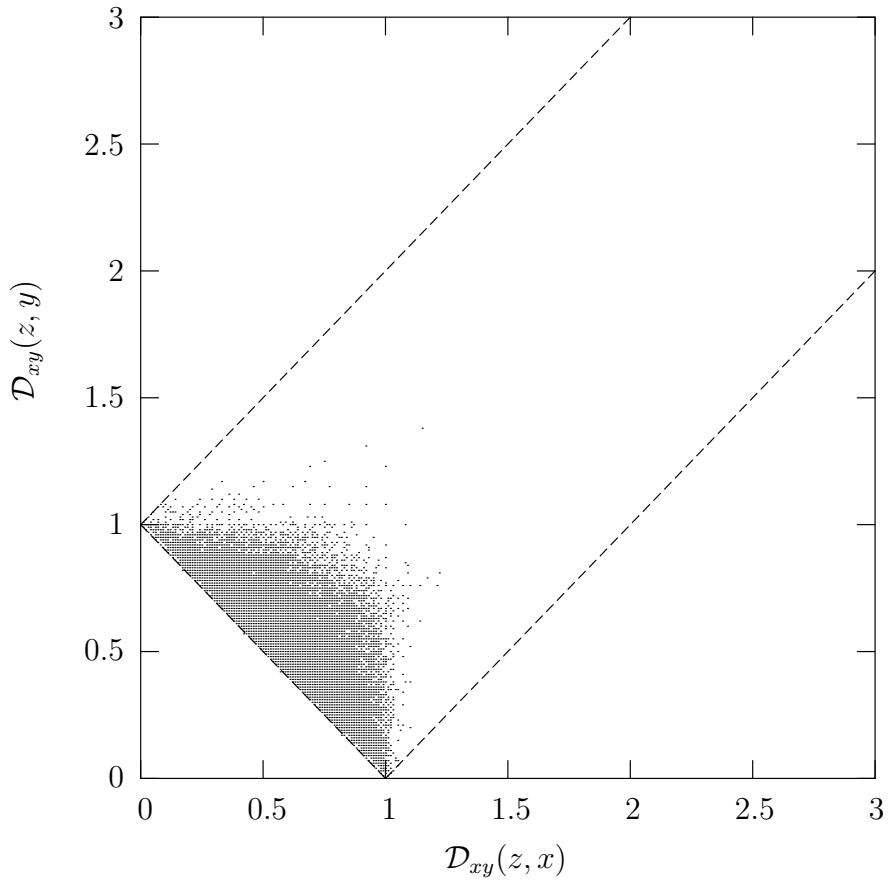

FIG. 2.8 – Opérateur CMH : Distance relative aux géniteurs d'une lignée

L'intérêt de ce type de figure est qu'il permet de comparer la recherche opérée par différents opérateurs de recombinaison. Dans notre cas, on en déduit que le CMH effectue une recherche plus locale que celle de l'opérateur CS.

2.3.4 Taille des programmes

Dans [80][90], les auteurs ont étudié l'évolution de la taille de programmes recombinés par l'opérateur CS sans pression de sélection, sans limite de taille et au sein d'une population infinie. Ils ont montré théoriquement et expérimentalement que la moyenne de la taille dans la population n'évolue pas mais que la distribution de la taille converge, quelque soit la distribution initiale, vers une distribution γ. Pour simplifier, une distribution γ est caractérisée par une sur-représentation des valeurs faibles et par la présence de valeurs extrêmement élevées par rapport à la moyenne.

Nous voulons comparer expérimentalement l'évolution de la distribution de la taille induite par l'opérateur CMH à celle du CS.

Évolution de la distribution

Les Figures 2.9 et 2.10 présentent, pour les deux opérateurs, la distribution de la taille des programmes, en échelle logarithmique, aux générations 0,1,100 et 1000. Ces distributions sont des moyennes sur 200 expériences et sont calculées à partir de l'évolution d'une population de 1000 individus sans pression de sélection.

L'évolution de la distribution obtenue expérimentalement avec l'opérateur CS est conforme aux conclusions de [80]. Dans le cas du CMH, l'évolution est beaucoup moins rapide mais semble également converger vers une distribution de type γ. A la génération 1000, quand le CS est utilisé, les programmes composés de 2 instructions sont les plus représentés avec 7.6% de la population, pour le CMH, il s'agit de la taille 23 avec 3.4%. Il semble donc que l'opérateur CMH soit moins biaisé, au sens de [80], que l'opérateur CS.

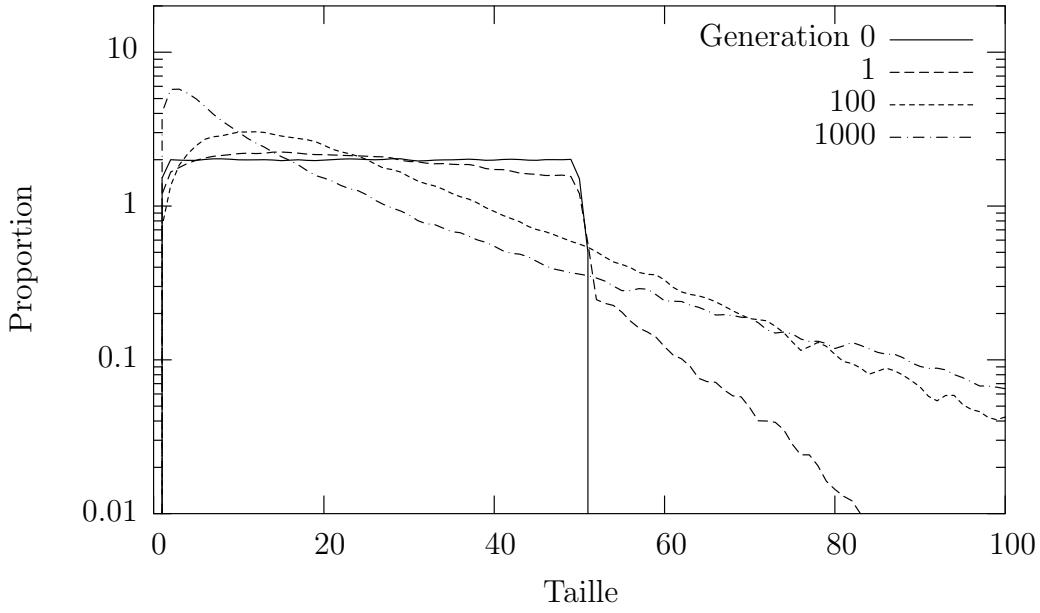

FIG. 2.9 – Opérateur CS : Évolution de la distribution de la taille des programmes

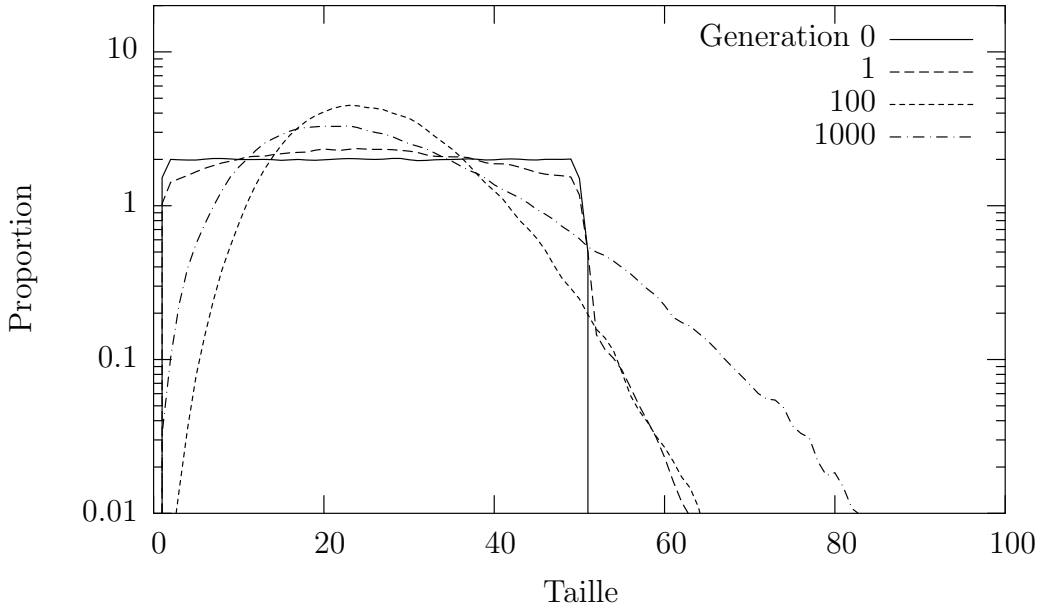

FIG. 2.10 – Opérateur CMH : Évolution de la distribution de la taille des programmes

2.3.5 Sélection des sites de croisement

Sites possibles

Nous avons déjà vu que l'opérateur CS échange aléatoirement des séquences d'instructions de tailles quelconques entre les parents. Ces séquences sont induites par le choix d'un couple quelconque de sites de croisement, c'est-à-dire un site indépendamment dans chacun des parents. Dans le cas d'un CMH, au contraire, il n'existe qu'un nombre restreint de couples de sites de croisement possibles. Ces couples sont déterminés par l'ensemble des meilleurs alignements qui à une instruction donnée dans un des parents fait correspondre un nombre limité d'instructions dans l'autre[8]. Plus précisément, un couple est toujours constitué de sites compris entre deux instructions d'une plus longue sous-séquence commune aux programmes recombinés.

Prenons l'exemple de deux programmes :

x =`CGCGCTCGTCCCGTGGCTTCCAAGACCTTATCGGCGTCATTAGTGA`

et

y=`CATCTGATGAAACGCGTGAAAGACCTTATACCTCAAA`

de longueurs respectives 46 et 37 instructions. La Figure 2.11 représente un meilleur alignement de longueur 56 entre x ($\overline{x}$ en partie supérieure) et y ($\overline{y}$ en partie inférieure). La distance d'édition entre x et y est $\mathcal{D}(x, y) = 29$. On peut voir que les instructions communes de x et de y ont été, dans la mesure du possible, appariées de sorte qu'une $lcs(x, y)$, *i.e.* `CTCGTGGCTAA-GACCTTATCCTCAAA`, de longueur 27 est présente aux mêmes positions de part et d'autre de l'alignement.

Examinons les couples de sites possibles entre les deux premières instructions `C` et `T` de $lcs(x, y)$. Tout d'abord, les sites 1 et 7 de l'alignement $(\overline{x}, \overline{y})$ correspondent aux couples (`C`,`C`) et (`T`,`T`). Ce sont des instructions de $lcs(x, y)$, elles seront donc conservées lors d'une recombinaison CMH. Les sites 2 à 6 de $(\overline{x}, \overline{y})$ correspondent aux couples (`G`,`A`), (`C`,`A`), (`G`,`A`), (`C`,`A`) et (`T`,`A`). Toutes les instructions de ces couples sont situées entre les deux instructions `C` et `T` de

[8]On rappelle que $\forall x, y \in P$, $\mathrm{CMH}(x, y) \subset \mathrm{CS}(x, y)$.

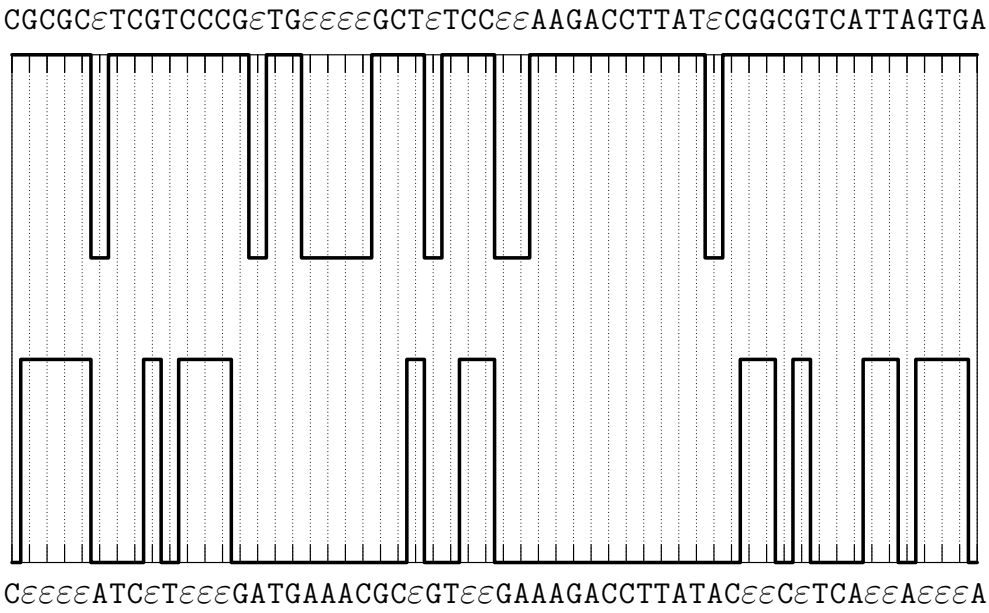

FIG. 2.11 – Le processus d'alignement révèle les régions plus ou moins homologues de x et de y.

$lcs(x, y)$. Les instructions `C`, `G`, `C`, `G`, `C` et `T` de x seront donc échangées avec les instructions `C`, `A` et `T` de y. On comprend bien, sur cet exemple, pourquoi la recombinaison CMH échange toujours des instructions comprises entre deux positions d'une plus longue sous-séquence commune.

Probabilité de sélection des sites

Dans le cas d'une recombinaison CS, le choix du site de croisement est réalisé uniformément parmi toutes les instructions. A contrario, avec le CMH, le processus d'alignement modifie la probabilité qu'un site de croisement soit choisi entre deux instructions données. Nous avons délimitée, Figure 2.11, les régions les moins homologues où des instructions ε ont été insérées. Du point de vue de la recombinaison CMH, ces zones constituent les parties fragiles de l'alignement, là où la probabilité que l'échange soit réalisé est la plus grande.

Pour bien comprendre la situation, nous avons représenté Figure 2.12 et Figure 2.13, les probabilités $\mathcal{S}(x, i)$ et $\mathcal{S}(y, i)$ que les $i^{\text{ième}}$ et $i + 1^{\text{ième}}$

instructions de x et de y soient séparées par une recombinaison. La ligne pleine correspond à une recombinaison CMH alors que la ligne pointillée correspond à une recombinaison par l'opérateur CS. Pour l'opérateur CS, les probabilités $\mathcal{S}(x,i)$ et $\mathcal{S}(y,i)$ sont indépendantes l'une de l'autre. Elles sont constantes sur tout le génotype et valent $1/(n+1)$ pour un programme de taille n. En ce qui concerne le CMH, $\mathcal{S}(x,i)$ et $\mathcal{S}(y,i)$ varient en fonction du nombre d'instructions ε insérées entre i et $i+1$ et dépendent donc du meilleur alignement choisi. A titre d'exemple, les deux premières instructions `C` et `A` de y sont séparées par 4 ε Figure 2.11, ce qui signifie que 5 sites de croisement dans l'alignement entraînent la séparation de `C` et `A`, d'où $\mathcal{S}(y,1) = 5/(56+1)$ ce qui fait trois fois plus qu'avec l'opérateur CS.

Ainsi, moins des régions sont homologues dans les programmes recombinés, plus elles ont de chances d'être modifiées lors d'une recombinaison par CMH. Cela révèle une forme d'efficacité de l'opérateur, puisque les efforts de calculs consentis lors du processus d'alignement sont dans un sens compensés par cette propriété qui assure que le matériel génétique échangé sera le plus différent possible. Autrement dit, cela permet de diminuer la probabilité que la recombinaison de parents différents ne soit qu'une simple recopie et que du temps de calcul ne soit dépensé dans de multiples évaluations de programmes identiques. En effet, une méthode simple et peu efficace, permettant de garantir la conservation de l'homologie, *i.e.* des caractéristiques communes aux parents, durant la recombinaison aurait pu consisté à limiter au maximum les variations liées à l'opérateur de croisement. Au contraire, avec le CMH, un maximum de variations peuvent être espérées tout en conservant l'homologie.

2.4 Conclusion

Pour faire face aux critiques formulées à l'encontre de l'opérateur de croisement utilisé de façon standard en PG, des auteurs ont préconisés l'usage de nouveaux opérateurs de croisement plus "homologues", c'est-à-dire garantissant que certaines caractéristiques communes aux parents soient conservés lors d'une recombinaison. Différentes tentatives avaient déjà été réalisées dans ce sens, avec des critères d'homologie variés, portant par exemple sur la taille,

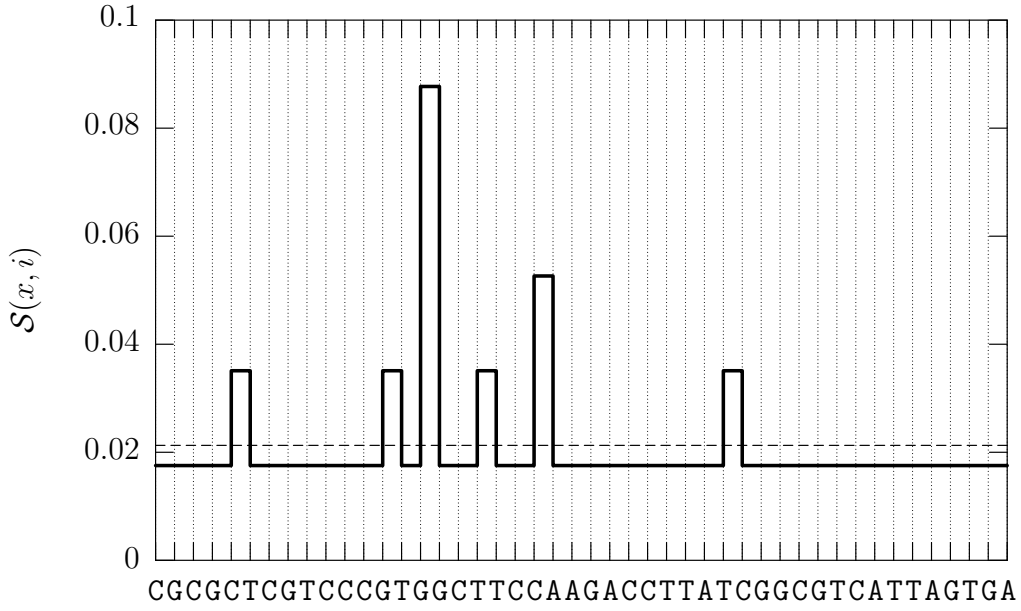

FIG. 2.12 – Le processus d'alignement modifie la probabilité du choix site de croisement dans x.

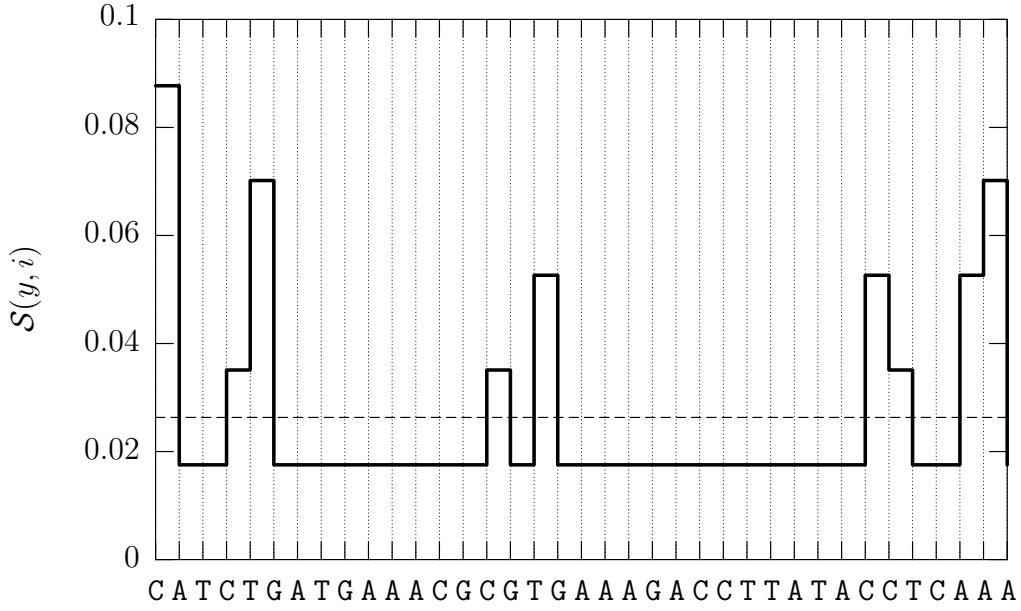

FIG. 2.13 – Le processus d'alignement modifie la probabilité du choix site de croisement dans y.

la forme, la position ou bien la fonction du matériel génétique échangé.

Dans ce chapitre, nous avons défini un nouvel opérateur de croisement pour la PGL, le Croisement par Maximum d'Homologie (CMH), où le critère d'homologie retenu est la similarité de séquences des programmes, mesurée par la distance d'édition. Lors d'une recombinaison CMH, nous avons la garantie qu'une des plus longues sous-séquences communes aux deux parents sera conservée chez les enfants. De plus, nous avons vu que dans la mesure du possible le produit du croisement ne sera pas une simple recopie des parents. Nous espérons qu'en définissant l'homologie au plus bas niveau possible, c'est-à-dire au niveau génétique, le CMH pourra à la fois conserver la taille, la position et la fonction du matériel génétique échangé et qu'ainsi, la brutalité de la recombinaison en PG, *i.e.* ses effets délétères sur la fitness (cf Section 1.2.3), pourra être évitée.

Notons enfin que l'opérateur CMH, tel que nous l'avons défini, peut être utilisé pour recombiner toutes formes de représentation linéaire de taille variable utilisées en Évolution Artificielle.

Chapitre 3

Homologie, neutralité et difficulté dans les paysages de fitness

Dans ce chapitre, nous réalisons des études préliminaires pour l'utilisation opérationnelle du CMH. Une des principales inconnues dans ce domaine est le réglage du taux de croisement optimal du CMH et sa capacité à fonctionner conjointement avec l'opérateur de mutation. En effet, le fonctionnement du CMH diffère largement de celui de l'opérateur de croisement standard et nous avons déjà noté, en dehors de toute notion de performance, diverses influences sur la taille et la nature des programmes recombinés.

L'utilisation de problèmes dits synthétiques pour caractériser les performances d'un algorithme, ou d'un opérateur, est devenue classique en EA. Cette démarche s'inscrit dans la problématique plus générale des algorithmes d'optimisation pour lesquels la nécessité de caractériser et de classer les problèmes à optimiser est avérée. En effet, les *No Free Lunch Theorems* [105] impliquent que si un algorithme est plus performant que la recherche aléatoire pour une classe de problèmes donnée alors il existe une classe de problèmes pour laquelle il le sera moins, et que donc, il n'existe pas d'algorithme universel qui soit toujours efficace. C'est pourquoi, une connaissance approfondie des relations entre la performance d'un algorithme et les propriétés des problèmes traités est nécessaire. Un problème synthétique possède l'avantage d'être, par construction, bien connu et permet souvent, de part la simplicité

de l'évaluation d'un individu, d'effectuer des expérimentations plus poussées. Par exemple, un parcours exhaustif de l'espace de recherche est parfois réalisable ce qui rend possible l'énumération des optima globaux. En outre, il possède généralement un nombre réduit de paramètres permettant d'ajuster ces propriétés (taille, neutralité, difficulté, épistasie, ...) et ainsi de mieux caractériser leurs relations avec un opérateur donné.

Cependant, les travaux existants dans le domaine de la synthèse de problèmes traitent rarement de l'Évolution Artificielle d'individus de Tailles Variables (EA-TV) comme c'est le cas en PG ou avec les Messy-GA, une exception serait par exemple [23]. Or il existe des problématiques spécifiques à l'EA-TV. L'identification des gènes, par exemple, qui, dans le cas des AG, est triviale et liée à la position dans le génotype, est beaucoup plus complexe en EA-TV, puisqu'un gène sera reconnu par sa forme (combinaison d'instructions) ou bien par sa fonction. Une autre spécificité de l'EA-TV est que la neutralité des problèmes à optimiser est généralement beaucoup plus forte[1]. Ceci peut s'expliquer d'une part parce qu'un gène peut se trouver à différentes positions dans le génotype alors que sa position n'influe pas durant l'évaluation de la performance (neutralité positionnelle), d'autre part par une très grande quantité d'introns (neutralité fonctionnelle), liée aux mécanismes d'évaluation par exemple. Dernière particularité : la taille variable des individus peut être considérée comme une nouvelle dimension dans l'espace de recherche qui peut nécessiter une exploration spécifique et, par la même, imposer l'utilisation d'opérateurs idoines.

Largement inspirés par les travaux réalisés dans le domaine des AG, nous introduisons trois problèmes synthétiques pour l'EA-TV : le problème de la *Fitness Guidée par l'Homologie*, le problème des *Routes Royales* et celui des *Routes Épistatiques*. Pour caractériser leurs propriétés respectives, nous utilisons la métaphore des paysages de fitness et un ensemble de mesures sur ces paysages.

[1]On parle de forte neutralité quand la proportion de génotypes possédant la même valeur de fitness est élevée.

3.1 Paysages de fitness en EA-TV

3.1.1 Définition

Introduit par Sewall Wright [106] en 1931, le concept de *paysage de fitness* est un des plus importants en biologie de l'évolution. Il est basé sur l'hypothèse qu'à chaque organisme, c'est-à-dire à chaque génotype, peut-être associée une valeur indiquant son adaptation, appelée fitness. Un arrangement des génotypes décrit alors l'espace dans lequel évolue une population soumise à la sélection naturelle. L'intérêt de cette métaphore pour les biologistes est que la topologie de tels paysages reflète directement la possibilité pour une population d'atteindre certains génotypes (ou phénotypes).

Le concept de paysage de fitness a été également utilisé en EA [46]. Dans ce cas, une fonction définit la fitness de chaque génotype et ceux-ci sont arrangés dans un ordre topologique, appelé espace des configurations, qui dépend des opérateurs génétiques utilisés (approche "un opérateur - un paysage" [44]). On considère souvent que deux génotypes sont voisins dans l'espace des configurations si l'on peut passer de l'un à l'autre en une seule mutation, mais certains travaux ont également étudié l'espace des configurations lié à l'opérateur de recombinaison [33]. Formellement, on définit un paysage de fitness comme un triplet $(G, \mathcal{V}, f)$, avec :

- l'ensemble des génotypes G, c'est-à-dire l'ensemble des configurations,
- la fonction de voisinage $\mathcal{V} : G \rightarrow \mathcal{P}(G)$. $\forall x, y \in G$, $\mathcal{V}(x)$ est appelé voisinage de x et on dira que y est voisin de x si $y \in \mathcal{V}(x)$.
- f : la fonction de fitness qui associe une valeur réelle à chaque élément de G.

Dans le cadre de nos travaux, nous nous intéressons aux paysages de fitness en EA-TV et plus particulièrement en PGL dont la spécificité réside dans la fonction de voisinage qui doit être définie sur des chaînes de taille variable. Nous définissons le voisinage par rapport à l'opérateur de mutation, donc aux trois opérations élémentaires qu'il réalise (insertion, délétion et substitution). Nous avons déjà vu que la distance d'édition $\mathcal{D}$ rend compte du nombre d'opérations élémentaires permettant de transformer un génotype en un autre, on prendra donc $\mathcal{V}(x) = \{y \in G | \mathcal{D}(x, y) = 1\}$.

On note qu'un génotype de longueur λ défini sur un alphabet Σ de taille N possède au plus $(2\lambda + 1) \times N$ voisins. En effet, une borne supérieure de la taille du voisinage peut être obtenue en comptant le nombre de mutations $\mathcal{M}$ qu'il est possible d'effectuer sur un individu et en supposant que toutes ces mutations donnent un génotype différent. On a donc :

$$\begin{array}{rcll} \mathcal{M} & = & \lambda \times (N-1) & \text{(nombre de substitutions)} \\ & + & \lambda & \text{(nombre de délétions)} \\ & + & (\lambda+1) \times N & \text{(nombre d'insertions)} \\ & = & (2\lambda+1) \times N & \end{array}$$

3.1.2 Mesures sur les paysages

Plusieurs mesures ont été proposées et largement utilisées pour décrire les paysages de fitness en terme de "difficulté". Ici, la difficulté fait référence à la capacité pour une heuristique de recherche locale d'atteindre un optimum, c'est-à-dire que ces mesures ne sont pas conçues pour directement rendre compte de la difficulté de l'optimisation par un algorithme manipulant une population de solutions et utilisant un opérateur de recombinaison. Nous verrons par la suite si ces mesures sont, dans la pratique, adaptées à la PG.

Marches Aléatoires

Soit G, l'ensemble des génotypes définis sur un alphabet Σ de taille N. Une marche aléatoire $\{g_1, g_2, \ldots\}$ sur un paysage de fitness est une série dans G dont g_1 est le génotype initial et où $g_{i+1} \in \mathcal{V}(g_i)$ est choisi aléatoirement. Dans [104], Weinberger définit la *fonction d'auto-corrélation* et la *longueur de corrélation* d'une marche aléatoire dans le but de mesurer l'épistasie, *i.e.* le degré d'interaction entre les gènes, dans un paysages de fitness.

La fonction d'auto-corrélation ρ d'une fonction de fitness f est la fonction d'auto-corrélation de la série $\{f(g_1), f(g_2), \ldots, f(g_l)\}$ telle que :

$$\rho(s) = \frac{\sum_{t=1}^{l-s}(f(g_t) - \overline{f})(f(g_{t+s}) - \overline{f})}{\sum_{t=1}^{l}(f(g_t) - \overline{f})^2}$$

avec l la longueur de la marche, $s \in [1, l]$ et $\overline{f} = (1/l)\sum_{t=1}^{l} f(g_t)$. On considère généralement que la valeur de $\rho(s)$ est significativement différente de 0

quand elle n'est pas comprise dans l'intervalle $[-2/\sqrt{l}, +2/\sqrt{l}]$ et donc que la fonction de fitness f n'est pas aléatoire.

La longueur de corrélation τ d'une marche aléatoire permet de mesurer comment la corrélation décroît avec la distance et donc d'estimer la rugosité[2] d'un paysage. Plus un paysage est rugueux plus la longueur de corrélation est faible. A titre de comparaison, on pourra utiliser dans cette étude la Table 3.1 qui indique la longueur de corrélation τ pour différents Paysages-NK (qui seront décrits Section 3.2.3) en fonction du paramètre K qui contrôle la rugosité du paysage.

TAB. 3.1 – Paysages-NK, N=100 : Longueur de corrélation τ d'une marche aléatoire.

K	τ	Rugosité
0	> 50	Très faible
5	47	Faible
25	17	Élevée
95	1	Très élevée

Pour calculer τ, on fait varier s de 0 à l, et la valeur de τ est donnée par $s-1$ quand pour la première fois $\rho(s)$ est comprise dans l'intervalle $[-2/\sqrt{l}, +2/\sqrt{l}]$. Autrement dit, il s'agit de la plus grande valeur de s pour laquelle l'auto-corrélation est significative.

Marches Adaptatives

Contrairement aux marches aléatoires, où seule la notion de voisinage est retenue pour choisir le génotype suivant de la série, dans le cas des marches adaptatives un critère de performance supplémentaire est appliqué, tel que seuls des déplacements améliorant la fitness soient acceptés. Il existe plusieurs variantes de marches adaptatives ([52]) avec différents critères de sélection, comme par exemple un des meilleurs voisins ou le premier voisin meilleur trouvé.

[2]Traduction libre de l'anglais *ruggedness* qui signifie accidenté, ayant une surface irrégulière.

Soit G, l'ensemble des génotypes définis sur un alphabet Σ de taille N. Dans notre cas, une marche adaptative est une série $\{g_1, g_2, \ldots, g_{t+l}\}$ dans G dont g_1 est le génotype initial et où g_{i+1} est le premier élément de $\mathcal{V}(g_i)$ trouvé qui soit plus performant que g_i. Contrairement aux marches classiques, la recherche n'est pas effectuée de manière exhaustive parmi tous les voisins de g_i mais au plus parmi $(2\lambda+1) \times N$ génotypes, avec λ la taille de g_i, choisis aléatoirement avec remise dans $\mathcal{V}(g_i)$. Ceci permet de rendre compte de la difficulté de découvrir un des voisins meilleurs dans le cas où leur densité est faible et correspond mieux au comportement d'un système PG où la recherche locale est effectuée "en aveugle".

Une marche adaptative s'arrête donc quand aucun voisin meilleur n'est découvert parmi les $(2\lambda + 1) \times N$ génotypes testés, c'est-à-dire que g_{t+l} est un optimum local. En calculant plusieurs marches, on peut donc estimer :

- la distribution de la fitness des optima locaux à partir de la distribution des fitness finales $f(g_{t+l})$.
- la distance moyenne aux optima locaux à partir de la longueur l des marches adaptatives.

3.2 Problèmes synthétiques pour l'EA-TV

Dans cette section, nous introduisons trois problèmes synthétiques qui tiennent compte des spécificités de l'EA-TV :

- le problème de la *Fitness Guidée par l'Homologie* où la fitness et la distance à l'optimum sont totalement corrélées ;
- le problème des *Routes Royales* dont la taille et la neutralité peuvent être ajustées et qui permet de mettre en évidence les effets disruptifs d'un opérateur de recombinaison ;
- le problème des *Routes Épistatiques* dont la taille, la neutralité et la rugosité peuvent être ajustées et qui permet de mettre en évidence la difficulté d'optimiser des fonctions non-linéaires.

3.2.1 Fitness Guidée par l'Homologie

Définition

Nous présentons ici un nouveau problème, noté FGH, pour *Fitness Guidée par l'Homologie* [27], où la fitness d'un individu est donnée par son homologie, sa distance d'édition $\mathcal{D}$, à un génotype donné choisi comme unique optimum.

La fonction de fitness du problème FGH est donc une fonction $f_{g^*} : G_\Sigma \rightarrow \mathbb{N}$ définie sur des génotypes de taille variable telle que la fitness d'un génotype $g \in G_\Sigma$, notée $f_{g^*}(g)$ soit :

$$f_{g^*}(g) = \mathcal{D}(g, g^*)$$

avec

- Σ un alphabet de taille N qui définit l'ensemble de tous les symboles s possibles par locus ;
- G_Σ l'ensemble fini de tous les génotypes g de taille $\lambda \leq \lambda_{max}$ définis sur Σ ;
- $g^* \in G_\Sigma$ un génotype de taille Λ choisi comme optimum.

Le problème FGH dans le cadre de l'EA-TV est analogue au problème de l'Unitation pour les AG. Dans ce problème, l'optimum est le génome composé exclusivement de bits à 1 et la fitness d'un individu est donné par son unitation, c'est-à-dire son nombre de bits à 1. Ainsi, la fitness est égale à la distance de Hamming à l'optimum.

Analyse du paysage

Nous avons déjà vu que la taille de l'alphabet est un des paramètres qui influence le plus la distance entre deux individus (cf. Section 2.2.1), c'est pourquoi elle doit être prise en compte pour l'étude du paysage FGH. Nous avons réalisé 10^3 marches aléatoires de longueur 1000 ainsi que 10^3 marches adaptatives sur le paysage FGH, en utilisant des alphabets Σ de tailles N différentes, N variant de 1 à 40. La taille de l'optimum g^* est fixée à Λ=80, la taille initiale des génotypes est choisie aléatoirement entre 1 et 80 gènes alors que la taille maximum autorisée est λ_{max}=100.

La Figure 3.1 présente la longueur de corrélation τ moyenne des marches aléatoires et son écart type en fonction de N. Pour les différentes valeurs de N testées, τ conserve une valeur élevée et n'est pas influencée par N, ce qui constitue un résultat attendu puisque N ne modifie pas la rugosité du paysage et que dans tous les cas il n'existe qu'un seul optimum global (aucun optimum local). On peut donc raisonnablement imputer les variations de τ à des erreurs d'échantillonnages.

La Figure 3.2 présente la longueur moyenne des marches adaptatives l et son écart type en fonction de N. On constate que, pour le cas théorique N=1, l est proche de 40 ce qui signifie qu'en moyenne 40 pas adaptatifs sont nécessaires pour atteindre l'optimum. Par contre, l augmente clairement avec N et converge asymptotiquement vers 65. Ces variations de l sont dues au fait que la distance moyenne entre deux génotypes, donc dans notre cas la distance à l'optimum, augmente avec N (cf Figure 2.1).

Enfin, la Figure 3.3 présente la fitness moyenne d'arrêt des marches adaptatives, notée $f(g_l)$, c'est-à-dire la fitness du meilleur génotype trouvé, et son écart type en fonction de N. On note que, pour N=1, $f(g_l)$ est nulle, ce qui signifie que l'optimum a été découvert au cours de toutes les marches. Cependant, quand N augmente, l'optimum semble être de plus en plus difficile à atteindre. Ce comportement est du à la définition des marches adaptatives que nous avons choisi (cf. 3.1.2). En effet, celles-ci rendent compte de la densité des meilleurs voisins et donc de la difficulté de découvrir un meilleur voisin quand la taille du voisinage augmente avec N. Cette définition apparaît satisfaisante, puisqu'on peut supposer que le problème FGH sera réellement plus difficile à optimiser par PG pour des valeurs de N élevées.

Nous avons calculé la proportion de voisins meilleurs, pires et neutres pour 10^7 génotypes. La Table 3.2 présente ces résultats en pourcentage pour N variant de 1 à 40. On constate que la découverte de voisins meilleurs est presque deux fois moins probable pour N=40 que pour N=1. De plus, il s'agit ici d'une analyse moyenne sur tout le paysage et ce phénomène sera plus important encore pour les génotypes proches de l'optimum. On remarque également que le problème FGH est fortement neutre dès N=1 et que ceci s'accentue pour des valeurs de N élevées.

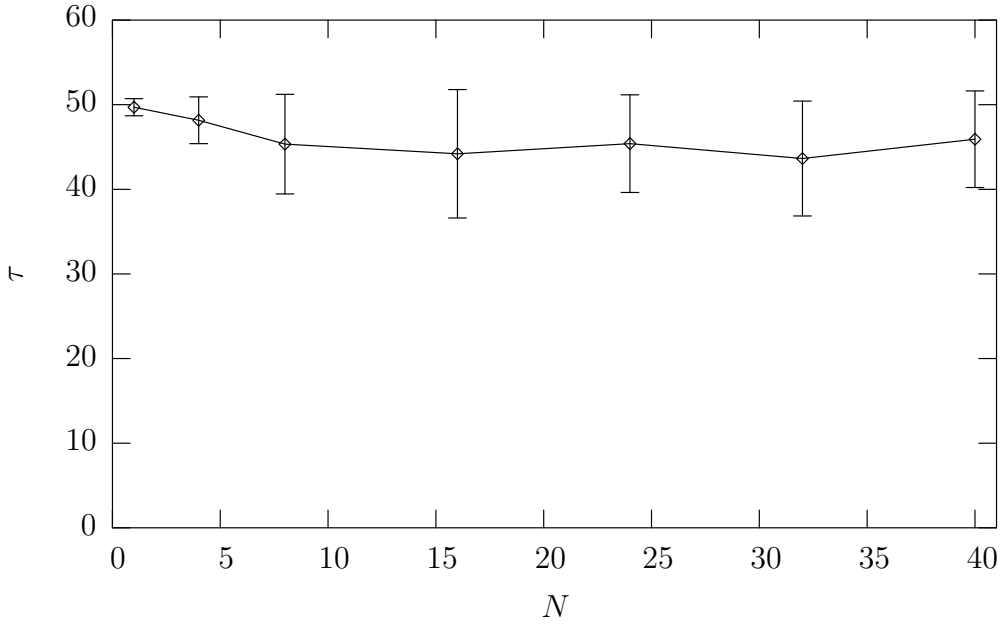

FIG. 3.1 – Paysage FGH, Λ=80 : Longueur de corrélation τ des marches aléatoires en fonction de la taille N de l'alphabet

TAB. 3.2 – Paysage FGH, Λ=80 : Proportions (en %) de voisins Pires, Neutres et Meilleurs

	Λ=80		
N	Pires	Neutres	Meilleurs
1	22.70	54.54	22.74
4	19.10	61.74	19.14
8	17.71	64.50	17.78
16	15.62	68.66	15.70
24	14.41	71.05	14.52
32	13.45	73.03	13.51
40	12.57	74.71	12.70

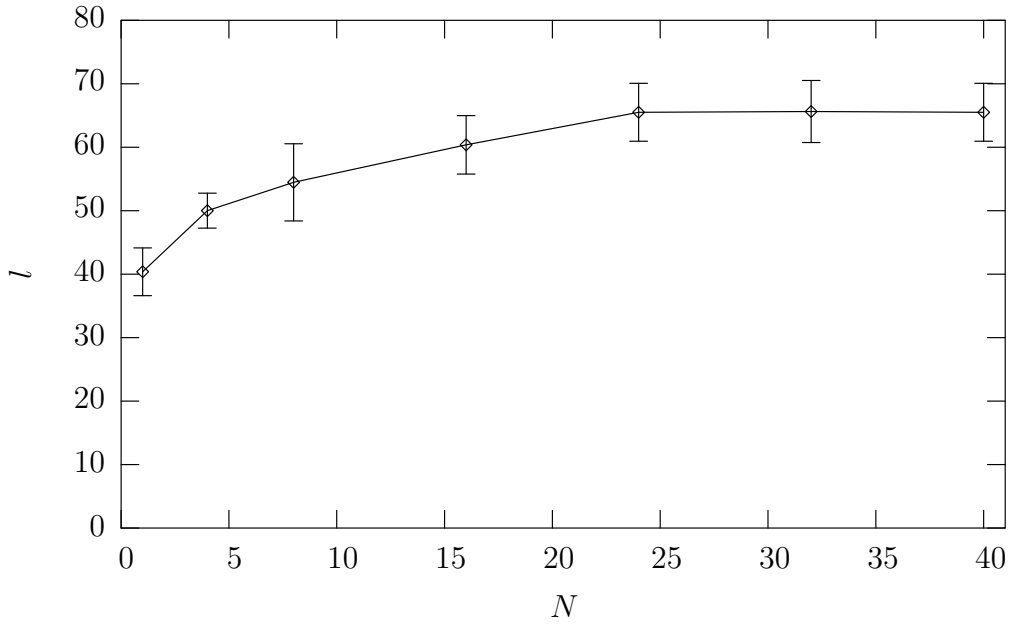

FIG. 3.2 – Paysage FGH, Λ=80 : Longueur l des marches adaptatives en fonction de la taille N de l'alphabet

3.2.2 Routes Royales

Définition

Les paysages *Routes Royales* (RR) ont été conçus pour étudier le fonctionnement des AG [32]. Ils permettent de décrire comment, durant le processus évolutionnaire, des parties de la solution optimale, appelés blocs de construction, sont combinées pour produire des solutions meilleures. Peu de travaux font référence aux RR en PG, on peut citer par exemple le problème *Arbre Royal* [83], conçu pour servir de problème de référence en PGA et qui fut utilisé dans [19] pour étudier des mesures de difficulté mais rien en ce qui concerne les chaînes de tailles variables et donc la PGL. Nous proposons ici un nouveau type de paysage, les paysages *Routes Royales* pour la PGL [28], dont l'objectif est de permettre l'étude des opérateurs de recombinaison, en particulier, de mettre en évidence leurs effets destructifs (ou constructifs) sur les blocs de construction.

Pour définir les RR pour l'EA-TV, nous avons choisi une famille de géno-

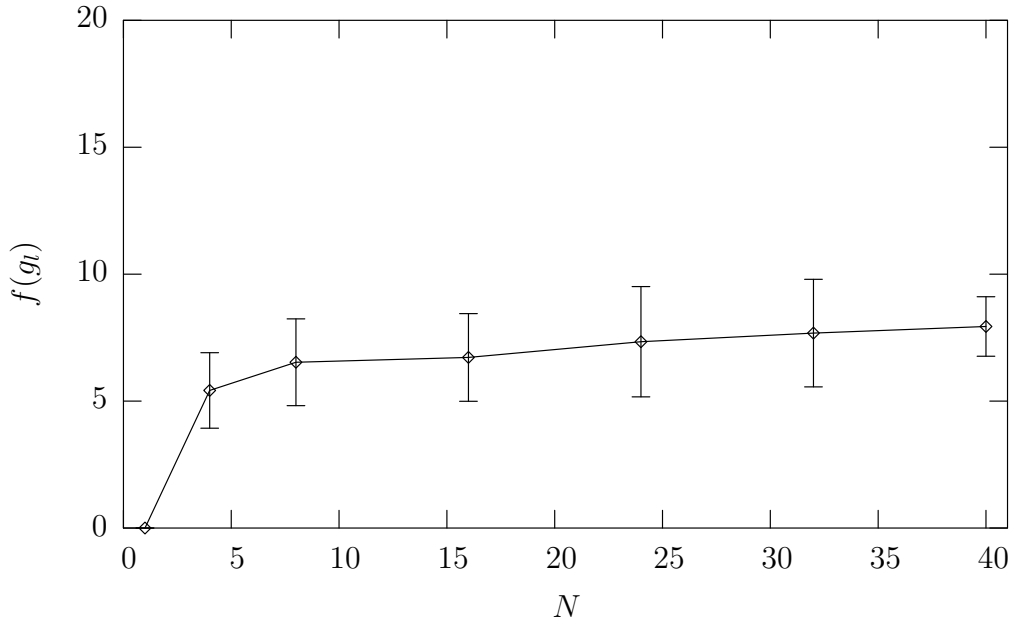

FIG. 3.3 – Paysage FGH, Λ=80 : Fitness d'arrêt $f(g_l)$ des marches adaptatives en fonction de la taille N de l'alphabet

types à qui nous avons affecté une performance optimale puis nous les avons "cassés" en plusieurs blocs de construction (différents).

Formellement, l'ensemble des optima est :

$$G^* = \{g \in G_\Sigma \mid \forall s \in \Sigma,\, B_b(g,s) = 1\},$$

avec

$$B_b(g,s) = \begin{cases} 1 & \text{si } \exists\, i \in [0, \lambda - b[\mid \forall\, j \in [0, b-1],\, g_{i+j} = s, \\ 0 & \text{sinon,} \end{cases}$$

et

- $b \geq 1$ la taille des blocs ;
- N le nombre de blocs opimal ;
- Σ un alphabet de taille N qui définit l'ensemble de tous les symboles s possibles par locus ;
- G_Σ l'ensemble fini de tous les génotypes g de taille $\lambda \leq \lambda_{max}$ [3] définis

[3] λ_{max} doit être supérieur à $N \times b$

sur Σ ;

– g_k le $k^{\text{ième}}$ locus de g.

$B_b(g, s)$ rend compte de la présence (ou de l'absence) dans g d'un bloc de s, c'est-à-dire d'une séquence contiguë d'un même symbole s. On remarque que seule la présence d'un bloc est prise en compte, mais ni sa position ni d'éventuelles répétitions. Le nombre de blocs correspond au nombre de symboles $s \in \Sigma$ pour lesquelles $B_b(g, s)$ vaut 1. La contribution de chaque bloc à la fitness totale est fixe, et donc la fitness $f_{Nb}(g)$ d'un génotype $f_{Nb}(g)$ possédant n blocs est simplement :

$$f_{Nb}(g) = N - n = N - \sum_{i=1}^{N} B_b(g, s_i)$$

On notera que la fonction $f_{Nb}(g)$ doit être minimisée.

Le génotype $g \in G^*$ suivant est un exemple d'optimum avec $\Sigma = \{A, T, G, C\}$, N=4, b=3 et λ=22 :

$$g = \mathbf{AAA}GTA\mathbf{GGG}TAA\mathbf{TTTCCC}TCCC$$

Nous avons mis en caractère gras les séquences qui contribuent à la fitness[4].

Pour pouvoir atteindre efficacement un optimum, un algorithme devra donc créer et combiner des blocs sans détruire les structures déjà existantes. Ce problème est dédié à l'étude des opérateurs de croisement en taille variable et il est important de noter que les éventuelles recombinaison délétères seront liées à la disparition d'un bloc, et jamais à une translocation ou une concaténation. En d'autre termes, il n'y a pas d'épistasie inter-blocs.

Influence des paramètres

Les propriétés du problème RR dépendent du nombre de blocs N et la taille des blocs b. Bien que ces deux paramètres ne soient pas complètement indépendants, ils permettent dans une certaine mesure de contrôler différents aspects du paysage. Le paramètre N détermine la taille de l'espace de recherche mais également la taille du voisinage. De plus, il contrôle le nombre

[4]Bien que la dernière séquence 'CCC' soit un bloc valide, elle ne contribue pas à la fitness puisqu'il s'agit d'une répétition.

de blocs à découvrir, c'est-à-dire l'effort de calcul à fournir pour résoudre le problème. Quant au paramètre b, il a une influence directe sur le degré de neutralité du paysage : plus b est grand, plus la taille des ensembles d'iso-fitness, c'est-à-dire des génotypes ayant la même fitness, est importante.

Analyse du paysage

Cette analyse sera présentée conjointement avec celle des Routes Épistatiques définies ci-après.

3.2.3 Routes Épistatiques

Dans cette section, nous cherchons à définir un problème synthétique plus proche encore des problèmes classiques de l'EA-TV, où un individu peut être vu comme un nombre variable de gènes interagissant entre eux. Pour modéliser ce type d'espace de recherche, on doit donc tout d'abord être capable d'identifier les différents gènes d'un individu puis d'en définir leurs relations. On peut atteindre cet objectif en partant des RR, définies précédemment, où les gènes sont des blocs et en rendant la contribution à la fitness globale de chacun des gènes dépendante de la présence des autres blocs, comme c'est le cas dans les Paysages-NK.

Les Paysages-NK (NK), définis par Kauffman [46], constituent une famille de problèmes permettant d'étudier comment l'épistasie est liée à la rugosité d'un paysage et donc à sa difficulté. L'épistasie correspond au degré d'interaction entre les gènes alors que la rugosité est liée à la densité des optima locaux.

La fonction de fitness d'un NK est une fonction $f_{NK} : \{0,1\}^N \rightarrow [0,1[$ définie sur des chaînes binaires composées de N loci, ie. N gènes. À chaque locus i est associé une composante de fitness $f_i : \{0,1\}^{K+1} \rightarrow [0,1[$. Ces composantes tiennent compte de la valeur du locus i mais également de la valeur de K autres loci (avec $0 \leq K \leq N-1$). On définit la fitness globale $f_{NK}(x)$ de $x \in \{0,1\}^N$ comme la moyenne des valeurs des N composantes de fitness :

$$f_{NK}(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x_i; x_{i_1}, \ldots, x_{i_K})$$

avec $\{i_1, \ldots, i_K\} \subset \{1, \ldots, i-1, i+1, \ldots, N\}$.

Dans un NK, l'épistasie peut donc être ajustée au moyen du paramètre K.

- Pour K=0 : la fonction de fitness est simplement additive.
  - il existe un optimum unique et très attractif ;
  - il existe toujours un voisin meilleur (excepté pour l'optimum global) ;
  - l'optimum global peut-être atteint en $N/2$ pas adaptatifs en moyenne.
- Pour K=$N-1$: la fonction de fitness est équivalente à un assignement aléatoire des valeurs de fitness sur l'espace des génotypes, c'est donc un problème difficile.
  - la probabilité qu'un génotype soit un optimum local vaut $\frac{1}{N+1}$;
  - le nombre d'optima locaux est environ $\frac{2^N}{N+1}$;
  - la distance moyenne entre les optima locaux est approximativement $2ln(N-1)$.

Définition

Nous proposons une extension des RR, appelée *Routes Épistatiques* (RE) [28], où l'épistasie entre les blocs peut-être contrôlée. On sait que plus un problème est épistatique, plus son paysage de fitness est rugueux donc plus il est difficile. Formellement, la fonction de fitness du problème RE est une fonction $f_{NKb} : G_\Sigma \to [0, 1[$ définie sur des génotypes de tailles variables. Comme dans les RR, l'ensemble G^* des optima possèdent N blocs de taille b, cependant la contribution f_i de chacun des blocs dépend de la présence ou de l'absence des autres blocs. Les composantes de fitness f_i sont définies comme dans les paysages NK et la fitness $f_{NKb}(g)$ d'un génotype $g \in G_\Sigma$ est la moyenne des N composantes f_i :

$$f_{NKb}(g) = \frac{1}{N} \sum_{i=1}^{N} f_i(B_b(g, s_i); B_b(g, s_{i_1}), \ldots, B_b(g, s_{i_K}))$$

Dans la pratique, une implémentation d'un NK sera utilisée pour le calcul des f_i et les K liens épistatiques seront choisis aléatoirement dans une chaîne binaire de taille N. La valeur du $i^{\text{ième}}$ bit de cette chaîne est égal à la valeur du prédicat $B_b(g, s_i)$ correspondant. Enfin, nous devons nous assurer que

l'ensemble des génotypes g possédant N blocs correspond bien à la famille des optima globaux G^*. Pour cela, après avoir exploré exhaustivement l'espace des chaînes binaires $\{0,1\}^N$ pour trouver la fitness optimale du NK, nous réalisons une permutation sur cet espace pour que l'optimum devienne 1^N.

Influence des paramètres

Les propriétés du problème RE dépendent des trois paramètres N, K et b. Les paramètres N et b ont exactement le même rôle que dans les RR, c'est-à-dire que N contrôle le nombre de blocs et b leur taille. Le paramètre K permet de contrôler les liens épistatiques entre les différents gènes et par la même le nombre d'optima locaux. Pour K=0, le problème RE sera très semblable au problème RR correspondant puisque l'insertion d'un nouveau bloc induira toujours une augmentation de la fitness. A l'opposé, pour K=$N-1$, avec un haut degré d'épistasie, la grande majorité des "routes" mène vers des optima locaux où l'insertion d'un nouveau bloc induira toujours une diminution de la fitness.

Analyse du paysage

Par construction, nous connaissons bien le rôle des trois paramètres N, K et b, cependant la définition formelle du problème RE ne nous permet pas de déduire directement comment ils interagissent. C'est pourquoi une analyse expérimentale est nécessaire. Nous avons donc réalisé diverses mesures sur le paysage RE avec N=8, 10 et 16, K variant de 0 à $N-1$ et b de 1 à 5. Pour diminuer l'influence de la création aléatoire du NK, les résultats des mesures présentées dans cette partie sont calculés en réalisant une moyenne sur 10 NK différents et pour chacun des couples N et K. Les génotypes initiaux ont été générés en choisissant aléatoirement leur taille entre 1 et $2Nb$ puis en choisissant chaque symbole s dans Σ de taille N. La taille maximum des génotypes autorisées est λ_{max}=100 gènes.

La longueur de corrélation τ , mesurée à partir de 10^3 marches aléatoires de longueur 1000, est présentée Figure 3.4. Pour b petit, on constate qu'elle décroît continûment avec K, ce qui est le résultat attendu puisque K contrôle

la rugosité du paysage[5]. Par contre, quand b augmente la longueur de corrélation n'est plus déterminée par K.

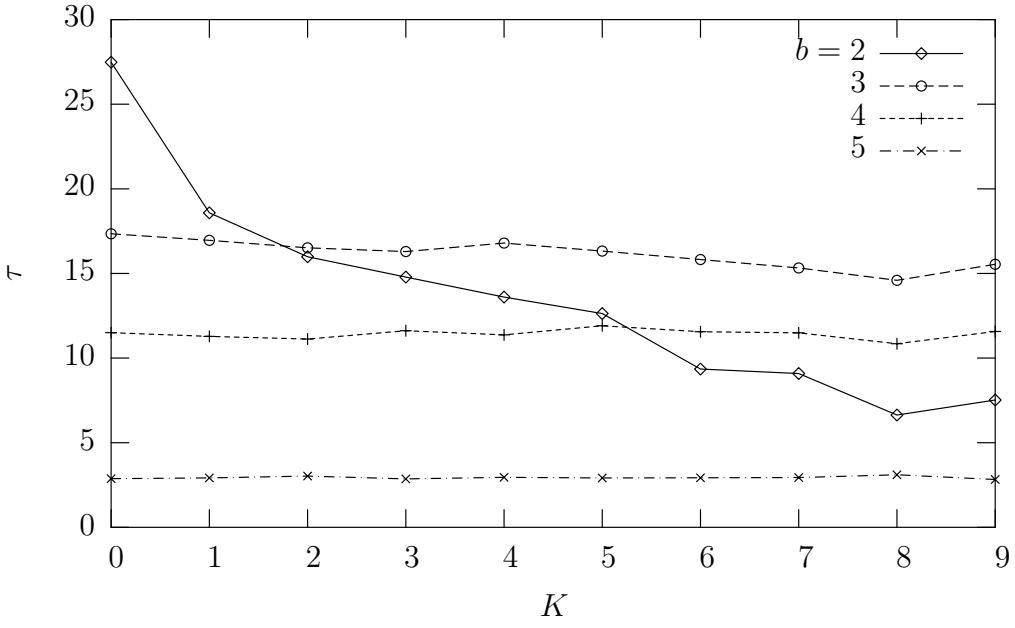

FIG. 3.4 – Paysage RE, N=10 : Longueur de corrélation τ des marches aléatoires en fonction de K

Ces résultats sont confirmés par l'étude de la distance moyenne entre les optima locaux, obtenue à partir de 10^3 marches adaptatives, qui est présentée Figure 3.5. Pour des valeurs de b faibles, la distance varie en fonction de K. Quand b augmente, l'influence de K s'estompe, et les marches deviennent plus courtes car elles s'arrêtent plus souvent sur les plateaux neutres.

Pour mesurer le taux de neutralité du problème RE, nous avons calculé la proportion de voisins neutres pour 10^7 génotypes. La Table 3.3 présente ces résultats en pourcentage pour N=10, K=0 et b variant de 1 à 5. On constate que dès b=1 le problème est majoritairement neutre et que quand b augmente la quasi totalité des voisins sont neutres.

[5]Les valeurs de τ présentées ici peuvent être comparées à celles de la Table 3.1 pour les Paysages-NK.

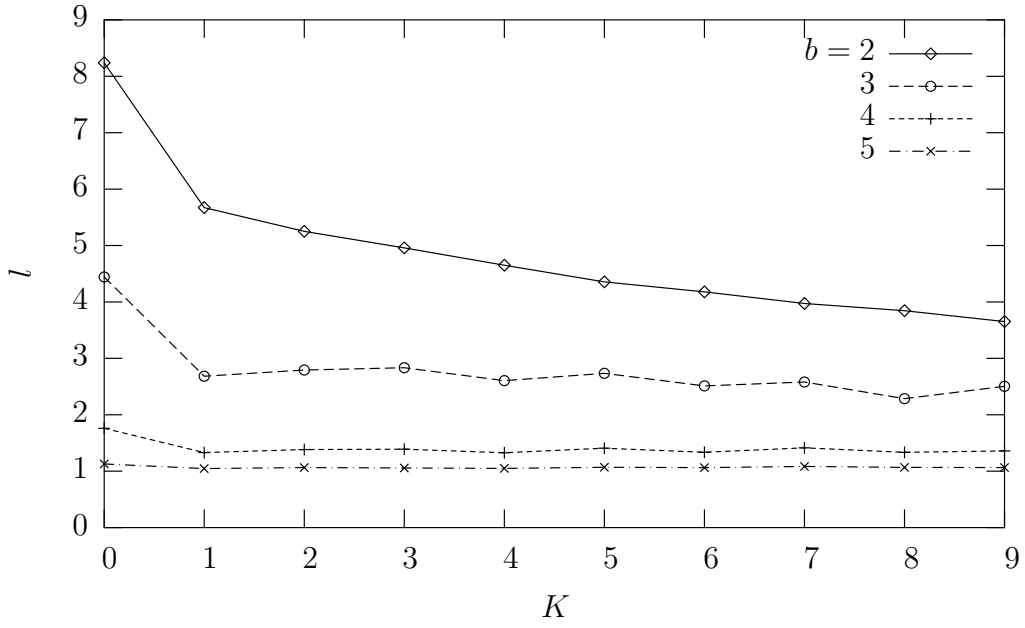

FIG. 3.5 – Paysage RE, N=10 : Longueur l des marches adaptatives en fonction de K

TAB. 3.3 – Paysage RE, N=10 : Proportions (en %) de voisins Pires, Neutres et Meilleurs

	N=10, K=0		
b	Pires	Neutres	Meilleurs
1	15.18	69.739	15.07
2	9.29	81.412	9.28
3	1.78	96.429	1.78
4	0.25	99.495	0.25
5	0.03	99.929	0.03

3.3 Résultats Expérimentaux

Dans cette section, nous comparons les résultats obtenus par PGL avec les opérateurs CS et CMH sur les trois problèmes synthétiques que nous avons introduits. Nous voulons principalement trouver les réglages optimaux des taux de mutation ($\mathcal{T}_m$) et de croisement ($\mathcal{T}_c$), c'est-à-dire les réglages qui donnent les meilleurs résultats en terme de fitness moyenne, c'est pourquoi nous avons fait varier $\mathcal{T}_m$ (de 0 à 2) et $\mathcal{T}_c$ (de 0 à 1). On notera que pour $\mathcal{T}_m$ =1, chaque individu impliqué dans le processus de reproduction subira, en moyenne, une mutation de chaque type (insertion, délétion et substitution). On notera également que le processus de recombinaison entraîne toujours la création de deux individus. Pour chacun de ces réglages, 50 expériences indépendantes sont réalisées. Un test de Student, avec 95% de confiance, est utilisé pour déterminer si les résultats obtenus sont statistiquement différents les uns des autres.

3.3.1 Paramètres de l'algorithme

Nous utilisons une population de 1000 individus créés aléatoirement. Dans la population initiale, la taille de création des individus est distribuée uniformément entre 1 et λ_c=50 gènes choisis aléatoirement dans un alphabet de N symboles, N étant dépendant du problème traité. Durant les expériences, la taille des individus est limitée à λ_{max}=100 gènes. L'évolution, avec élitisme, sélection par tournoi de 16[6] individus et remplacement *steady-state* est limitée à T générations[7], T étant dépendant du problème traité.

3.3.2 Fitness Guidée par l'Homologie

Nous avons réalisé un ensemble d'expériences sur le problème FGH avec T=50 générations, la taille de l'optimum Λ=80 et la taille de l'alphabet N=10. On rappelle que la longueur des marches adaptatives pour Λ=80 et

[6] L'influence de la taille du tournoi sera étudiée Section 4.1.4.

[7] Dans un système steady-state, le concept de génération est artificiel et n'est utilisé que pour permettre la comparaison avec les systèmes générationels. Ici, une génération correspond à un nombre de remplacements égal au nombre d'individus dans la population, *i.e.* 1000.

N=10 (cf fig 3.2) est supérieure à 50, ce qui signifie que si l'opérateur de croisement utilisé n'a pas d'effets positifs sur la convergence, l'optimum sera rarement découvert.

TAB. 3.4 – Meilleurs résultats trouvés sur le problème FGH.

Type de Croisement	Fitness Moyenne	Taux de Succès	Génération Moyenne
CS	$2.2_{(\sigma=1.5)}$	10%	49.4
CMH	$\mathbf{0.1}_{(\sigma=0.4)}$	**90**%	**28.6**

La Table 3.4 rapporte les résultats trouvés avec les deux opérateurs CS et CMH pour les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$. On rappelle que le but est de minimiser la fitness puisqu'il s'agit de la distance à l'optimum. On constate que pour ce problème, le CMH est plus efficace que l'opérateur CS, puisque l'optimum est découvert beaucoup plus souvent, environ 90% des expériences, avec CMH et seulement 10% dans le cas du CS. De plus, nous avons rapporté la génération moyenne de découverte de l'optimum, on constate qu'en utilisant le CMH, la convergence est également plus rapide.

Les figures 3.6 et 3.7 présentent la fitness moyenne du meilleur individu trouvé en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour les deux opérateurs de croisement. Ce type de figure permet de visualiser les zones d'efficacité des opérateurs. Dans les deux cas, l'application de l'opérateur de mutation est nécessaire mais $\mathcal{T}_m$ doit rester limité. Par contre, le taux de recombinaison optimal est faible pour CS (0.2) et très élevé pour CMH (1.0). Un taux de croisement élevé associé à un taux de mutation relativement faible est proche de ce qui est préconisé pour les AG. Ce constat ainsi que les résultats expérimentaux sur ce problème tendent à confirmer que le CMH induit une dynamique de recherche plus proche de celle de l'opérateur de recombinaison des AG que de celle du CS. En effet, dans les AG, la corrélation entre la fitness et la distance à l'optimum a été proposée comme une mesure de difficulté des paysages (cf.[44]).

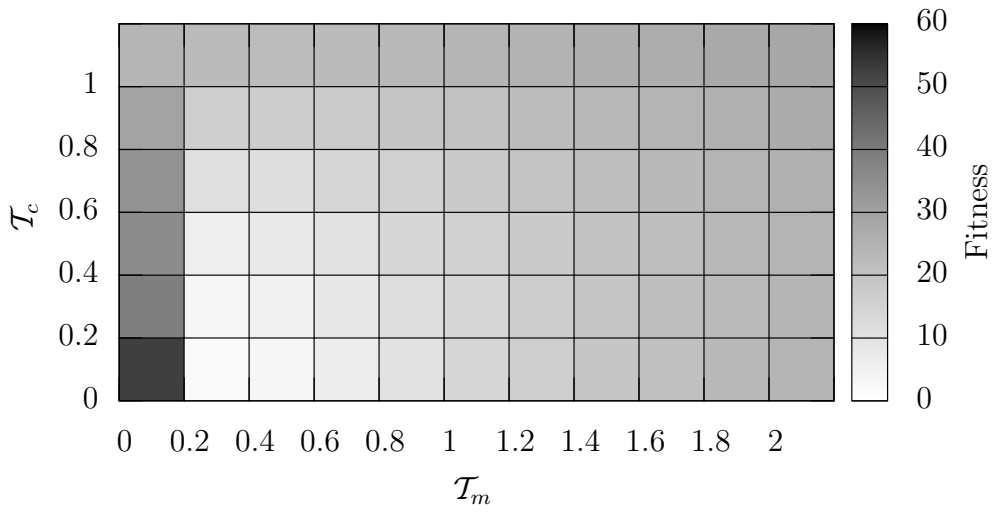

FIG. 3.6 – Problème FGH, Λ=80, N=10 et CS : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

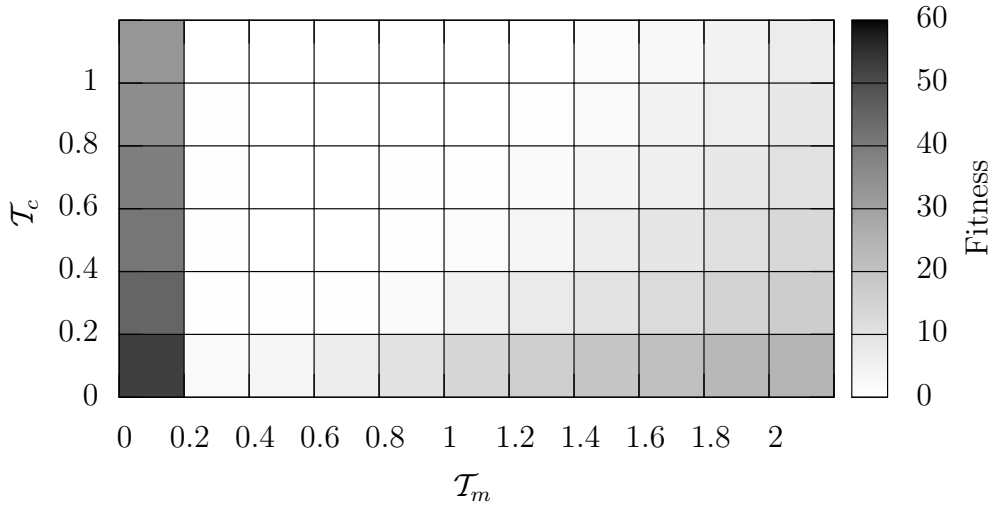

FIG. 3.7 – Problème FGH, Λ=80, N=10 et CMH : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

Effets du croisement

On s'intéresse aux effets du croisement dans le processus de recherche, en particulier quelle part de la convergence et de l'amélioration de la fitness durant l'évolution est due à la recombinaison. Dans ce qui suit, nous distinguerons, (cf. Figure 3.8) :

- les croisements dits Avantageux (notés A), pour lesquels au moins un enfant (E_i) est plus performant que le meilleur de ses parents (P_1) ;
- les croisements dits Délétères (notés D), pour lesquels les deux parents (P_1 et P_2) sont plus performants que le meilleur de leurs enfants ;
- les croisements dits Neutres (notés N), pour les autres cas.

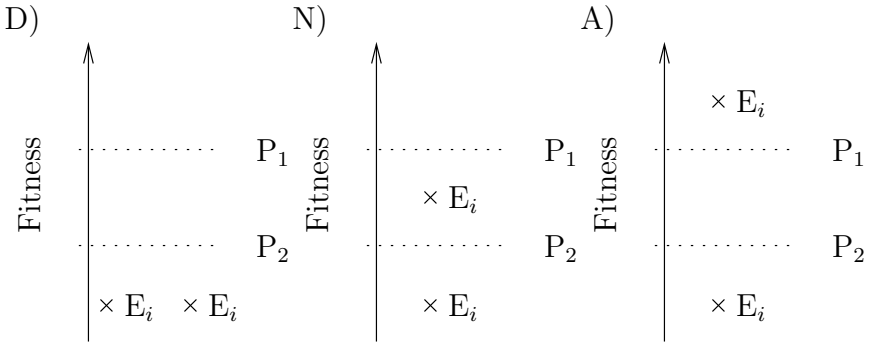

FIG. 3.8 – Exemples de croisements Délétères, Neutres et Avantageux.

Nous avons rapporté, Table 3.5, les proportions moyennes (exprimées en pourcentage) des croisements de type D, N et A obtenues pour le réglage optimal de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour CS et CMH. On constate que les deux opérateurs ont des effets très différents, puisque la quasi totalité des croisements réalisés avec l'opérateur CS entraînent une dégradation de la fitness, alors qu'avec l'opérateur CMH, la très grande majorité des croisements sont neutres[8]. De plus, on constate une proportion de croisements A significative quand l'opérateur CMH est utilisé.

Nous avons également tracé les effets des deux opérateurs en fonction de la fitness parentale (cf. Figures 3.9 et 3.10). Les valeurs de fitness élevées

[8]On note que pour pour N=10, l'analyse du paysage a montré qu'environ 65% des voisins sont neutres.

TAB. 3.5 – Proportions (en %) des croisements Délétères, Neutres et Avantageux sur le problème FGH.

Type de Croisement	Λ=80, N=10		
	D	N	A
CS	93.66	5.18	1.15
CMH	0.17	93.00	6.82

correspondent au début des expériences, où la probabilité de détruire des structures existantes est faible, alors que les valeurs de fitness faibles correspondent à la fin, où les individus recombinés sont complexes et la probabilité d'améliorer leurs performances est plus faible.

Dans le cas du CS, les croisements de type D, N et A sont en proportion comparable au début de l'évolution, puis, rapidement, la proportion de croisements D augmente pour dépasser 90%, tandis que la proportion de croisements N diminue pour atteindre un niveau résiduel d'environ 5%. Enfin, la proportion de croisements A est rapidement négligeable et laisse supposer que les gains en fitness obtenus durant la convergence ne peuvent être imputés à l'opérateur de recombinaison.

Les effets du CMH sont, pour les valeurs de fitness élevées, majoritairement neutres avec une proportion de croisement A significative, autour de 40%. Les proportions des croisements A et N ont par la suite une évolution symétrique où les croisements N deviennent largement prépondérants tandis que les croisements A tendent lentement à disparaître. On remarque que la proportion de croisements D est extrêmement faible et qu'ils n'apparaissent qu'au début de la convergence, sans doute quand des individus trop peu homologues sont recombinés, c'est-à-dire que peu de structures communes existent et donc ne peuvent être conservées.

3.3.3 Routes Royales

Nous avons réalisé un ensemble d'expériences sur le problème RR avec T=400 générations, pour N=8, 10, 16 et b=5.

La Table 3.6 présente les résultats obtenus avec les deux opérateurs CS

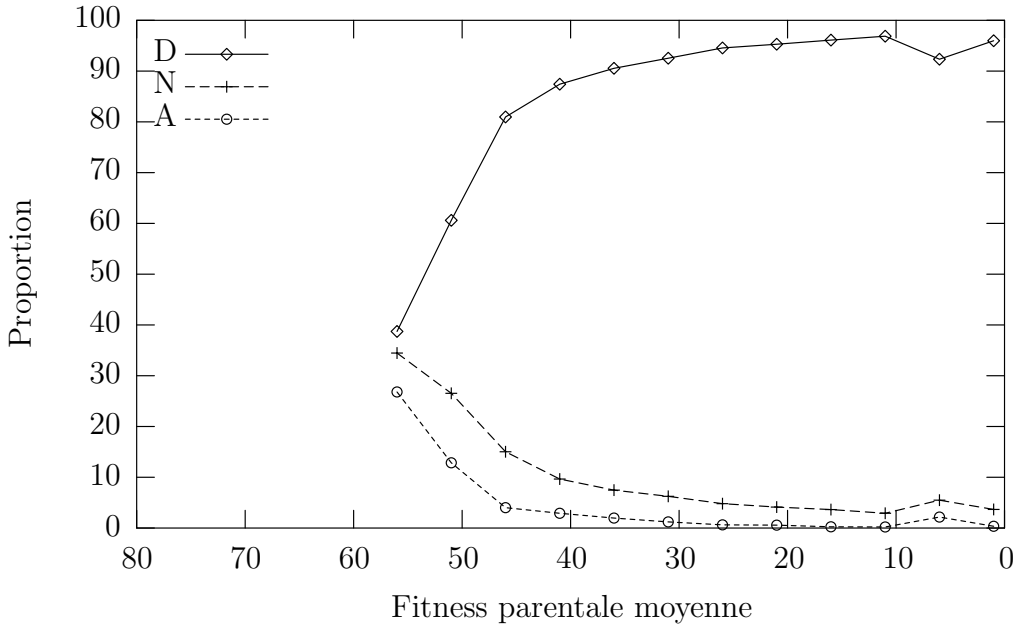

FIG. 3.9 – Problème FGH, Λ=80, N=10 et CS : Proportions (en %) des croisements Délétères, Neutres et Avantageux en fonction de la fitness parentale moyenne.

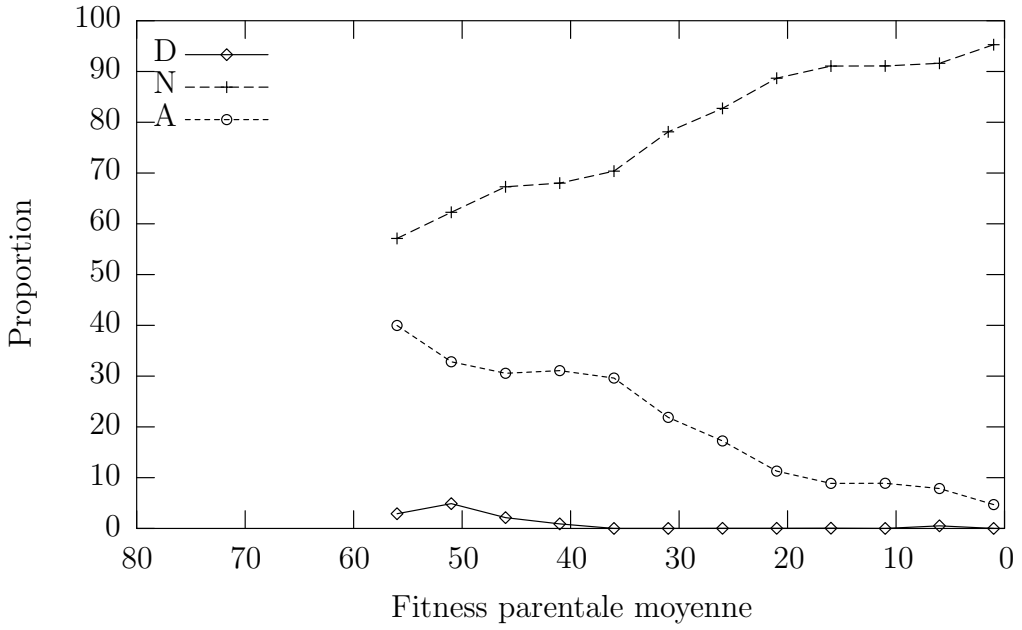

FIG. 3.10 – Problème FGH, Λ=80, N=10 et CMH : Proportions (en %) des croisements Délétères, Neutres et Avantageux en fonction de la fitness parentale moyenne.

TAB. 3.6 – Meilleurs résultats trouvés sur le problème RR.

Type de Croisement	Fitness Moyenne	Taux de Succès	Génération Moyenne
N=8			
CS	$0.0_{(\sigma=0)}$	100%	$131.1_{(\sigma=42.7)}$
CMH	$0.0_{(\sigma=0)}$	100%	$\mathbf{82.4}_{(\sigma=32.4)}$
N=10			
CS	$1.0_{(\sigma=0.6)}$	36%	$367.5_{(\sigma=73.4)}$
CMH	$\mathbf{0.4}_{(\sigma=0.5)}$	**62**%	$\mathbf{308.7}_{(\sigma=71.8)}$
N=16			
CS	$8.8_{(\sigma=0.6)}$	0%	-
CMH	$\mathbf{7.8}_{(\sigma=0.6)}$	0%	-

et CMH pour les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$. La valeur de fitness, qui correspond au nombre moyen de blocs non trouvés et doit donc être minimisée, indique une performance supérieure du CMH par rapport au CS pour les différentes expériences réalisées. Les taux de succès montrent que la difficulté du problème est croissante avec N. Ceci s'explique d'une part par une probabilité de découvrir un bloc plus faible pour N grand, mais également parce que, par exemple pour le cas N=16, la taille de l'optimum représente 80% de la taille maximum autorisée, ce qui constitue une contrainte suplémentaire pour l'algorithme. Enfin, la génération moyenne de découverte de l'optimum est inférieure dans le cas du CMH, indiquant une convergence plus rapide.

Les Figures 3.11 et 3.12 présentent la fitness moyenne du meilleur individu trouvé en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour les deux opérateurs de croisement sur le problème RR avec N=10 ; l'influence de $\mathcal{T}_m$ et $\mathcal{T}_c$ étant comparable pour les autres valeurs de N testées. On remarque que pour CS et CMH, les meilleures performances ont été obtenues avec $\mathcal{T}_m$ très élevé (2.0). Ceci est vraisemblablement lié à la recherche de nouveaux blocs qui, n'étant pas guidée par la fitness, nécessite une forte exploration locale. Quant à l'influence de $\mathcal{T}_c$, elle diffère totalement entre les opérateurs CS et CMH. En effet, quand CS est utilisé, le réglage optimal de $\mathcal{T}_c$ est très faible (0.2) alors qu'il est maximal (1.0) avec le CMH.

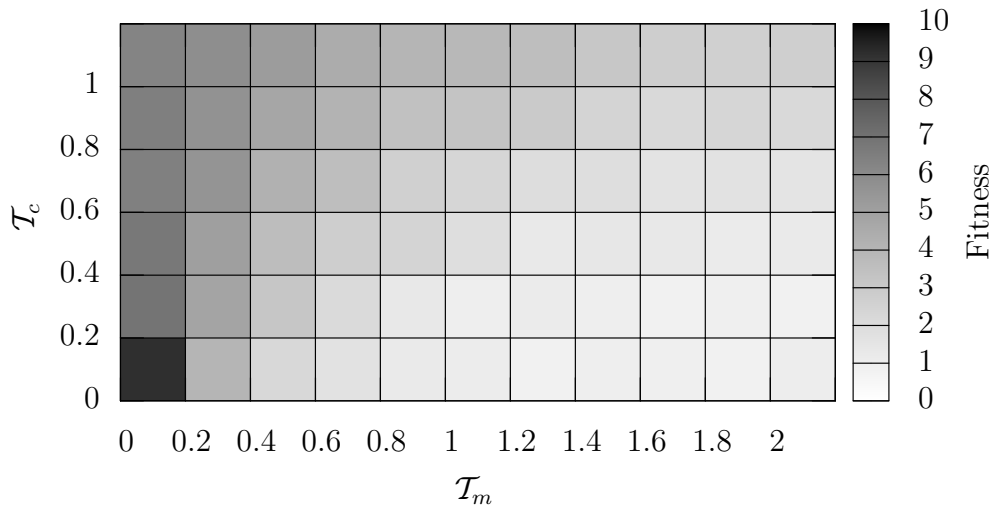

FIG. 3.11 – Problème RR, N=10 et CS : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

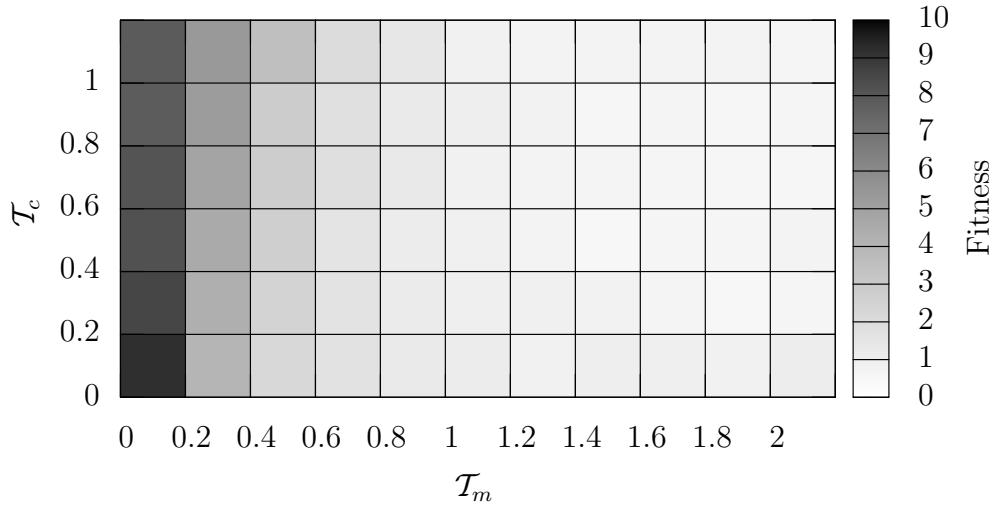

FIG. 3.12 – Problème RR, N=10 et CMH : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

Évolution de la taille

Pour N=8 et $b = 5$, le plus petit des optima est de taille 40 et donc, pour résoudre ce problème l'algorithme n'a pas besoin de manipuler des génotypes de grandes tailles. Cependant, en étudiant la taille moyenne du meilleur individu dans la population finale en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ (cf. Figures 3.13 et 3.14), on constate que dans la grande majorité des expériences réalisées avec le CS, la taille maximum autorisée (100 gènes) est atteinte. Plus précisément, dès que l'opérateur de recombinaison est appliqué ($\mathcal{T}_c \geq 0.2$), et indépendamment de $\mathcal{T}_m$ (sauf pour $\mathcal{T}_m$ =0), la taille moyenne est supérieure à 90 gènes. On reconnaît ici le phénomène de bloat propre au CS. Au contraire, pour les expériences réalisées avec le CMH, c'est l'opérateur de mutation qui influe le plus sur le nombre de gènes. L'opérateur CMH est donc plus parcimonieux que le CS sur ce problème.

Pour N=16 et b=5, les performances du CMH sont assez proche de celles du CS. On peut expliquer ce phénomène par un excès de parcimonie, c'est-à-dire par une difficulté spécifique pour l'opérateur CMH d'orienter la recherche vers des individus composés d'au moins 80 gènes, la taille du plus petit des optima. La Figure 3.15 montre l'évolution de la taille moyenne des individus pour les opérateurs CS et CMH. On note qu'avec l'opérateur CS la taille croit très rapidement et ne semble être limitée que par la taille maximum autorisée (100 gènes). Dans le cas du CMH, l'évolution est plus lente et n'atteint la valeur critique de 80 que tardivement entre les générations 250 et 300.

Effets du croisement

Nous avons rapporté, Table 3.7, les proportions moyennes (exprimées en pourcentage) des croisements D, N et A obtenues pour le réglage optimal de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour CS et CMH. On constate que les deux opérateurs ont des effets très différents, puisque la quasi totalité, autour de 80%, des croisements réalisés avec l'opérateur CS entrainent une dégradation de la fitness, alors qu'avec l'opérateur CMH, la très grande majorité, plus de 90%, des croisements sont

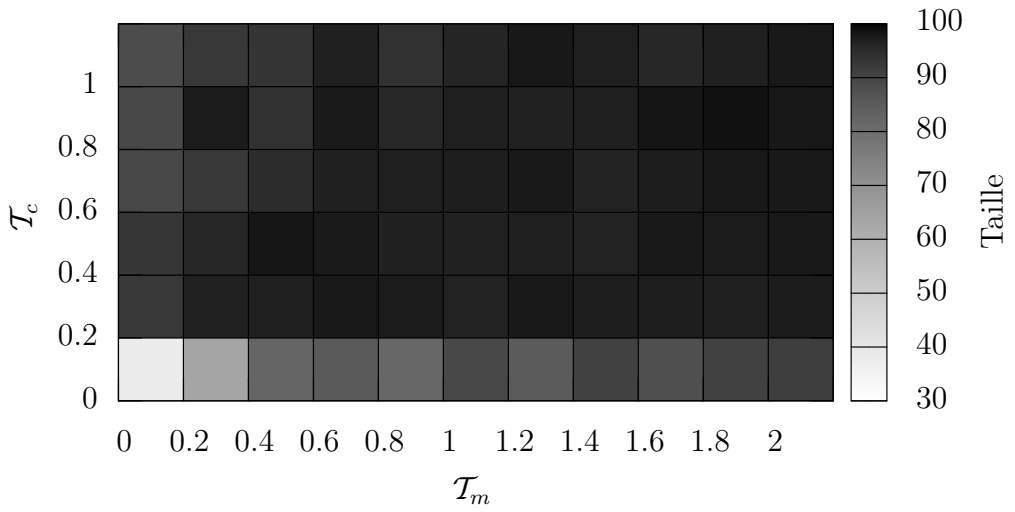

FIG. 3.13 – Problème RR, N=8 et CS : Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

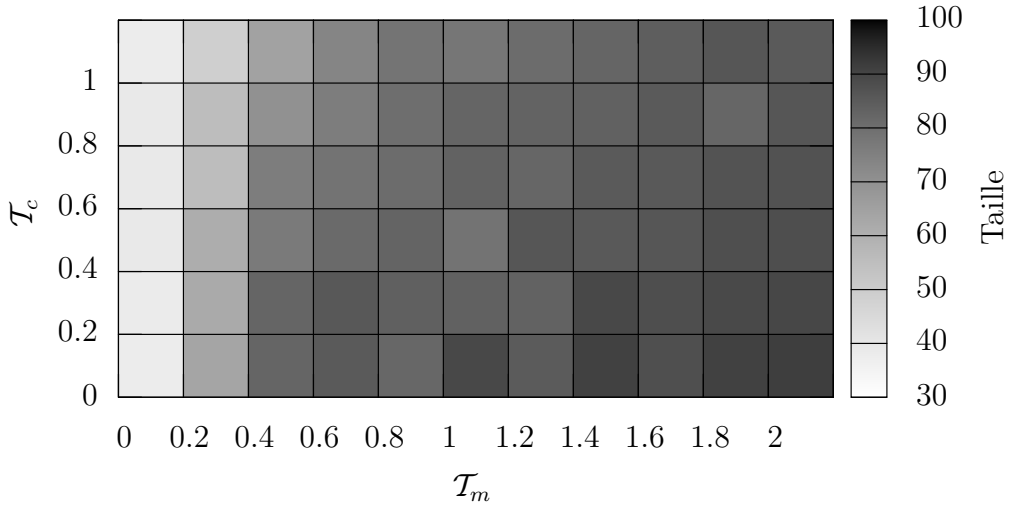

FIG. 3.14 – Problème RR, N=8 et CMH : Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

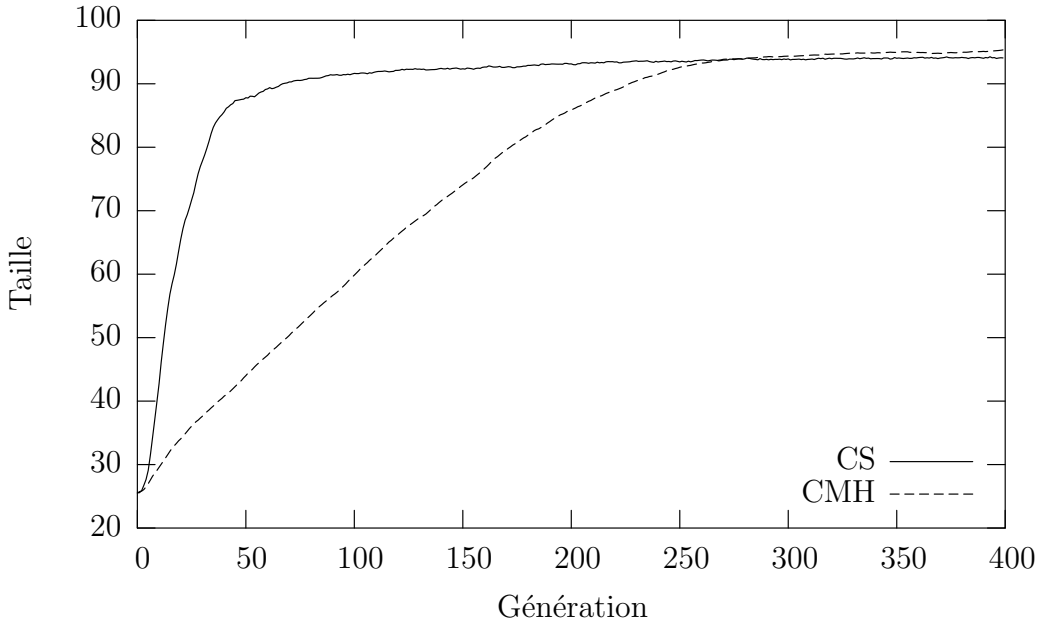

FIG. 3.15 – Problème RR, N=16 : Évolution de la taille moyenne.

neutres [9]. De plus, on constate une proportion de croisements A très faible, mais significative, quand l'opérateur CMH est utilisé.

TAB. 3.7 – Proportions (en %) des croisements Délétères, Neutres et Avantageux sur le problème RR.

Type de Croisement	N=8 et K=5			N=16 et K=5		
	D	N	A	D	N	A
CS	79.47	20.43	0.09	76.12	23.78	0.08
CMH	9.21	90.68	0.29	7.44	93.33	0.21

Nous avons également tracé les effets des deux opérateurs en fonction du nombre de blocs parental moyen. (cf. Figures 3.16 et 3.17). Les nombres de blocs élevés correspondent à la fin des expériences, où les individus recombinés sont complexes et la probabilité d'améliorer leurs performances est plus faible.

[9] On note que pour pour N=8 ou 16, l'analyse du paysage a montré qu'environ 99.9% des voisins sont neutres.

L'opérateur CS est massivement neutre au début de la convergence, puis devient majoritairement délétère avec l'augmentation du nombre de blocs, et donc une probabilité de détruire un bloc plus élevée. Cependant, on constate une proportion de croisements N toujours significative et jamais inférieure à 30%. Enfin, les croisements A sont peu nombreux et disparaissent entre 5 et 6 blocs. Les effets du CMH sont, au début des expériences, fortement neutres avec une proportion de croisement A significative, plus de 10%. Les proportions des croisements A et N ont par la suite une évolution symétrique où la proportion de croisements N augmentent tandis que les croisements A tendent lentement à disparaître. On remarque que la proportion de croisements D est extrêmement faible et qu'ils n'apparaissent qu'au début de la convergence, sans doute quand des individus trop peu homologues sont recombinés, c'est-à-dire que les blocs contenus dans les génotypes parentaux différent et ne sont pas protégés par le processus d'alignement.

Les problèmes synthétiques du type des RR ont pour objectif, comme dans les AG, de confirmer l'hypothèse des blocs de construction qui suppose que des blocs de construction présents dans les individus lors d'un croisement peuvent être combinés pour former des blocs plus longs et ainsi créer des individus meilleurs. On constate que le comportement de l'opérateur CS ne permet pas de tenir cette hypothèse puisque CS tend à casser les blocs de grande dimension. Au contraire, les résultats expérimentaux du CMH montrent qu'il préserve les blocs et permet parfois de les recombiner, même quand ceux-ci représentent une part importante du génome, comme quand N=16 et b=5, où 80 gènes, sur 100 au maximum, participent à l'évaluation de la performance.

3.3.4 Routes Épistatiques

Nous avons réalisé une série d'expériences sur le problème RE sur T=400 générations, pour N=8, 10, 16, b=5 et K variant de 0 à $N/2$. Les réglages de $\mathcal{T}_m$ et $\mathcal{T}_c$ sont choisis d'après les résultats obtenus sur le problème RR, c'est-à-dire $\mathcal{T}_m$ =2.0, identique pour les expériences avec CMH et CS, et $\mathcal{T}_c$ =1.0 pour le CMH alors que $\mathcal{T}_c$ =0.2 pour le CS.

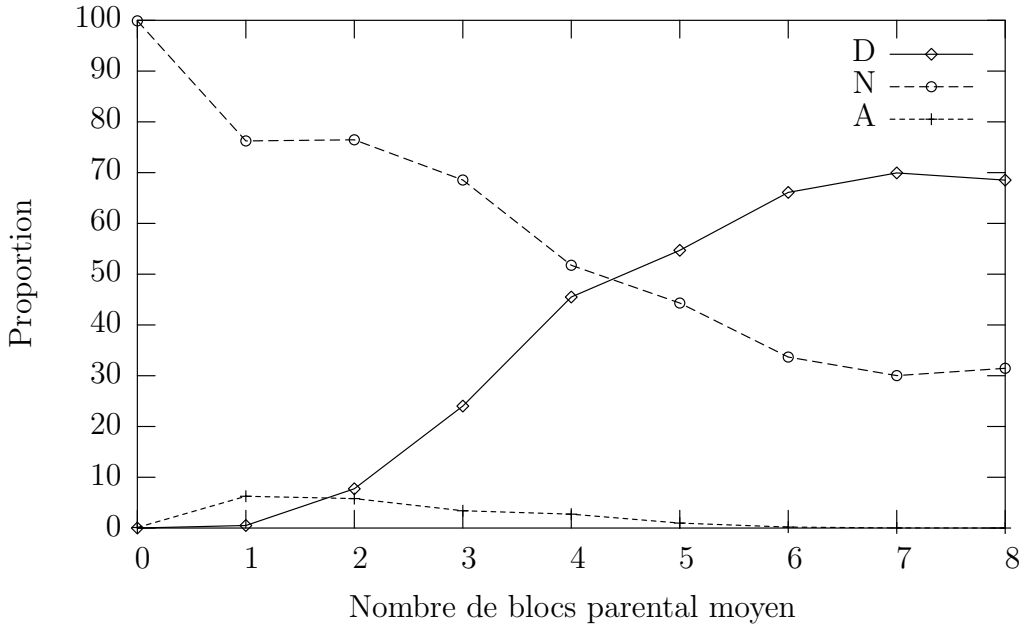

FIG. 3.16 – Problème RR, N=8 et CS : Proportions (en %) des croisements Délétères, Neutres et Avantageux en fonction du nombre de blocs moyen parental moyen.

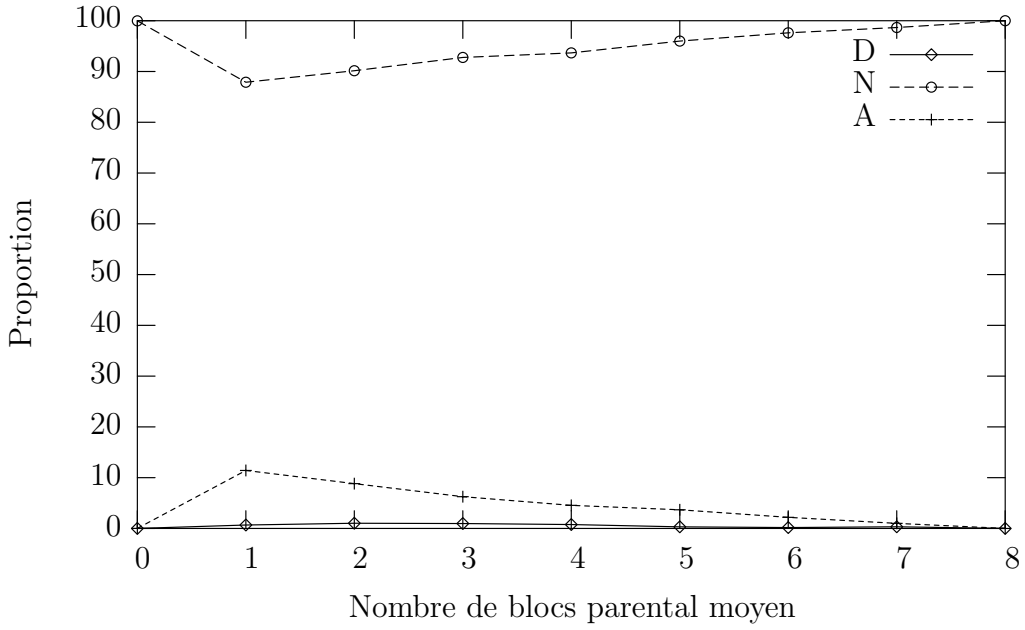

FIG. 3.17 – Problème RR, N=8 et CMH : Proportions (en %) des croisements Délétères, Neutres et Avantageux en fonction du nombre de blocs moyen parental moyen.

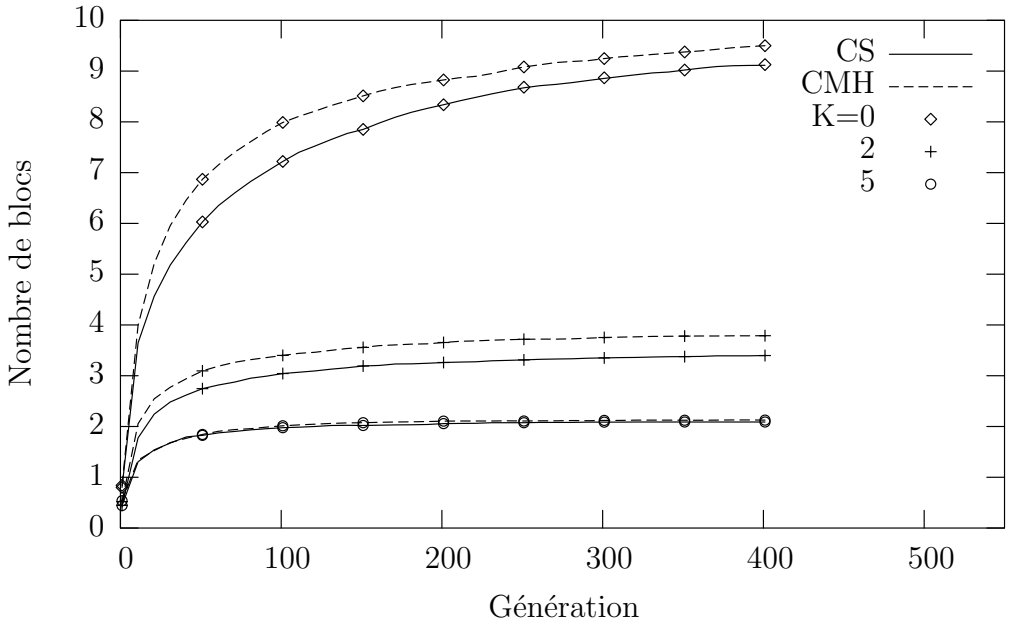

FIG. 3.18 – Problème RE, N=10 : Évolution du nombre de blocs trouvés pour CS et CMH.

Nous avons tracé (cf. Figure 3.18) l'évolution du nombre moyen de blocs trouvés pour N=10 [10] et K=0, 2, 5. On note, conformément à nos attentes, une difficulté croissante avec K puisque le problème est presque complètement résolu pour K=0 alors que moins de deux blocs sont découverts en moyenne pour K=5. On sait que pour K grand, il n'existe pas de routes vers l'optimum par agrégations successives de blocs car les liens épistatiques qui rendent dépendante la contribution de chacun des blocs à la présence (ou à l'absence) des autres impose une dynamique particulière de création et de suppression des blocs que ni le CMH ni le CS ne peuvent assurer. Pour K=0 et 2, le CMH permet de trouver, en moyenne, un demi bloc de plus que lorsque le CS est utilisé, cependant cette différence est nulle pour K=5.

Le paramètre K permet de contrôler la rugosité du paysage RE, c'est-à-dire le nombre d'optima locaux. Contrairement au paysage RR où un optimum unique existe et peut-être atteint en augmentant continûment le nombre

[10] Les autres valeurs de N testées ont donné des résultats comparables.

de blocs différents dans les programmes, la présence d'optima locaux dans les paysages RE implique que le système doit être capable à la fois de conserver certains blocs et d'en détruire d'autres.

En comparant les courbes d'évolution des deux opérateurs, on remarque que l'avantage du CMH sur le CS apparaît dès les premières générations puis que celui-ci reste stable au cours de l'évolution. On peut raisonnablement supposer que la convergence s'effectue en fait en deux temps. Au début du processus, les blocs présents dans la population initiale sont assemblés pour former des individus plus performants : c'est là que le comportement homologue du CMH est prépondérant. Puis, dans un deuxième temps, quand la population s'est homogénéiser, le rôle de l'opérateur de croisement diminue et c'est l'opérateur de mutation qui permet d'obtenir d'éventuels gains en fitness.

3.4 Conclusion

Dans ce chapitre nous proposons trois problèmes synthétiques pour l'Évolution Artificielle en Taille Variable dont certaines caractéristiques, comme la neutralité et la rugosité, peuvent être ajustées. Tout d'abord, nous avons voulu vérifier, à l'aide de la métaphore des paysages de fitness et de mesures de difficulté sur ces paysages, si les paramètres d'ajustement des problèmes avaient les effets attendus. Nous avons vu que la présence de neutralité affecte la qualité des mesures ce qui rend délicate leur utilisation pour prédire la difficulté des problèmes en PG.

Sur ces trois problèmes, nous avons comparé les performances de l'opérateur de Croisement utilisé de façon Standard (CS) en PGL et de l'opérateur de Croisement par Maximum d'Homologie (CMH). Nous avons vu que quand la fitness et la distance à l'optimum sont fortement corrélées dans un paysage de recherche, et en dépit d'une neutralité élevée, le CMH est plus efficace que le CS. Nous pensons que cela illustre la propriété théorique du respect (cf Équation 2.11) basée sur la distance dans les ensembles de recombinaison. Nous avons observé également que le CMH préserve les structures communes durant les opérations de recombinaison, ce qui permet d'exploiter, mieux que

dans le cas de l'opérateur CS, les différentes qualités des individus présents dans la population. C'est ici la propriété de conservation de l'homologie (cf. Section 2.3.1) par le CMH qui est mise en évidence expérimentalement. Cependant, cet avantage du CMH sur le CS ne semble plus décisif si l'épistasie du problème à résoudre est trop élevée. Ceci s'explique par l'incapacité du CMH à détruire les structures communes qui le rend fortement sensible à la présence d'optima locaux dans les paysages et au phénomène de convergence prématurée de la population.

L'étude des effets délétères du croisement sur la fitness confirme empiriquement notre hypothèse de départ, sur les relations entre la brutalité de la recombinaison et l'homologie. En effet, contrairement au CS dont les effets sont majoritairement délétères, l'opérateur CMH est très souvent sélectivement neutre et permet même d'espérer des gains en fitness plus fréquents. De plus, conformément à l'hypothèse de protection (cf. Section 1.2.3) qui considère que la croissance incontrôlée des programmes, appelée bloat, est liée à une forme de protection de la population vis à vis de la brutalité de la recombinaison, l'utilisation du CMH, c'est-à-dire, comme nous l'avons vu, d'un opérateur moins brutal, diminue significativement la taille des programmes dans la population. Cependant, cette limitation pourrait poser certaines difficultés, selon le problème à résoudre, par exemple en ne permettant pas au système d'explorer suffisamment l'espace de la taille des programmes si cela s'avérait nécessaire.

Les problèmes synthétiques présentés dans ce chapitre ont l'avantage d'être par construction bien définis et de mettre en évidence certaines propriétés spécifiques des opérateurs génétiques. Cependant, ils sont loin de refléter entièrement la complexité des problèmes réels, c'est pourquoi l'étude expérimentale des propriétés du CMH sera poursuivie dans le prochain chapitre.

Chapitre 4

Homologie, taille et diversité des programmes

Dans ce chapitre, nous nous intéressons aux propriétés des populations recombinées par les deux opérateurs CS et CMH. Nous avons déjà vu que sur un paysage plat, la distribution de la taille des programmes diffère nettement entre les deux opérateurs. Nous voulons maintenant connaître, quand une pression de sélection est appliquée, comment le nombre d'instructions varie dans la population et plus généralement comparer l'évolution des populations recombinées, en particulier en ce qui concerne la diversité des programmes. Pour cela, nous utiliserons un problème de Régression Symbolique Booléenne qui fait référence en PG. Par la suite nous verrons comment grâce au CMH il est possible de contrôler la taille des programmes pour lutter contre le phénomène de bloat tout en garantissant un bon compromis fitness *vs* taille. Enfin, nous discuterons de ce qui différencie la recherche effectuée par les deux opérateurs CMH et CS.

4.1 Propriétés des populations recombinées

Pour confirmer les résultats expérimentaux obtenus avec le CMH et le CS dans le chapitre précédent et étudier les populations recombinées, nous utilisons un problème de Régression Symbolique Booléenne appelé Even Parity.

4.1.1 Régression Symbolique Booléenne

La régression symbolique consiste à rechercher une expression mathématique, dans notre cas sous la forme d'un programme de PGL, qui permette d'approximer au mieux (ce qui reste à définir) une fonction donnée. De manière générale, cette fonction n'est pas connue et la recherche s'effectue à partir d'un ensemble d'exemples, le jeu d'apprentissage, illustrant les relations entre les entrées et les sorties de la fonction à approximer.

En Régression Symbolique Booléenne (RSB), la fonction recherchée est une fonction dont les entrées et les sorties sont binaires. Des problèmes de référence de ce type existent depuis les origines de la PG [54], dans les domaines du multiplexage et de la parité par exemple, et ont pour objectif l'évaluation de la capacité de la PG à découvrir une expression logique directement utilisable en électronique programmable. Nous nous intéressons à la RSB tout d'abord parce qu'elle constitue une première étape vers la Régression Symbolique Réelle, qui est un de nos objectifs. Le choix du jeux d'instructions est simple et il n'existe pas de problématiques spécifiques à la génération et à l'optimisation des constantes. De plus des travaux récents [81] ont proposé une implémentation très efficace de l'exécution et de l'évaluation de la fitness de programmes booléens, basée sur le constat qu'un unique processeur n bits peut être vu comme n processeurs 1 bit fonctionnant en parallèle. Cette implémentation a permis dans notre cas d'accélérer le système en divisant par 32 le temps de calcul.

Définition du problème

Dans cette section, nous traitons un problème de parité appelé en anglais Even Parity dont l'objectif est de découvrir une fonction E recevant en entrée un vecteur $(b_1, b_2, \ldots, b_N)$ de N bits et renvoyant 1 si le nombre de bits à 1 dans ce vecteur est paire et 0 sinon. Une expression simple utilisant des `OU` exclusifs (`XOR` en anglais)[1], notés $\oplus$, permet de résoudre ce problème :

$$E(b_1, b_2, \ldots, b_N) = \overline{(b_1 \oplus b_2 \ldots \oplus b_N)}$$

[1] On rappelle que $\texttt{XOR}(a, b) = (a\overline{b}) + (\overline{a}b)$.

Nous évaluons cette expression pour construire le jeux d'apprentissage $\mathcal{A}$ qui contient, dans tous les cas, pour les 2^N vecteurs v_i d'entrées possibles, la valeur de la fonction E.

Le système PGP décrit Section 1.3 est utilisé et la sortie $o(p)$ d'un programme p sera lue au sommet de la pile d'opérandes. En PG, le problème est trivial et n'a pas d'intérêt si le `OU` exclusif est utilisé dans le jeux d'instructions. Le jeux d'instructions Σ comprend donc :

- une instruction logique, notée `NOT`, d'arité 1 ;
- deux instructions logiques, notées `AND` et `OR`, d'arité 2 ;
- une instruction logique, notée `IF`, d'arité 3 telle que [2] :
$$\texttt{IF}(c, x, y) = (cx) + (\overline{c}y) ;$$
- une instruction spécifique de manipulation de la pile d'opérandes, notée `DUP`, d'arité 1 qui duplique le sommet de la pile.

On remarque l'absence de terminaux dans le jeux d'instructions. En effet, des test préliminaires ont montré que le système était plus performant quand la pile d'opérandes est systématiquement initialisée avec les N bit d'entrée du problème. Plus généralement, quand toutes les variables d'entrées présentes dans un ensemble d'apprentissage sont nécessaires à la résolution d'un problème, une stratégie similaire pourra être étudiée.

Pour le problème Even Parity, on définit la fonction de fitness $f_N : \{0,1\}^N \to \mathbb{R}$ d'un programme $p \in P$, avec P l'ensemble des programmes définit sur Σ et $\mathcal{A}$ l'ensemble d'apprentissage contentant les 2^N vecteurs d'entrées v_i, comme la somme des erreurs absolues commises sur $\mathcal{A}$ telle que :

$$f_N(x) = \frac{1}{2^N} \sum_{i=1}^{2^N} |o_i(p) - E(v_i)|$$

avec $o_i(p)$ la sortie de p pour le $i^{\text{ième}}$ vecteur de $\mathcal{A}$. Il convient donc de minimiser cette fonction pour résoudre le problème Even Parity.

Analyse du paysage

Nous avons réalisé 10^3 marches aléatoires de longueur 1000 et autant de marches adaptatives sur le paysage Even Parity avec N=8 et 10. La longueur

[2] Notons que $\texttt{IF}(a, \overline{b}, b) = \texttt{XOR}(a, b)$

de corrélation τ moyenne est égale à 17.35 pour N=8 et 14.16 pour N=10, ce qui correspond à un paysage corrélé mais rugueux (cf Table 3.1). On peut comparer ces valeurs à celles obtenues pour le problème RE (cf Figure 3.4) et ainsi supposer que le problème ne sera cependant pas très difficile à résoudre par PG. La longueur des marches adaptatives l est extrêmement faible avec l=1.19 pour N=8 et l=1.07 pour N=10, ce qui signifie que la quasi totalité des programmes évalués se sont avérés être des optima locaux. On note également que la fitness moyenne des optima locaux pour N=8 et 10 est égale à 0.5, c'est-à-dire que pour la moitié des vecteurs d'entrées testés les programmes évalués ont donné la bonne réponse. Ce phénomène correspond en fait à des programmes très nombreux (plus de 90% de l'espace de recherche) dont la sortie est une constante (0 ou 1) quelque soit le vecteur d'entrée. Enfin, nous avons trouvé, en testant 10^7 programmes, une proportion de voisins neutres de plus de 99% pour N=8 et 10, ce qui indique un paysage massivement neutre. Cependant, il faut tenir compte de la remarque précédente et donc du fait que près de 90% des programmes testés avaient la même fitness. On peut supposer que le nombre de voisins neutres diminue quand on explore des régions de l'espace de fitness élevée.

4.1.2 Paramètres de l'algorithme

Nous utilisons une population de 1000 individus créés aléatoirement. Dans la population initiale, la taille de création des individus est distribuée uniformément entre 1 et λ_c gènes choisis aléatoirement dans le jeux d'instruction. Nous avons testé les performances du système pour λ_c=25, 50 et 75. Durant les expériences, la taille des individus est limitée à λ_{max}=100 gènes. L'évolution, avec élitisme, sélection par tournoi de 16 individus[3] et remplacement *steady-state* est limitée à 100 générations.

4.1.3 Résultats expérimentaux

Dans cette section, nous comparons les résultats obtenus par PGL avec les opérateurs CS et CMH sur le problème Even Parity pour N=8 et 10.

[3]L'influence de la taille du tournoi sera étudiée plus loin dans cette section.

Nous voulons, entre autre, trouver les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$, c'est-à-dire les réglages qui donnent les meilleurs résultats en terme de fitness moyenne, c'est pourquoi nous avons fait varier $\mathcal{T}_m$ (de 0 à 2) et $\mathcal{T}_c$ (de 0 à 1). Pour chaque réglage, 50 expériences indépendantes sont réalisées. Un test de Student, avec 95% de confiance, est utilisé pour déterminer si les résultats obtenus sont statistiquement différents les uns des autres.

La Table 4.1 rapporte les résultats trouvés pour N=8 avec les deux opérateurs CS et CMH pour les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$. Le taux de succès indique que le problème est facilement résolu, sans différence statistiquement significative, quelque soit l'opérateur de croisement utilisé. La génération moyenne de découverte de l'optimum montre que l'opérateur CS est plus performant que le CMH en ce qui concerne le nombre de programmes évalués. Cependant, si l'on considère la taille moyenne et la taille effective[4] moyenne du meilleur programme, on note qu'avec le CMH, les programmes sont au moins deux fois plus petits. Ainsi en dépit de l'effort de calcul lié au processus d'alignement du CMH et du nombre de générations nécessaires à la convergence, les expériences réalisées avec le CMH se sont avérées plus rapides et donc, en terme de temps de calcul, le CMH accélère le processus évolutionnaire.

TAB. 4.1 – Meilleurs résultats trouvés sur le problème Even Parity avec N=8.

Type de Croisement	Taux de Succès	Génération Moyenne	Taille Moyenne	Taille Effective Moyenne
$\lambda_c = 25$				
CS	98%	$\mathbf{63}_{(\sigma=13)}$	$84_{(\sigma=17)}$	$54_{(\sigma=12)}$
CMH	95%	$74_{(\sigma=9)}$	$\mathbf{31}_{(\sigma=5)}$	$\mathbf{27}_{(\sigma=4)}$
$\lambda_c = 50$				
CS	98%	$63_{(\sigma=14)}$	$87_{(\sigma=17)}$	$59_{(\sigma=12)}$
CMH	100%	$60_{(\sigma=11)}$	$\mathbf{32}_{(\sigma=4)}$	$\mathbf{27}_{(\sigma=3)}$
$\lambda_c = 75$				
CS	98%	$67_{(\sigma=15)}$	$84_{(\sigma=16)}$	$60_{(\sigma=11)}$
CMH	98%	$65_{(\sigma=12)}$	$\mathbf{40}_{(\sigma=5)}$	$\mathbf{31}_{(\sigma=4)}$

[4]On rappelle que la taille effective correspond à la taille des programmes après la phase de compilation décrite Section 1.3.

Exemples : Nous présentons un programme de fitness optimale et de taille effective 63 obtenu avec le CS pour $\lambda_c = 50$:

```
X1 X2 X3 X4 X5 X6 X7 X8 DUP OR DUP NOT IFT DUP NOT IFT DUP OR
DUP NOT IFT NOT NOT DUP AND DUP DUP AND NOT NOT NOT DUP NOT
IFT DUP DUP AND AND DUP NOT NOT NOT DUP DUP AND AND DUP NOT
NOT OR DUP OR NOT AND DUP AND NOT NOT NOT DUP OR NOT
```

ainsi qu'un programme de fitness optimale et de taille effective 26 obtenu avec le CMH pour $\lambda_c = 50$:

```
X1 X2 X3 X4 X5 X6 X7 X8 DUP NOT IFT NOT NOT DUP DUP AND NOT
IFT NOT OR DUP NOT IFT DUP NOT NOT
```

On notera que pour faciliter la compréhension, nous avons ajouté, aux deux programmes précédents, un préfixe `X1 X2 X3 X4 X5 X6 X7 X8` qui est présent de manière implicite suite à l'initialisation automatique de la pile d'opérandes.

La Table 4.2 présente les résultats obtenus pour N=10 avec les deux opérateurs CS et CMH pour les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$. Avec l'opérateur CS, le problème est facile à résoudre pour toutes les valeurs de λ_c alors que lorsque le CMH est utilisé, un taux de succès très faible (30%) est rapporté pour λ_c=25 mais statistiquement équivalent à celui du CS pour $\lambda_c \geq 50$. Les performances du système quand l'opérateur CMH est utilisé sont donc dépendantes de λ_c. On note également que la taille ainsi que la taille effective des programmes obtenus en utilisant l'opérateur CMH sont nettement inférieures à celles rapportées pour le CS.

Les Figures 4.1 et 4.2 présentent la fitness moyenne du meilleur individu trouvé en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour les deux opérateurs de croisement sur le problème Even Parity avec N=8 et λ_c=50 ; l'influence de $\mathcal{T}_m$ et $\mathcal{T}_c$ étant comparable pour N=10 ainsi que pour les autres valeurs de λ_c testées. Quand l'opérateur CS est utilisé conjointement avec l'opérateur de mutation, la résolution du problème Even Parity ne nécessite pas un réglage très précis de $\mathcal{T}_m$ et $\mathcal{T}_c$. Cependant, on note que les meilleurs résultats du CS ont été obtenus pour $\mathcal{T}_m$ =0.4 et $\mathcal{T}_c$ =0.6. Par contre, avec l'opérateur CMH, une recherche fine du réglage optimal, en particulier de $\mathcal{T}_m$, est nécessaire. La plus petite

TAB. 4.2 – Meilleurs résultats trouvés sur le problème Even Parity avec N=10.

Type de Croisement	Taux de Succès	Génération Moyenne	Taille Moyenne	Taille Effective Moyenne
$\lambda_c = 25$				
CS	**90**%	**65**$_{(\sigma=17)}$	80$_{(\sigma=19)}$	61$_{(\sigma=17)}$
CMH	30%	80$_{(\sigma=11)}$	**28**$_{(\sigma=2)}$	**26**$_{(\sigma=2)}$
$\lambda_c = 50$				
CS	88%	**69**$_{(\sigma=13)}$	90$_{(\sigma=14)}$	67$_{(\sigma=14)}$
CMH	82%	83$_{(\sigma=6)}$	**30**$_{(\sigma=4)}$	**27**$_{(\sigma=3)}$
$\lambda_c = 75$				
CS	84%	**72**$_{(\sigma=17)}$	83$_{(\sigma=19)}$	66$_{(\sigma=19)}$
CMH	88%	80$_{(\sigma=5)}$	**41**$_{(\sigma=10)}$	**36**$_{(\sigma=7)}$

valeur de fitness trouvée correspond à $\mathcal{T}_m$ =1.2 et $\mathcal{T}_c$ compris entre 0.4 et 1.0. Nous avons déjà vu que le comportement du CMH est plus proche de celui de l'opérateur de recombinaison des AG que celui de l'opérateur CS, c'est pourquoi des méthodes permettant de lutter contre la convergence prématurée, comme la mutation, sont adaptées.

Évolution de la taille

Nous avons rapporté (Figures 4.3 et 4.4) la taille moyenne du meilleur individu à la génération 100[5] en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour N=8 et λ_c=50. Dans les populations recombinées par CS (avec $\mathcal{T}_c > 0$), la taille est toujours supérieure à 50 gènes et l'on note que celle-ci augmente avec $\mathcal{T}_m$ et $\mathcal{T}_c$. Dans le cas du CMH, elle est toujours inférieure à 50 gènes et n'est que très légèrement influencée par $\mathcal{T}_c$ alors qu'elle augmente régulièrement avec $\mathcal{T}_m$.

Avec l'opérateur CS, la courbe d'évolution de la taille moyenne du meilleur individu trouvé (cf. Figure 4.5), pour N=8 et λ_c=50 et pour le réglage optimal de $\mathcal{T}_m$ et $\mathcal{T}_c$, montre une croissance très importante qui apparaît dès les premières générations et n'est stoppée que par la limite $\lambda_{max} = 100$, c'est le phénomène de bloat. Lorsque le CMH est utilisée, cette croissance est lente pendant les quarante premières générations et devient quasi nulle jusqu'à la

[5]Les échelles utilisées différent entre les deux figures.

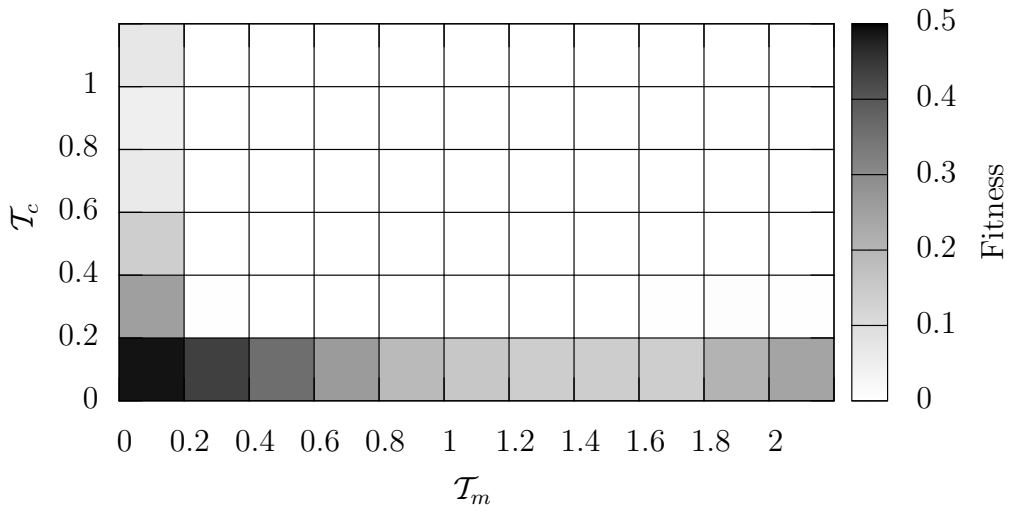

FIG. 4.1 – Problème Even Parity, N=8 et CS : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

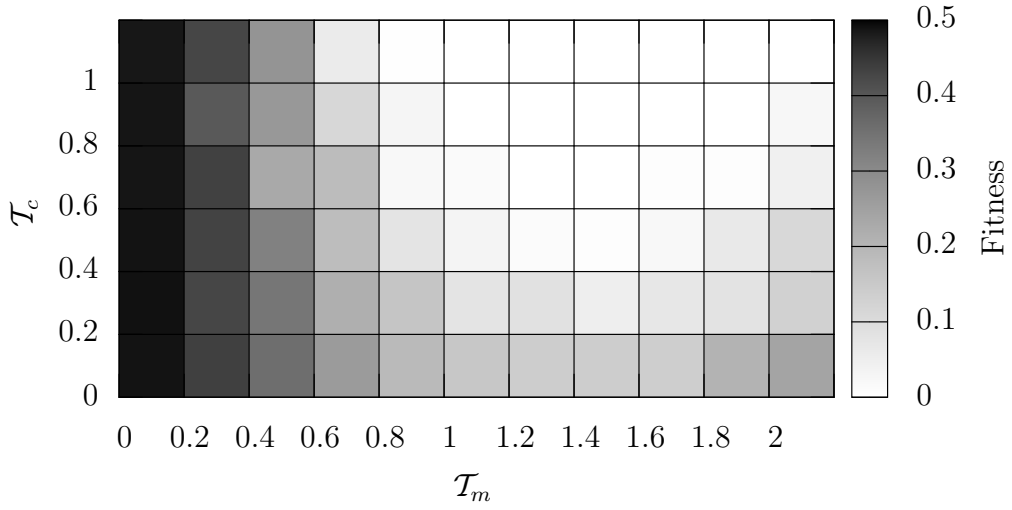

FIG. 4.2 – Problème Even Parity, N=8 et CMH : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

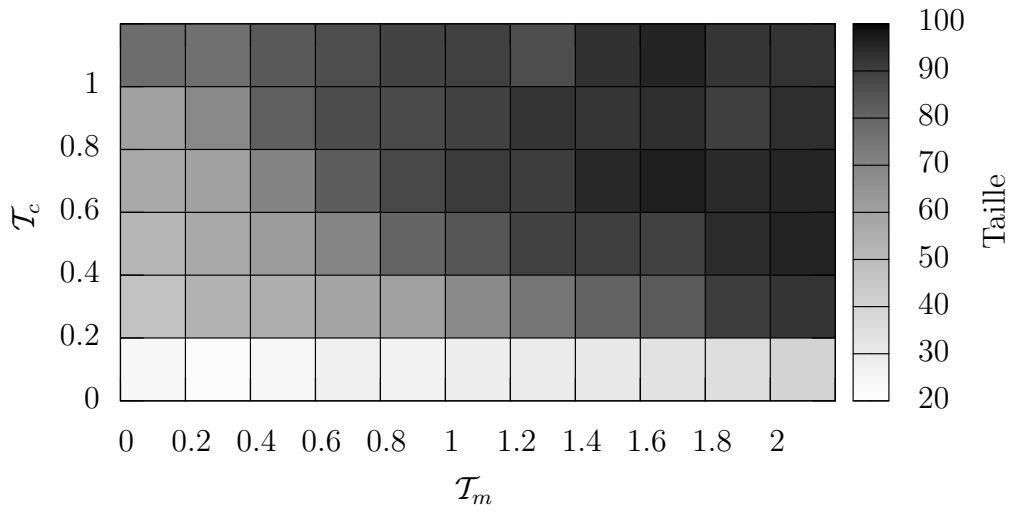

FIG. 4.3 – Problème Even Parity, N=8 et CS : Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

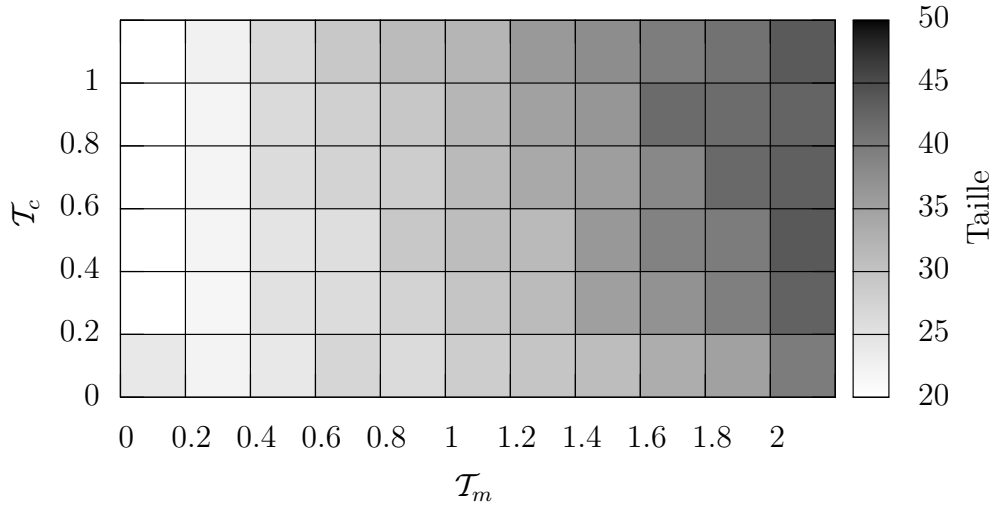

FIG. 4.4 – Problème Even Parity, N=8 et CMH : Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$.

génération d'arrêt. C'est cette faible augmentation de la taille durant l'évolution qui explique les performances réduites obtenues avec le CMH pour des valeurs de λ_c trop petites.

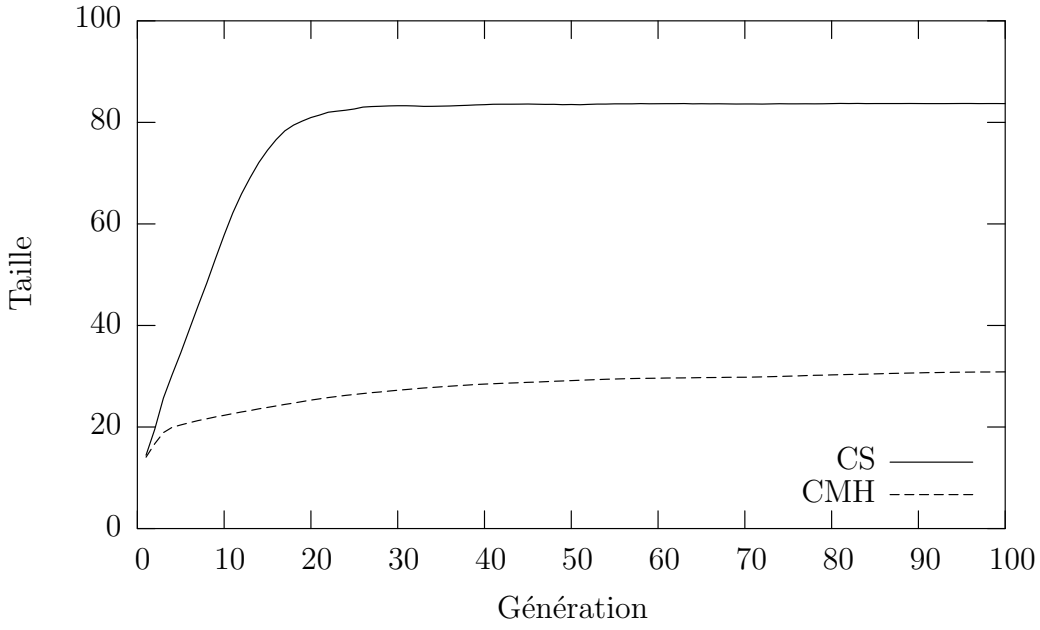

FIG. 4.5 – Problème Even Parity, N=8 : Évolution de la taille moyenne.

Nous avons recherché les valeurs de $\mathcal{T}_m$ et $\mathcal{T}_c$ optimales sur le problème Even Parity avec N=10 et $\lambda_c = 50$, avec les mêmes paramètres que précédemment mais nous avons fixé λ_{max} à 200 gènes. La Table 4.3 donne les résultats obtenus pour le réglage optimal qui s'est avéré être identique à celui des expériences précédentes. On remarque clairement une dégradation des performances pour l'opérateur CS, en particulier le taux de succès chute de près de 25% alors que les expériences menées avec le CMH ont donné des résultats identiques, ce qui signifie que la limite en taille n'a jamais été atteinte.

On peut donc décrire le comportement des deux opérateurs comme suit :

– Avec le CS, les performances dépendent fortement de la taille maximum autorisée λ_{max} et sont peu (voire pas) influencées par la taille de création λ_c ;

TAB. 4.3 – Meilleurs résultats trouvés sur le problème Even Parity avec N=10, λ_c=50 et λ_{max}=200.

Type de Croisement	Taux de Succès	Génération Moyenne	Taille Moyenne	Taille Effective Moyenne
CS	66%	$\mathbf{56}_{(\sigma=16)}$	$126_{(\sigma=27)}$	$92_{(\sigma=23)}$
CMH	**82**%	$83_{(\sigma=6)}$	$\mathbf{30}_{(\sigma=4)}$	$\mathbf{27}_{(\sigma=3)}$

– Avec le CMH, les performances dépendent fortement de la taille de création λ_c et sont peu (voire pas) influencées par la taille maximum autorisée λ_{max}.

Effets du croisement

Nous avons rapporté, Table 4.4, les proportions moyennes (exprimées en pourcentage) des croisements D, N et A obtenues pour le réglage optimal de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour CS et CMH. On constate que les deux opérateurs ont des effets différents, en fait presque symétriques, puisque près des deux tiers des croisements réalisés avec l'opérateur CS entraînent une dégradation de la fitness, alors qu'avec l'opérateur CMH, une large majorité, plus de 75% des croisements sont neutres[6]. La proportion de croisements A est très faible pour les deux opérateurs avec un petit avantage pour le CMH.

TAB. 4.4 – Proportions (en %) des croisements Délétères, Neutres et Avantageux sur le problème Even Parity.

Type de Croisement	N=8			N=10		
	D	N	A	D	N	A
CS	66.90	33.02	0.08	70.08	29.81	0.10
CMH	22.45	77.44	0.10	23.47	76.34	0.17

Nous avons également tracé les effets des deux opérateurs en fonction de la fitness parentale (cf. Figures 4.6 et 4.7). Les valeurs de fitness élevées correspondent au début des expériences, où la probabilité de détruire des

[6]On note que pour pour N=10, l'analyse du paysage a montré qu'environ 99% des voisins sont neutres.

structures existantes est faible, alors que les valeurs de fitness faibles correspondent à la fin, où les individus recombinés sont complexes et la probabilité d'améliorer leurs performances est plus faible. Ces figures confirment les résultats obtenus au chapitre précédent (cf. Section 3.3.2 et 3.3.3) et montrent expérimentalement que le CMH est moins brutal que l'opérateur CS.

4.1.4 Étude de la population

Les performances d'un opérateur génétique dépendent du problème à résoudre et de la structure de la population. Cette population étant, elle même, le produit des opérateurs génétiques, nous nous intéressons aux différentes propriétés de la population en fonction de l'opérateur de recombinaison utilisé.

Populations finales

Nous avons représenté (cf. Figures 4.8 et 4.9) pour les deux opérateurs, un échantillon composé de 200 individus (sur 1000) d'une des populations finales obtenues avec le meilleur réglage de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour N=8 et λ_c=50. Pour construire cet échantillon, nous avons sélectionné uniquement des programmes dont la valeur de fitness est nulle, c'est-à-dire ceux qui répondent parfaitement au problème Even Parity. Chaque ligne correspond au génotype d'un des 200 individus après la phase de compilation, c'est-à-dire que les instructions non exécutées sont supprimées. Nous avons donné une couleur (nuance de gris) différente à chacune des 5 instructions qui compose les programmes puis nous les avons disposées, de gauche à droite, suivant leur ordre d'exécution. Enfin, nous nous sommes servi de la couleur des instructions pour établir un ordre lexicographique qui permet de trier les programmes verticalement.

La différence la plus remarquable entre ces deux figures, et sans doute la plus importante, concerne la taille des individus qui est presque trois fois moins importante quand l'opérateur CMH est utilisé. La variabilité des génotypes semble, même en tenant compte de la différence de taille, plus importante avec l'opérateur CS et l'on pourra s'étonner, dans les deux cas, du

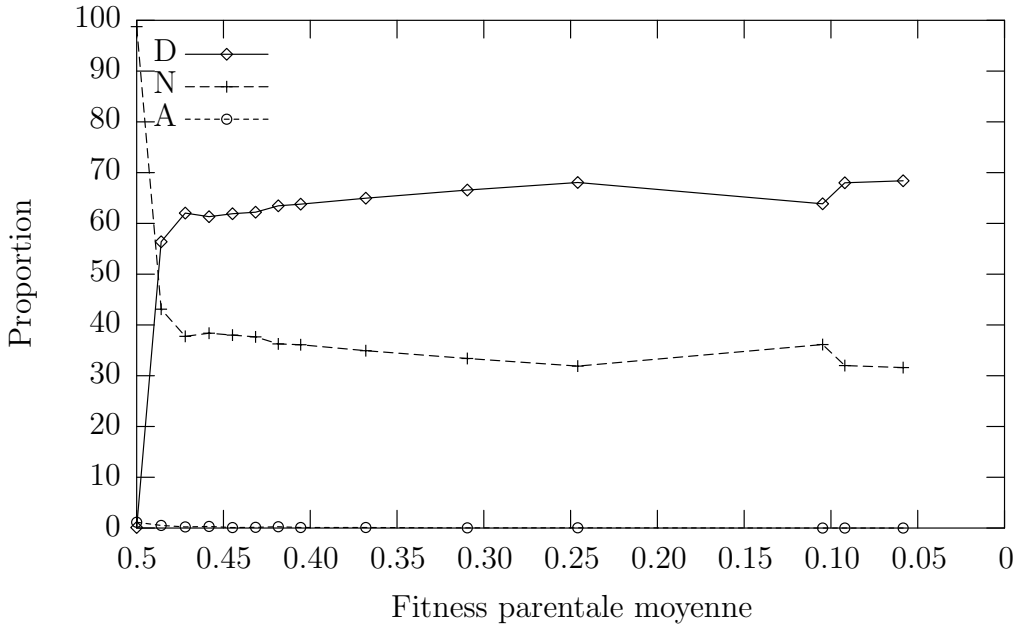

FIG. 4.6 – Problème Even Parity, N=10 et CS : Proportions (en %) des croisements Délétères, Neutres et Avantageux en fonction de la fitness parentale moyenne.

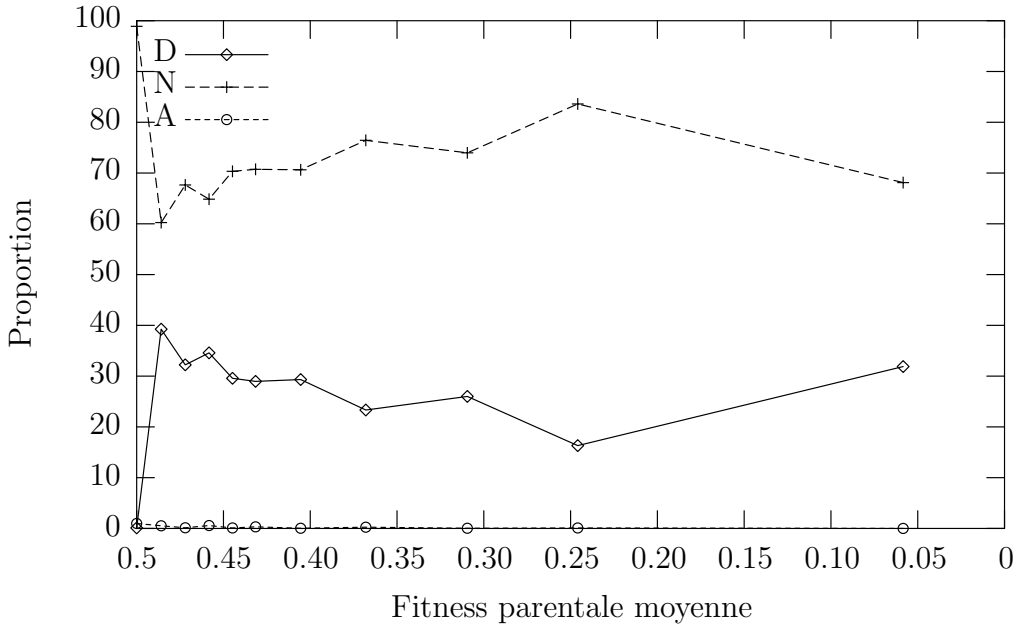

FIG. 4.7 – Problème Even Parity, N=10 et CMH : Proportions (en %) des croisements Délétères, Neutres et Avantageux en fonction de la fitness parentale moyenne.

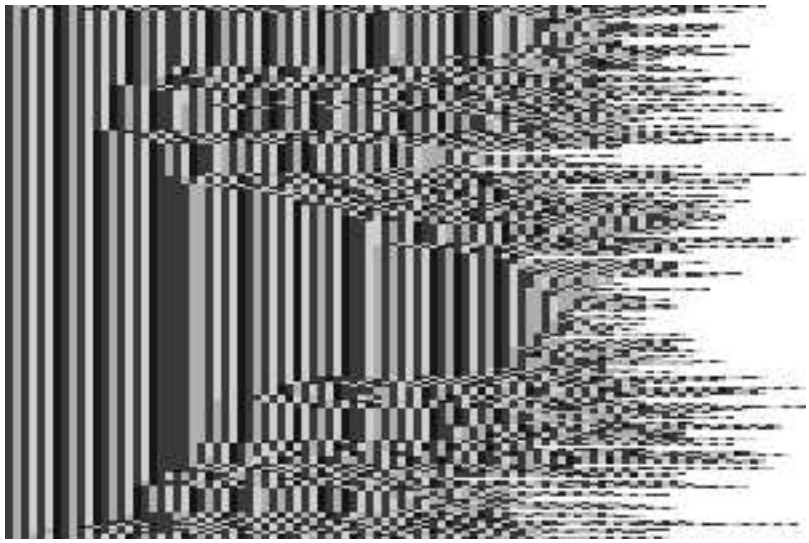

FIG. 4.8 – Problème Even Parity, N=8 et CS : Échantillon d'une population finale.

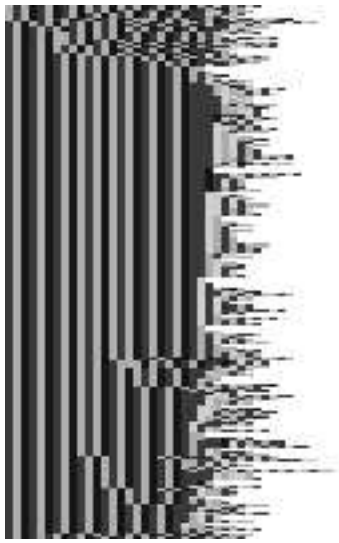

FIG. 4.9 – Problème Even Parity, N=8 et CMH : Échantillon d'une population finale.

nombre de génotypes différents ayant une valeur de fitness nulle. Il semble également, pour l'opérateur CS, que plusieurs sous-populations coexistent dans l'échantillon final alors que ce phénomène n'apparaît pas pour le CMH. Cela nous invite à envisager, dans le futur, l'utilisation conjointe du CMH et de modèles de populations en îles pour "forcer" l'existence de multiples populations.

On remarque, pour les deux opérateurs, deux parties distinctes dans les génotypes. La première, à gauche des figures, où les programmes sont plus ou moins similaires entre eux et semblent partager des structures communes et une deuxième partie, à droite, où de nettes différences existent[7]. Tout ce passe comme si les premières instructions des programmes s'étaient stabilisées et que la recherche ne s'effectuait plus que sur les dernières instructions. On se souviendra d'une part, que nous avons initialisé, avant chaque exécution d'un programme, la pile d'opérandes avec les N entrées, ici 8, du problème à résoudre et d'autre part qu'aucun terminal n'est présent dans le jeux d'instructions. Ceci implique que les effets des premières instructions des programmes soient décisifs puisque une entrée pourrait être définitivement perdue, avec une séquence "`DUP`, `NOT`, `AND`" par exemple. D'un autre côté, les dernières instructions exécutées sont également déterminantes puisque la valeur de sortie d'un programme est lue au sommet de la pile.

Polymorphisme

On s'intéresse au polymorphisme de la population, c'est-à-dire à la diversité des programmes. Plusieurs mesures de la diversité existent et ont d'ores et déjà été utilisées en PG [15].

Nous utilisons l'entropie comme mesure de la diversité phénotypique, c'est-à-dire de la variabilité des valeurs de fitness dans une population. On notera $Z_{phen}(M)$, l'entropie d'une population M, dont les individus possèdent

[7]Pour nous en assurer, nous avons réalisé les mêmes figures mais en parcourant les programmes dans l'ordre inverse, c'est-à-dire que les dernières instructions dans l'ordre d'exécution ont été examinées en premier pour déterminer l'ordre lexicographique. Dans ces cas, nous avons constaté que pour les deux opérateurs à peine 5 instructions étaient partagées par tous les programmes.

n valeurs de fitness différentes, telle que :

$$Z_{phen}(M) = -\sum_{k=1}^{n} p_k \times log(p_k)$$

avec p_k la proportion des individus de M possédant la $k^{\text{ième}}$ valeur de fitness. Une valeur de $Z_{phen}(M)$ élevée indique une forte diversité phénotypique.

Nous utilisons l'homologie relative moyenne de la population comme mesure de la diversité génotypique, c'est-à-dire de la variabilité des génotypes dans une population. La taille moyenne des individus diffère largement en fonction de l'opérateur de recombinaison utilisé, c'est pourquoi l'homologie est rapportée au nombre d'instructions des programmes. Ainsi, on notera $Z_{gen}(M) \in [0, 1]$, l'homologie relative moyenne d'une population M, composée de n individus, telle que :

$$Z_{gen}(M) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{\mathcal{D}(M_i, M_j)}{\mathcal{S}(M_i) + \mathcal{S}(M_j)}$$

avec M_i le $i^{\text{ième}}$ individu de M et $\mathcal{S}(M_i)$ la taille de M_i. Une valeur de $Z_{gen}(M)$ élevée indique une forte diversité génotypique.

Dans un algorithme évolutionnaire, la taille du tournoi de remplacement détermine la force de la pression de sélection exercée sur les individus d'une population et permet de contrôler le polymorphisme. Nous avons réalisé une série de 50 expériences pour différentes tailles de tournoi de remplacement sur le problème Even Parity avec N=8, λ_c=50 en recherchant le réglage optimal de $\mathcal{T}_m$ et $\mathcal{T}_c$. En terme de performance, nous n'avons pas noté de différence remarquable en fonction de la taille du tournoi, seul le réglage optimal de $\mathcal{T}_m$ a été modifié : plus la taille du tournoi est élevée, plus le $\mathcal{T}_m$ optimal est élevé. En effet, quand la taille du tournoi augmente, la pression de sélection augmente également, ce qui entraîne une convergence plus rapide de la population d'où la nécessité de maintenir le polymorphisme a un niveau élevé entre autre grâce à l'opérateur de mutation. Nous pensons qu'il existe néanmoins des tailles de tournoi pour lesquels l'opérateur ne pourrait maintenir la diversité sans dégrader les performances du système.

L'évolution moyenne de $Z_{phen}(M)$ sur 50 expériences, pour $\mathcal{T}_m$ =1.0 et $\mathcal{T}_c$ =1.0, est présentée Figure 4.10 et 4.11. Pour les deux opérateurs, chaque

courbe présente deux phases distinctes. Durant la première phase l'entropie croît, cela correspond à une augmentation du nombre de valeur de fitness dans la population. En effet, on rappelle que les programmes créés aléatoirement ont presque toujours une valeur de fitness égale à 0.5. La deuxième phase montre une lente diminution de l'entropie qui correspond à la diffusion des meilleurs individus dans la population et donc à la sur-représentation de certaines valeurs de fitness. On note, avec le CS et le CMH, que l'entropie décroît d'autant plus vite que la taille du tournoi est élevée, c'est-à-dire que la pression de sélection est forte. L'évolution de $Z_{phen}(M)$ est comparable pour les deux opérateurs, en particulier pour les valeurs finales obtenues à la génération 100. On remarque cependant que la valeur de $Z_{phen}(M)$ maximum atteinte (entre les générations 10 et 40) est plus influencée par la taille du tournoi dans le cas du CMH.

L'évolution moyenne de $Z_{gen}(M)$ sur 50 expériences, pour $\mathcal{T}_m$ =1.0 et $\mathcal{T}_c$ =1.0, est présentée Figure 4.12 et 4.13. La création aléatoire des génotypes garantit une forte diversité au début des expériences et comme attendu, celle-ci décroît au cours de l'évolution. Pour les deux opérateurs, plus la taille du tournoi est élevée, plus la diversité génotypique est faible. Ce phénomène est constaté dès la première génération avec le CMH et après quelques générations (10 environ) pour l'opérateur CS. Plus généralement, la valeur de $Z_{gen}(M)$ est plus faible quand le CMH est utilisé, c'est-à-dire que les programmes sont plus homologues ce qui est conforme à nos attentes, en particulier à la propriété de respect (cf. Équation 2.11) de l'ensemble de recombinaison du CMH.

Distribution de la taille des programmes

Nous avons déjà vu que la taille moyenne des individus dépend fortement de l'opérateur de recombinaison utilisé. Les Figures 4.14 et 4.15 présente la distribution de la taille des programmes, en échelle logarithmique, pour le CS et le CMH, aux générations 0,1,10 et 100. Ces distributions sont des moyennes sur 50 expériences et sont calculées à partir de l'évolution de la population sur le problème Even Parity avec N=8, λ_c=50 et le réglage optimal de $\mathcal{T}_c$ et $\mathcal{T}_m$ trouvé précédemment. Elles peuvent être comparées aux distributions

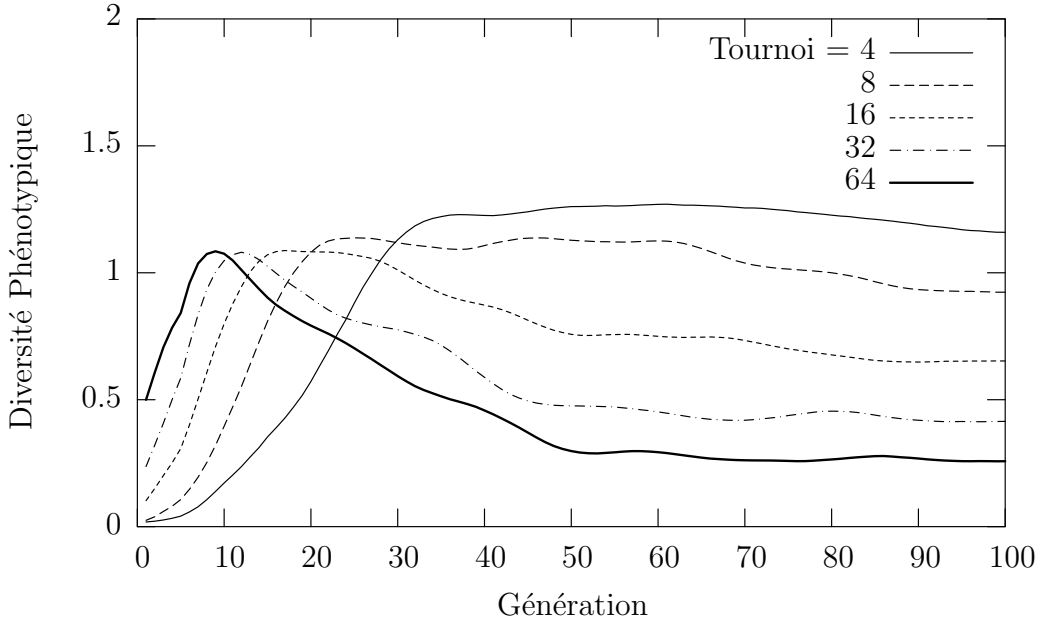

FIG. 4.10 – Problème Even Parity, N=8 et CS : Évolution de l'entropie de la population.

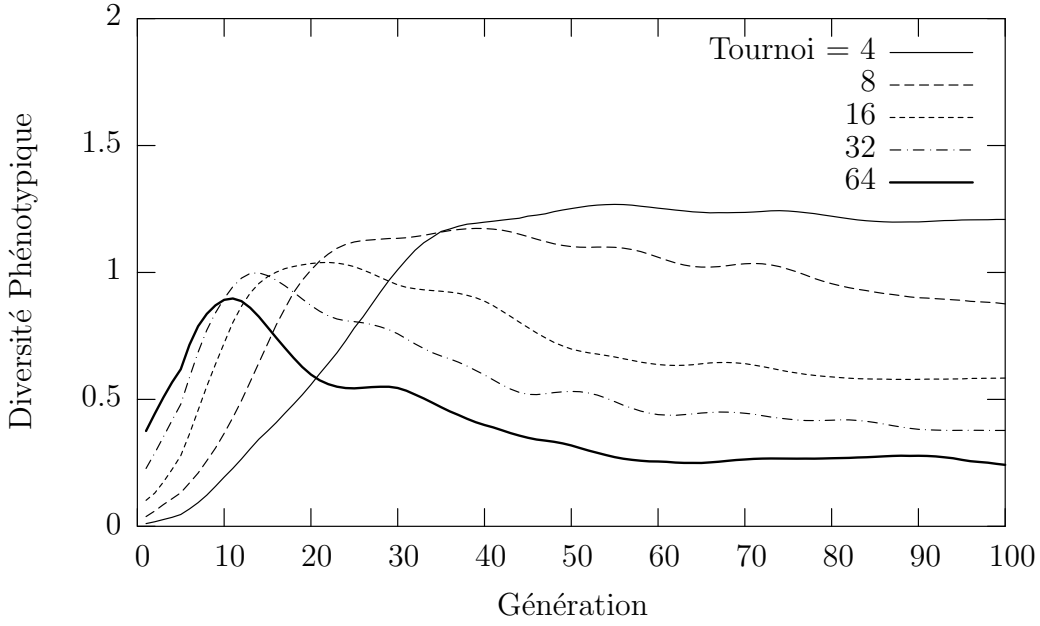

FIG. 4.11 – Problème Even Parity, N=8 et CMH : Évolution de l'entropie de la population.

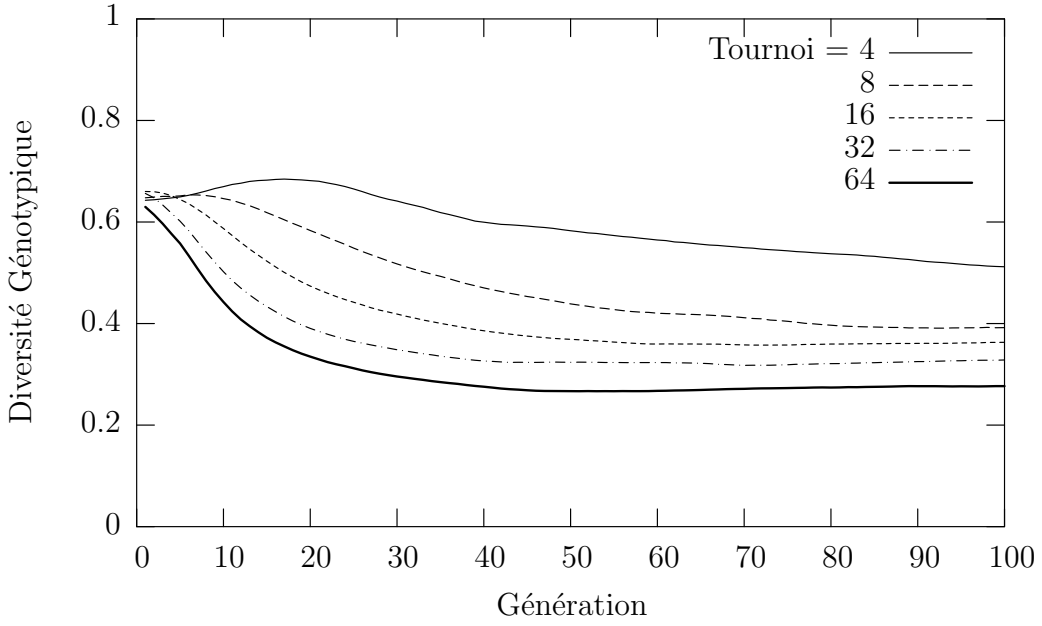

FIG. 4.12 – Problème Even Parity, N=8 et CS : Évolution de l'homologie relative moyenne de la population.

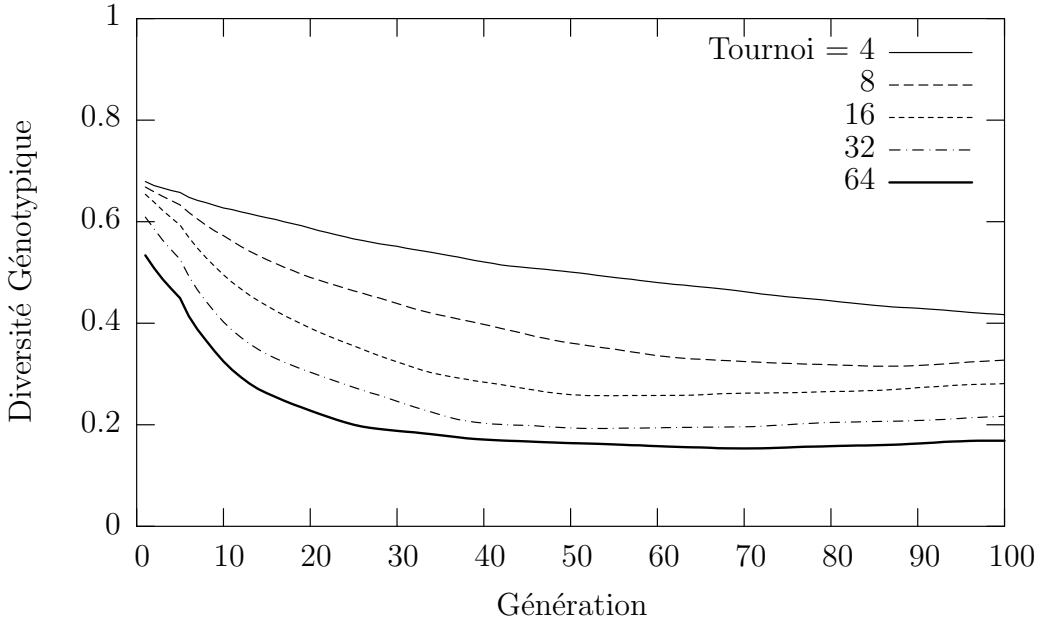

FIG. 4.13 – Problème Even Parity, N=8 et CMH : Évolution de l'homologie relative moyenne de la population.

(cf. Figures 2.9 et 2.10) obtenues sans pression de sélection.

Nous avons vu que, sans pression de sélection, la distribution de la taille des programmes recombinés par CS converge vers une distribution γ et donc que les programmes de petites tailles sont sur-représentés. L'évolution de la distribution sur le problème Even Parity est tout autre. En effet, dès la génération 10, on constate que de nombreux programmes ont atteint la taille maximum et que très peu de programmes possèdent moins de 10 instructions. A la génération 100, plus de 20% des programmes sont composés de 100 instructions. En ce qui concerne le CMH, la distribution de la taille de programmes est proche d'une gaussienne et celle-ci se décale lentement vers les tailles supérieures.

Parmi les explications avancées au sujet du phénomène de bloat, *l'hypothèse de la dérive* et *l'hypothèse du biais* (cf Section 1.2.3) suggèrent toutes deux que la nature de l'espace de recherche est en cause : à partir d'un programme ayant un taille t et une fitness f données, il est plus probable de trouver des programmes dont la fitness soit supérieure à f en explorant l'espace de recherche dans les régions où les programmes sont plus grands que t.

Nous avons dessiné des cercles, sur chaque distribution des figures précédentes, correspondant à la taille du meilleur individu trouvé[8] à chaque génération. On note que dans le cas du CS les programmes de même taille que le meilleur sont de moins en moins nombreux et représentent seulement 1% de la population à la génération 100. A l'opposé, avec le CMH, leur proportion augmente et ils constituent jusqu'à 5% de la population à la fin des expériences. Nous avons calculé le nombre de programmes dont la taille est supérieure à celle du meilleur individu trouvé, à la dernière génération, ils sont près de 70% avec le CS et seulement 45% dans le cas du CMH.

Ainsi, quand le CS est utilisé, l'opérateur de sélection est appliqué majoritairement sur des programmes plus grands que le meilleur individu à chaque génération et donc, suivant les hypothèses rappelées précédemment, la probabilité d'améliorer la fitness en diminuant la taille est d'autant plus réduite. On peut dire que le CS amplifie les "défauts" de l'espace de recherche. Avec

[8]Seule l'abscisse est significative.

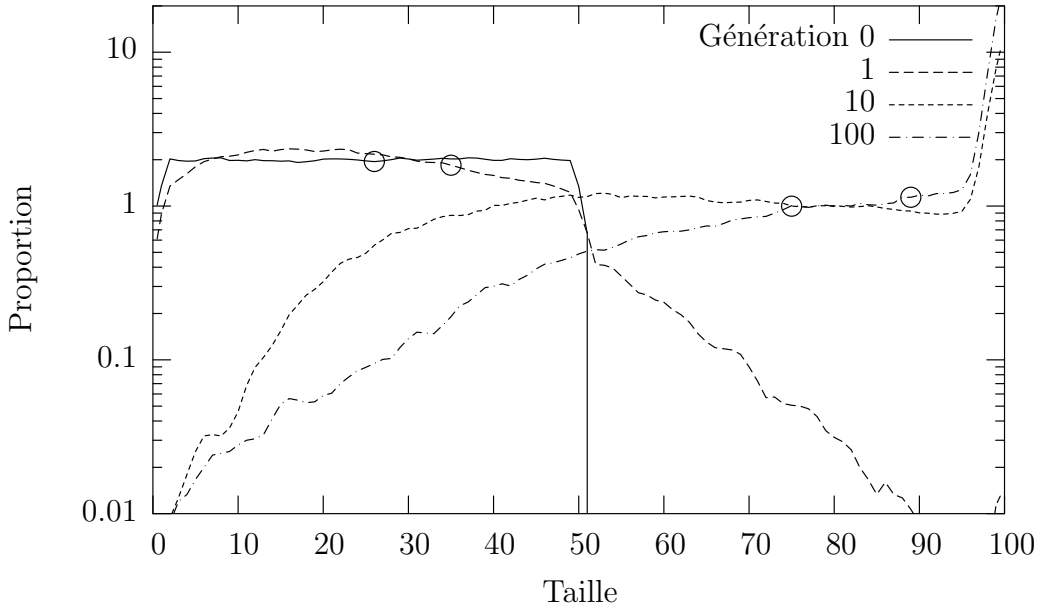

FIG. 4.14 – Problème Even Parity, N=8 et CS : Évolution de la distribution de la taille dans la population.

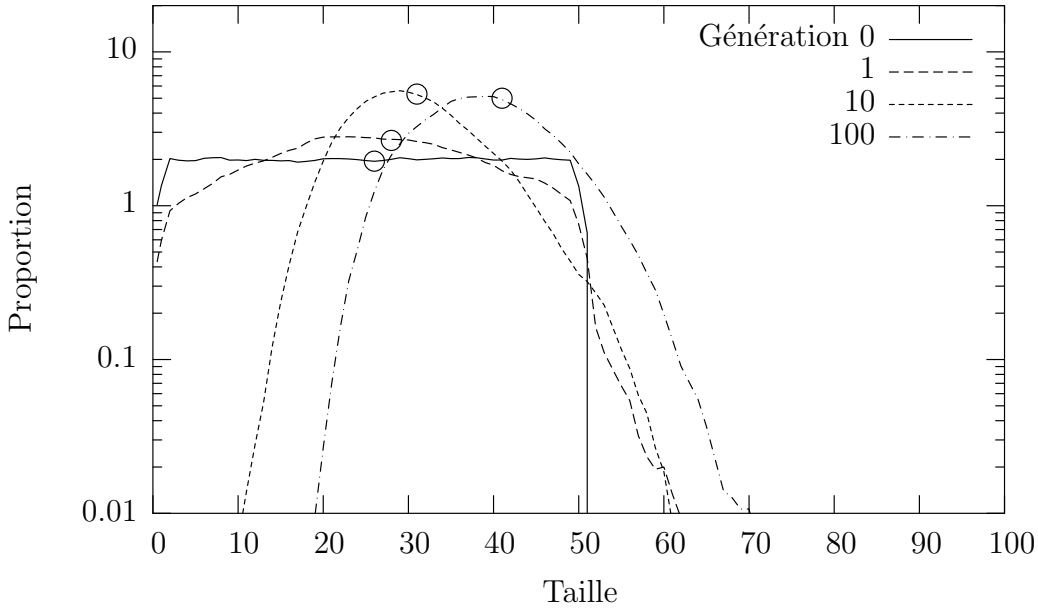

FIG. 4.15 – Problème Even Parity, N=8 et CMH : Évolution de la distribution de la taille dans la population.

l'opérateur CMH, la sélection opère sur une population dont la distribution des tailles est a peu près centrée autour de la taille du meilleur individu et donc seule la nature de l'espace de recherche est responsable du bloat.

4.2 Vers un contrôle de la taille

Les résultats obtenus précédemment ont montré que si le CMH est utilisé, les performances du système dépendent du réglage de la taille de création maximum des programmes λ_c, mais plus généralement de la distribution de la taille des individus dans la population. De plus, nous avons vu que si des expériences sont réalisées avec un réglage adapté de λ_c, des performances équivalentes à celles de l'opérateur CS peuvent être obtenues et que le phénomène de bloat est largement réduit. Cependant, il semble difficile de trouver ce réglage *a priori* pour un problème quelconque. Dans cette section, en dépit de l'incapacité pour l'opérateur CMH d'explorer la taille des programmes, nous étudions la possibilité de réaliser un contrôle précis de la taille des programmes durant l'évolution.

4.2.1 Sans pression de sélection

On rappelle que l'opérateur de mutation, utilisé depuis le début de cette étude, réalise en fait trois opération différentes : l'insertion, la délétion ou la substitution d'instructions. C'est pourquoi, quand nous évoquons le taux d'application $\mathcal{T}_m$ de l'opérateur de mutation, il s'agit en fait des taux d'application des opérations d'insertion, de délétion et de substitution qui ont toujours été identiques jusqu'ici. Nous introduisons de nouvelles notations pour identifier le réglage de chacune de ses opérations avec $\mathcal{T}_i$ pour l'insertion, $\mathcal{T}_d$ pour la délétion et enfin $\mathcal{T}_s$ pour la substitution.

Mutation équilibrée

Nous avons tracé, Figure 4.16 et 4.17, à partir de 50 expériences indépendantes, l'évolution de la taille moyenne des programmes recombinés par, respectivement CS et CMH, avec $\mathcal{T}_c$ =1.0 , $\mathcal{T}_i$ =$\mathcal{T}_d$ =1.0 et $\mathcal{T}_s$ =0.0. Le pay-

sage est plat, c'est-à-dire qu'il n'y a pas de pression de sélection. Différentes valeurs de λ_c (50, 250 et 450) ont été testées et la valeur de λ_{max} a été fixée à 500 instructions. La population contient 1000 individus créés aléatoirement avec un jeux de 10 instructions.

Il est clair, que quand l'opérateur CS est utilisé, la taille moyenne des programmes varie et que les trois courbes, correspondant aux 3 valeurs de λ_c, semblent converger après 500 générations autour de 75 instructions. Le comportement théorique de la taille moyenne des programmes recombinés par CS en population infinie et ce, sans limite de taille, a été décrit dans [80] et [90]. Les auteurs ont montré que celle-ci ne varie pas au cours de l'évolution et ne dépend que de la moyenne initiale. Bien que, dans notre cas, la taille de la population ne soit pas infinie, nous pensons qu'elle est suffisamment grande pour ne pas introduire d'accidents dans la sélection d'autant qu'une moyenne sur 50 expériences a été réalisée. Nous pensons plutôt que les causes principales du comportement constaté ici résident d'une part dans la limite λ_{max} mais également dans l'utilisation conjointe de l'opérateur de mutation. En effet, quand la recombinaison par CS produit des programmes plus longs que ne l'autorise λ_{max}, ce qui est plus fréquent avec $\lambda_c = 450$, nous avons choisi (cf. Section 1.3) d'enlever les instructions excédentaires. Ainsi, le nombre d'instructions moyen dans la population diminue, entraînant la décroissance de la taille moyenne des programmes. A l'inverse, quand la recombinaison par CS produit des programmes de très petites tailles, ce qui est plus fréquent avec $\lambda_c = 50$, bien que les réglages de $\mathcal{T}_i$ et $\mathcal{T}_d$ soient équilibrés, toutes les opérations de délétions prévues ne peuvent être appliquées, la taille nulle constituant également une limite infranchissable. Ainsi, le nombre d'instructions moyen dans la population augmente, entraînant la croissance de la taille moyenne des programmes. La convergence des trois courbes serait donc le produit de ces deux phénomènes.

En ce qui concerne la recombinaison par CMH, nous avons vu que, sans pression de sélection, la distribution de la taille des programmes est moins éparse que celle liée à l'opérateur CS. Les deux limites (taille nulle et taille maximum) sont, quelque soit λ_c, rarement atteintes et le nombre d'instructions dans la population reste stable.

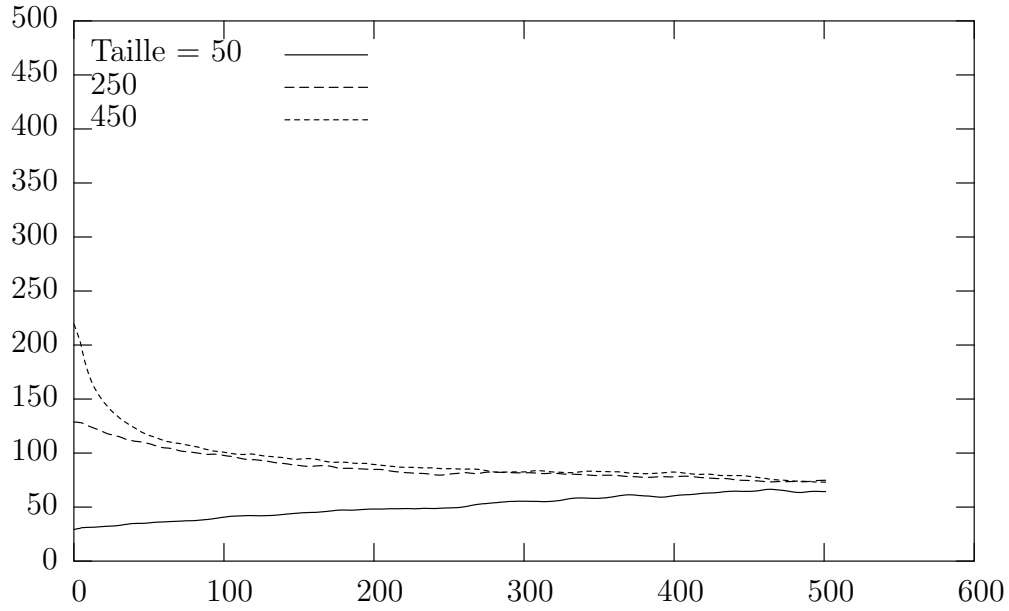

FIG. 4.16 – Paysage plat et CS : Évolution de la taille moyenne avec mutation équilibrée

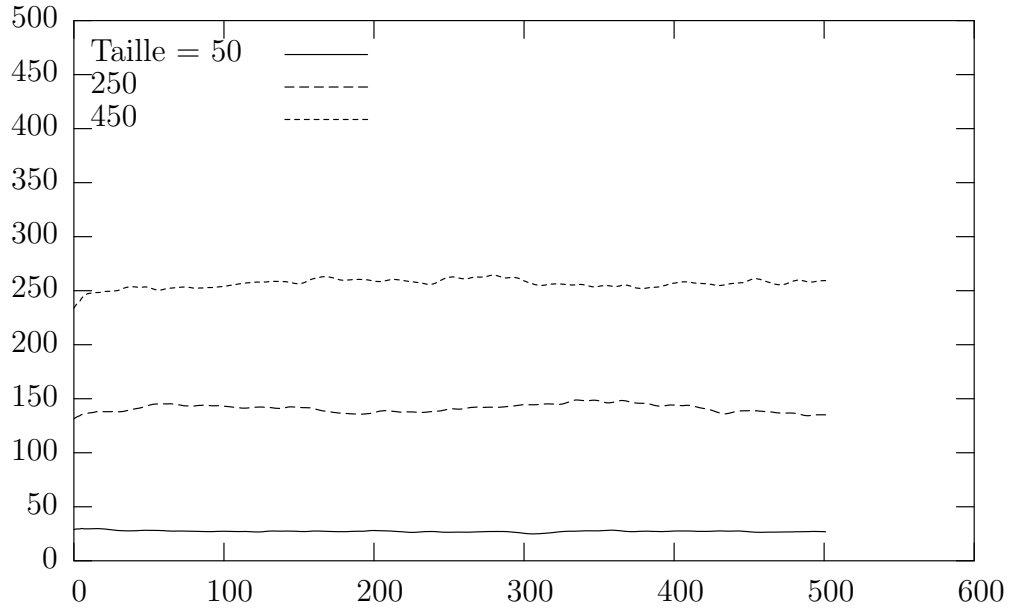

FIG. 4.17 – Paysage plat et CMH : Évolution de la taille moyenne avec mutation équilibrée

Mutation déséquilibrée

Nous proposons d'utiliser l'opérateur de mutation pour modifier la taille moyenne des programmes durant l'évolution. Nous avons reproduit les expériences précédentes (mêmes populations initiales) mais avec un réglage déséquilibré de la mutation, c'est-à-dire un excès d'insertions : $\mathcal{T}_i$ =1.5, $\mathcal{T}_d$ =0.5 et $\mathcal{T}_s$ =0.0. Ainsi chaque programme va "gagner" une instruction en moyenne à chaque génération. Les Figures 4.18 et 4.19 présentent l'évolution de la taille moyenne pour les deux opérateurs de croisement. On constate qu'avec le CS, la taille moyenne converge très rapidement (moins de 150 générations) autour de 130 instructions. On remarque que la valeur de convergence a été déplacée par rapport aux expériences avec mutation équilibrée (elle était de 75 instructions). Avant de converger, seule la courbe d'évolution du cas $\lambda_c = 50$ adopte le comportement attendu avec un coefficient directeur de la pente proche de 1. Quand la recombinaison est effectuée avec l'opérateur CMH, les trois courbes d'évolution ont un comportement croissant, avec un coefficient directeur de la pente proche de 1, jusqu'à ce que la taille moyenne atteigne λ_{max}.

Nous avons reproduit les expériences précédentes (même populations initiales) avec un réglage déséquilibré de la mutation, mais avec un excès de délétions : $\mathcal{T}_i$ =0.5, $\mathcal{T}_d$ =1.5 et $\mathcal{T}_s$ =0.0. Ainsi chaque programme va '"perdre" une instruction en moyenne à chaque génération. Les Figures 4.20 et 4.21 présentent l'évolution de la taille moyenne pour les deux opérateurs de croisement. On constate, pour les trois valeurs de λ_c, une décroissance de la taille moyenne, avec une valeur finale proche de 2.5 instructions pour le CMH et de 4 instructions pour le CS. L'évolution, dans le cas du CS, est plutôt asymptotique alors qu'elle est plus linéaire pour le CMH avec un coefficient directeur de la pente proche de -1.

Ainsi, une utilisation conjointe de l'opérateur CMH et de taux déséquilibrés d'insertions et de délétions permet de contrôler la taille moyenne des programmes sur un paysage plat.

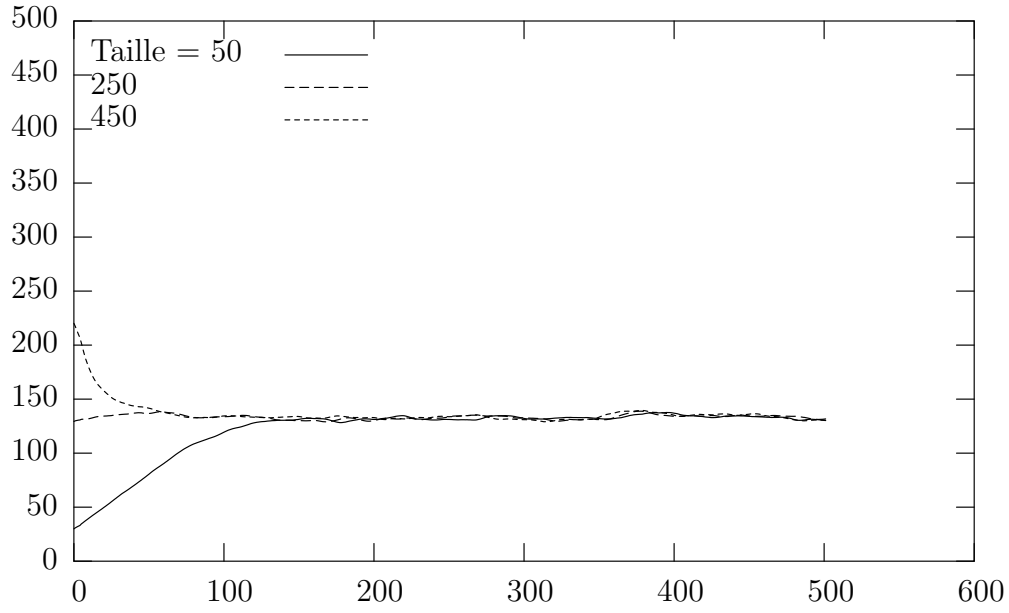

FIG. 4.18 – Paysage plat et CS : Évolution de la taille moyenne avec un excès d'insertions

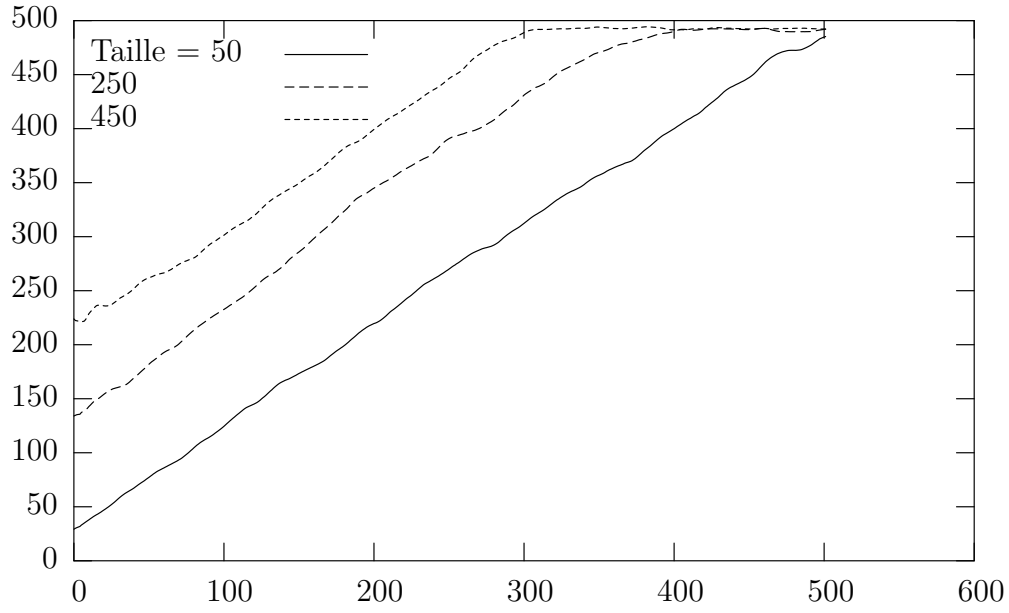

FIG. 4.19 – Paysage plat et CMH : Évolution de la taille moyenne avec un excès d'insertions

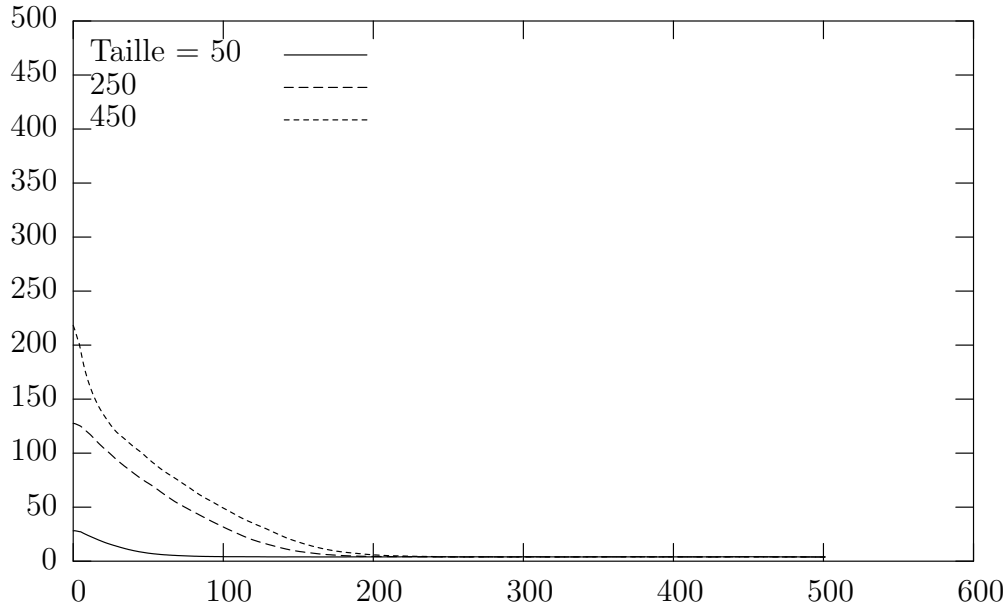

FIG. 4.20 – Paysage plat et CS : Évolution de la taille moyenne avec un excès de délétions

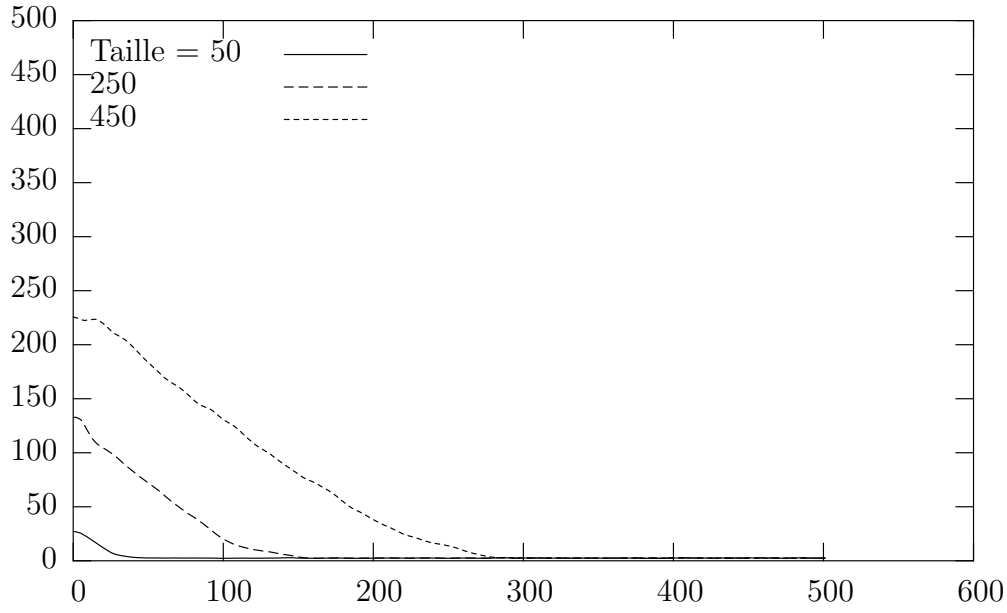

FIG. 4.21 – Paysage plat et CMH : Évolution de la taille moyenne avec un excès de délétions

4.2.2 Avec pression de sélection

Nous voulons vérifier si il est également possible de contrôler la taille des programmes quand la sélection guide la dynamique de la population. Nous avons réalisé une série de 50 expériences dans les mêmes conditions que précédemment sur le problème Even Parity pour N=10, avec une taille du tournoi de remplacement de 16 et avec un excès de délétions. Le taux de croisement optimal a été choisi à partir des résultats obtenus Table 4.2, c'est-à-dire $\mathcal{T}_c$ =0.8 pour le CMH et $\mathcal{T}_c$ =0.6 pour le CS.

Nous voyons, Figure 4.22, que pour le CS les trois courbes convergent vers 170 instructions et donc que la taille moyenne ne peut être que difficilement contrôlée avec l'opérateur de mutation. Pour le CMH, Figure 4.23, les courbes d'évolution converge vers 30 instructions ce qui est inférieur aux tailles obtenues pour un réglage équilibré. Ainsi, un réglage déséquilibré de la mutation permet d'imaginer de nouveaux types d'exploration de l'espace de recherche. En effet, à notre connaissance, c'est la première fois, qu'un système PG débute la recherche à partir de grands programmes pour finir avec des programmes de petites tailles. Ceci ne constitue certainement pas une stratégie optimale, bien que 100% des expériences aient permis de découvrir un optimum, mais montre expérimentalement qu'un contrôle de la taille des programmes peut être envisagé.

4.3 Deux méthodes pour contrôler la taille

Nous avons vu que le CMH permet de réduire sensiblement le phénomène de bloat mais que la taille des programmes recombinés évolue peu. Elle ne peut donc être optimisée et cela peut influencer grandement les performances du système. Nous avons déjà montré [26], sur un problème de référence, que nous pouvons orienter la recherche et réaliser un contrôle de la taille des programmes quand le CMH est utilisé conjointement avec un réglage déséquilibré de la mutation.

Plus généralement, nous pensons que c'est la distribution des tailles induite par la recombinaison qui détermine la dynamique des populations sou-

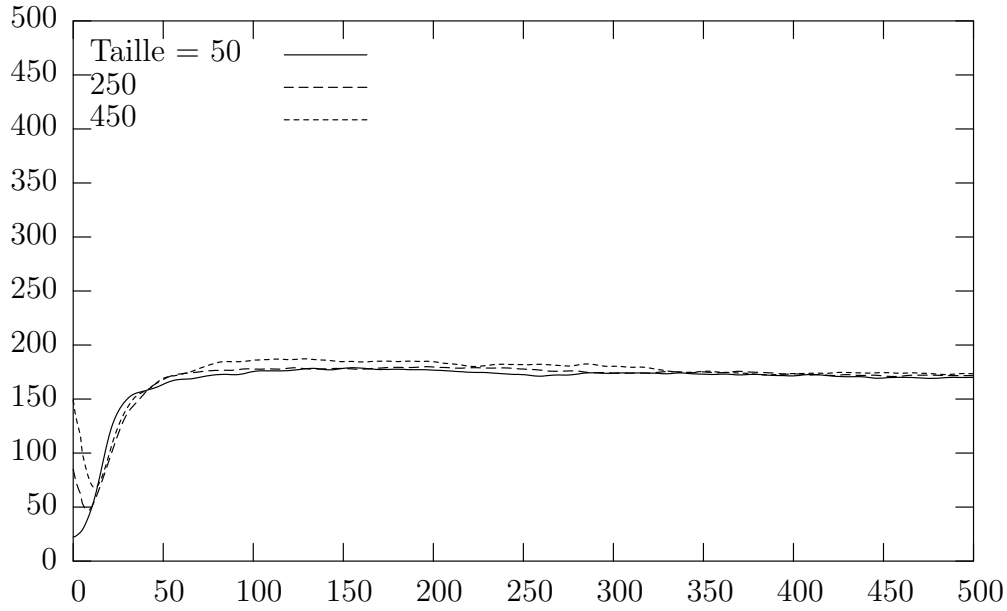

FIG. 4.22 – Problème Even Parity, N=10 et CS : Évolution de la taille moyenne avec délétion et substitution

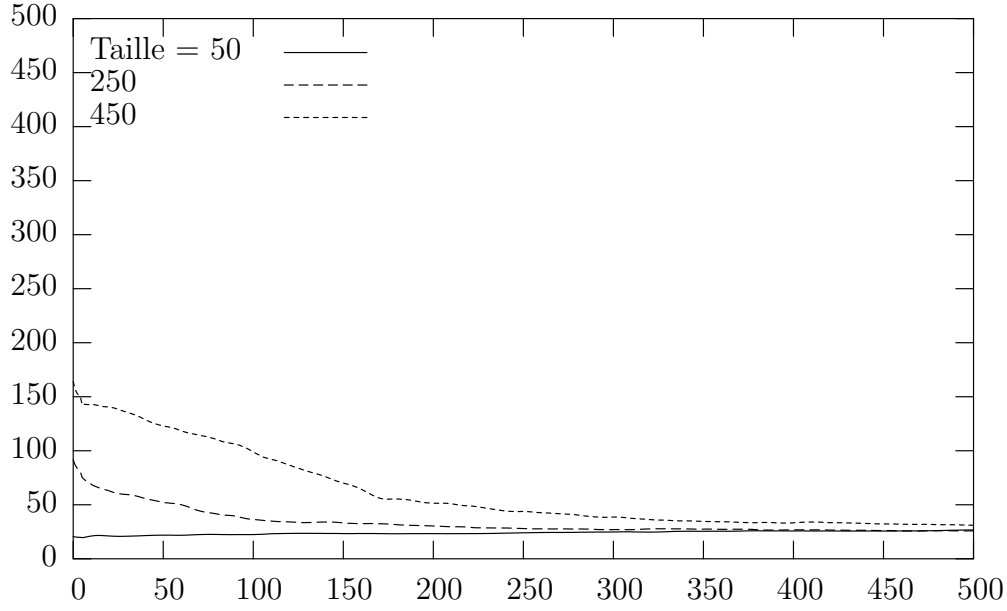

FIG. 4.23 – Problème Even Parity, N=10 et CMH : Évolution de la taille moyenne avec délétion et substitution

mises à la pression de sélection. En effet, dans le cas du CS, au tout début d'une expérience, la sélection opère sur des programmes majoritairement beaucoup plus petits que la moyenne de la population, qu'elle a tendance à éliminer, conformément aux hypothèses de la dérive et du biais (cf. Section 1.2.3). Les programmes autorisés à se reproduire sont donc souvent plus longs que la moyenne de la population, ce qui entraîne une croissance rapide de la taille des programmes. Par la suite, les programmes sont majoritairement plus longs que la moyenne et il devient de plus en plus difficile de trouver de bons programmes courts. Quand l'opérateur CMH est utilisé, les programmes soumis à la sélection ont des tailles distribuées quasi symétriquement par rapport à la moyenne et sont rarement disproportionnés. C'est pourquoi, l'évolution de la taille moyenne est lente et dépend de la distribution initiale dans la population.

Parce que la distribution de la taille des programmes recombinés par CS est fortement biaisée, les méthodes de contrôle de la taille existantes agissent, la plupart du temps, sur le schéma de sélection des individus, comme par exemple la pression de parcimonie, l'optimisation multi-objectifs ou la méthode Tarpéienne (cf Section 1.2.3). Au contraire, le CMH permet d'envisager des stratégies de contrôle où la sélection est laissée libre. Nous proposons deux méthodes de contrôle qui modifient dynamiquement la distribution de la taille des programmes pour influencer la recherche et ainsi permettre à la sélection de découvrir les régions de l'espace de fitness élevée. Les performances de ces méthodes seront évaluées dans le prochain chapitre.

4.3.1 Translation de la distribution

A partir du constat que l'utilisation de taux de mutation déséquilibrés permet de modifier la taille moyenne des programmes durant l'évolution, nous définissons l'opérateur CMH+INS$_r$ comme la conjonction de l'opérateur CMH et de l'opérateur d'insertion appliqué avec un taux r. L'opérateur CMH+INS$_r$ utilisé avec r positif permet de translater la distribution des tailles vers des valeurs élevées. On note que dans [13], un réglage déséquilibré de la mutation a été préconisé dans le contexte de la PGL avec un opérateur

de croisement dit homologue.

La Figure 4.24 présente la distribution de la taille des programmes, en échelle logarithmique, pour l'opérateur CMH+INS$_{1.0}$ aux générations 0,1,100 et 1000. Ces distributions sont des moyennes sur 200 expériences et sont calculées à partir de l'évolution de la population sur un paysage plat. Nous avons fixé $\mathcal{T}_m$ =0 et $\mathcal{T}_c$ =0.5. Elles peuvent être comparées aux distributions du CS et du CMH (cf. Figures 2.9 et 2.10) obtenues dans des conditions expérimentales équivalentes.

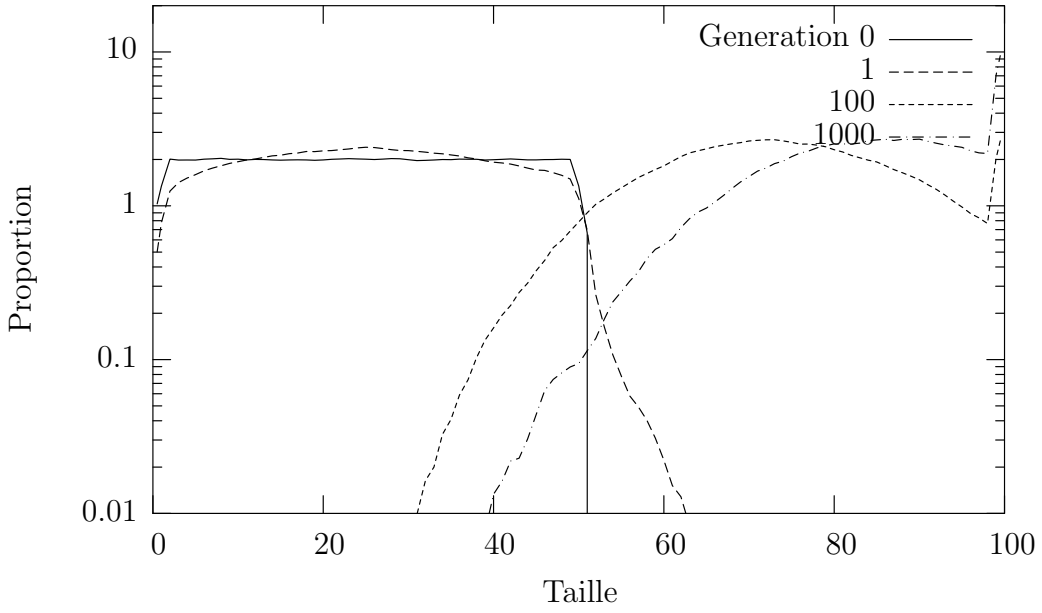

FIG. 4.24 – Paysage plat et CMH+INS$_{1.0}$: Évolution de la distribution de la taille des programmes

On constate, conformément à nos attentes, que la distribution est effectivement translatée, au cours du temps, vers les tailles élevées. Elle converge vers une distribution gaussienne et l'on note, aux générations 100 et 1000, de fréquentes "collisions" avec la limite λ_{max} qui n'était jamais atteinte quand le CMH était utilisé seul. La taille 100 est la plus représentée à la génération 1000 avec 16.7% de la population.

4.3.2 Transformation de la distribution

Nous savons que l'opérateur CS induit une distribution particulière de la taille des programmes avec un sur-échantillonage des programmes courts mais également la présence de programmes anormalement longs par rapport à la moyenne de la population. En combinant l'opérateur CMH et l'opérateur CS, on peut transformer la distribution et ainsi permettre à la recherche de tirer parti de ces programmes "anormaux". Nous définissons l'opérateur CMH+CS$_p$ avec $(1-p)$ la probabilité que les recombinaisons soient effectuées avec l'opérateur CMH et p la probabilité que les recombinaisons soient réalisées avec l'opérateur CS. Ainsi, CMH+CS$_{0.5}$ correspond à l'utilisation équiprobable du CMH et du CS. On note que dans [22], une combinaison d'opérateurs de croisement dits *size fair* (qui respectent la taille) et de l'opérateur standard a été utilisée avec succès.

La Figure 4.25 présente la distribution de la taille des programmes, en échelle logarithmique, pour l'opérateur CMH+CS$_{0.2}$ aux générations 0,1,100 et 1000. Ces distributions sont des moyennes sur 200 expériences et sont calculées à partir de l'évolution de la population sur un paysage plat. Nous avons fixé $\mathcal{T}_m$ =0 et $\mathcal{T}_c$ =0.5. Elles peuvent être comparées aux distributions du CS et du CMH (cf. Figures 2.9 et 2.10) obtenues dans des conditions expérimentales équivalentes.

On peut vérifier que la distribution des tailles correspond bien à un compromis entre celles des deux opérateurs. En effet, des programmes plus longs, que dans le cas où le CMH est employé seul, sont présents et la taille 10 est la plus représentée à la génération 1000 avec 3.7% de la population, alors qu'il s'agissait de la taille 2 pour l'opérateur CS. De plus, en calculant la somme pondérée des distributions du CS et CMH, nous avons obtenu des distributions comparables à celles de l'opérateur CMH+CS$_p$.

4.4 Discussion

L'étude des propriétés des populations recombinées, nous permet de mieux comprendre les effets des opérateurs de croisement. Nous avons vu que la dis-

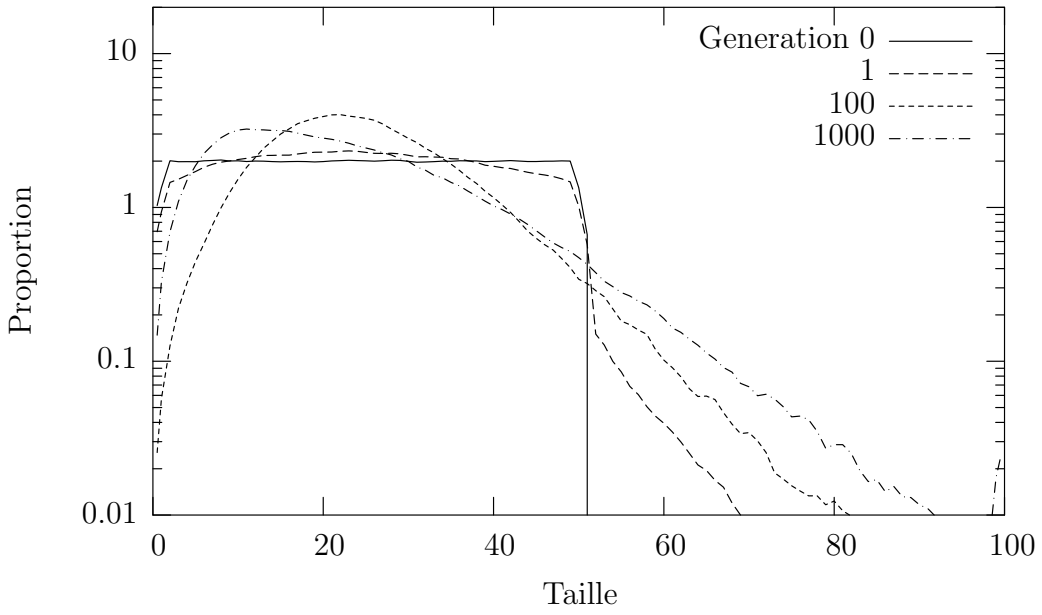

FIG. 4.25 – Paysage plat et CMH+$CS_{0.2}$: Évolution de la distribution de la taille des programmes

tribution de la taille des programmes est plus compacte avec le CMH qu'avec le CS, et ce avec ou sans pression de sélection. C'est pourquoi, pour évaluer la diversité génotypique des populations soumises à une pressions de sélection, nous avons utilisé l'homologie relative moyenne, $Z_{gen}(M)$, qui prend en compte la variabilité de la taille des programmes dans la population. Malgré cela, nous avons vu que $Z_{gen}(M)$ diminue plus rapidement quand le CMH est utilisé, ce qui est conforme à la propriété de respect (cf. Équation 2.11) que seul le CMH vérifie. Les effets des deux opérateurs sont profondément différents ce qui a des conséquences sur la nature de la recherche réalisée et donc sur les performances du système.

4.4.1 Mélange, diffusion et exploration de la taille

Dans [60], Poli met en avant deux types de recherche réalisées par l'opérateur CS : le mélange et la diffusion.

Le mélange est un échange d'instructions entre les programmes, c'est cette

opération qui permet d'exploiter les différentes qualités des individus de la population. Conformément au théorème de Geiringer pour la PGL [82], un opérateur est un "mélangeur" si, quand il est appliqué de façon répétée sur une population infinie sans pression de sélection et sans mutation, la proportion d'un symbole s à une position donnée des programmes dépend de la proportion de s à cette même position dans la population initiale. c'est-à-dire qu'elle ne dépend pas de la proportion des autres symboles de l'alphabet aux autres positions possibles. Autrement dit, à chaque génération, la proportion d'un symbole à une position donnée est conservée par l'opérateur.

La diffusion représente quant à elle, les déplacements, au sein d'un individu, de ses propres gènes. Un opérateur est un "diffuseur" pour la PGL si, quand il est appliqué de façon répétée sur une population infinie sans pression de sélection et sans mutation, la proportion d'un symbole s est indépendante de sa position dans les programmes. c'est-à-dire que tous les symboles sont présents dans leurs proportions initiales à toutes les positions possibles. Autrement dit, à chaque génération, la proportion d'un symbole dans la population est conservée par l'opérateur.

Nous pensons que l'opérateur CS réalise également un troisième type de recherche : l'exploration en taille. Cette recherche semble indispensable en PG et plus généralement en EA-TV mais elle est réalisée de manière implicite et surtout incontrôlée par l'opérateur CS.

A l'opposé, nous avons vu que le CMH ne permet pas d'explorer l'espace des tailles et que cela peut constituer une réelle faiblesse en terme de performance. Le CMH est, au même titre que le CS, un mélangeur pour la PGL puisqu'il réalise un échange d'instructions entre les individus de la population et que toutes les instructions d'un programme sont susceptibles d'être échangées. En revanche, nous avons vu Section 2.3.5 que la position des instructions est prise en compte durant la recombinaison, c'est pourquoi le CMH n'est pas un diffuseur pour la PGL. En effet, les instructions mélangées sont toujours comprises entre deux positions consécutives d'une plus longue sous-séquence commune aux parents. Ainsi, la diffusion au sein des individus d'une population est très limitée.

Contrairement à Poli, nous ne disposons pas d'un cadre formel, comme

le théorème exact des schémas, pour quantifier les phénomènes de diffusion dans la population, c'est pourquoi nous proposons de les étudier empiriquement. Nous avons effectué une série de 50 expériences indépendantes pour les opérateurs CS et MHC. Durant ces expériences, la taille maximum des programmes autorisée est λ_{max}=1000 gènes. L'évolution, sans pression de sélection, avec $\mathcal{T}_m$ =0 et $\mathcal{T}_c$ =1.0, est limitée à 100 générations. La population, contenant 2.10^4 individus, est initialisée de façon très particulière (cf. Figure 4.26). La taille des individus est distribuée uniformément entre 1 et λ_c=50

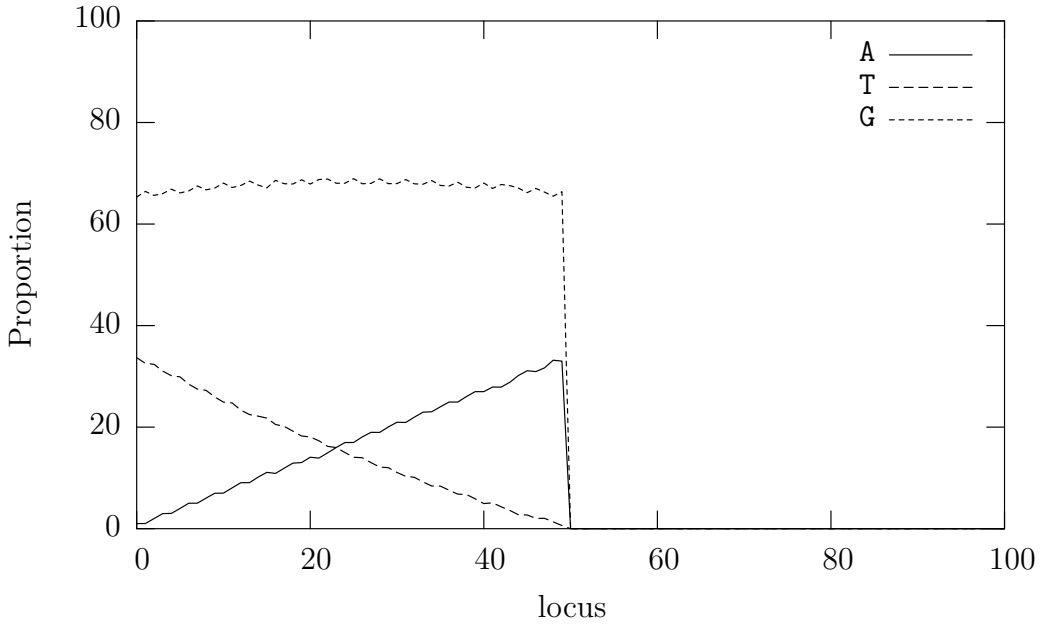

FIG. 4.26 – Paysage plat : Proportions (en %), à la première génération, des instructions `A`, `T` et `G` en fonction de leur position

gènes. Les programmes sont exclusivement composés des trois instructions `A`, `T` et `G` dont les proportions par locus dans la population sont connues :

- la proportion d'instruction `A` croit linéairement entre 0% au locus 1 et 33% au locus 50 ;
- la proportion d'instruction `T` décroît linéairement entre 33% au locus 1 et 0% au locus 50 ;
- la proportion d'instruction `G` est constante et vaut 66% pour tous les

locus.

Nous avons tracé, Figure 4.27 et 4.28, les proportions, à la génération 100, des instructions `A`, `T` et `G` en fonction de la position dans les programmes (entre les locus 1 et 100), pour les opérateurs CS et CMH. On peut vérifier, pour le CS, que les phénomènes de diffusion des instructions ont réparti les trois instructions `A`, `T` et `G` entre toutes les positions des programmes. Les proportions sont donc bien indépendantes des loci et sont proches des proportions moyenne dans la population initiale, *i.e.* 16.6% d'instructions `A` et `T` et 66.6% d'instruction `G`. Les différences constatées sont probablement dues à des erreurs d'échantillonage puisque la population n'est pas infinie. Au contraire, avec l'opérateur CMH, les proportions des trois instructions sont restées proches des proportions initiales et dépendent de la position dans les programmes. Les différences constatées sont également dues à la taille de la population, mais aussi à la disparition des programmes composés de plus de 40 instructions[9]. On peut déduire de cette expérience, que le CMH n'est pas un diffuseur pour la PGL.

4.4.2 Découplage

L'opérateur CS n'a pas été conçu pour réaliser le mélange, la diffusion et l'exploration de la taille. En fait, ce sont vraisemblablement plutôt des choix techniques et d'implémentation qui ont amené Koza à définir le CS tel que nous le connaissons. Cependant ces trois types de recherches peuvent, en fonction du type de problème à résoudre, être nécessaires pour espérer pouvoir découvrir des programmes performants. Nous pensons qu'un système PG devrait disposer de trois opérateurs spécifiques pour le mélange, la diffusion et l'exploration en taille, ou tout au moins qu'il soit possible d'ajuster la part des trois types de recherches dans les mécanismes de variations. On se souviendra que dans les AG, à l'origine [42], Holland préconise l'utilisation d'un opérateur d'inversion dont la fonction pourrait être comparée à la diffusion.

Nous avons vu que le CMH est, quasi exclusivement, un mélangeur pour la PGL. Il s'agit donc d'un bon candidat pour le découplage du mélange,

[9]On notera que moins de 1% de la population est composée d'individus ayant une taille supérieure à 35 instructions.

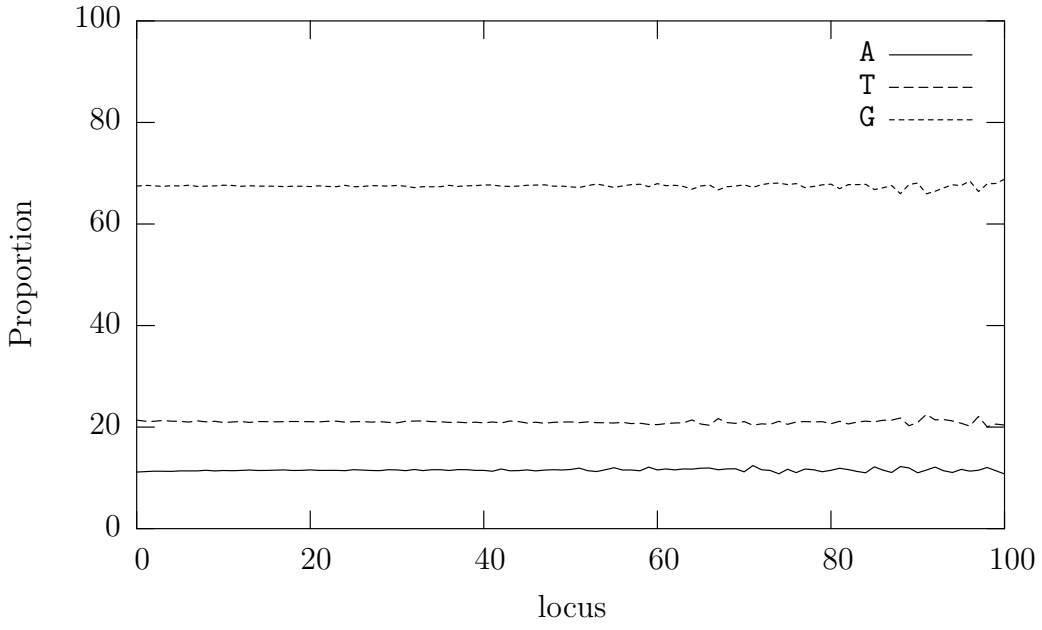

FIG. 4.27 – Paysage plat et CS : Proportions (en %), à la génération 100, des instructions `A`, `T` et `G` en fonction de la position

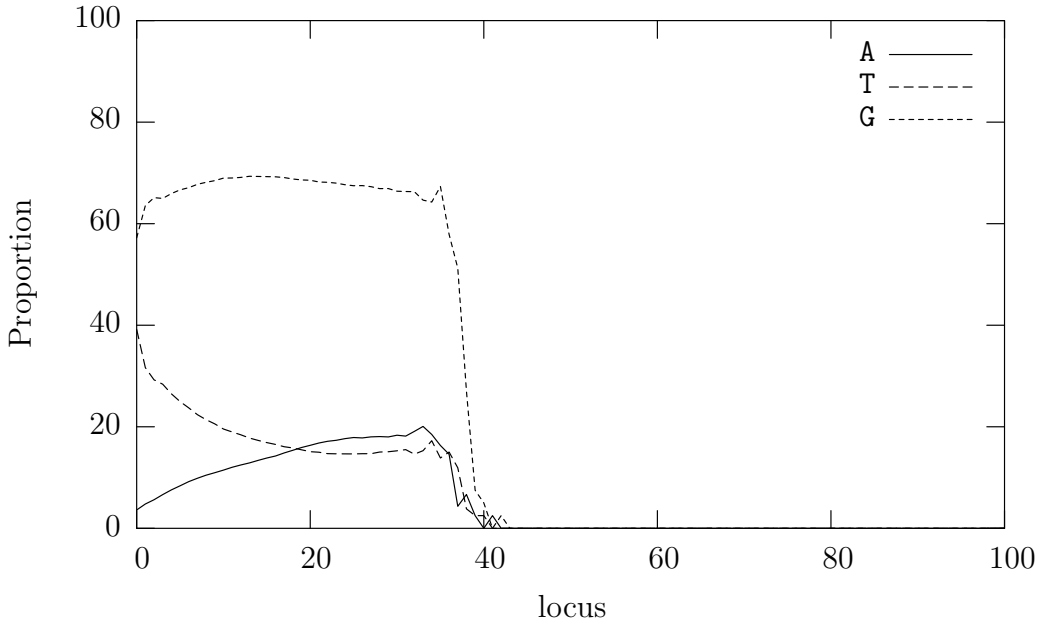

FIG. 4.28 – Paysage plat et CMH : Proportions (en %), à la génération 100, des instructions `A`, `T` et `G` en fonction de la position

de la diffusion et de l'exploration dans le parcours de l'espace de recherche. Cette hypothèse devra être examinée par des travaux ultérieurs. Cependant, on notera que l'étude des performances des deux opérateurs de contrôle de la taille présentée dans le chapitre suivant pourra être considérée comme une première confirmation expérimentale de cette hypothèse.

Chapitre 5

Homologie, application à la résolution d'un problème inverse

Alors que pour résoudre un problème direct, on cherche à décrire une relation de causes à effets, dans le cas d'un problème inverse, on doit retrouver les causes à partir d'une série de mesures des effets. Les problèmes inverses sont souvent plus difficiles à résoudre que les problèmes directs, car la quantité et la qualité des mesures disponibles sont généralement insuffisantes pour décrire entièrement tous les effets et, par là même, pour retrouver l'ensemble des causes. De plus, comme différentes causes peuvent produire les mêmes effets, la solution du problème peut ne pas être unique : dans ce cas, le problème est dit "mal posé".

La résolution de problèmes inverses est devenue, dans de nombreux domaines scientifiques, une thématique importante. On peut expliquer ce phénomène par, à la fois un net accroissement de la quantité de mesures disponibles et, suite à l'essor des sciences et techniques de l'information, des moyens de les traiter et de les stocker, mais également par l'arrivée à maturité de méthodes, analytiques ou stochastiques, permettant de réaliser l'inversion. En particulier les Réseaux de Neurones Artificiels (RNA) ont déjà démontré avec succès leur capacité à modéliser des problèmes inverses ; nous citerons par exemple [85]. Dans ce contexte, la PG s'est avérée, d'après des travaux récents [20][17], être également une méthode adaptée.

Dans ce chapitre, nous évaluons les performances du système PGP appliqué à la résolution d'un problème inverse. Nous présentons également une amélioration de ce système, appelée approche par équipe de prédicteur, qui facilite la recherche et qui permet d'accroître la quantité d'instructions réellement utilisées dans les programmes. Nous étudions aussi le comportement des méthodes de contrôle de la taille, en particulier en ce qui concerne le compromis fitness *vs* taille.

5.1 Inversion des composantes atmosphériques

5.1.1 Présentation du problème

Le problème inverse que nous allons aborder dans ce chapitre est un problème réel qui a trait aux caractéristiques des aérosols atmosphériques[1]. Une connaissance précise de ces caractéristiques est décisive car les aérosols ont un rôle essentiel dans le bilan radiatif terrestre, *i.e.* le bilan de l'énergie reçue et émise par la terre, et qu'ils ont donc une incidence significative sur le climat. Dans cette étude, nous nous intéressons aux caractéristiques des aérosols dans le domaine de la télédétection, entre autre pour permettre la validation des mesures réalisées par des capteurs satellitaires.

La société ACRI-ST , qui a co-financé ce travail de thèse, concentre ses activités dans le domaine des sciences de la terre et plus particulièrement dans la surveillance de l'environnement. Dans le contexte écologique mondial, cette problématique est devenue un enjeu majeur et le nombre de projets réalisés dans ce domaine est en continuelle expansion. A l'heure actuelle, nous disposons d'une multitude d'instruments de mesures et la quantité de données physiques, chimiques ou optiques ne cessent de croître. La société ACRI-ST est particulièrement impliquée dans le cycle de vie de différents projets spatiaux liés à la surveillance de l'environnement, de la mise au point à la validation des capteurs et jusqu'à l'exploitation et la livraison des données scientifiques.

Dans le cas d'un satellite d'observation, réalisant la surveillance des océans

[1]Les aérosols sont des particules en suspension dans l'atmosphère (aérosol atmosphérique est un abus de langage couramment utilisé).

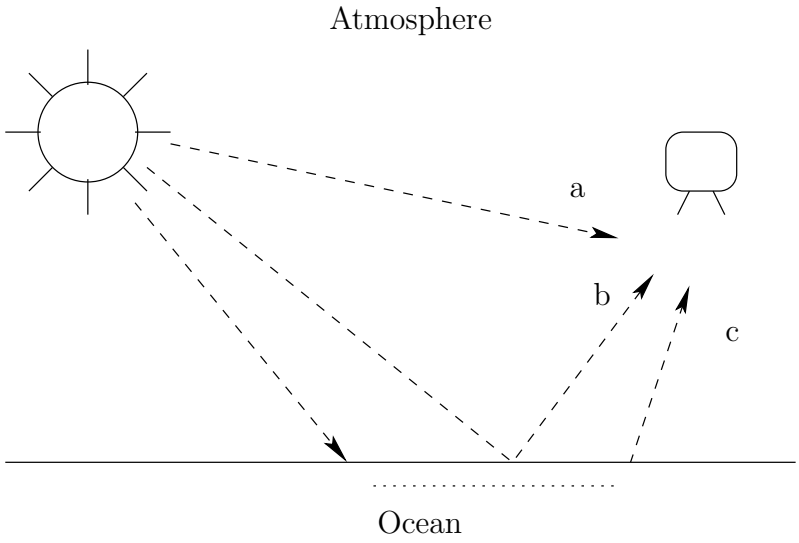

FIG. 5.1 – Les signaux perçus par un satellite sont influencés par les milieux qu'ils rencontrent : a) l'atmosphère ; b) l'interface océan-atmosphère ; c) l'océan.

par exemple, le signal lumineux mesuré nous informe de la nature et de la composition des milieux traversés, cf Figure 5.1. On peut donc raisonnablement espérer, après analyse du signal marin, déterminer, par exemple, les concentrations des différentes composantes océaniques. Cependant, le signal mesuré par un satellite est également largement influencé par l'atmosphère qu'il traverse et si l'on ne s'intéresse qu'exclusivement au signal marin, il convient donc de corriger le signal mesuré des "perturbations" engendrées par l'atmosphère. Dans certain cas, en océan ouvert par exemple, cette correction atmosphérique est relativement aisée et pourra être réalisée en comparant le signal reçu à différentes longueurs d'ondes. A l'inverse, en milieu côtier, cette comparaison n'est pas suffisante et certaines ambiguïtés ne peuvent être levées.

Pour résoudre ce problème, une idée consiste à déterminer, à partir de mesures réalisées au sol, les différentes composantes atmosphériques. Le réseau AERONET (Aerosol Robotic Network) fournit des relevés réguliers de luminances[2] du ciel mesurées à partir de radiomètres situés dans de très nom-

[2]La luminance correspond à l'énergie parvenant à un récepteur (ou capteur) par unité

breuses régions côtières du monde, voir Figure 5.2. Notre objectif consiste donc à trouver des modèles d'inversion de ces mesures de luminances pour caractériser les propriétés de l'atmosphère et ainsi permettre la correction des données satellitaires. On s'intéressera tout particulièrement aux caractéristiques des aérosols atmosphériques.

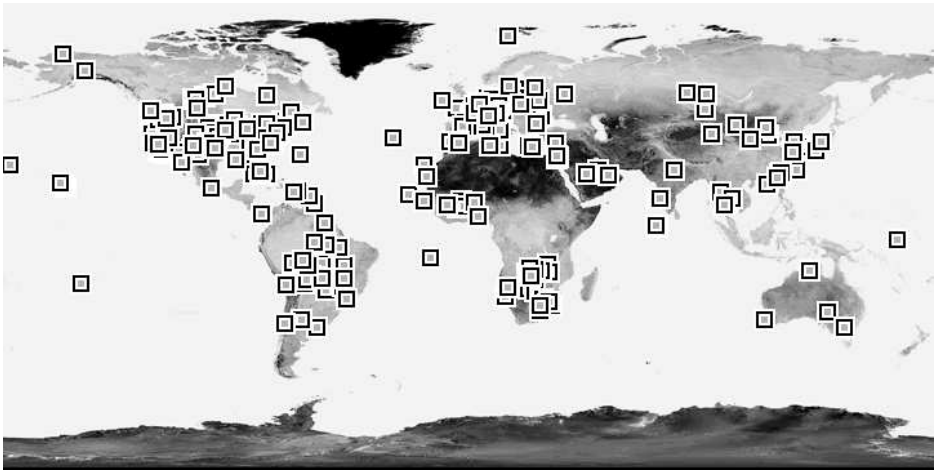

FIG. 5.2 – Le réseau AERONET.

5.1.2 Méthodologie

Nous proposons de résoudre le problème de l'inversion des composantes atmosphérique par modélisation statistique. De façon générale, un modèle statistique est construit à partir d'une base d'exemples reflétant les relations entre les entrées et les sorties du problème à résoudre et la phase de construction du modèle est appelée apprentissage ou entraînement.

Une grande quantité de données est souvent nécessaire pour réaliser correctement l'apprentissage de modèles inverses. Par ailleurs, l'apprentissage doit être effectué à partir de données les plus réalistes possible pour que les modèles soient entraînés en tenant compte, par exemple, des variabilités naturelles et des erreurs de mesures. Cependant, en fonction du problème inverse traité, il peut s'avérer difficile de collecter assez de mesures pour couvrir

de surface, par unité de temps et dans une direction donnée.

entièrement, et de façon suffisamment dense, le domaine d'entrée. Il est alors communément admis, pour réaliser l'apprentissage, de recourir à des données issues d'une simulation puis de bruiter ces données[3] pour tenir compte des erreurs de mesures [35][17]. Pour synthétiser le jeu d'apprentissage, nous utilisons un code de transfert radiatif comme modèle direct.

Nous voulons inverser un modèle de transfert radiatif, appelé modèle *Ordres Successifs Océan Atmosphère* (OSOA), qui est décrit en détail dans [18]. Ce modèle direct résout une équation de transfert radiatif par la méthode des ordres successifs de diffusion pour le système océan-atmosphère, mais dans notre cas, l'océan ne sera pas considéré. Il prend en compte les multiples évènements de diffusion et de polarisation de la lumière pour le milieu atmosphérique. L'atmosphère est vue comme un mélange de molécules et d'aérosols que l'on suppose être des sphères homogènes. Le modèle d'aérosol est défini par quatre paramètres :

- l'épaisseur optique à 675 nm et la pente de Junge qui caractérisent la distribution de la taille ;
- la partie réelle et la partie imaginaire de l'indice de réfraction.

Ces propriétés optiques sont calculées à l'aide de la théorie de la diffusion de Mie [69]. Le modèle OSOA fournit la distribution angulaire du champ de luminance ainsi que son degré de polarisation.

Pour limiter le nombre d'ambiguïtés dans le problème d'inversion et ainsi s'assurer qu'il n'est pas "mal posé", les modèles sont spécialisés pour chacun des quatre paramètres, pour certains angles solaires et pour tenir compte des spécificités régionales et de la saisonnalité. Cependant, dans cette étude, notre objectif n'est ni de réaliser l'ensemble de ces modèles d'inversions ni de montrer l'intérêt des données de polarisation, ce qui a d'ores et déjà été fait à l'aide de RNA (cf [16]), mais plutôt d'étudier la faisabilité de l'inversion par PG. C'est pourquoi nous nous limitons au modèle d'inversion des luminances pour l'épaisseur optique des aérosols à 675 nm, appelée τ_a, pour un angle solaire compris entre 70 et 75 degrés.

[3]Les connaissances *a priori* des experts du domaine, concernant par exemple les lois physiques et les relations entre les variables, peuvent être directement prises en compte pour réaliser l'opération de bruitage.

5.1.3 Inversion par PG

Comme notre problème inverse est relativement bien posé, réaliser une inversion par PG revient à rechercher une expression mathématique, c'est-à-dire un programme, permettant d'approximer au mieux la relation entre les entrées et les sorties du problème à résoudre. Dans notre cas, il s'agit donc d'effectuer une Régression Symbolique Réelle (RSR). Cette problématique a été largement abordée en PG et de nombreux problèmes de références existent avec chacun leur spécificité. Pour étudier les performances du système PGP en RSR et réaliser des tests préliminaires, nous utilisons le problème Poly-10, introduit dans [78], en particulier parce que la fonction à découvrir est définie sur un nombre de variables assez important, comme c'est le cas pour ce problème d'inversion.

Une amélioration récente, présentée dans [48], appelée Linear Scaling (LS), permet d'améliorer grandement les performances de PG en RSR. L'idée est simple et très efficace, elle consiste à effectuer une transformation affine de la sortie des programmes de la forme :

$$Ls(o(p)) = a \times o(p) + b$$

avec $o(p)$ la sortie d'un programme. La valeur des paramètres a et b peut être obtenue par régression linéaire avec une complexité en temps $\mathcal{O}(N)$ pour N le nombre de vecteurs dans le jeu d'apprentissage. Le LS permet en fait de "soulager" un système PG de la recherche des deux constantes a et b et lui permet de se "concentrer" sur la recherche des programmes dont l'expression à une forme la plus proche possible de la fonction à approximer, on peut donc dire que la recherche s'effectue à une translation et une homothétie près. Nous utilisons le LS pour toutes les expériences réalisées dans ce chapitre et pour simplifier les notations la sortie transformée d'un programme sera notée $o(p)$.

5.2 Équipe de Prédicteurs

L'objectif des méthodes d'apprentissage est de trouver des prédicteurs qui, après entraînement sur un ensemble d'exemples, sont capables d'approximer la fonction qui a généré ces exemples. Différentes méthodes sont

connues pour être des approximateurs universels[4], c'est-à-dire qu'à partir d'un ensemble d'exemples, une fonction donnée peut être "apprise" avec une précision arbitraire. Cependant, la recherche de prédicteurs optimaux peut être difficile à réaliser, par exemple en raison de la complexité dans le choix d'une architecture adaptée ou bien encore à cause du sur-apprentissage sur le jeu d'entraînement. On parle de sur-apprentissage quand un prédicteur a appris "par coeur" une fonction ou bien quand il reflète plus les variations aléatoires des données d'entrées, *i.e.* le bruit, que les propriétés de la fonction à approximer, ce qui entraîne souvent de faibles performances en terme de capacité de généralisation du prédicteur.

Pour lutter contre le sur-apprentissage, plusieurs solutions ont été proposées et l'on peut citer entre autres : la sélection de modèles, l'arrêt de l'apprentissage ou la combinaison de prédicteurs (voir [93]). Dans cette section, nous nous intéressons aux méthodes de combinaisons de prédicteurs, également appelées méthodes d'ensembles ou de comités. Un comité correspond à une équipe de prédicteurs obtenus par apprentissage, où la prédiction réalisée est une combinaison de la prédiction des membres de l'équipe. Les performances d'une équipe sont, la plupart du temps, in-envisageables avec un prédicteur unique, car les erreurs commises par chacun des membres peuvent se compenser quand les différentes prédictions sont combinées. De multiples techniques de combinaison ont été envisagées, comme par exemple la moyenne simple, les mixtures d'expert, le Bagging ou le Boosting. Dans la pratique, les méthodes d'ensemble appliquées aux RNA ont démontré leur intérêt, voir [57]. En PG, des travaux ont également été menés avec succès dans le domaine de la combinaison de prédicteurs [66][108][49][11][76].

Nous proposons d'utiliser un des aspects du système PGP pour réaliser l'évolution d'équipes de prédicteurs dont le nombre, contrairement aux études précédentes dans ce domaine, peut varier durant la convergence.

[4]En faisant quelques hypothèses sur le jeu d'instructions utilisés pour construire les programmes, il a déjà été montré [107] que la PG est un approximateur universel.

5.2.1 PGP et sous-programmes

Nous avons vu, Section 1.3, comment est réalisée l'évaluation d'un programme en PGP, en particulier nous en avons étudié un exemple présenté Figure 1.4. En revenant à cet exemple, on constate qu'après l'étape *f*, la pile d'opérandes contient les valeurs 1 et 3. Ceci signifie que la séquence évaluée, *i.e.* `1 ADD 5 2 SUB`, correspond, en fait, à deux sous-programmes indépendants `1` et `5 2 SUB`. De manière classique, la sortie d'un programme p, notée o(p) est lue au sommet de la pile. Dans notre cas, seule l'exécution de `5 2 SUB` est prise en compte.

Nous proposons d'inclure l'ensemble des sous-programmes dans le processus d'évaluation de la fitness, en combinant la sortie de chacun des sous-programmes, *i.e.* toutes les valeurs de la pile d'opérandes finale. Naturellement, le type de combinaison à réaliser est dépendant du type de problème à résoudre ; dans le cas d'une RSR toute les combinaisons linéaires, par exemple, peuvent être envisagées.

Grâce à cette nouvelle méthode d'évaluation, le système peut contrôler les effets des opérateurs génétiques en changeant le nombre de sous-programmes et donc en modifiant la contribution de chacun d'eux à la fitness globale. Ainsi, la recherche est de plus en plus fine (locale) à mesure que le nombre de sous-programmes augmente car la contribution de chacun d'eux diminue, et inversement. De plus, la quantité d'instructions inutilisées, souvent très importante en PGP (cf Table 4.1), est significativement réduite ce qui implique une diminution proportionnelle du nombre de variations génétiques (mutations et croisements) neutres et ainsi une meilleure performance générale du système. Cette approche peut être vue comme l'évolution d'équipes d'un nombre variable de prédicteurs formées par la combinaison des sous-programmes.

5.2.2 Différentes combinaisons

Soit x un programme composé de m instructions pouvant être interprétées comme une équipe de k prédicteurs p_i, avec $i \in [1, k]$ et $k \in [1, m]$. La

sortie du programme, c'est-à-dire la sortie de l'équipe, notée $O(x)$[5], est une combinaison des sorties de chaque prédicteur $o(p_i)$. Nous proposons d'étudier différentes combinaisons de prédicteurs :

- Somme des sorties des prédicteurs, notée `Sm`
$$O(x) = \sum_{i=1}^{k} o(p_i)$$
- Moyenne Arithmétique des sorties des prédicteurs, notée `MoyA`
$$O(x) = \frac{1}{k} \sum_{i=1}^{k} o(p_i)$$
- Produit des sorties des prédicteurs, notée `Prd`
$$O(x) = \prod_{i=1}^{k} o(p_i)$$
- Moyenne Géométrique des sorties des prédicteurs, notée `MoyG`
$$O(x) = \sqrt[k]{\prod_{i=1}^{k} o(p_i)}$$
- Moyenne Pondérée des sorties des prédicteurs, notée `MoyP`
$$O(x) = \frac{1}{\sum_{i=1}^{k} w_i} \sum_{i=1}^{k} w_i o(p_i)$$
avec $w_i = e^{-\beta E(p_i)}$, β un réel positif et $E(p_i)$ l'erreur RMS d'apprentissage du prédicteur p_i
- Meilleure des sorties des prédicteurs, notée `Best`
$$O(x) = o(argmin_{p_i, i \in [1,k]}(E(p_i)))$$
avec $E(p_i)$ l'erreur RMS d'apprentissage du prédicteur p_i
- Sortie du prédicteur au sommet de la pile, notée `Top`, c'est la méthode classique d'évaluation en PGP, où une seule sortie est prise en compte.
$$O(x) = o(p_k)$$

5.3 Régression symbolique réelle

Des tests préliminaires sont nécessaires avant d'étudier l'inversion des composantes atmosphériques. En effet, la taille du jeu d'apprentissage du problème inverse est très importante et le temps de calcul requis ne permet pas, par exemple, de rechercher le réglage optimal de $\mathcal{T}_m$ et $\mathcal{T}_c$. C'est pourquoi, dans cette section, nous recherchons ce réglage sur un problème de référence en RSR, le problème Poly-10, pour différentes combinaisons de prédicteurs. Puis, nous analysons, sur ce problème, les performances des deux

[5]La technique de LS décrite précédemment (cf Section 5.1.3) sera appliquée à $O(x)$ et pas à la sortie de chaque prédicteur.

méthodes de contrôle de la taille.

Définition du problème

Dans le problème Poly-10, la fonction à découvrir est un polynôme $\mathcal{P}$ de degré 3 défini sur 10 variables : $\mathcal{P}(x_1, \ldots, x_{10}) = x_1x_2 + x_3x_4 + x_5x_6 + x_1x_7x_9 + x_3x_6x_{10}$. Le jeu d'apprentissage $\mathcal{A}$ contient 50 vecteurs d'entrées v_i et a été généré en assignant aléatoirement des valeurs comprises entre $[-1, 1]$ aux 10 variables du polynôme $\mathcal{P}$.

Le système PGP décrit Section 1.3 est utilisé et la sortie $O(p)$ d'un programme p sera évaluée comme une combinaison des différents sous-programmes de p. Le jeu d'instructions Σ comprend :

- quatre instructions correspondant aux opérations mathématiques de base, notées `ADD`, `SUB`, `MUL` et `DIV`[6], d'arité 2 ;
- dix instructions représentant les variables d'entrées, notées `X1` à `X10`, d'arité 0 ;
- une instruction spécifique de manipulation de la pile d'opérande, notée `DUP`, d'arité 1 qui duplique le sommet de la pile.

Bien que le jeu d'instructions contienne les terminaux nécessaires, nous initialisons la pile d'opérandes avec les 10 variables d'entrée du problème. Nous avons en effet constaté que cette initialisation permet d'éviter un phénomène de convergence prématurée des expériences où certains programmes découverts dès les premières générations se répandent dans la population et où la fitness ne progresse que très peu.

Pour le problème Poly-10, on définit la fonction de fitness $f : p \to \mathbb{R}$ d'un programme $p \in P$, avec P l'ensemble des programmes définis sur Σ et $\mathcal{A}$ l'ensemble d'apprentissage contentant les 50 vecteurs d'entrées v_i, comme l'erreur RMS sur $\mathcal{A}$ telle que :

$$f(x) = \sqrt{\frac{1}{50} \sum_{i=1}^{50} \big(O_i(p) - \mathcal{P}(v_i)\big)^2}$$

avec $O_i(p)$ la sortie de p pour le $i^{\text{ième}}$ vecteur de $\mathcal{A}$. Il convient donc de minimiser cette fonction pour résoudre le problème Poly-10.

[6]DIV est une opération dite protégée qui n'autorise pas la division par 0.

Analyse du paysage

Nous avons réalisé 10^3 marches aléatoires de longueur 1000 et autant de marches adaptatives sur le paysage Poly-10 pour deux combinaisons de prédicteurs `Top` et `MoyA`. La longueur de corrélation τ moyenne des marches aléatoires est égale à 24 pour `Top` et 48 pour `MoyA`, ce qui montre que le paysage est plus facile à optimiser avec `MoyA`. Ce résultat est confirmé par l'analyse de la longueur des marches adaptatives l, qui est faible avec l=8.31 pour `Top` et beaucoup plus élevée avec l=397.05 pour `MoyA`. De même, on note que la fitness moyenne des optima locaux diffère largement entre les deux types d'évaluation avec 0.56 pour `Top` et 0.25 pour `MoyA`. Enfin, nous avons trouvé, en testant 10^7 programmes, une proportion de voisins neutres de près de 95% pour `Top`, ce qui indique un paysage massivement neutre alors qu'il n'est que de 31% pour `MoyA`. Cette analyse, avec des paysages moins neutres et moins rugueux pour `MoyA`, confirme nos attentes sur l'approche par équipes de prédicteurs.

5.3.1 Paramètres de l'algorithme

Nous utilisons une population de 500 individus créés aléatoirement. Dans la population initiale, la taille des individus est distribuée uniformément entre 1 et 50 gènes choisis aléatoirement dans le jeu d'instructions Σ. Durant les expériences, la taille des individus est limitée à λ_{max}=500 gènes. L'évolution, avec élitisme, sélection par tournoi de 16 individus et remplacement *steady-state* est limitée à 100 générations, c'est-à-dire 100×500 évaluations de fitness.

5.3.2 Résultats expérimentaux

Dans cette section, nous comparons les résultats obtenus avec les opérateurs CS et CMH sur le problème Poly-10. Nous voulons, entre autre, trouver les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$, c'est-à-dire les réglages qui donnent les meilleurs résultats en terme de fitness moyenne, c'est pourquoi nous avons fait varier $\mathcal{T}_m$ (de 0 à 2) et $\mathcal{T}_c$ (de 0 à 1). Cette recherche est effectuée pour toutes les combinaisons de prédicteurs décrites précédemment. Pour chaque réglage et pour chaque combinaison, 50 expériences indépendantes sont réa-

lisées. Un test de Student, avec 95% de confiance, est utilisé pour déterminer si les résultats obtenus sont statistiquement différents les uns des autres.

Réglages des paramètres

Les Figures 5.3 et 5.4 présentent la fitness moyenne du meilleur individu trouvé en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour les deux opérateurs de croisement sur le problème Poly-10 pour la combinaison de prédicteur `MoyA` ; l'influence de $\mathcal{T}_m$ et $\mathcal{T}_c$ étant comparable pour les autres types de combinaisons. Quand l'opérateur CS est utilisé conjointement avec l'opérateur de mutation, la résolution du problème Poly-10 nécessite un réglage précis de $\mathcal{T}_m$ et $\mathcal{T}_c$. Les meilleurs résultats du CS, avec une fitness moyenne de 0.13, ont été obtenus pour $\mathcal{T}_m$ =0.4 et $\mathcal{T}_c$ =0.2. Par contre, avec l'opérateur CMH, une recherche fine du réglage optimal ne semble pas nécessaire. La plus petite valeur de fitness trouvée est 0.25 et correspond à $\mathcal{T}_m$ =0.4 et $\mathcal{T}_c$ =0.8.

On peut largement expliquer ces faibles performances de l'opérateur CMH par son incapacité à explorer l'espace des tailles des programmes. Les Figures 5.5 et 5.6 présentent la taille moyenne du meilleur individu dans la population finale en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour les deux opérateurs de croisement sur le problème Poly-10 pour la combinaison de prédicteur `MoyA`. On constate une quasi indépendance de la taille au réglage de $\mathcal{T}_m$ et $\mathcal{T}_c$, avec des valeurs proches de λ_{max}=500 pour l'opérateur CS et, à l'opposé, des valeurs toujours inférieures à 100 gènes pour le CMH.

Équipes de prédicteurs

La Table 5.1 présente les résultats obtenus sur le problème Poly-10 avec les différentes équipes de prédicteurs, pour le réglage optimal de $\mathcal{T}_m$ et $\mathcal{T}_c$ et ce pour l'opérateur CS puisqu'il s'est avéré être le plus performant. Sur la première ligne et à titre de comparaison, nous avons également rapporté les performances obtenues par un système PGA (cf. Section 1.2.1), dont les paramètres ont été choisis pour correspondre au mieux à ceux du système PGP et dont le réglage de $\mathcal{T}_m$ et $\mathcal{T}_c$ a été également optimisé. Nous constatons que les combinaisons `Top` et `Best` ne donnent pas de bons résultats, ce qui

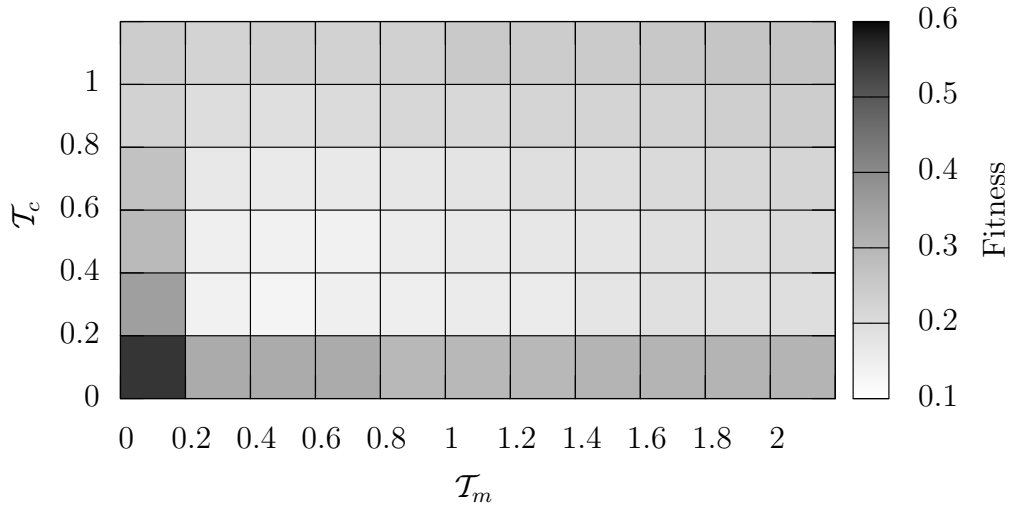

FIG. 5.3 – Problème Poly-10 et CS : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`

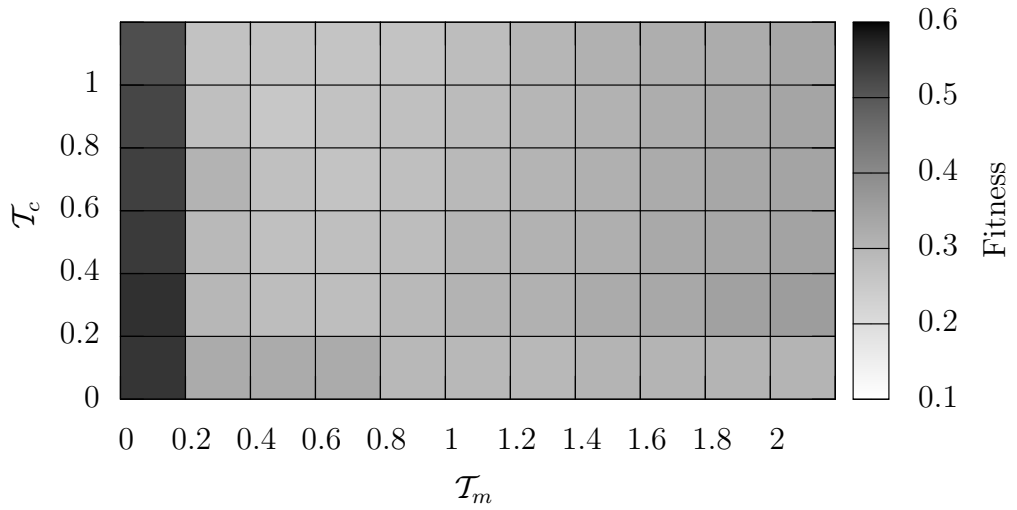

FIG. 5.4 – Problème Poly-10 et CMH : Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`

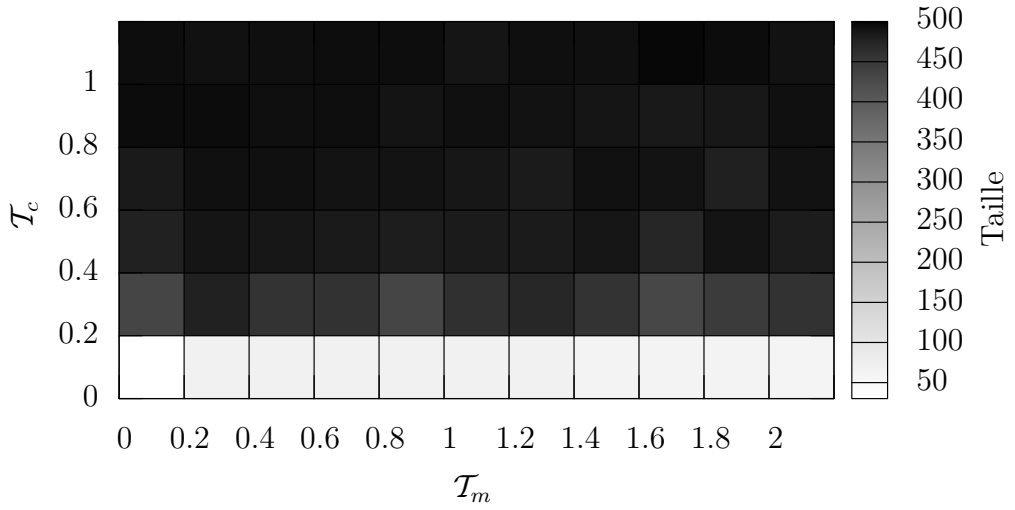

FIG. 5.5 – Problème Poly-10 et CS : Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`

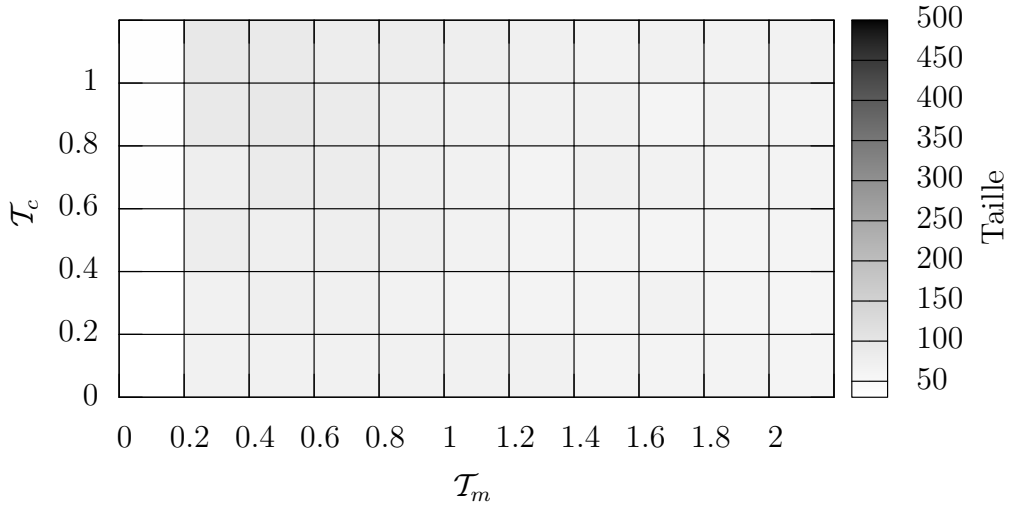

FIG. 5.6 – Problème Poly-10 et CMH : Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`

TAB. 5.1 – Meilleurs résultats trouvés sur le problème Poly-10 avec l'opérateur CS pour les différentes combinaisons de prédicteurs.

Combinaison	Fitness Moyenne	Écart Type	Meilleure	Pire
PGA	0.25	0.07	0.08	0.40
`Top`	0.35	0.10	0.16	0.54
`Best`	0.43	0.04	0.13	0.52
`Sm`	0.14	0.02	0.08	0.27
`MoyA`	0.13	0.02	0.09	0.26
`Prd`	0.21	0.05	0.15	0.41
`MoyG`	0.20	0.05	0.14	0.40
`MoyP`	**0.06**	**0.01**	0.02	0.12

était attendu puisque dans les deux cas, la prédiction réalisée par l'équipe se résume à celle d'un prédicteur unique et que seule une petite portion des programmes est utilisée (avec les conséquences sur l'espace de recherche que nous avons déjà décrites). Les combinaisons `Sm` et `MoyA` sont performantes comparées au système PGA[7]. Par contre, les combinaisons `Prd` et `MoyG` ne semblent pas adaptées à ce problème, peut être parce que par construction il est plus aisé de le décomposer comme une somme. Enfin, les équipes de prédicteurs `MoyP` ont largement surclassé les autres.

On note que les résultats présentés pour `MoyP` correspondent à un facteur β=10. Nous avons tracé, Figure 5.7, l'erreur d'apprentissage moyenne en fonction de β. On constate que l'erreur dépend de la valeur de β et qu'elle présente un minimum pour $\beta = 10$.

Nombres de prédicteurs

Dans cette étude, une équipe de prédicteurs correspond à la combinaison des sorties des sous-programmes d'un individu PGP. Une des originalités de cette approche est que le nombre de prédicteurs n'est pas fixé *a priori* mais peut varier durant le processus évolutionnaire. Nous espérions que la dynamique de l'algorithme permettrait d'optimiser le nombre de prédicteurs,

[7]Avec le mécanisme de Linear Scaling, décrit Section 5.1.3, les deux combinaisons `Sm` et `MoyA` sont en fait équivalentes.

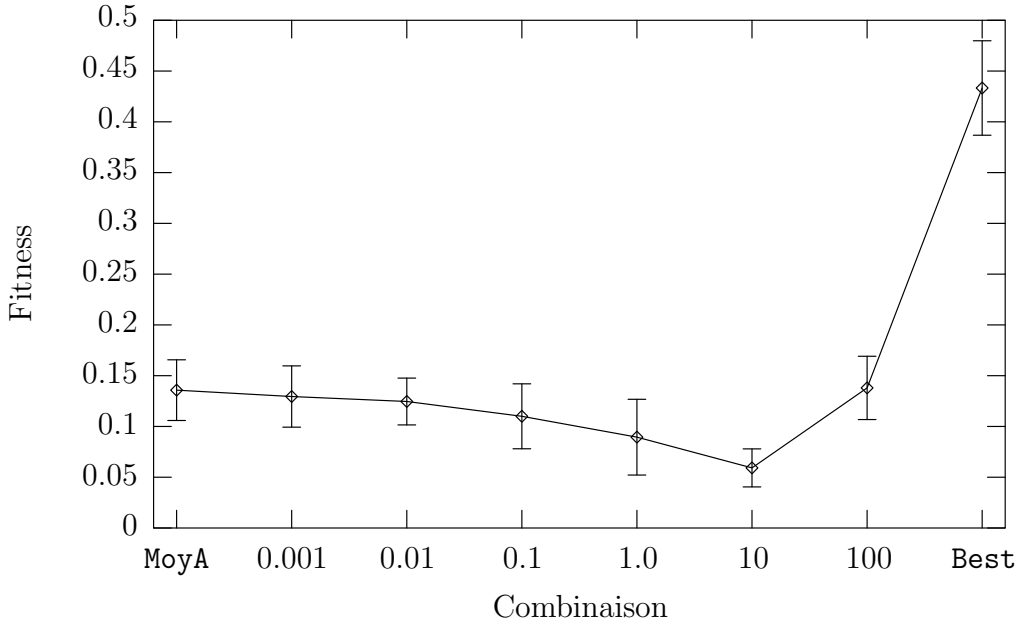

FIG. 5.7 – Problème Poly-10 et CS : Fitness moyenne pour les combinaisons `MoyA`, `Best` et `MoyP` avec différentes valeurs de β.

cependant on constate que la taille de l'équipe est fortement corrélée avec le nombre d'instructions des individus, un coefficient de corrélation 0.89 pour la combinaison `MoyA` par exemple. Ainsi, le nombre de prédicteurs n'est pas directement contrôlé par le système.

Nous pensons que la combinaison `MoyP` permet de résoudre ce problème. Nous avons tracé, Figure 5.8, l'évolution du nombre moyen de prédicteurs pour `MoyP` avec $\beta = 0.1$. La courbe montre une forte croissance dès les premières générations comme c'est le cas pour l'évolution du nombre d'instructions. Nous avons également tracé le nombre de prédicteurs ayant une contribution négligeable à la fitness de l'équipe avec un poids w_i proche de 0 ainsi que le nombre de prédicteurs ayant une contribution significative. On constate qu'à la fin des expériences environ un tiers des prédicteurs sont "éliminés" des équipes et que, malgré une croissance continue du nombre de prédicteurs total, le nombre de prédicteurs réellement pris en compte dans l'évaluation reste stable, autour de 60, à partir de la dixième génération. Ainsi, la combinaison `MoyP` permet, indirectement, d'effectuer un contrôle de la taille des équipes.

Cependant à ce stade, on peut raisonnablement supposer que le réglage optimal de β dépende du problème à résoudre[8], et plus précisément de l'erreur d'apprentissage moyenne des prédicteurs. La Figure 5.9 présente la valeur des poids w_i en fonction de l'erreur d'apprentissage pour différentes valeurs de β. Pour cet exemple, nous avons fait l'hypothèse d'une erreur comprise entre 0 et 1. Nous voyons que pour des faibles valeurs de β, les poids sont, indépendamment de l'erreur, très proche de 1, ce qui est comparable à la pondération uniforme de la combinaison `MoyA`. A l'opposé, pour β=100, seuls les meilleurs prédicteurs ont un poids non nul, ce qui peut être comparé à la combinaison `Best`. En revenant à la Figure 5.7, on peut vérifier que la combinaison `MoyP` offre, en faisant varier β, une solution de continuité entre les équipes `MoyA` et `Best`.

[8]En fait, cette hypothèse sera confirmée plus loin dans ce chapitre.

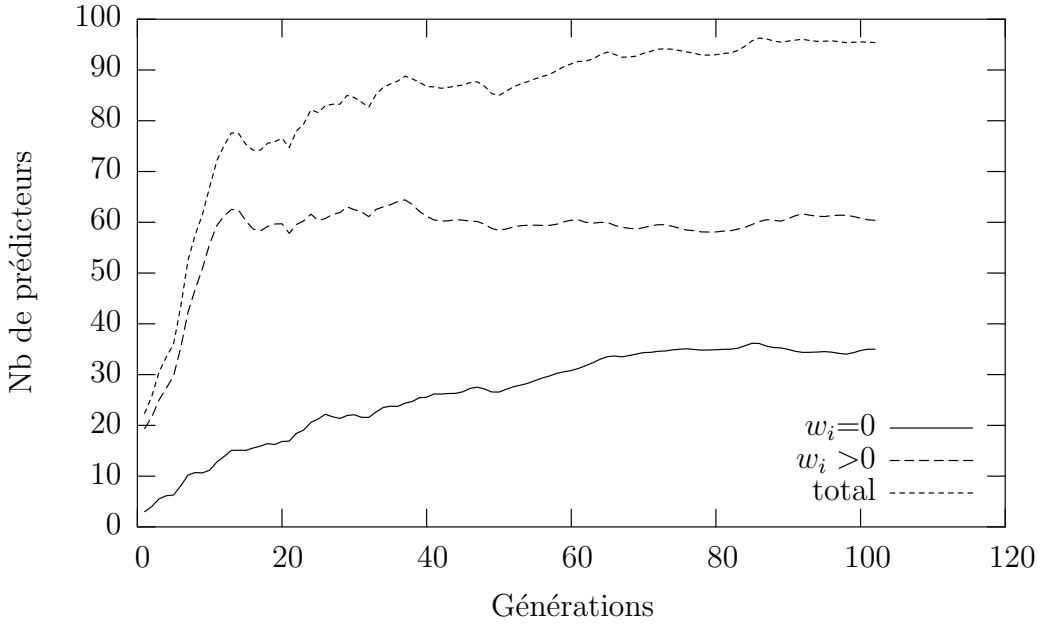

FIG. 5.8 – Problème Poly-10 et CS : Évolution du nombre de prédicteurs moyen avec $\beta = 0.1$ pour l'équipe de prédicteur `MoyP`.

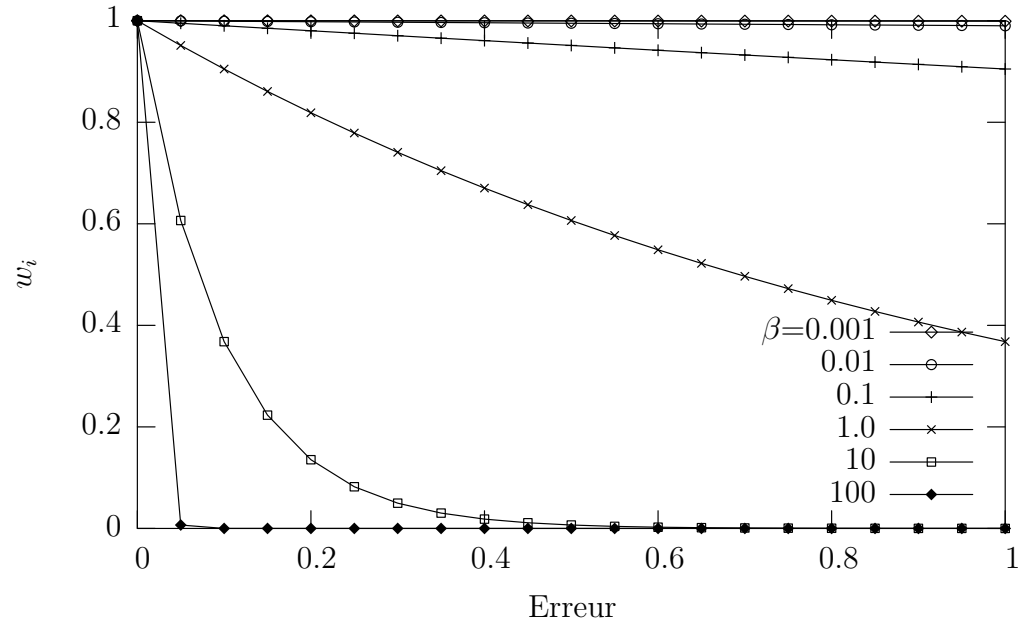

FIG. 5.9 – Poids w_i en fonction de l'erreur pour différentes valeurs de β.

5.3.3 Contrôle de la taille

Nous voulons comparer les résultats obtenus par la combinaison de prédicteurs `MoyA` avec les opérateurs CMH+$INS_{2.0}$ et CMH+$CS_{0.1}$ sur le problème Poly-10. Pour cela, nous recherchons les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$, c'est-à-dire les réglages qui donnent les meilleurs résultats en terme de fitness moyenne, c'est pourquoi nous avons fait varier $\mathcal{T}_m$ (de 0 à 2) et $\mathcal{T}_c$ (de 0 à 1). Pour chaque réglage, 50 expériences indépendantes sont réalisées. Un test de Student, avec 95% de confiance, est utilisé pour déterminer si les résultats obtenus sont statistiquement différents les uns des autres.

TAB. 5.2 – Meilleurs résultats trouvés sur le problème Poly-10 avec la combinaison `MoyA` pour différents opérateurs de recombinaison.

Opérateur	Fitness Moyenne	Taille Moyenne	Taille Effective Moyenne
CS	$\mathbf{0.13}_{(\sigma=0.03)}$	$457.42_{(\sigma=79.07)}$	$457.14_{(\sigma=79.25)}$
CMH	$0.25_{(\sigma=0.05)}$	$\mathbf{92.28}_{(\sigma=31.39)}$	$\mathbf{91.74}_{(\sigma=31.45)}$
CMH+$INS_{2.0}$	$\mathbf{0.14}_{(\sigma=0.03)}$	$247.18_{(\sigma=90.36)}$	$245.00_{(\sigma=90.75)}$
CMH+$CS_{0.1}$	$\mathbf{0.11}_{(\sigma=0.02)}$	$419.12_{(\sigma=96.90)}$	$418.80_{(\sigma=96.82)}$

La Table 5.2 rapporte les résultats trouvés avec les différents opérateurs pour les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$. Comme nous l'avons déjà mentionné, avec l'opérateur CMH, le système a trouvé des programmes moins performants mais plus petits qu'avec l'opérateur CS. Statistiquement, les meilleures fitness obtenues avec les opérateurs CS, CMH+$INS_{2.0}$ et CMH+$CS_{0.1}$ sont équivalentes. Par contre, la taille des programmes diffère largement. On note, pour l'opérateur CMH+$INS_{2.0}$, qu'un bon compromis taille *vs* fitness est obtenu, avec une taille moyenne deux fois moindre que dans le cas du CS. Nous remarquons également que pour tous les opérateurs testés, la taille moyenne est très proche de la taille effective moyenne. Ce taux d'instructions "utiles" d'environ 99%, en moyenne, est une des conséquences de l'approche par équipe de prédicteurs et on peut le comparer aux taux obtenus précédemment, avec des prédicteurs uniques (cf. Table 4.1), où il était compris entre 70% et 80%.

Les Figures 5.10 et 5.11 présentent la fitness moyenne du meilleur individu trouvé en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour les opérateurs CMH+INS$_{2.0}$ et CMH+CS$_{0.1}$ sur le problème Poly-10. Les meilleurs résultats du CMH+INS$_{2.0}$, ont été obtenus pour $\mathcal{T}_m$ =0 et $\mathcal{T}_c$ =1.0. Il convient de rappeler que, par construction, cet opérateur réalise, en moyenne, 2 insertions par individu. Avec l'opérateur CMH+CS$_{0.1}$, l'utilisation de la mutation est nécessaire mais doit rester faible. En effet, la plus petite valeur de fitness trouvée correspond à $\mathcal{T}_m$ =0.2 et $\mathcal{T}_c$ =0.8. Plus généralement, pour les deux opérateurs, des valeurs de $\mathcal{T}_c$ élevées et des valeurs de $\mathcal{T}_m$ faibles sont préférables.

Nous avons rapporté (Figures 5.12 et 5.13) la taille moyenne du meilleur individu dans la population finale en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$. Dans le cas du CMH+INS$_{2.0}$, la translation de la distribution des tailles, induite par l'insertion de deux instructions en moyenne par individu, détermine largement la taille des programmes trouvés qui ne semble que peu influencée par $\mathcal{T}_m$ et $\mathcal{T}_c$. De plus, il est évident que si aucune recombinaison n'est réalisée, le contrôle de la taille n'est pas effectif (environ 400 instructions) puisque aucun mécanisme ne compense le réglage déséquilibré de la mutation. A l'opposé, avec l'opérateur CMH+CS$_{0.1}$, la taille varie en fonction du $\mathcal{T}_c$. On comprend donc que la taille de programmes ne dépend pas uniquement de la proportion de CMH et de CS, c'est-à-dire du paramètre p, mais aussi du nombre de recombinaisons CS effectuées par génération, qui augmente avec $\mathcal{T}_c$.

Compromis fitness *vs* taille

Pour visualiser le compromis fitness *vs* taille, nous avons tracé Figure 5.14, pour les quatre opérateurs de recombinaison, un nuage de points avec la fitness moyenne en abscisse et la taille moyenne en ordonnée. Chaque point correspond à un des réglages de $\mathcal{T}_m$ et $\mathcal{T}_c$ testés et les lignes reliant les points représentent les frontières de Pareto, l'objectif étant de minimiser les deux critères. On constate que le compromis diffère largement pour les quatre opérateurs. Quand le CS ou le CMH sont utilisés, les variations de la taille sont relativement faibles alors que pour les deux opérateurs de contrôle de la taille, les frontières de Pareto couvrent une zone plus importante.

On s'intéresse maintenant à l'influence sur le compromis taille *vs* . fitness

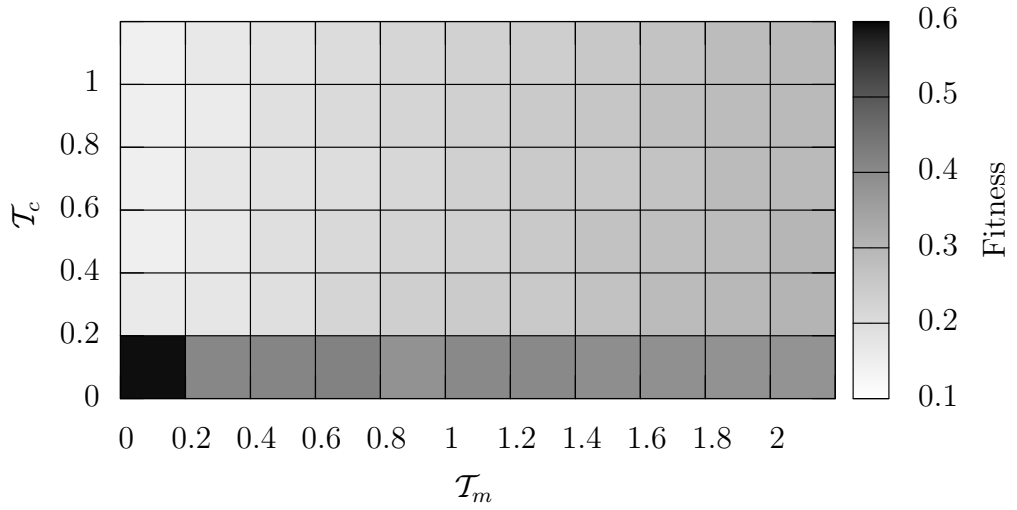

FIG. 5.10 – Problème Poly-10 et CMH+$\text{INS}_{2.0}$: Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`.

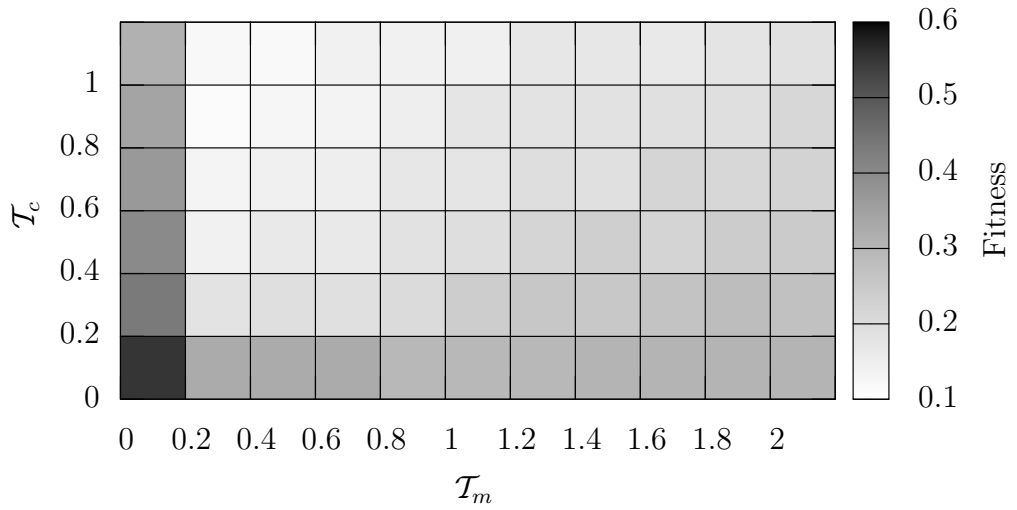

FIG. 5.11 – Problème Poly-10 et CMH+$\text{CS}_{0.1}$: Fitness moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`.

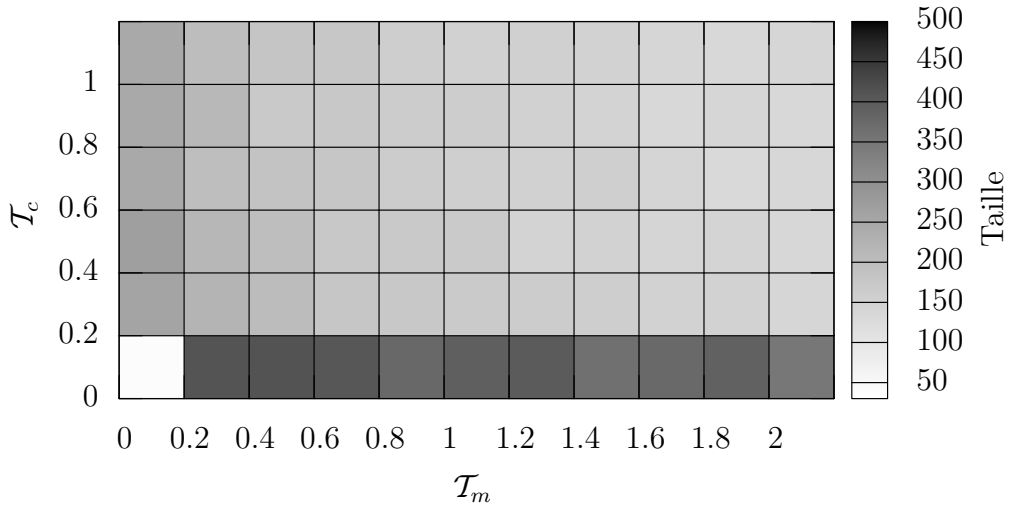

FIG. 5.12 – Problème Poly-10 et CMH+$\text{INS}_{2.0}$: Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`.

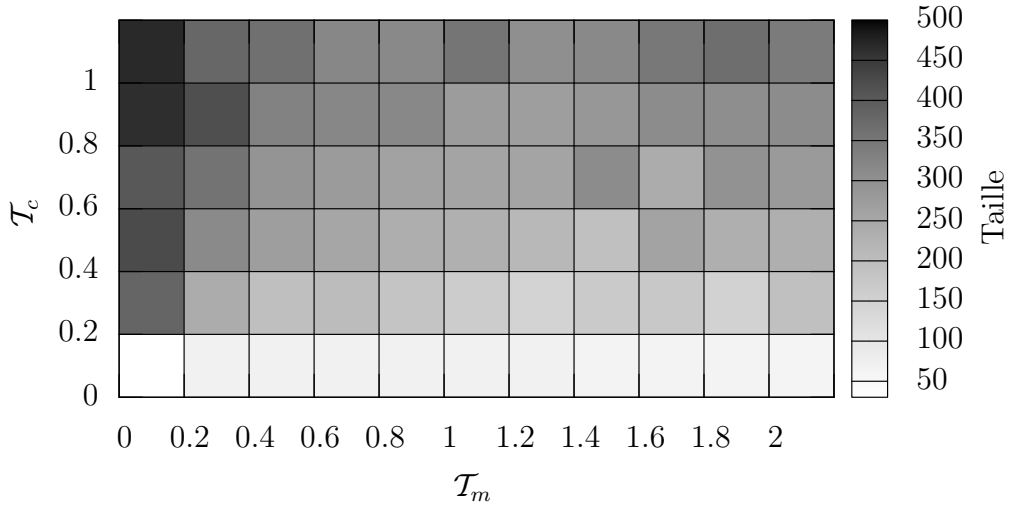

FIG. 5.13 – Poly-10 et CMH+$\text{CS}_{0.1}$: Taille moyenne en fonction de $\mathcal{T}_m$ et $\mathcal{T}_c$ pour l'équipe de prédicteur `MoyA`.

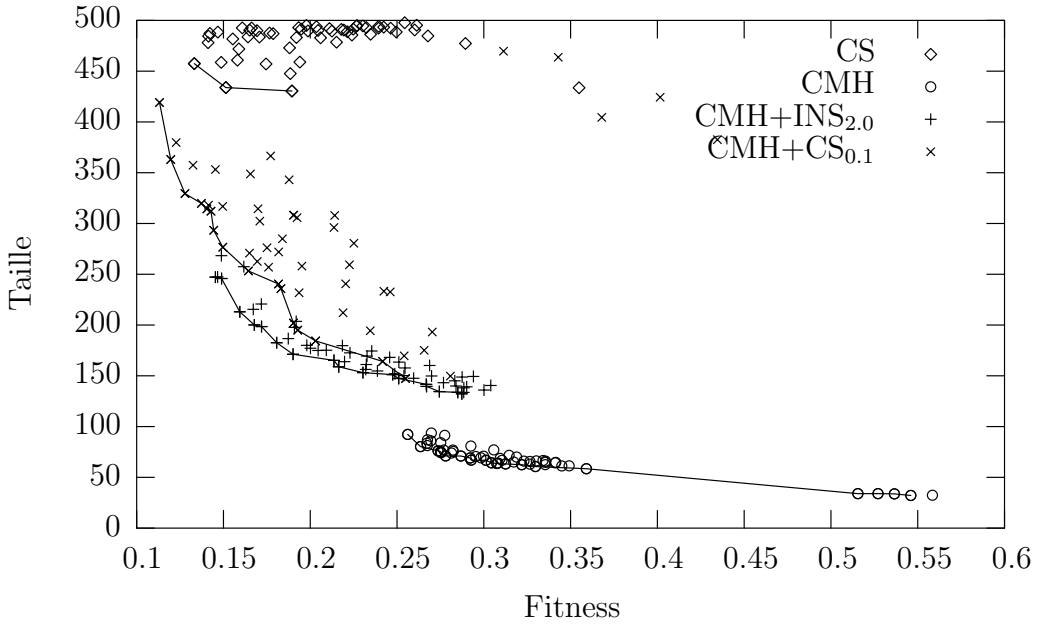

FIG. 5.14 – Problème Poly-10 et différents opérateurs : Nuage de points Fitness *vs* Taille pour l'équipe de prédicteur `MoyA`. Les lignes reliant les points correspondent aux frontières de Pareto.

des deux paramètres r et p des opérateurs CMH+INS$_r$ et CMH+CS$_p$. Nous avons donc réalisé une série d'expériences spécifiques avec $\mathcal{T}_m$ =0.2 et $\mathcal{T}_c$ =0.8.

Les Figures 5.15 et 5.16 montrent les variations respectives de la taille et la fitness moyenne du meilleur programme trouvé en fonction du paramètre r pour l'opérateur CMH+INS$_r$. On note que l'insertion d'instructions, contrôlée par r, entraîne une diminution de la valeur de la fitness et que ce phénomène disparaît quand r est supérieur à 2.0. La taille moyenne est fortement corrélée au paramètre r et toutes les tailles possibles de l'espace de recherche peuvent être atteintes. En comparant les résultats obtenus avec le CMH à ceux de l'opérateur CMH+INS$_{2.0}$, on note que des programmes deux fois plus performants mais également deux fois plus longs ont été découverts.

Les Figures 5.17 et 5.18 montrent les variations respectives de la taille moyenne et la fitness moyenne du meilleur programme trouvé en fonction du paramètre p pour l'opérateur CMH+CS$_p$. Nous voyons que, pour toutes les valeurs de p testées, la fitness obtenue est meilleure que celles obtenues avec les opérateurs CMH et CS. Cela signifie qu'une utilisation conjointe de ces deux opérateurs est toujours préférable. Cependant, on remarque que la taille moyenne des programmes augmente rapidement avec le paramètre p et qu'elle semble atteindre la valeur λ_{max} quand p est supérieur à 0.5. De plus, la courbe de fitness présente un minimum pour p=0.2. C'est pourquoi, le réglage de p doit être très précis pour espérer obtenir le meilleur compromis taille *vs* fitness possible. Ainsi, dans le contexte de la RSR, on peut définir une "région d'intérêt" de l'opérateur CMH+CS$_p$ pour p dans l'intervalle [0, 0.2].

5.4 Problème inverse

Les tests préliminaires réalisés sur le problème Poly-10 ont montré expérimentalement l'intérêt de l'approche par équipe de prédicteurs ainsi que des méthodes de contrôle de la taille. Dans cette section, notre objectif est de résoudre le problème de l'inversion des composantes atmosphériques. Nous vérifions dans un premier temps si les résultats précédents peuvent être confirmés sur ce problème, puis nous réalisons une recherche plus intensive avec

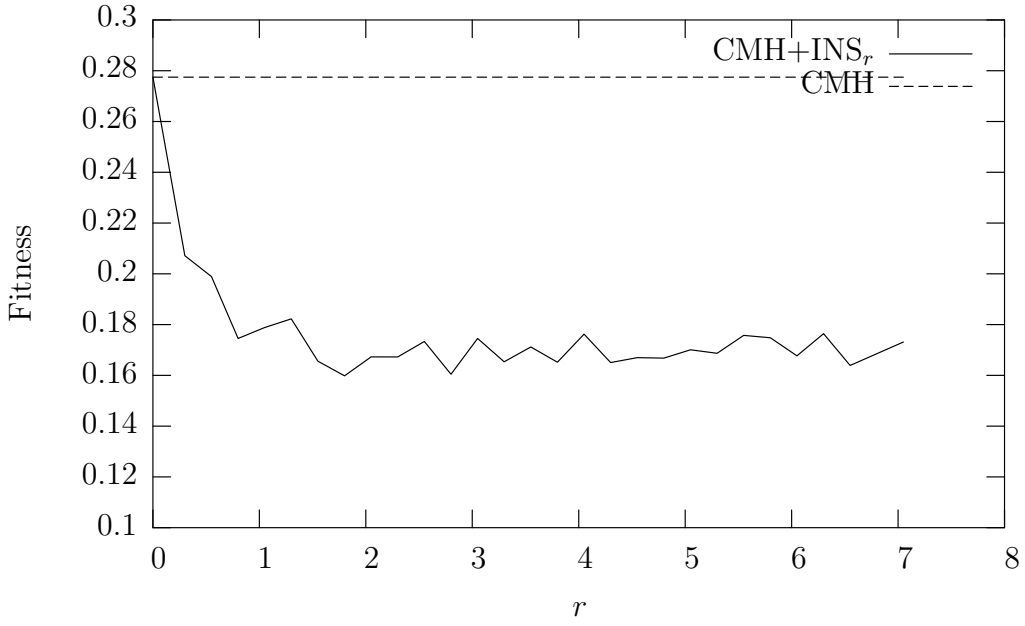

FIG. 5.15 – Problème Poly-10 et CMH+INS$_r$: Fitness moyenne en fonction de r pour l'équipe de prédicteur `MoyA`.

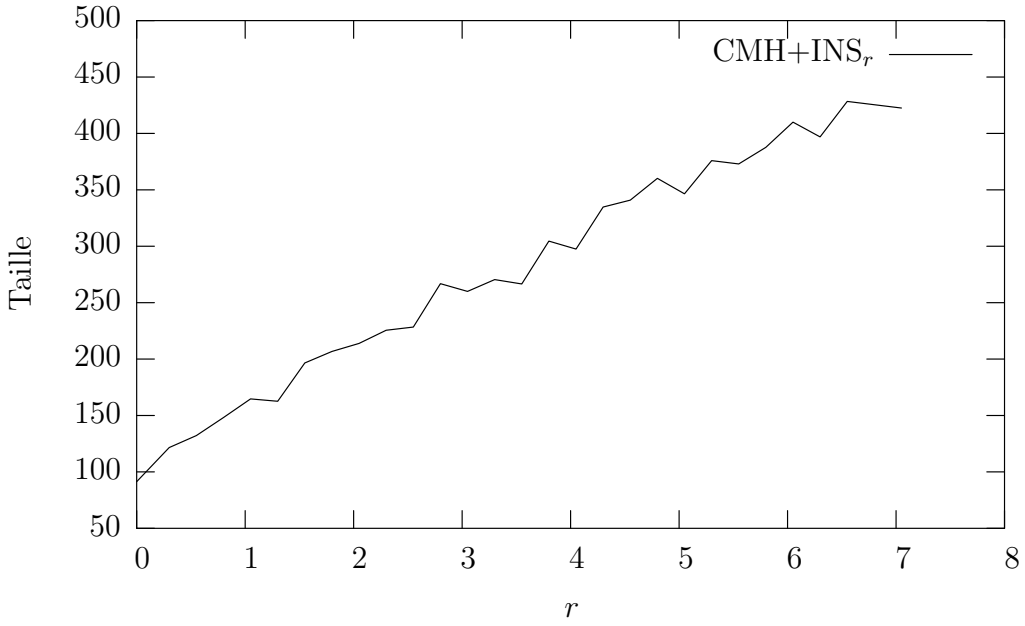

FIG. 5.16 – Problème Poly-10 et CMH+INS$_r$: Taille moyenne en fonction de r pour l'équipe de prédicteur `MoyA`.

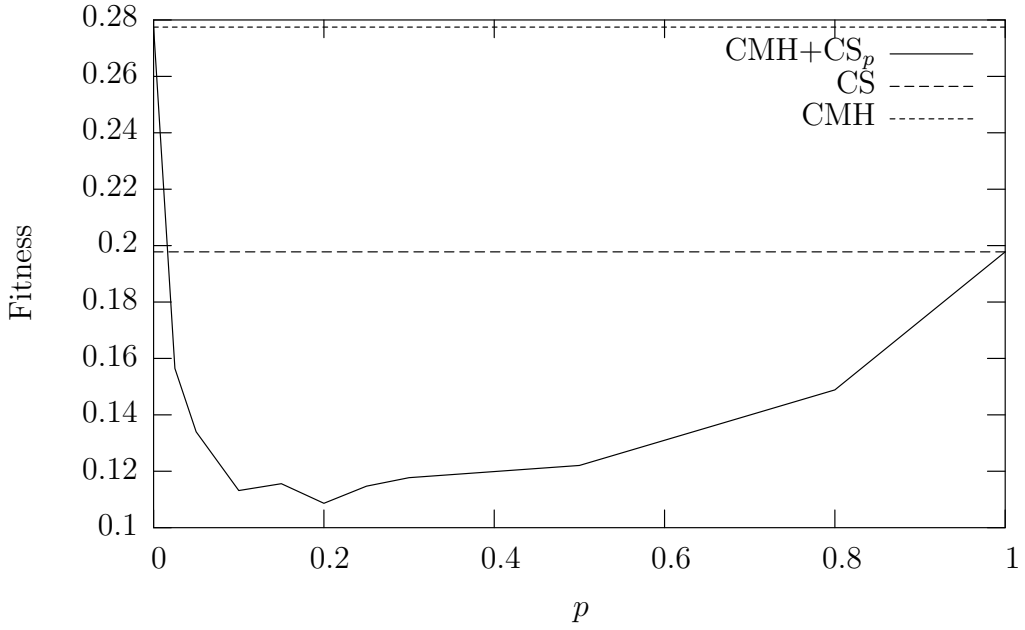

FIG. 5.17 – Problème Poly-10 et CMH+CS$_p$: Fitness moyenne en fonction de p pour l'équipe de prédicteur `MoyA`.

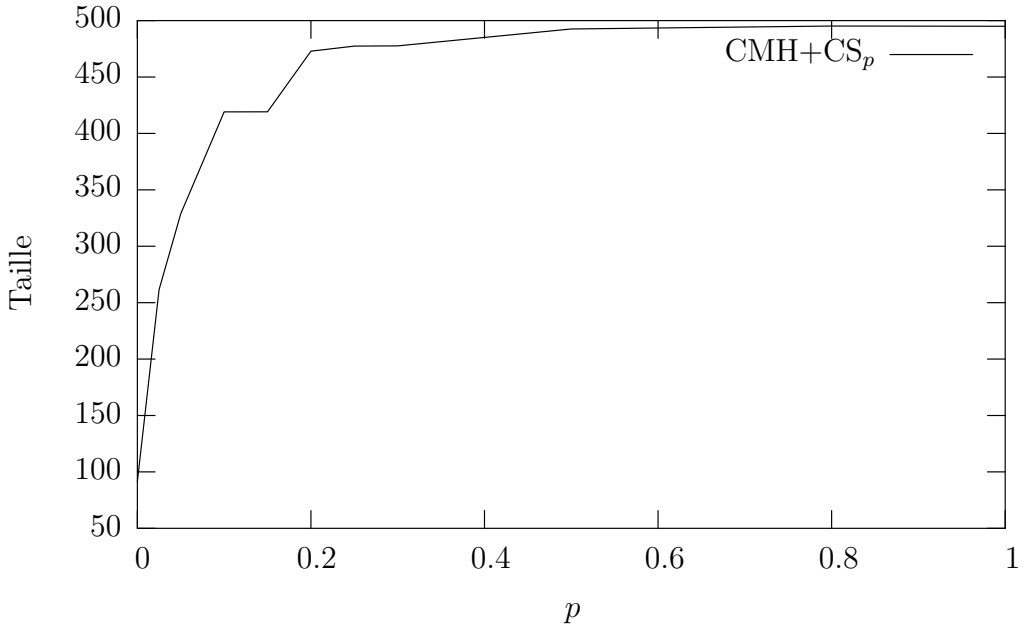

FIG. 5.18 – Problème Poly-10 et CMH+CS$_p$: Taille moyenne en fonction de p pour l'équipe de prédicteur `MoyA`.

entre autre plus de générations et plus d'individus dans la population.

Définition du problème

Pour éviter le phénomène de sur-apprentissage, nous utilisons une méthode d'arrêt de l'apprentissage, nommée Backwarding [87], basée sur l'utilisation d'un jeu d'entrées indépendant appelé ensemble de validation. Nous avons donc réparti les données d'entrées du problème inverse en trois groupes, l'ensemble d'apprentissage $\mathcal{A}$ qui comprend 500 vecteurs et à partir duquel est calculée la fitness durant l'évolution, l'ensemble de validation qui comprend 250 vecteurs et qui permet de détecter l'apparition du sur-apprentissage et enfin l'ensemble de test qui comprend également 250 vecteurs et qui permet de calculer l'erreur de généralisation du modèle. Un vecteur est constitué des 30 variables d'entrées qui correspondent pour trois longueurs d'ondes différentes (440, 675 et 870 nm) aux valeurs de la luminance du ciel obtenues pour dix angles de diffusion compris entre 3 et 150 degrés. Un exemple de 10 vecteurs d'entrées extraits du jeu d'apprentissage $\mathcal{A}$ est présenté Figure 5.19. A partir de ces 30 variables d'entrées, le modèle inverse doit prédire la valeur de l'épaisseur optique τ_a. Pour les trois jeux d'entrées, τ_a est distribué uniformément dans l'intervalle [0.1,0.6].

Le système PGP décrit Section 1.3 est utilisé et la sortie $O(p)$ d'un programme p sera évaluée comme une combinaison des différents sous-programmes de p. Le jeu d'instructions Σ comprend :

- une instruction correspondant au logarithme népérien, notée `LOG`[9], d'arité 1 ;
- une instruction correspondant à la fonction sigmoïde, souvent utilisée en RNA, notée `SIGM`[10], d'arité 1 ;
- quatre instructions correspondant aux opérations mathématiques de base, notées `ADD`, `SUB`, `MUL` et `DIV`[11], d'arité 2 ;
- trente instructions représentant les variables d'entrées, notées `X1` à `X30` ;
- une instruction représentant une constante aléatoire dans l'intervalle

[9] LOG est une opération dite protégée qui permet de gérer les valeurs négatives et nulles.
[10] SIGM$(x) = 1/(1 + e^{-x})$
[11] DIV est une opération dite protégée qui n'autorise pas la division par 0.

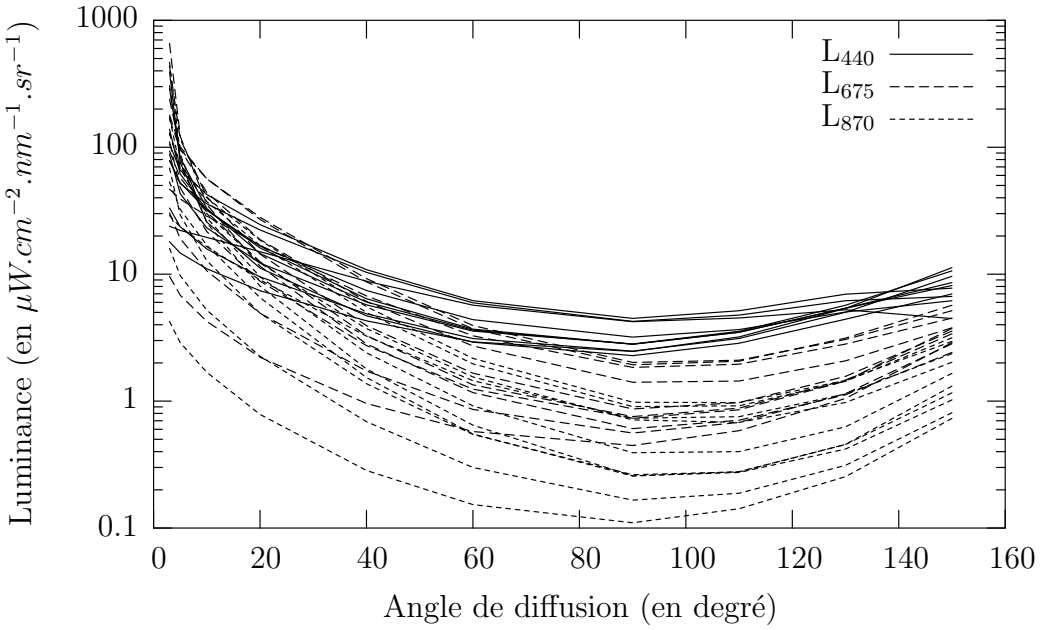

FIG. 5.19 – Exemples de données d'entrées du problème inverse.

[-10, 10], inspirée des Ephemeral Random Constant [54], notée `ERC` ;

– une instruction spécifique de manipulation de la pile d'opérande, notée `DUP`, d'arité 1 qui duplique le sommet de la pile.

Bien que le jeu d'instructions contienne les terminaux nécessaires et suite aux constations faites en RSR, nous initialisons la pile d'opérandes avec les 30 variables d'entrée du problème.

Pour le problème inverse, on définit la fonction de fitness $f : p \rightarrow \mathbb{R}$ d'un programme $p \in P$, avec P l'ensemble des programmes définis sur Σ et $\mathcal{A}$ l'ensemble d'apprentissage contentant les 500 vecteurs d'entrées v_i, comme l'erreur RMS sur $\mathcal{A}$ telle que :

$$f(x) = \sqrt{\frac{1}{500}\sum_{i=1}^{500}\left(O_i(p) - \mathcal{P}(v_i)\right)^2}$$

avec $O_i(p)$ la sortie de p pour le $i^{\text{ième}}$ vecteur de $\mathcal{A}$. Il convient donc de minimiser cette fonction pour résoudre le problème inverse.

Analyse du paysage

Nous avons réalisé 10^3 marches aléatoires de longueur 100 et autant de marches adaptatives sur le paysage du problème inverse pour deux combinaisons de prédicteurs `Top` et `MoyA`. La longueur de corrélation τ moyenne des marches aléatoires est égale à 44 pour `Top` et 67 pour `MoyA`, ce qui montre que le paysage est plus facile à optimiser avec `MoyA`. Ce résultat est confirmé par l'analyse de la longueur des marches adaptatives l, qui est faible avec l=8.12 pour `Top` et extrêmement élevée avec l=975.05[12] pour `MoyA`. De même, on note que la fitness moyenne des optima locaux diffère largement entre les deux types d'évaluation avec 0.065 pour `Top` et 0.026 pour `MoyA`. Enfin, nous avons trouvé, en testant 10^7 programmes, une proportion de voisins neutres de plus de 96% pour `Top`, ce qui indique un paysage massivement neutre alors qu'il n'est que de 29% pour `MoyA`.

Par rapport aux résultats obtenus sur le problème Poly-10, avec la combinaison `MoyA` le problème inverse semble plus facile à optimiser par une méthode de recherche locale.

5.4.1 Paramètres de l'algorithme

Nous utilisons une population de 500 individus créés aléatoirement. Dans la population initiale, la taille des individus est distribuée uniformément entre 1 et 50 gènes choisis aléatoirement dans le jeu d'instructions Σ. Durant les expériences, la taille des individus est limitée à λ_{max}=500 gènes. L'évolution, avec élitisme, sélection par tournoi de 16 individus et remplacement *steady-state* est limitée à 100 générations.

5.4.2 Résultats expérimentaux

Dans cette section, nous comparons les résultats obtenus, en terme de fitness moyenne, avec différentes combinaisons de prédicteurs sur le problème inverse. Pour chaque combinaison, 50 expériences indépendantes sont réalisées avec l'opérateur CS. Le réglage des opérateurs génétiques est choisi

[12]En fait, il s'agit ici d'une borne inférieure car la recherche est arrêtée systématiquement après 1000 déplacements.

à partir des conclusions tirées des expériences précédentes sur le problème Poly-10, c'est-à-dire $\mathcal{T}_m$ =0.4 et $\mathcal{T}_c$ =0.2. Un test de Student, avec 95% de confiance, est utilisé pour déterminer si les résultats obtenus sont statistiquement différents les uns des autres.

La Table 5.3 présente les résultats (apprentissage, validation et test) obtenus sur le problème inverse pour différentes combinaisons de prédicteurs. Pour des raisons équivalentes à celles évoquées pour le problème Poly-10, les combinaisons `Top` et `Best` ne permettent pas de découvrir des programmes performants, ils sont même moins bons que ceux obtenus avec une heuristique de recherche locale (voir l'analyse du paysage précédente). Les équipes `MoyP` sont les seules à s'être significativement mieux comportées que le système PGA, en particulier en ce qui concerne l'erreur de test, qui est la plus importante puisqu'elle reflète la précision du modèle d'inversion dans des conditions opérationnelles, c'est-à-dire avec des données d'entrées non apprises. De façon générale, on ne note pas de forte tendance au sur-apprentissage, sans doute parce que les expériences ne comportaient pas suffisamment de générations pour que les prédicteurs se spécialisent trop.

On notera que les résultats présentés pour `MoyP` correspondent à un facteur β=1. La Figure 5.20 présente les variations de l'erreur de test moyenne en fonction de β. On peut voir que l'erreur est dépendante de β et que la courbe admet un minimum pour β=1.0[13].

Nous avons également réalisé l'apprentissage de deux RNA sur les mêmes jeux d'entrées :

- un perceptron sans couche cachée permettant de modéliser uniquement les relations linéaires entre les entrées et les sorties du problème ;
- un perceptron avec une couche cachée (une recherche préalable ayant permis de déterminer son nombre de noeuds optimal) permettant de modéliser les relations linéaires et non-linéaires entre les entrées et les sorties du problème.

Les erreurs de test correspondantes ont également été tracées Figure 5.20. On constate que l'approche par équipe de prédicteur permet au système PGP d'être plus performant qu'un RNA linéaire et que l'optimisation du

[13]les résultats obtenus pour β=0.1 et β=10 sont équivalents.

TAB. 5.3 – Meilleurs résultats trouvés sur le problème inverse avec l'opérateur CS pour les différentes combinaisons de prédicteurs.

Combinaison	RMS Moyenne	Écart Type	Meilleure	Pire
Apprentissage				
PGA	0.019	0.004	0.012	0.034
`Top`	0.033	0.006	0.015	0.050
`Best`	0.039	0.013	0.022	0.078
`Sm`	0.019	0.004	0.011	0.037
`MoyA`	0.020	0.006	0.009	0.035
`Prd`	0.024	0.006	0.013	0.043
`MoyG`	0.017	0.003	0.012	0.026
`MoyP`	**0.015**	**0.002**	0.009	0.022
Validation				
PGA	0.019	0.004	0.011	0.042
`Top`	0.031	0.005	0.015	0.051
`Best`	0.037	0.012	0.020	0.072
`Sm`	0.018	0.003	0.010	0.033
`MoyA`	0.019	0.005	0.009	0.032
`Prd`	0.023	0.006	0.012	0.047
`MoyG`	0.017	0.003	0.013	0.025
`MoyP`	**0.013**	**0.002**	0.008	0.020
Test				
PGA	0.021	0.006	0.014	0.044
`Top`	0.034	0.006	0.015	0.052
`Best`	0.040	0.013	0.022	0.080
`Sm`	0.021	0.004	0.011	0.040
`MoyA`	0.021	0.006	0.012	0.038
`Prd`	0.026	0.007	0.015	0.048
`MoyG`	0.020	0.005	0.013	0.038
`MoyP`	**0.015**	**0.002**	0.010	0.021

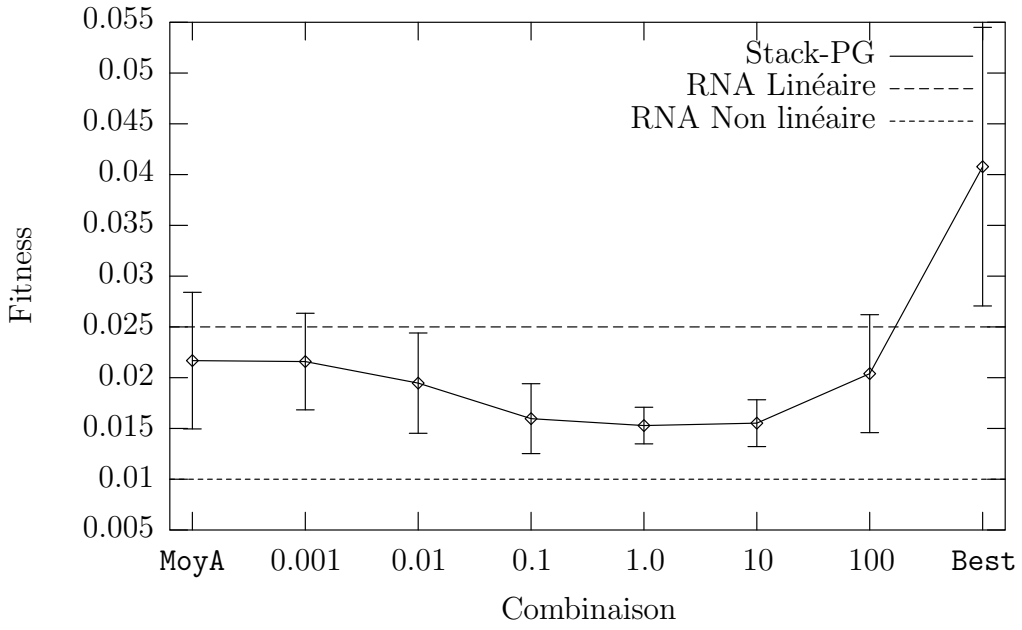

FIG. 5.20 – Problème inverse et CS : Fitness moyenne pour les combinaisons `MoyA`, `Best` et `MoyP` avec différentes valeurs de β.

TAB. 5.4 – Meilleurs résultats trouvés sur le problème inverse avec la combinaison `MoyA` et β=1.0 pour différents opérateurs de recombinaison.

Opérateur	Fitness Moyenne	Taille Moyenne	Taille Effective Moyenne
CS	$\mathbf{0.015}_{(\sigma=0.002)}$	$454.86_{(\sigma=128.85)}$	$444.61_{(\sigma=128.87)}$
CMH	$0.021_{(\sigma=0.004)}$	$\mathbf{90.18}_{(\sigma=19.73)}$	$\mathbf{90.00}_{(\sigma=19.70)}$
CMH+$INS_{3.0}$	$\mathbf{0.016}_{(\sigma=0.002)}$	$154.56_{(\sigma=32.32)}$	$154.16_{(\sigma=32.41)}$
CMH+$CS_{0.15}$	$\mathbf{0.013}_{(\sigma=0.002)}$	$360.36_{(\sigma=117.33)}$	$357.61_{(\sigma=117.07)}$

paramètre β permet de se rapprocher significativement de la précision d'un RNA non-linéaire.

5.4.3 Contrôle de la taille

Nous voulons comparer les résultats obtenus par la combinaison de prédicteurs `MoyP` avec β=1.0 pour les opérateurs CS, CMH , CMH+$INS_{3.0}$ et CMH+$CS_{0.15}$ sur le problème inverse. Les réglages optimaux de $\mathcal{T}_m$ et $\mathcal{T}_c$ sont repris des expériences menées sur le problème Poly-10, c'est-à-dire $\mathcal{T}_m$ =0.4 et $\mathcal{T}_c$ =0.2 pour le CS, $\mathcal{T}_m$ =0.4 et $\mathcal{T}_c$ =0.8 pour le CMH, $\mathcal{T}_m$ =0 et $\mathcal{T}_c$ =1.0 pour le CMH+$INS_{3.0}$ et enfin $\mathcal{T}_m$ =0.2 et $\mathcal{T}_c$ =0.8 pour le CMH+$CS_{0.15}$. Pour chaque opérateur, 50 expériences indépendantes sont réalisées. Un test de Student, avec 95% de confiance, est utilisé pour déterminer si les résultats obtenus sont statistiquement différents les uns des autres.

La Table 5.4 rapporte les résultats trouvés avec les différents opérateurs de recombinaison. L'opérateur CMH a permis de découvrir des programmes plus petits mais moins performants que les autres opérateurs. Les erreurs de test moyennes obtenues avec les trois opérateurs CS, CMH+$INS_{3.0}$ et CMH+$CS_{0.15}$ sont statistiquement équivalentes, mais les tailles moyennes des meilleurs individus trouvés différent largement. L'opérateur CMH+$INS_{3.0}$ réalise un bon compromis fitness *vs* taille.

Compromis fitness *vs* taille

On s'intéresse maintenant à l'influence sur le compromis taille *vs* fitness des deux paramètres r et p des opérateurs CMH+INS$_r$ et CMH+CS$_p$. Nous avons donc réalisé une série d'expériences spécifiques avec $\mathcal{T}_m$ =0.2 et $\mathcal{T}_c$ =0.8.

Les Figures 5.21 et 5.22 montrent les variations respectives de la taille moyenne et la fitness moyenne du meilleur programme trouvé en fonction du paramètre r pour l'opérateur CMH+INS$_r$. Comme pour le problème Poly-10, l'insertion d'instructions, contrôlée par r, entraîne une diminution de la valeur de la fitness quand r est inférieur à 3.0. Par contre, pour des valeurs de r supérieures, celle-ci se dégrade régulièrement. La taille moyenne est fortement corrélée au paramètre r et toutes les tailles possibles de l'espace de recherche peuvent être atteintes.

Les Figures 5.23 et 5.24 montrent les variations respectives de la taille et la fitness moyenne du meilleur programme trouvé en fonction du paramètre p pour l'opérateur CMH+CS$_p$. Nous voyons que, pour toutes les valeurs de p testées, la fitness obtenue est meilleure que celles obtenues avec les opérateurs CMH et CS. Cependant, on remarque que la taille moyenne des programmes augmente rapidement avec le paramètre p et qu'elle semble atteindre la valeur λ_{max} quand p est supérieur à 0.5. De plus, la courbe de fitness présente un minimum pour p=0.15.

5.4.4 Expérience finale

A partir des résultats précédents, nous voulons effectuer une recherche plus poussée sur le problème inverse. Nous effectuons une série de 50 expériences avec l'opérateur CMH+CS$_{0.15}$. Le réglage de $\mathcal{T}_m$ et $\mathcal{T}_c$ est fixé à, respectivement, 0.2 et 0.8. Nous utilisons l'approche par équipe de prédicteur avec la combinaison de sous-programmes `MoyP` et β=1.0. La population est composée de 1000 individus créés aléatoirement. Dans la population initiale, la taille des individus est distribuée uniformément entre 1 et 50 gènes choisis aléatoirement dans le jeu d'instructions Σ décrit précédemment. Durant les expériences, la taille des individus est limitée à λ_{max}=1000 gènes. L'évolution, avec élitisme, sélection par tournoi de 16 individus et remplacement

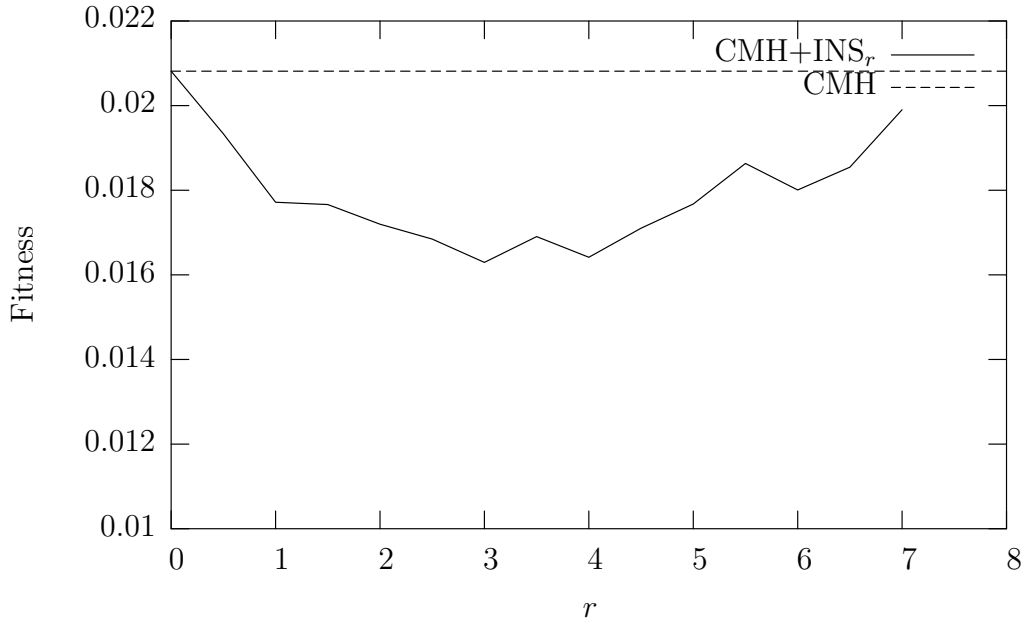

FIG. 5.21 – Problème inverse et CMH+INS$_r$: Fitness moyenne en fonction de r.

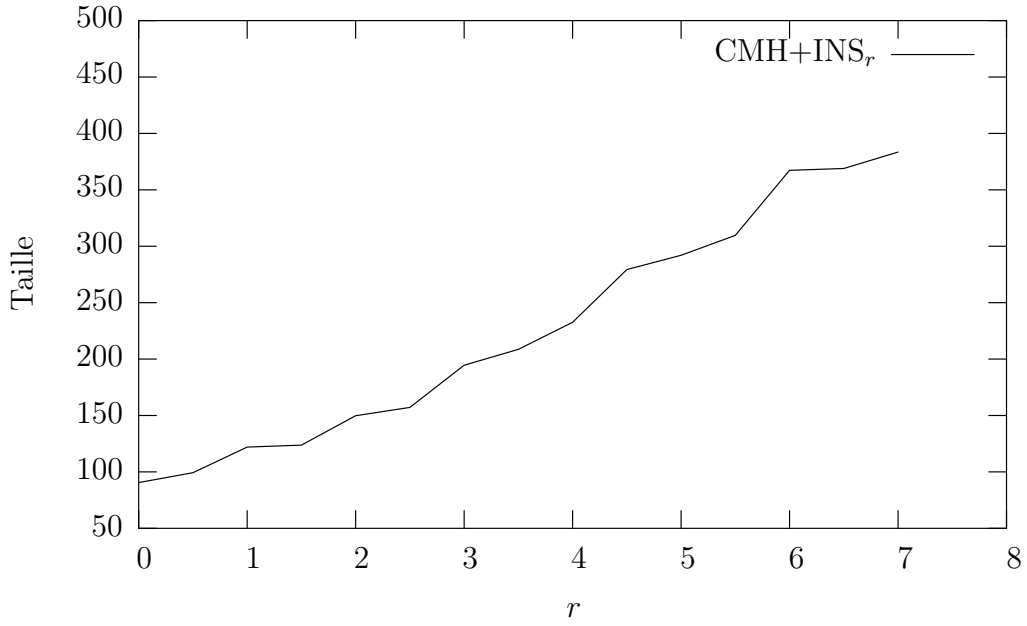

FIG. 5.22 – Problème inverse et CMH+INS$_r$: Taille moyenne en fonction de r.

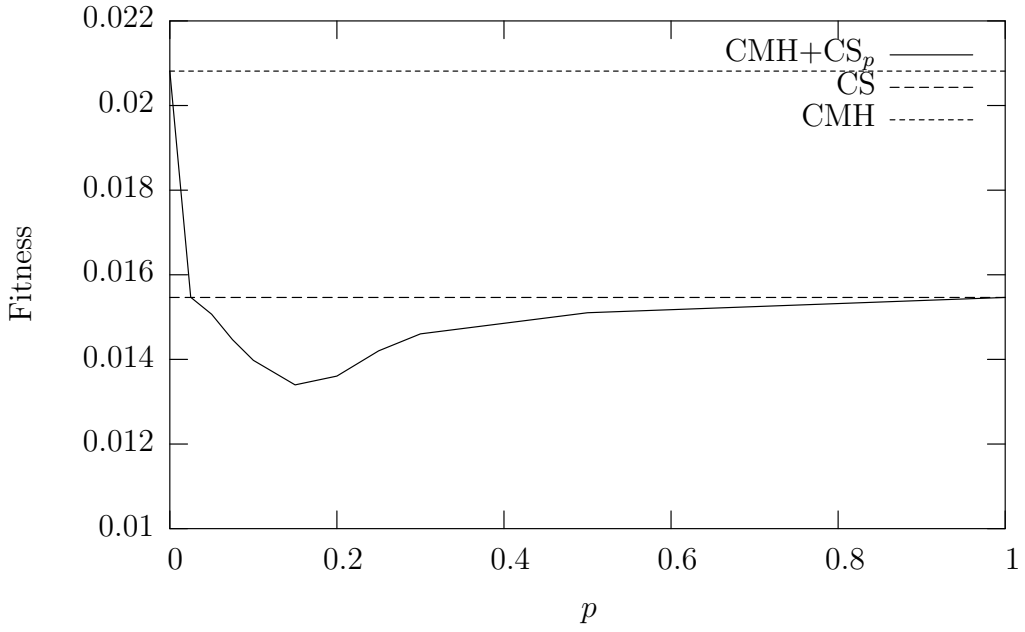

FIG. 5.23 – Problème inverse et CMH+CS$_p$: Fitness moyenne en fonction de p.

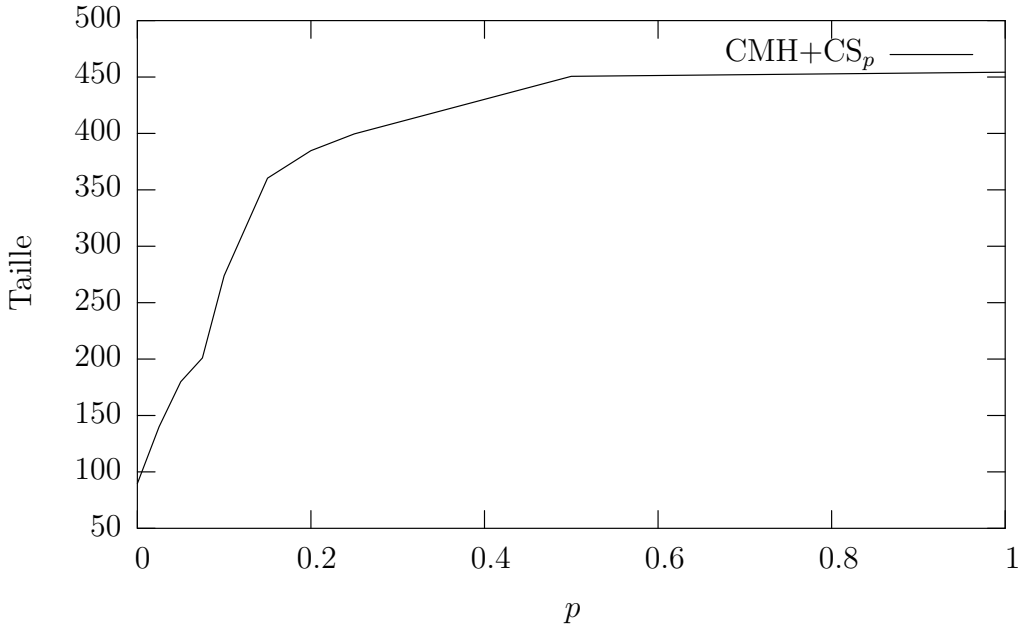

FIG. 5.24 – Problème inverse et CMH+CS$_p$: Taille moyenne en fonction de p.

TAB. 5.5 – Recherche poussée sur le problème inverse avec l'opérateur MHC+$CS_{0.15}$ et la combinaison de prédicteur `MoyP` avec β=1.0.

Erreur RMS	Moyenne	Écart Type	Meilleur	Pire
Apprentissage	0.009	0.001	0.007	0.022
Validation	0.009	0.001	0.007	0.020
Test	0.011	0.001	0.008	0.021

steady-state est limitée à 200 générations.

La Table 5.5 présente les résultats obtenus. L'erreur de test moyenne est proche de celle du RNA non-linéaire rapportée précédemment ce qui permet d'espérer de bonnes performances en ce qui concerne la capacité de généralisation des modèles. Bien que le nombre de générations ait été doublé, nous n'avons pas constaté de tendance au sur-apprentissage. Les meilleurs programmes découverts ont une taille moyenne de 748.4 instructions et constituent des équipes composées de 302 prédicteurs, en moyenne.

Meilleur modèle d'inversion

Le meilleur des programmes découverts, avec une erreur de test de 0.008, est composé de 709 instructions et forme une équipe de 240 prédicteurs. Nous avons étudié la distribution des poids w_i normalisés par la somme des poids pour ce programme, voir Figure 5.25. On peut voir que plus de 70% des prédicteurs ne sont pas pris en compte durant l'évaluation et que les prédicteurs les plus performants ne représentent qu'une très faible partie de l'équipe, c'est-à-dire environ 10%.

On s'intéresse à la qualité de l'inversion réalisée par le meilleur programme. On peut voir, Figure 5.26, en abscisse les τ_a espérés issus du l'ensemble de test et en ordonnée les τ_a prédits issus de l'évaluation du meilleur programme sur les vecteurs d'entrées correspondants. On constate que l'inversion est quasiment parfaite, avec une erreur RMS relative de 7.2% qui ne semble pas dépendre de valeur τ_a, autrement dit cette erreur est uniformément répartie sur l'intervalle $[0.1, 0.6]$

Une des caractéristiques les plus importantes d'un modèle d'inversion est

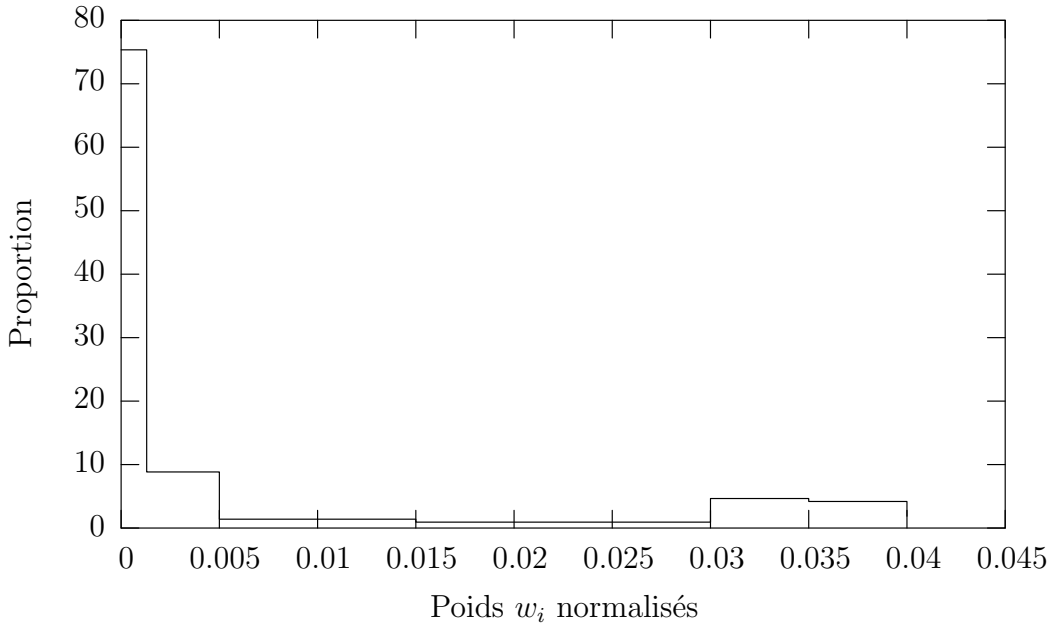

FIG. 5.25 – Problème inverse : Distribution des poids w_i normalisés de la meilleure équipe de prédicteur.

la robustesse, c'est-à-dire sa capacité de résistance au bruit. Pour évaluer la robustesse du meilleur programme, nous avons appliqué un bruit gaussien de plus ou moins 5%, indépendant pour chacune des 30 variables des 250 vecteurs d'entrée de l'ensemble de test. On peut voir, Figure 5.27, en abscisse les τ_a espérés issus du l'ensemble de test et en ordonnée les τ_a prédits issus de l'évaluation du meilleur programme sur les vecteurs d'entrées bruités correspondants. La qualité de l'inversion semble encore satisfaisante, avec une erreur RMS relative de 9.8%, ce qui signifie que le bruit n'a pas été amplifié par le processus d'inversion.

L'erreur relative commise par le meilleur programme n'est pas biaisée et sa distribution est proche d'un gaussienne, voir Figure 5.28. On peut donc raisonnablement penser pouvoir directement utiliser ce modèle de façon opérationnelle sur des données fournies par le réseau AERONET dont les radiomètres effectuent des mesures avec plus ou moins 5% d'erreur.

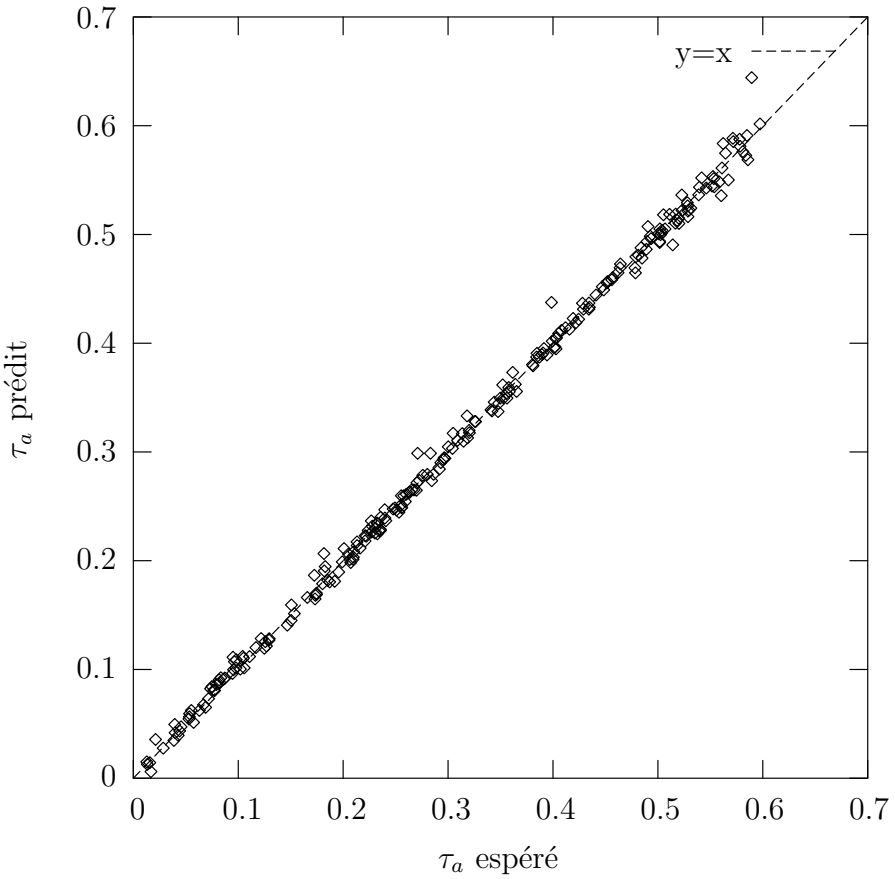

FIG. 5.26 – Problème inverse : Diagramme de prédiction du meilleur programme.

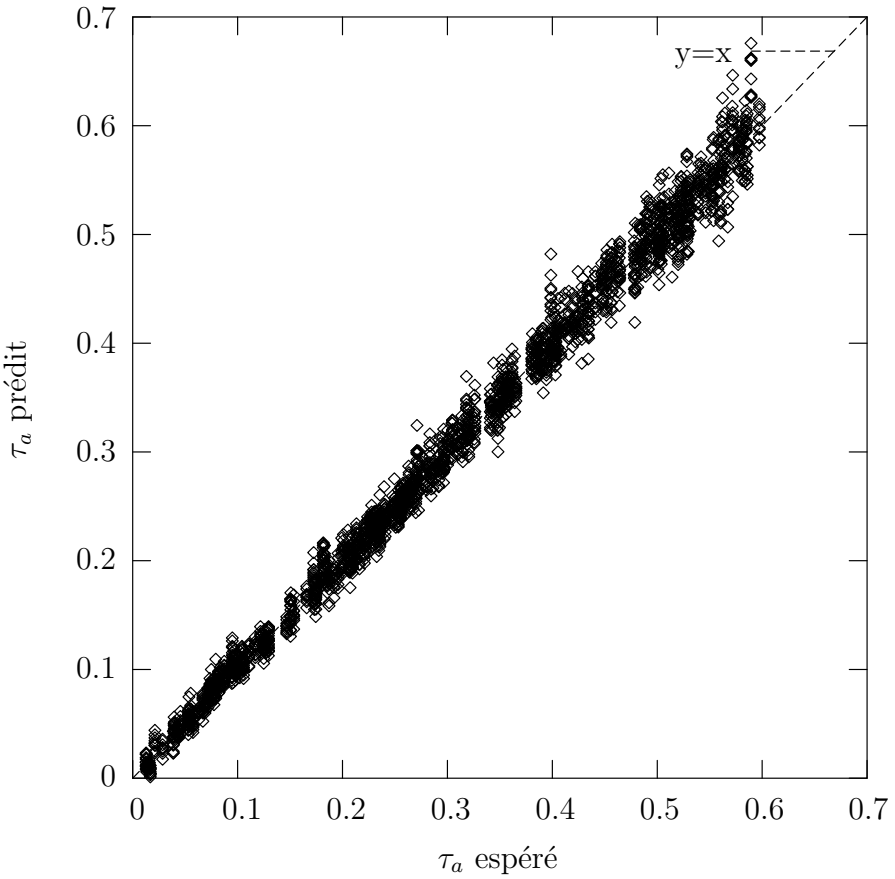

FIG. 5.27 – Problème inverse : Diagramme de prédiction du meilleur programme avec bruit gaussien de plus ou moins 5%.

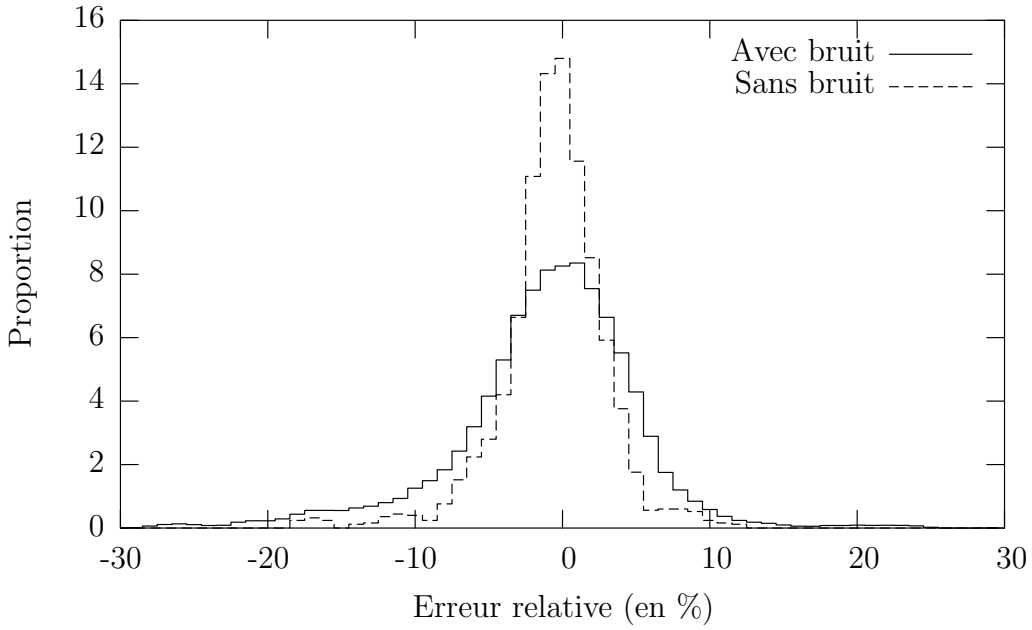

FIG. 5.28 – Problème inverse : Distribution de l'erreur relative du meilleur programme.

5.5 Conclusion

Dans ce chapitre, nous avons constaté l'efficacité des deux méthodes de contrôle de la taille, par translation et transformation de la distribution de la taille des programmes recombinés avec le CMH, qui ont principalement permis d'obtenir de meilleur compromis fitness *vs* taille. De plus, nous avons vu que l'approche par équipe de prédicteurs améliore significativement les performances du systèmes PGP, en particulier quand la performance de chaque prédicteur est prise en compte dans la combinaison. Enfin, nous avons montré expérimentalement la faisabilité de l'inversion par PG de caractéristiques des aérosols atmosphériques, les performances du modèle d'inversion, donné par le meilleur programme obtenu, étant comparables à celles de modèles obtenus avec des RNA.

Cependant, pour atteindre nos objectifs nous avons consacré un effort de calcul considérable pour rechercher la valeur optimale des paramètres du système, comme le taux d'application des opérateurs de croisement et

de mutation $\mathcal{T}_c$ et $\mathcal{T}_m$, le facteur β qui modifie la pondération en fonction de l'erreur des prédicteurs dans l'équipe ainsi que les paramètres r et p de contrôle de la taille. Il est clair que cette démarche ne peut être appliquée dans un cadre opérationnel, notamment parce que dans ce cas la comparaison avec les RNA est beaucoup trop défavorable (en temps). Par contre, nous pensons qu'en répétant cette démarche avec des problèmes différents, il sera possible de déterminer des valeurs optimales moyennes de ces paramètres qui pourront être par la suite utilisées de façon systématique. Une autre possibilité serait de déterminer ces valeurs dynamiquement grâce à des mécanismes d'auto-adaptation. Ainsi du temps de calcul sera disponible pour augmenter le nombre d'évaluation des programmes, notamment en manipulant des populations de plus grandes tailles et durant plus de générations.

Conclusion

L'opérateur de Croisement utilisé de façon Standard (CS) en Programmation Génétique (PG) a fait l'objet de nombreux travaux qui ont montré, théoriquement et expérimentalement, son caractère brutal, c'est-à-dire principalement ses effets délétères sur la fitness. Une des conséquences de cette brutalité est le phénomène de bloat, cette croissance exagérée des programmes qui pénalise la PG et la rend difficilement utilisable de manière opérationnelle. C'est ce constat qui a motivé les travaux présentés dans ce mémoire.

Pour lutter contre la brutalité de la recombinaison en PG, une idée consiste à utiliser des opérateurs, dits homologues, qui transmettent aux enfants des caractéristiques communes aux parents. Pour définir un nouvel opérateur, nous avons choisi d'utiliser l'acception moderne de l'homologie en biologie qui repose sur la similarité des séquences d'ADN, le Croisement par Maximum d'Homologie (CMH), pour la Programmation Génétique Linéaire (PGL). Nous avons démontré que le CMH conserve l'homologie en garantissant que la plus longue sous-séquence d'instructions communes aux parents est transmise aux enfants durant la recombinaison tout en assurant qu'un maximum de matériel génétique différent est échangé. Les expériences réalisées sur des problèmes variés ont montré que le CMH est beaucoup moins brutal que le CS et qu'ainsi le phénomène de bloat est nettement réduit. On notera qu'avec ce résultat, l'objectif principal de cette étude a été atteint.

Nous avons également étudié les performances de PG et de l'opérateur CMH pour la résolution d'un problème inverse. La faisabilité de l'inversion par PG a été montrée et des résultats comparables à ceux obtenus avec des Réseaux de Neurones Artificiels ont été rapportés. Nous pensons que la qualité des modèles d'inversion trouvés classe la PG parmi les méthodes

efficaces, compétitives et directement opérationnelles pour la résolution de problèmes inverses.

Nous avons vu que l'opérateur CS réalise trois types de recherches : le mélange, la diffusion et l'exploration de la taille des programmes. Ces trois types de recherche semblent nécessaires au bon fonctionnement des systèmes PG mais l'opérateur CS ne permet pas de les dissocier durant l'évolution. L'opérateur CMH est quasi exclusivement un mélangeur pour la PGL, ce qui nous permet d'envisager des stratégies originales pour le parcours de l'espace de recherche grâce au découplage des opérations de mélange, de diffusion et d'exploration. Nous pensons que ceci constitue une des perspectives les plus intéressantes pour le CMH. En effet, que ce soit d'un point de vue théorique ou expérimental, aucune raison ne justifie que la combinaison optimale des trois opérations de recherches soit constante au cours de l'évolution ni même qu'elle soit identique en fonction du problème à résoudre. Deux méthodes de contrôle de la taille, par translation et transformation de la distribution de la taille des programmes dans la population, ont été testées sur deux problèmes différents. Elles ont toutes deux montré que le découplage du mélange et de l'exploration de la taille des programmes permet d'améliorer le compromis fitness *vs* taille.

De nombreux travaux restent a mener pour comprendre et améliorer le comportement du CMH. Le premier d'entre eux serait certainement d'étendre le cadre théorique du théorème exact des schémas proposé par Poli pour qu'il intègre la recombinaison homologue du CMH. La définition et le calcul de la distance d'édition sur lesquels repose le CMH tel que nous l'avons défini pourraient être certainement discutés et modifiés ; par exemple, pour permettre de prendre en compte le typage des instructions, d'accélérer le calcul de l'ensemble des meilleurs alignements, ou bien même d'envisager la recombinaison homologue de structures arborescentes.

Compte tenu des résultats présentés dans ce mémoire, nous pensons que l'homologie, comme c'est le cas dans la nature, doit être prise en compte, quelque soit la définition retenue, durant les opérations de croisement et ce pour la plupart des algorithmes évolutionnaires manipulant des structures de taille variable.

Bibliographie

[1] E. Alba, C. Cotta, and J. M. Troya. Type-constrained genetic programming for rule-base definition in fuzzy logic controllers. In *Genetic Programming 1996 : Proceedings of the First Annual Conference*, pages 255–260, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[2] L. Altenberg. The evolution of evolvability in genetic programming. In *Advances in Genetic Programming.* MIT Press, 1994.

[3] D. Andre and A. Teller. A study in program response and the negative effects of introns in genetic programming. In *Genetic Programming 1996 : Proceedings of the First Annual Conference*, page 12, Stanford University, CA, USA, July 1996. MIT Press.

[4] D. Andre and A. Teller. Evolving team darwin united. *RoboCup-98 : Robot Soccer World Cup II. Lecture Notes in Computer Science*, 1604 :pages 346–352, 1999.

[5] P. J. Angeline. Two self-adaptive crossover operators for genetic programming. In *Advances in Genetic Programming 2*, chapter 5, pages 89–110. MIT Press, Cambridge, MA, USA, 1996.

[6] P. J. Angeline. Subtree crossover : Building block engine or macromutation ? In *Genetic Programming 1997 : Proceedings of the Second Annual Conference*. Morgan Kaufmann, July 1997.

[7] P. J. Angeline. An historical perspective on the evolution of executable structures. *Informaticae*, 36(1-4) :179–195, 1998.

[8] W. Banzhaf and W.B. Langdon. Some considerations on the reason for bloat. *Genetic Programming and Evolvable Machines*, 3(1) :81–91, 2002.

[9] T. Blickle and L. Thiele. Genetic programming and redundancy. In *Genetic Algorirhms within the Framework of Evolutionary Computation (Workshop at KI-94)*, pages 33–38, Saarbrücken, 1994. Max-Planck-Institut für Informatik.

[10] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1) :17–26, 2001.

[11] M. Brameier and W. Banzhaf. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines*, 2(4) :381–407, 2001.

[12] M. Brameier and W. Banzhaf. Neutral variations cause bloat in linear GP. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 290–299, Essex, 14-16 April 2003. Springer-Verlag.

[13] M. Brameier and W. Bhanzhaf. Explicit control of diversity and effective variation distance in linear genetic programming. In *Genetic Programming, Proceedings of the 5th European Conference, EuroGP 2002*, volume 2278 of *LNCS*, pages 37–49, Kinsale, Ireland, 3-5 April 2002. Springer-Verlag.

[14] W. S. Bruce. The lawnmower problem revisited : Stack-based genetic programming and automatically defined functions. In *Genetic Programming 1997 : Proceedings of the Second Annual Conference*, pages 52–57, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.

[15] E. Burke, S. Gustafson, and G. Kendall. A survey and analysis of diversity measures in genetic programming. In *GECCO 2002 : Proceedings of the Genetic and Evolutionary Computation Conference*, pages 716–723, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[16] M. Chami, M. Defoin Platel, and D. Antoine. Operational retrieval of aeorsol refractive index from ground based measurements of sky radiance and degree of polarization with a neural network approach. *Journal of Geophysical Research*, A paraître, 2004.

[17] M. Chami and D. Robilliard. Inversion of oceanic constituents in case i and case ii waters with genetic programming algorithms. *Applied Optics*, 40(30) :6260–6275, 2002.

[18] M. Chami, R. Santer, and E. Dilligeard. Radiative transfer model for the computation of radiance and polarization in an ocean-atmosphere system : polarization properties of suspended matter for remote sensing. *Applied Optics*, 40(15) :2398–2416, 2001.

[19] M. Clergue, P. Collard, M. Tomassini, and L. Vanneschi. Fitness distance correlation and problem difficulty for genetic programming. In *GECCO 2002 : Proceedings of the Genetic and Evolutionary Computation Conference*, pages 724–732, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[20] P. Collet, E. Lutton, F. Raynal, and M. Schoenauer. Polar IFS+parisian genetic programming=efficient IFS inverse problem solving. *Genetic Programming and Evolvable Machines*, 1(4) :339–361, 2000.

[21] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of an International Conference on Genetic Algorithms and the Applications*, pages 183–187, Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 July 1985.

[22] R. Crawford-Marks and L. Spector. Size control via size fair genetic operators in the PushGP genetic programming system. In *GECCO 2002 : Proceedings of the Genetic and Evolutionary Computation Conference*, pages 733–739, New York, 9-13 July 2002. Morgan Kaufmann Publishers.

[23] J. M. Daida, J. A. Polito, S. A. Stanhope, R. R. Bertram, J. C. Khoo, and S. A. Chaudhary. What makes a problem GP-hard ? analysis of a tunably difficult problem in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 982–989, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

[24] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. John Murray, London, 1859.

[25] E. D. de Jong, R. A. Watson, and J. B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, San Francisco, 2001. Morgan Kaufmann.

[26] M. Defoin Platel, M. Clergue, and P. Collard. Homolgy gives size control in genetic porgramming. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*. IEEE Press, 2003.

[27] M. Defoin Platel, M. Clergue, and P. Collard. Maximum homologous crossover for linear genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 200–210, Essex, 14-16 April 2003. Springer-Verlag.

[28] M. Defoin Platel, S. Verel, M. Clergue, and P. Collard. From royal road to epistatic road for variable length evolution algorithm. In *Evolution Artificielle, 6th International Conference*, volume 2936 of *Lecture Notes in Computer Science*, pages 3–14, Marseilles, France, 27-30 October 2003. Springer. Revised Selected Papers.

[29] P. D'haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, Orlando, Florida, USA, 27-29 1994. IEEE Press.

[30] J-P. Drecourt. Using artificial neural networks and genetic programming in rainfall/runoff modeling. In *3rd DHI Software Conference & DHI Software Courses*, Helsingør, Denmark, 7-11 June 1999.

[31] L. Fogel, A. Owers, and M. Walsh. Artificial intelligence through simulated evolution. John Wiley, 1966.

[32] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundation of Genetic Algorithms 2*, pages 109–126. Morgan Kaufman, 1993.

[33] P. Gitchoff and G. P. Wagner. Recombination induced hypergraphs : a new approach to mutation-recombination isomorphism. *Complex.*, 2(1) :37–43, 1996.

[34] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, 1989.

[35] L. Gross, S. Thiria, R. Frouin, and B. G. Mitchell. Artificial neural networks for modeling the transfer function between marine reflectances and phytoplankton pigment concentration. *J. Geophys. Res.*, C2(105) :3483–3495, 2000.

[36] D. Gusfield. *Algorithms on Strings, Tree and Sequences*. Cambridge University Press, 1997.

[37] S. Gustafson, A. Ekart, E. Burke, and G. Kendall. Problem difficulty and code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 5(3) :271–290, September 2004.

[38] S. Handley. On the use of a directed acyclic graph to represent a population of computer programs. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 154–159, Orlando, Florida, USA, 27-29 June 1994. IEEE Press.

[39] K. Harries and P. Smith. Exploring alternative operators and search strategies in genetic programming. In *Genetic Programming 1997 : Proceedings of Second Annual Conference*, pages 81–88, Stanford University, 147-155. Morgan Kaufmann.

[40] K. Harries and P. W. H. Smith. Code growth, explicitly defined introns and alternative selection schemes. Under Submission, 1998.

[41] M. I. Heywood and A. N. Zincir-Heywood. Dynamic page based crossover in linear genetic programming. *IEEE Transactions on Systems, Man, and Cybernetics : Part B - Cybernetics*, 32(3) :380–388, June 2002.

[42] J. H. Holland. *Adaptation in natural and artificial systems*. Ann Arbor : University of Michigan Press, first edition, 1975. Second printing in 1993, The MIT Press.

[43] M. Huynen, P. Stadler, and W. Fontana. Smoothness within ruggedness : The role of neutrality in adaptation, 1996.

[44] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann Publishers Inc., 1995.

[45] W. Kantschik and W. Banzhaf. Linear-tree GP and its comparison with other GP structures. In *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 302–312, Lake Como, Italy, 18-20 April 2001. Springer-Verlag.

[46] S. A. Kauffman. *"The origins of order". Self-organization and selection in evolution.* Oxford University Press, New-York, 1993.

[47] M. Keijzer. *Scientific Discovery using Genetic Programming.* PhD thesis, Danish Technical University, Lyngby, Denmark, March 2002.

[48] M. Keijzer. Improving symbolic regression with interval arithmetic and linear scaling. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 71–83, Essex, 14-16 April 2003. Springer-Verlag.

[49] M. Keijzer and V. Babovic. Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations. In *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 76–90, Edinburgh, 15-16 April 2000. Springer-Verlag.

[50] M.J. Keith and M.C. Martin. Genetic programming in c++ : Implementation issues. In *Advances in Genetic Programming*, chapter 13, pages 285–310. MIT Press, 1994.

[51] M. Kimura. *The Neutral Theory of Molecular Evolution.* Cambridge University Press, Cambridge, UK, 1983.

[52] K. E. Kinnear, Jr. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1, pages 142–147, Orlando, Florida, USA, 27-29 1994. IEEE Press.

[53] J. R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, volume 1, pages 768–774. Morgan Kaufman, 1989.

[54] J. R. Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[55] J. R. Koza. *Genetic Programming II : Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[56] J. R. Koza, III Bennett, H. Forrest, Andre, David, Keane, and A. Martin. *Genetic Programming III : Darwinian Invention and Problem Solving*. MIT Press, 1999.

[57] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *NIPS*, 7 :231–238, 1995.

[58] W. B. Langdon. Size fair and homologous tree genetic programming crossovers. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1092–1097, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[59] W. B. Langdon and R. Poli. Fitness causes bloat. In *Second Online World Conference on Soft Computing in Engineering Design and Manufacturing*, pages 13–22. Springer-Verlag London, 23-27 1997.

[60] W. B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[61] E.R. Lankester. On the use of the term homology in modern zoology. *Annals and Magazine of Natural History*, 6(4) :34–43, 1870.

[62] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics-Doklady, 1966.

[63] S. Luke. Code growth is not caused by introns. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 228–235, Las Vegas, Nevada, USA, 8 2000.

[64] S. Luke and L. Panait. A survey and comparaison of tree generation algorithms. In *Proceedings of the Genetic and Evolutionary Compu-*

tation Conference (GECCO-2001), pages 81–88, San Francisco, 2001. Morgan Kaufmann.

[65] S. Luke and L. Panait. Fighting bloat with nonparametric parsimony pressure. In *Parallel Problem Solving fro Nature - PPSN VII*, page 441. Springer-Verlag, 2002.

[66] S. Luke and L. Spector. Evolving teamwork and coordination with genetic programming. In *Genetic Programming 1996 : Proceedings of the First Annual Conference*, pages 150–156, Stanford University, CA, USA, 28–31 1996. MIT Press.

[67] J. Maynard-Smith. *The Theory of Evolution*. Penguin Books, New York, third edition, 1975.

[68] N. F. McPhee and J. D. Miller. Accurate replication in genetic programming. In *Genetic Algorithms : Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, Pittsburgh, PA, USA, 15-19 1995. Morgan Kaufmann.

[69] G. Mie. Beitrage zur optik truber medien, speziell kolloidalen metallosungen. *Annale Physics*, 25 :377–442, 1908.

[70] D. J. Montana. Strongly typed genetic programming. Technical Report #7866, Cambridge University, 10 Moulton Street, Cambridge, MA 02138, USA, 7 1993.

[71] P. Nordin, W. Banzhaf, and F. D. Francone. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In *Advances in Genetic Programming 3*, chapter 12, pages 275–299. MIT Press, Cambridge, MA, USA, June 1999.

[72] P. Nordin, F. Francone, and W. Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In *Proceedings of the Workshop on Genetic Programming : From Theory to Real-World Applications*, pages 6–22, Tahoe City, California, USA, 9 1995.

[73] M. O'Neill, C. Ryan, and M. Nicolau. Grammar defined introns : An investigation into grammars, introns, and bias in grammatical evolution., 2001.

[74] U. O'Reilly. Using a distance metric on genetic programs to understand genetic operators. In *Late Breaking Papers at the Genetic Programming 1997 Conference, J. R. Koza, Ed., Stanford University*, 1997.

[75] R. Owen. *Lectures on the comparative anatomy and physiology of the vertebrate animals.*, page 374. London : Longman, Brown, Green and Longmans, 1843.

[76] G. Paris, D. Robilliard, and C. Fonlupt. Applying boosting techniques to genetic programming. In *Artificial Evolution 5th International Conference, Evolution Artificielle, EA 2001*, volume 2310 of *LNCS*, pages 267–278, Creusot, France, October 29-31 2001. Springer Verlag.

[77] T. Perkis. Stack-based genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, Orlando, Florida, USA, 27-29 1994. IEEE Press.

[78] R. Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 200–210, Essex, 14-16 April 2003. Springer-Verlag.

[79] R. Poli and W. B. Langdon. Genetic programming with one-point crossover. In *Soft Computing in Engineering Design and Manufacturing*, pages 180–189. Springer-Verlag London, 23-27 June 1997.

[80] R. Poli and N. F. McPhee. Exact schema theorems for GP with one-point and standard crossover operating on linear structures and their application to the study of the evolution of size. In *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038, pages 126–142. Springer-Verlag, 18-20 2001.

[81] R. Poli and J. Page. Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code GP and demes. *Genetic Programming and Evolvable Machines*, 1(1/2) :37–56, 2000.

[82] R. Poli, J. E. Rowe, C. R. Stephens, and A. H. Wright. Allele diffusion in linear genetic programming and variable-length genetic algorithms with subtree crossover. In *Proceedings of the 4th European Conference*

on Genetic Programming, EuroGP 2002, volume 2278, pages 213–228, Kinsale, Ireland, 3-5 2002. Springer-Verlag.

[83] W. F. Punch, D. Zongker, and E. D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In *Advances in Genetic Programming 2*, chapter 15, pages 299–316. MIT Press, Cambridge, MA, USA, 1996.

[84] N. Radcliffe. Forma analysis and random respectful recombination, 1991.

[85] C. Rajanayake, S. Samarasinghe, and D. Kulasiri. Solving the inverse problem in stochastic groundwater modelling with artificial neural networks. In *Proceeding of Biennial Congress of the International Environmental Modelling and Software Society (IEMSS)*, volume 2, pages 154–159. IEMSS, 2002.

[86] I. Rechenberg. Cybernetic solution path of an experimental problem. Royal Aircraft Establishment, Library translation No. 1122, Farnborough, Hants, UK., Aug 1965.

[87] D. Robilliard and C. Fonlupt. Backwarding : An overfitting control for genetic programming in a remote sensing application. In *Selected Papers from the 5th European Conference on Artificial Evolution*, pages 245–254. Springer-Verlag, 2002.

[88] J. Rosca and D. Ballard. Causality in genetic programming. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 256–263, Pittsburgh, 1995. Morgan Kaufmann.

[89] J. P. Rosca. *Hierarchical Learning with Procedural Abstraction Mechanisms*. PhD thesis, Department of Computer Science, The College of Arts and Sciences, University of Rochester, Rochester, NY 14627, USA, February 1997.

[90] J. E. Rowe and N. F. McPhee. The effects of crossover and mutation operators on variable length linear structures. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 535–542, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[91] C. Ryan, J. J. Collins, and M. O Neill. Grammatical evolution : Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming*, volume 1391, pages 83–95, Paris, 14-15 1998. Springer-Verlag.

[92] C. Ryan, M. O'Neill, and J. J. Collins. Grammatical evolution : Solving trigonometric identities. In *Proceedings of Mendel 1998 : 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Sets.*, pages 111–119, Brno, Czech Republic, 24-26 1998. Technical University of Brno, Faculty of Mechanical Engineering.

[93] W. Sarle. Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on Interface*, 1995.

[94] H.P. Schwefel. Evolution and optimum seeking. John Wiley & Sons, 1995.

[95] P. W. H. Smith. Controlling code growth in genetic programming. In *Advances in Soft Computing*, pages 166–171, De Montfort University, Leicester, UK, 2000. Physica-Verlag.

[96] T. Soule and J.A. Foster. Removal bias : A new cause of code growth in tree-based evolutionary programming. In *Proceedings of the International Conference on Evolutionary Computation (ICEC'98)*, pages 781–786, 1998.

[97] T. Soule and J.A. Foster. Effects of code growth and parsimony pressure on population in genetic programming. *Evolutionary Computation*, 6(4) :293–309, 1999.

[98] T. Soule and J.A. Foster. An analysis of the causes of code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 3(1) :283–309, 2002.

[99] L. Spector. Evolving control structures with automatically defined macros. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 99–105, MIT, Cambridge, MA, USA, 10–12 1995. AAAI.

[100] L. Spector. Autoconstructive evolution : Push, pushGP, and pushpop. In *Proceedings of the Genetic and Evolutionary Computation Confe-*

rence (GECCO-2001), pages 137–146, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[101] L. Spector and H.J. Bernstein. Communication capacities of some quantum gates, discovered in part through genetic programming. In *Proceedings of the Sixth International Conference on Quantum Communication, Measurement, and Computing*, Paramus, NJ, 2002. Rinton Press.

[102] K. Stoffel and L. Spector. High-performance, parallel, stack-based genetic programming. In *Genetic Programming 1996 : Proceedings of the First Annual Conference*, pages 224–229, Stanford University, CA, USA, 28–31 1996. MIT Press.

[103] R.E. Keller W. Banzhaf, P. Nordin and F.D. Francone. *Genetic Programming - An Introduction.* Morgan Kaufmann, 1998.

[104] E. D. Weinberger. Local properties of kauffman's N-k model, a tuneably rugged energy landscape. *Physical Review A*, 44(10) :6399–6413, 1996.

[105] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1) :67–82, April 1997.

[106] S. Wright. Evolution in mendelian population. *Genetics 16 : pp. 97-159*, 1931.

[107] Xin Yao. Universal approximation by genetic programming. In *Foundations of Genetic Programming*, Orlando, Florida, USA, 13 1999.

[108] Byoung-Tak Zhang and Je-Gun Joung. Time series prediction using committee machines of evolutionary neural trees. In *Proceedings of the Congress of Evolutionary Computation*, volume 1, pages 281–286. IEEE Press, 1999.

Homologie en Programmation Génétique
Application à la résolution d'un problème inverse

Les Algorithmes Évolutionnaires (AE) sont des méthodes de recherche inspirées par la théorie darwinienne de l'évolution, travaillant, en général, sur une population de solutions potentielles, par itération de phases de sélections et de variations aléatoires. La Programmation Génétique (PG) est un AE dont l'objectif est la recherche automatique de programmes au sens large. L'une des particularités de la PG est qu'elle manipule des représentations complexes : arbres (PGA) ou listes de longueur variables (PGL), par exemple. Les variations aléatoires permettant de créer de nouveaux programmes peuvent être soit des modifications locales (mutations), soit des recombinaisons de programmes (croisements). L'opérateur de croisement est souvent considéré comme le plus important et constitue un axe de recherche majeur en PG. Cet opérateur, dans sa version d'origine, recombine aléatoirement des sous-parties de programmes sans tenir compte du contexte : c'est une opération «brutale» qui est une des causes supposées de la croissance incontrôlée de la taille des programmes.

Inspirés par la recombinaison homologue de l'ADN, nous définissons, le Croisement par Maximum d'Homologie (CMH). A partir d'une mesure de similarité entre les expressions à recombiner, le CMH favorise les échanges qui respectent les structures communes préexistantes. Dans un premier temps, nous étudions les propriétés théoriques remarquables du CMH ; puis, nous déterminons expérimentalement son paramétrage et ses propriétés dynamiques. La capacité du CMH à effectuer une recherche moins brutale et à permettre un contrôle précis de la taille des programmes est par la suite mise en évidence sur des problèmes classiques de PG, en particulier dans le domaine de l'approximation de fonctions par régression symbolique.

En partant des différents résultats obtenus, nous appliquons notre méthode à la résolution d'un problème inverse complexe, l'inversion des composantes atmosphériques. Nous montrons comment, à coût constant, il est possible de rechercher des combinaisons de modèles inverses dont les performances sont supérieures aux modèles standards.

Mots Clés : Programmation Génétique, Algorithmes Évolutionnaires, Homologie, Recombinaison, Problème Inverse

---

Homology in Genetic Programming
Application to inverse problem solving

Evolutionary Algorithms (EA) are search methods working iteratively on a population of potential solutions that are randomly selected and modified. Genetic Programming (GP) is an EA that allows automatic search for programs, usually represented as syntax trees (TGP) or linear sequences (LGP). Two mechanisms perform the random variations needed to obtain new programs : the mutation operator (local variation) and the crossover operator (programs recombination). The crossover operator blindly exchanges parts of programs without taking the context into account, this is a "brutal" operation that may be responsible of the uncontrolled growth of programs during evolution.

Mainly inspired by the homologous recombinaison of DNA strands, we introduce the Maximum Homologous Crossover for LGP. The MHC ensures, thanks to a measure of similarity, that recombinaison of programs is respectful. We show on classical GP benchmarks, e.g. the symbolic regression problem, that when using MHC the search process is less brutal and that an accurate control of programs size is also possible. These results are used to address a real world problem : the inversion of atmospheric components.

We show that, with a constant computational effort, it is also possible to find teams of inversion predictors that outperform standard models.

Keywords : Genetic Programming, Evolutionary Algorithm, Homology, Recombination, Inverse Problem