50376
1998
241

Numéro d'ordre: 2336

UNIVERSITE DES SCIENCES ET TECHNOLOGIES DE LILLE

U.F.R. I.E.E.A.

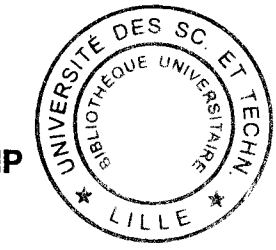
THESE

pour obtenir le grade de

DOCTEUR DE
L'UNIVERSITE DES SCIENCES ET TECHNOLOGIQUES DE LILLE

Discipline: Productique: Automatique et Informatique Industrielle

présentée et soutenue publiquement



par

Catherine Khamphang BOUNSAYTHIP

le 9 octobre 1998

Algorithmes Heuristiques et Evolutionnistes : Application à la Résolution du Problème de Placement de Formes Irrégulières

---

Directeur de thèse: Prof. Salah MAOUCHE

JURY

M. Pierre VIDAL	Président
Mlle Marie-Claude PORTMANN	Rapporteur
M. René SOENEN	Rapporteur
M. Pierre BORNE	Examinateur
M. Gilles ROUSSEL	Examinateur
M. Salah Maouche	Directeur

AVANT-PROPOS

L'étude rapportée dans ce document a été menée au sein du Laboratoire d'Automatique I³D, anciennement Centre d'Automatique de Lille (C.A.L.). Et avant tout propos, je voudrais exprimer ma profonde reconnaissance à l'égard de **Monsieur le Professeur Pierre VIDAL,** professeur Emérite de l'USTL, ancien directeur du C.A.L. , pour m'avoir accueillie dans son équipe et pour m'avoir permis d'effectuer ma thèse dans de bonne condition.

Ensuite, je voudrais présenter mes plus vifs et profonds remerciements à **Monsieur le Professeur Salah MAOUCHE**, co-directeur de cette thèse. Il a été tout au long de cette étude, mon guide spirituel, aussi bien dans le domaine de la recherche que dans ma vie personnelle. Il m'a beaucoup guidée dans mes premiers pas de chercheur, sur la façon d'analyser les problèmes, sur la façon de rédiger un article scientifique etc. Ses qualités humaines et son esprit d'initiative m'ont beaucoup inspiré et encouragée dans mes moments difficiles. Je lui dois vraiment toute ma gratitude. Au passage, je salue également toute sa sympathique famille.

J'apprécie beaucoup l'honneur que me font les rapporteurs extérieurs en jugeant ce travail. Leur appréciation, commentaire, et correction méritent une reconnaissance particulière. J'exprime donc toute ma reconnaissance à:

* **Mademoiselle Marie-Claude PORTMANN**, Professeur à l'Institut National Polytechnique de Lorraine (Ecole des Mines de Nancy) et présidente de la ROADEF, Société Française de Recherche Opérationnelle et d'Aide à la Décision. Par sa position et sa grande expérience dans le domaine des algorithmes, je suis honorée de son acceptation de rapporter cette thèse.

* **Monsieur René SOENEN**, Professeur à l'université de Valenciennes et du Hainaut-Cambresis, directeur du laboratoire de Génie Industriel et Logiciel. Sa connaissance et son expérience réputées dans le domaine du placement et de la confection, apportent un poids supplémentaire dans le jugement de ce travail.

Je suis aussi très honorée de la présence dans le jury et du jugement des examinateurs:

- **Monsieur Pierre E. BORNE,** Professeur à l'École Centrale de Lille, spécialiste du domaine "Contrôle et Analyse de Systèmes Complexes", auteurs de plusieurs livres dans le domaine et chairman de nombreuses conférences internationales, dont IEEE SMC, IMACS, IFAC...

- **Monsieur Gilles ROUSSEL,** Maître de Conférences à l'Université du Littoral. A **Gilles ROUSSEL**, je voudrais adresser un remerciement particulier, pour son immense travail qui m'a servi de base solide dans cette étude. Aussi, son intelligence et sa sympathie m'ont donné confiance dans les débuts de ma thèse.

J'adresse également mes remerciements à tout le personnel du laboratoire d'Automatique I³D, notamment **Prof. Christian Vasseur, Prof. Jack-Gérard Postaire**, et **Prof. Jean-Marc Toulotte.**

Ensuite, à **Dr. Michel Letellier**, Maître de Conférences à l'I.U.T. de Tours et chercheur au CNRS d'Orléans, j'adresse ma profonde reconnaissance, pour son constant encouragement tout au long de mon parcours universitaire.

Je dédie cette thèse à **ma mère, mes frères et soeurs** ainsi qu'à mon fiancé **Prof. Timo Honkela,** pour leur patience et soutien à tout moment de ma vie.

Je salue aussi **tous mes amis, toutes celles** et **tous ceux** qui, à tous moments, m'ont témoigné leur sympathie et leur soutien.

In Finland, I would like to thank all my colleagues at VTT Information Technology, namely the team "Information Systems" to which I belong. I address a special thanks to ***Prof. Seppo Linnainmaa, Aarno Lehtola,*** *and* ***Juha Savola*** *for their encouragement and their arrangements so that I could work on my thesis and also develop my research interests.*

Finally, I would like to thank my correspondents in Genetic Algorithms : ***Prof. J.T. Alander*** *(Vaasa, Finland) and* ***Dr. Helmut Meyer*** *(Salzburg, Austria).*

Fait à Villeneuve d'Ascq, le 9 octobre 1998

Catherine K. Bounsaythip

RESUME

Cette thèse présente les techniques d'optimisation telles que les recherches en arbre A^*_ε et $R^*_{\delta\varepsilon}$, le recuit simulé et les algorithmes évolutionnistes pour résoudre le problème de placement de formes irrégulières. Les méthodes de recherche en arbre sont basées sur l'algorithme A*, mais ayant des propriétés semi-admissibles pour accepter également des solutions sous-optimales. La seconde méthode est une hybridation entre le recuit simulé et l'algorithme en arbre dans lequel le paramètre *température* permet de contrôler la descente dans l'arbre. Cette technique permet les déplacements uniquement dans le voisinage du point courant de l'espace de recherche, c'est pourquoi la technique évolutionniste, basée sur une population de points de recherche, est appliquée. La première phase de l'approche évolutionniste consiste à étudier l'encodage du problème en utilisant le code de contour en "peignes", dérivant du codage discret de contour. La deuxième phase utilise une représentation hiérarchique en arbre comme en programmation génétique. Les tests sur la résolution de problème de placement de formes textiles ont montré que les différentes méthodes ont des propriétés complémentaires, qui peuvent être utilisées en coopération, dans un environnement dynamiquement changeant.

HEURISTIC AND EVOLUTIONARY ALGORITHMS: APPLICATION TO IRREGULAR SHAPE PLACEMENT PROBLEM

ABSTRACT: This thesis presents optimisation techniques such as tree search, simulated annealing and, evolutionary algorithms to solve a 2D irregular shape placement. The tree search methods are basically A^*, with ε-admissible features, so that sub-optimal solutions are also accepted. The simulated annealing is hybridised with tree search with *temperature* as a control parameter, which helps in escaping local minima by the "re-annealing" process. This technique permits to search only in the neighbourhood moving point per point. To solve this drawback, evolutionary algorithms is used to evolve a population of solution candidates "in parallel", based on the theory of genetic evolution. The encoding used in the first phase is based on "comb code", derived from the encoding of discrete rectangular deficiency computation, and in the second phase, a hierarchical structure representation like in genetic programming is used. Results show that the methods are competitive and can be used as co-operating processes in a dynamically changing environment.

DISCIPLINE: Productique: Automatique et Informatique Industrielle

Mots-clés: placement textile, recherche en arbre, recuit simulé, algorithmes évolutionnistes.

Key-words: textile shape placing, tree search, simulated annealing, evolutionary algorithms.

Laboratoire d'Automatique I^{3}D, Bât P2, 2ème étage
Université des Sciences et Technologies de Lille, 59655 Villeneuve d'Ascq CEDEX, France

TABLE DE MATIERES

Chapitre Introduction

0. Introduction 1
0.1 Contribution de cette thèse 1
0.2 Organisation de ce document 4

Chapitre I

I.1. Recherche et développement en textile 9
I.2. Problèmes autour du problème de placement 14
I.3. Les différents types de placement 18
I.4. Complexité du problème de placement 19
I.5. Définition du problème de placement 20
I.6. Les données pour tester les algorithmes 33
I.7. Conclusion 38
I.8. Références Bibliographiques 39

Chapitre II

II.1. Introduction 45
II.2. Algorithmes de recherche en arbre 46
II.3. Algorithme A^* 49
II.4. Algorithmes Semi-admissibles 53
II.4.1. Algorithme ε-admissible A_ε^* 54
II.4.2. Algorithme R_δ^* : Introduction aux principes 56
II.4.3. Algorithme $R_{\delta\varepsilon}^*$ 62
II.5. Application au problème de placement 63
II.6. Résultats avec l'algorithme A_ε^* 69
II.7. Résultats obtenus avec l'algorithme $R_{\delta\varepsilon}^*$ 72
II.8. Comparaison des deux techniques en arbre 77
II.9. Conclusion sur les recherches heuristiques en arbre 81
II.10 Exemples de placements 83
II.11 Références bibliographiques 93

Chapitre III

III.1. Introduction 97
III.2. Recuit Simulé 98
III.3. Caractéristiques du recuit simulé 100
III.4. Application au placement 104
III.5. Comparaison avec les algorithmes en arbres 118
III.6. Conclusion 119
III.7. Références Bibliographiques 125

Chapitre IV

IV.1. Introduction 131
IV.2. Principe des Algorithmes Evolutionnistes 133
IV.2.1. Codage pour les AG 136
IV.2.2. Décodage 137
IV.2.3. Construction de la population initiale 137
IV.2.4. Fonction de fitness 138

IV.2.5. Fitness Scaling 139
IV.2.6. Sélection 139
IV.2.7. Modèle de Reproduction 142
IV.2.8. Opérateurs génétiques 142
IV.2.9. Critère d'arrêt 145
IV.2.10. Compromis entre l'exploration et l'exploitation 148
IV.2.11. Le choix des valeurs des paramètres 149
IV.2.12. Théorème des schéma 150
IV.3. Aspects pratiques des AG 153
IV.3.1. Le concept de sous population 153
IV.3.2. Les AGs Parallèles 154
IV.4. Application au problème de placement 157
IV.4.1. Le problème de codage 157
IV.4.2. Première approche : Codage en peignes réduits 161
IV.4.3. Deuxième approche : Codage arborescent 166
IV.4.4. Implantation 174
IV.4. RESULTATS 177
IV.5. Discussion 183
IV.6. Conclusion 183
IV.7 Références bibliographiques 185

Chapitre V

V.1. Comparaison des quatre méthodes 193
V.2. Conclusions 195
V.2.1. Expérience acquise 196
V.3. Perspectives 197
V.3.1. Organisation entre les agents de recherche 198
V.4. Références Bibliographiques 200

ANNEXES

ANNEXE 1 : Techniques de recherche 203
ANNEXE 2 : Stratégie évolutionniste 205
ANNEXE 3 : Programmation génétique 209
ANNEXE 4 : Opérateurs génétiques 219

INTRODUCTION

CHAPITRE INTRODUCTION

Introduction

Les problèmes d'optimisation combinatoire suscitent beaucoup d'intérêts. Malgré les progrès considérables de l'outil informatique, les méthodes d'énumération, exhaustive ou partielle, sont encore peu satisfaisantes en temps d'exécution ou en efficacité. Comme ces problèmes contiennent souvent beaucoup de solutions à intérêts pratiques acceptables, les spécialistes de l'optimisation combinatoire ont orienté leur recherche vers le développement des méthodes heuristiques. Le but est de trouver une solution de qualité satisfaisante en un temps de calcul raisonnable. D'autant plus que pour des problèmes réels, il n'est pas toujours impératif de trouver la solution optimale, mais des solutions dont la qualité et le temps pour l'obtenir restent dans l'acceptable. Ces performances étant de nature opposée, il s'agit alors de trouver un compromis selon le contexte du problème.

0.1.1. Contribution de Cette Thèse

L'objet de cette thèse porte sur l'étude de trois méthodes heuristiques pour résoudre le problème du placement. Le but final est de constituer une bibliothèque d'algorithmes à utiliser selon le choix de l'utilisateur (carnet de commande, critère d'optimalité, temps de calcul...). Des analyses des résultats et les comportements des différents algorithmes permettront de conclure sur l'efficacité de chaque algorithme en fonction des critères à optimiser.

0.1.1.1. Le problème de placement

Le but du placement est de positionner un ensemble de formes sur un support à deux dimensions, en respectant un ensemble de contraintes et en optimisant une fonction objectif. Dans un contexte d'automatisation de fabrication, la cellule flexible de coupe a été imaginée pour faire intervenir différentes tâches de production participant à la confection. Les nouvelles contraintes de production imposent aux industries de la confection d'améliorer les performances de ces tâches afin de réduire le coût du produit fini en optimisant avant tout l'utilisation de la matière première. Celle-ci se présente sous forme d'une bande de tissu dont la longueur peut être considérée comme non bornée. Le nombre de pièces à produire est variable et peut être connu ou non à l'avance. De plus, la nécessité de produire vite conduit à inclure des contraintes temporelles sévères et à définir un objectif qui est d'obtenir une solution acceptable dans un délai donné. A cause de ces contraintes, le problème est très complexe; et selon les objectifs à atteindre et les modélisations choisies, les stratégies de résolution diffèrent d'un problème à l'autre. L'état de l'art montre que les méthodes utilisées sont très diverses, mais elles restent encore limitées soit en efficacité de recherche, soit en nombre de formes considérées. Il est donc nécessaire de trouver des algorithmes qui permettent de satisfaire ces exigences.

Un algorithme de placement doit être en mesure de fournir le placement d'un sous-ensemble de ces formes. Il cherche à optimiser une fonction de but visant à minimiser les pertes de matière, tout en respectant un temps de calcul acceptable par rapport au rythme des opérations connexes. Les trois types d'approches étudiées ici sont: *les recherches en arbre*, *le recuit simulé*, et *les méthodes évolutionnistes*.

0.1.1.2. Travail existant

Un important travail a été effectué par G. Roussel [Roussel94] dans la réalisation d'une plate-forme de simulation de placement en utilisant un langage orienté objet: le langage Smalltalk. Le codage des formes par peignes de contour et deux algorithmes de placement ont été implantés (A^*_ε et le recuit simulé). Le résultat obtenu est satisfaisant pour l'imbrication des formes et pour le placement sur une bande de tissu. La suite du travail est de trouver un algorithme qui soit capable de placer sur plusieurs bandes et capable de s'adapter à un environnement changeant, par exemple, en fonction des commandes. La première phase contient les améliorations des algorithmes existants, la seconde phase concerne l'approche

évolutionniste. Le but est de pouvoir utiliser ou combiner les différentes stratégies de recherche qu'offrent ces méthodes pour effectuer une recherche de solution de façon optimale.

0.1.1.3. Les recherches heuristiques en arbre

Dans cette thèse, les recherches en arbres sont des méthodes déterministes issues du fameux algorithme A*. La recherche est basée sur le principe du "meilleur d'abord" dans lequel la sélection d'un chemin à parcourir repose sur l'estimation de son coût futur. La difficulté de cette stratégie réside dans cette estimation. En effet, la condition pour que la recherche aboutisse à une solution optimale est que celle-ci soit inférieure ou égale au coût optimal. Or le coût optimal n'est pas connu à l'avance. Alors, il existe deux possibilités:

1) la première opte pour le chemin de coût toujours largement inférieur au coût optimal estimé, c'est le meilleur coût obtenu depuis le début de la recherche (il est mis à jour au fur et à mesure des nouvelles découvertes); l'utilisateur peut alors fixer un écart de tolérance par rapport à cet optimal estimé. C'est le principe de A_{ε}^{*};

2) la deuxième autorise le choix d'un chemin dont le coût surestime légèrement l'optimal sans trop s'éloigner en valeur supérieure, dont le seuil est fixé par un paramètre d; c'est la variante $R_{\delta\varepsilon}^{*}$.

Ces méthodes nécessitent une évaluation systématique de tous les noeuds à chaque étape de la recherche. Parfois, lorsqu'il n'est pas possible d'énumérer tous les états, les préférences sont orientées vers les méthodes aléatoires. Mais, ces dernières peuvent ne jamais aboutir à une solution, c'est pourquoi il est nécessaire de les *informer*. Dès lors, elles ne sont plus tout à fait aléatoires, mais "semi-aléatoire" puisqu'elles utilisent les informations enregistrées au cours de la recherche. Les méthodes semi-aléatoires ne font pas un examen systématique de toutes les possibilités à chaque étape, et leur recherche est guidée soit par une probabilité d'acceptation d'un état (*recuit simulé*), soit par le maintien d'un ensemble de points potentiels (*algorithmes évolutionnistes*).

0.1.1.4. L'algorithme du recuit simulé

Le recuit simulé peut être comparé à la méthode de descente du gradient avec possibilité de sortir des optima locaux. L'algorithme est basé sur le principe thermodynamique dont un des paramètres de contrôle est assimilé à la température. A hautes températures, les molécules ont une plus grande mobilité et peuvent occuper plusieurs configurations même non stables, au fur et à mesure que la température descend, la structure des molécules se fige peu à peu pour être définitivement bloquée dans une configuration quelconque; un nouveau réchauffement (*recuit*) permet de déformer cette configuration. Le processus de refroidissement doit être très lent afin de permettre au système de se stabiliser dans un état stable à énergie minimale. Car, si la température décroît trop rapidement, la recherche s'arrête sans avoir eu le temps de converger un quelconque changement Parfois, la recherche peut ne pas atteindre un état minimal global.

0.1.1.5. L'algorithme évolutionniste

La méthode évolutionniste est une métaphore du principe de la sélection naturelle et de la loi de survie du plus fort de Darwin. Elle utilise un ensemble (ou *population*) de solutions, la loi de sélection, et les opérateurs de transformation qui sont, entre autres, le croisement et la mutation. Une solution au problème est *codée* en terme de *chromosome*, appelé parfois aussi *génotype* dont le décodage donne le *phénotype*. Les chromosomes sont croisés entre eux ou mutés pour générer d'autres chromosomes. Pour éviter la génération de solutions invalides, l'idée est d'incorporer dans les opérateurs de croisement ou de mutations des contraintes ou des heuristiques.

Pourquoi utiliser les algorithmes évolutionnistes?

La réponse à cette question n'est pas facile. Bien qu'il y ait plusieurs expériences dans divers domaines, la théorie des algorithmes évolutionnistes n'offre pas de réponse concluante. On peut cependant citer la principales caractéristiques qui les distingue des méthodes classiques, notamment le **parallélisme** et la sélection naturelle. A chaque itération, les points candidats (solutions potentielles) soumis à la sélection naturelle aléatoire, ou semi-aléatoire évoluent "en parallèle", ce qui permet une recherche globale et distribuée sur tout l'espace de recherche. La population constitue une base de données compacte qui résume toute l'information acquise par la recherche jusqu'à la génération considérée. On utilise les algorithmes évolutionnistes lorsque les méthodes d'optimisation classiques ne permettent pas d'obtenir de bons résultats.

0.1.2. Organisation de ce Document

L'objectif de cette étude étant de définir les heuristiques applicables au problème de placement; l'accent sera mis plus sur l'aspect algorithmique des méthodes présentées. Ce document est divisé en cinq grandes parties : une partie introductive au contexte du problème, trois parties sur les trois types d'algorithmes utilisés et une partie conclusion. Cette organisation vise à faciliter la lecture lorsqu'on ne s'intéresse qu'à un algorithme particulier de ce document.

Ainsi, le chapitre 1 sera consacré à la présentation du problème de placement et à situer le contexte de ce travail. Ce chapitre inclut l'état de l'art, la formulation du problème, la fonction objectif le et le codage des formes utilisé. Les collections de formes utilisées pour tester les performances des différents algorithmes sont également introduites dans ce chapitre.

Les chapitres 2 et 3 sont consacrés aux méthodes de recherche heuristique en arbre A^*, A^*_ε, $R^*_{\delta\varepsilon}$ et le recuit simulé respectivement. Ces méthodes étant des reprises des travaux de G. Roussel [Roussel94], nous nous contentons de rappeler certains détails théoriques et ensuite nous décrivons les améliorations apportées. Les résultats de l'application de ces algorithmes sont donnés en fin de chaque chapitre. L'intérêt de cette reprise permet d'observer le comportement des anciens algorithmes pour des placements non limités à une seule bande.

Le chapitre 4 est consacré aux algorithmes évolutionnistes. La première partie de ce chapitre est une revue des algorithmes génétiques. La deuxième partie décrit notre approche pour implanter ces principes de recherche à la résolution d'un problème de placement. Deux approches ont été testées. La première utilise les peignes de contour en tant que *chromosome*. La population est donc constituée de formes simples et composées qui peuvent évoluer par mutation ou par croisement. Cette approche est très proche d'un recuit simulé basé sur une population de points de recherche. L'inconvénient réside dans l'aspect fortement contraint (*épistatique*) du code utilisé lors des opérations de croisement, car l'échange de matériel génétique se traduit par l'échange de peignes. La deuxième méthode vise donc à réduire ce caractère épistatique du premier codage en se basant sur le *codage arborescent* dans lequel le chromosome est constitué d'un ensemble de bandes de placement. Ainsi, les effets des opérateurs se situent à un niveau supérieur à la précédente. La méthode revient à considérer deux niveaux de résolution: le niveau des peignes et le niveau du regroupement des formes.

Après cette phase de description, nous donnons des résultats de leur application au placement. Les résultats montrent les comportements des algorithmes en fonctions des paramètres. Par manque de temps, nous n'avions pas pu étendre l'étude sur tous les carnets de tests comme dans les chapitres précédents. Toutefois, dans ce chapitre, nous montrons la faisabilité d'une nouvelle approche à la résolution d'un problème de placement en utilisant les principes évolutionnistes. La dernière partie concerne la conclusion et les perspectives où nous discutons sur le comportement général et l'efficacité des algorithmes utilisés ainsi que leur évolution possible vers une stratégie combinant leur potentialité de recherche à chaque stade de la résolution.

Chapitre I:

Introduction au Problème de Placement

TABLE DE MATIÈRES

I.1. Recherche et Développement en Textile 9

I.1.1. Efforts Déployés 9

I.1.2. Nouveaux Objectifs 10

I.1.3. Cellule Flexible de Préparation à la Confection 11

I.2. Problèmes Autour du Problème de Placement 15

I.2.1. Problème du Carnet de Commande 15

I.2.2. Problème de Coupe et Contraintes de Placement 15

I.3. Différents Types de Placement 18

I.3.1. Placement Interactif 18

I.3.2. Placement Automatique 18

I.4. Complexité du Problème de Placement 19

I.4.1. Réduction de la Complexité 19

I.5. Définition du Problème de Placement par Bande 21

I.5.1. Fonction Objectif 22

I.6. Le problème de placement dans la littérature 23

I.6.1. Programmation Linéaire 24

I.6.2. Méthodes Heuristiques 24

I.6.3. Recherche par Enumération en Arbre 26

I.6.4. Recherche Stochastique par le Recuit Simulé 28

I.6.5. Méthodes Evolutionnistes 30

I.7. Données pour Tester les Algorithmes 34

I.7.1. Carnets de Commande 34

I.7.2. Format d'un contour 36

I.8. Conclusion 38

I.9. Références Bibliographiques 39

CHAPITRE I

Introduction au Problème de Placement

I.1. Recherche et Développement en Textile

I.1.1. Efforts Déployés

La recherche et le développement dans l'industrie textile et la confection sont essentiellement motivés par le besoin des pays industrialisés de se protéger contre la concurrence utilisant la main d'oeuvre à bon marché. Elle est caractérisée par une multitude d'exigences conflictuelles: accroître la flexibilité et la qualité du produit tout en diminuant le temps de fabrication et les délais de livraison. En outre, les coûts liés aux encours, à la non-qualité, à l'utilisation de la matière, à la main d'œuvre et au fonctionnement de l'atelier ne sont pas négligeables. Dans un contexte de "juste à temps", de nouveaux critères d'optimisation exigent un équilibre optimal entre les degrés de flexibilité, le niveau de qualité, le volume de déchets acceptable et le coût de main d'oeuvre. De plus, l'implantation des systèmes de gestion, de manipulation de matière, de découpe et de gestion de personnel etc., était jusqu'à présent plus basée sur le savoir-faire et le sens d'organisation que sur l'utilisation de nouvelles technologies.

Des recherches et développements dans ce sens ont été sérieusement menés au Japon, aux USA et en Europe dans le but de permettre à l'industrie textile de déployer tous les moyens techniques pour satisfaire les nouvelles exigences du marché. Par exemple, en 1982 au Japon, le projet TRAASS[1] s'est vu doté d'un budget de plus de 10 millions de yens par le MITI pour le

[1] Technology Research Association for Automated Sewing Systems.

développement de la confection automatique [**Hewit95**] et en 1993, de nouveaux objectifs de recherche en textile ont été également lancés [**Byrne95**]. Aux USA, l'accord entre le gouvernement et AMTEX[2] a permis une subvention de $25 millions à plusieurs centres de recherche textile dont *Harvard Center for Textile and Apparel Research*[3] [**Milenkovic91**], *National Apparel Technology Center*, Caroline du Nord[4]. L'idée d'origine était d'utiliser des ordinateurs et des robots pour automatiser les opérations complexes de la confection mais depuis 1993, certains aspects, trop complexes, ont été abandonnés pour d'autres tâches plus simples telles que le placement, la découpe et la couture. En Europe, la section "Matières flexibles" du programme BRITE a également accordé des millions d'ECUs aux projets de confection automatique de vêtements [**Delgrange93**, **Byrne95**]. Et des collaborations universités-industries ont été aussi à l'origine de nombreux projets de productique [**Soenen77**, **Vidal85**, **Okat91**, **Roussel94**].

I.1.2. Nouveaux Objectifs

Les efforts déployés dans les années 1980 n'ont pas été inutiles puisque des progrès ont été apportés surtout au niveau de l'utilisation de l'informatique pour la gradation, le placement des patrons, la découpe de matière et le transport des matières entre stations de travail. Ces nouvelles méthodes ont beaucoup contribué à la diminution des prix de revient. Notamment, les systèmes DAO ont permis de réduire les pertes de matières. Dans beaucoup d'entreprises les pertes de tissu ne sont pas contrôlées rigoureusement, alors qu'elles peuvent représenter jusqu'à 25% de dépenses sur la matière [**Laplante89**]. On estime qu'une diminution des pertes de 2% peut faire augmenter de 10% le bénéfice de l'entreprise [***Confection91***]. La recherche a donc été particulièrement active dans l'intégration du système de DAO directement dans la fabrication "en ligne" de vêtements [**Draou86**, **Delaporte89**, **Jayaraman95**].

Il est évident que l'impact de la nouvelle technologie textile a été surtout profitable aux grandes entreprises aux dépens des plus petites. Il a été observé que pour une petite entreprise de 700 milles ECUs de chiffres d'affaire, un retour d'investissement n'est qu'en moyenne de 0.2%, contre 15% pour une grosse entreprise de plus de 70 millions ECUs de CA [**Byrne95**]. Pour diminuer les charges salariales qui peuvent représenter jusqu'à 14% du budget de l'entreprise, il est donc plus intéressant de distribuer certaines opérations simples (assemblage et

---

[2] AMTEX: *American Textile*, organisation qui finance des R&D en industrie textile américaine, http://amtex.sandia.gov/

[3] *Harvard Center for Textile and Apparel Research* (HCTAR), http://www.people.hbs.edu/sgarvin/HCTAR/

[4] *National Textile Center*, http://ntc.tx.ncsu.edu/html/

couture) dans les entreprises "satellites" à coût de main d'oeuvre intéressant. Au centre de ces satellites, sont réalisées les tâches plus difficiles, la création, la gradation, le placement et la découpe de façon intense et en utilisant les équipements les plus modernes.

Actuellement, même si les gouvernements ont changé de priorités industrielles, les entreprises textiles continuent leur recherche, soit en développant leur propre unité de recherche et développement, soit en finançant des projets avec des centres de recherche[5]. Par exemple, *Lectra Systems*[6] a consacré un budget de R&D de plus de 50 millions de francs en CFAO et "*plus d'une centaine d'années-hommes de développement*". En bref, la clé de la compétition entre les entreprises textiles actuelles est de maîtriser la technologie *mécatronique*[7][**Taylor95**], visant à contribuer à la meilleure compréhension et implantation des systèmes flexibles utilisant les autoroutes de l'information.

I.1.3. Cellule Flexible de Préparation à la Confection

C'est dans cet élan d'automatisation de la confection qu'a été conçu le projet d'atelier flexible au Centre d'Automatique de Lille. Cette structure propose d'utiliser un système de découpe en continu sur pli unique au lieu de la technique classique de coupe de *matelas*[8]. La découpe en continue sur pli unique permet d'économiser la matière en évitant les zones de défauts et de gagner du temps dans le cas de placement à raccords. Une telle idée nécessite une conception différente du chargement de la matière et du mode de découpe. Ceci a abouti à un projet portant sur la réalisation d'un atelier expérimental de découpe de vêtements, appelé "*Cellule flexible de coupe*" (Figure I. 1). Le schéma des flux de données entre les différentes composantes est donné à la Figure I. 2. L'étude des problèmes théoriques soulevés par cette structure est décrite dans [**Vidal85, Maouche92, Maouche95**].

---

[5] *TEXTILE NESTING*, GMD-SCAI, supporté par l'Association de Recherche Allemande (DFG) et en partenariat avec *Lectra*, http://www.gmd.de/SCAI/alg/nesting/tnest_main.html

[6] *Lectra systems*, http://www.lectra.com

[7] La mécatronique est un nouveau terme pour désigner une technologie combinant la mécanique, l'électronique et l'informatique dans un même processus de fabrication [Hewit95].

[8] Le *matelas* est faite de plusieurs couches de tissus de dimensions bornées sur lesquelles est posé un gabarit de placement suivant lequel se fera la découpe. Mais cette découpe produit une chute importante de matière. En effet, les zones de défauts sont généralement supprimées par élimination d'une bande entière de tissu ou par remplacement de la pièce défectueuse.

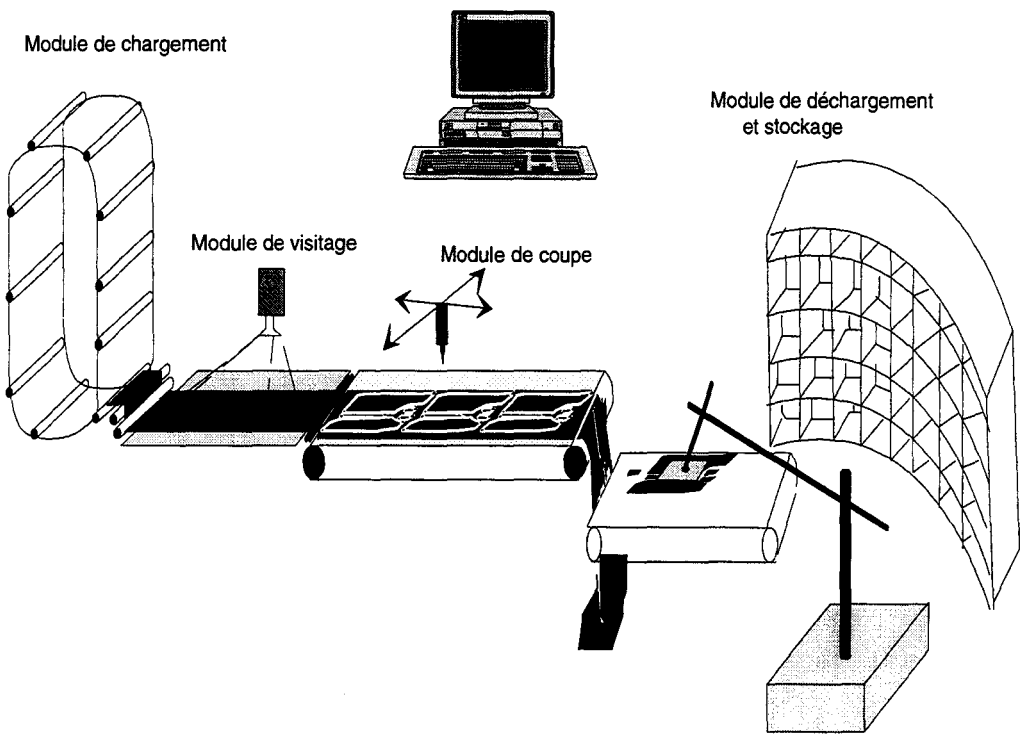


Figure I. 1: Cellule flexible de coupe. L'automatisation se trouve en amont de la confection, c-à-d, les opérations préalables à l'assemblage, au piquage et à la finition du vêtement.

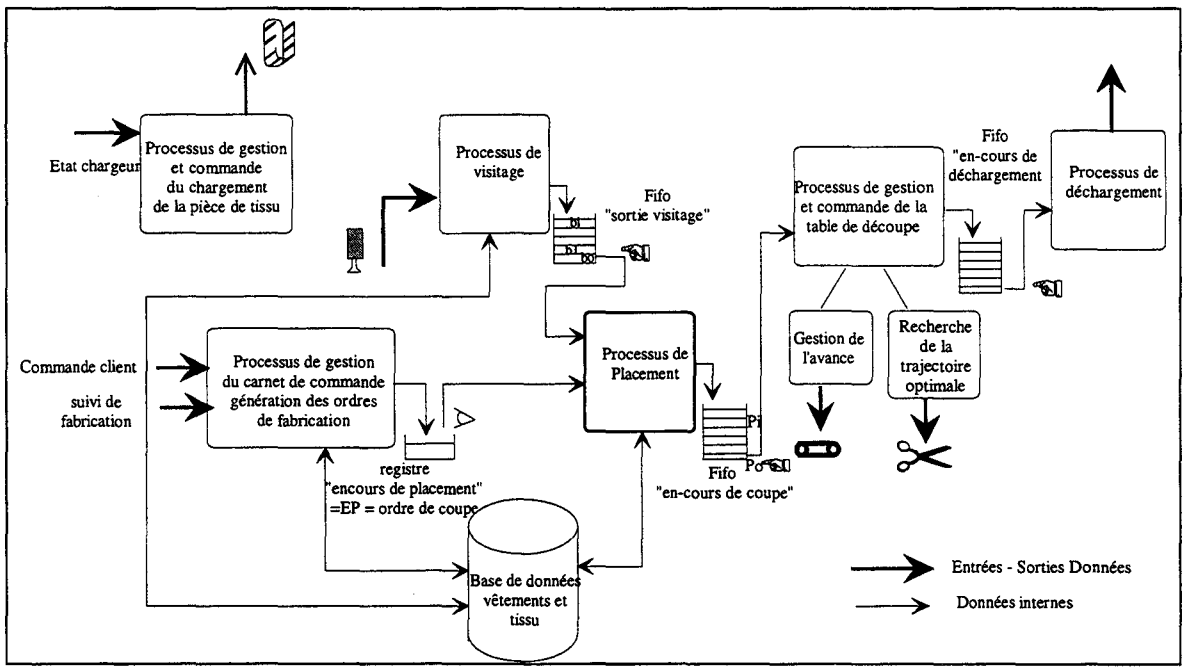


Figure I. 2: Exemple d'une architecture FAO avec les flux de données entre différentes composantes d'une cellule flexible [Roussel94].

I.1.3.1. Les composantes de la cellule flexible

L'intégration de l'atelier flexible fait intervenir les modules suivants : le chargement de la matière, la détection des défauts, le placement automatique, la découpe, et le déchargement de pièces.

I.1.3.1.1. Chargement de la matière

Pour faire une découpe en continue sur une seule épaisseur de la matière, la partie de chargement de tissus doit supporter un système de déroulement et d'enroulement de la matière. Il est constitué d'une structure métallique sur laquelle est disposé un ensemble de rouleaux de tissu. Le choix du rouleau de tissu à engager sur la table de découpe est réalisé par le processus de gestion d'ordres de fabrication.

I.1.3.1.2. Détection de défauts

La détection de défauts est réalisée grâce à un système de traitement d'image, capable de réaliser l'acquisition de la largeur du tissu (ou laize) et de détecter la position de défauts à éviter.

I.1.3.1.3. Découpe

Ce module pilote l'outil de coupe par *contournage* (laser ou jet d'eau). Les informations qu'il utilise sont issues du module de placement. En outre, il a accès à toutes les données que le module de placement utilise, en particulier les coordonnées de tous les points du contour de chaque forme placée.

I.1.3.1.4. Placement automatique

L'unité de placement utilise les informations issues d'une bibliothèque de formes dont les références sont fournies par l'ordre de coupe. L'algorithme de placement doit être en mesure de fournir le placement d'un sous-ensemble de ces formes. Il cherche à optimiser une fonction objectif visant à minimiser les critères tels que la perte en matière première et le temps de calcul.

I.1.3.1.5. Déchargement de pièces

Après la coupe, les objets complètement découpés doivent être évacués du tapis roulant puis stockés dans le magasin d'encours à l'assemblage. Un robot manipulateur doit intervenir de façon autonome pour cette tâche de déchargement. Son travail est rendu difficile par la nature souple du tissu, et la technique utilisée pour le préhenseur est capitale vis-à-vis de l'efficacité de la manipulation. Les pièces peuvent être ensuite convoyées vers un système d'assemblage automatique. Mais l'automatisation de ce dernier est encore trop complexe et son fonctionnement reste encore expérimental [**Taylor95**].

Comme nous avons discuté dans l'introduction, l'automatisation complète d'une cellule de confection s'est avérée beaucoup plus difficile que ce qui a été prévu. Cependant, certains aspects de cette automatisation ont été bénéfiques et restent un domaine actif de recherche, notamment en technologie CAO/DAO qui commence à acquérir une place importante dans l'industrie de confection. De plus, il est toujours souhaitable d'avoir un système automatique de placement, car, on estime qu'un opérateur humain requiert une durée de six mois à un an pour fournir de bons placements [**Milenkovic91**]. Dans une grande entreprise, on a parfois besoin d'une douzaine de stations CAO/DAO travaillant 24 heures sur 24. Il est alors nécessaire d'avoir des algorithmes de placement efficaces pour une production "juste à temps" ou lorsque les données, telles que le style demandé, le modèle à produire, la quantité dans chaque taille, la référence du tissu, la priorité de lancement etc., arrivent de manière imprévisible.

I.2. Problèmes Autour du Problème de Placement

Le problème du placement est lié au problème de découpe. Le processus physique de découpe peut exiger une présentation de la matière sous forme d'unité de dimensions compatibles comme c'est le cas en confection où la longueur est bornée par celle de la table de découpe et le nombre de pièces défini à l'avance. Le problème de découpe se présente alors sous forme de deux problèmes distincts: un problème de placement et un problème du carnet de commande.

I.2.1. Problème du Carnet de Commande

Le problème du carnet de commande consiste à satisfaire l'ensemble des demandes connues à l'avance et trouver un ensemble d'amalgames pour les satisfaire. Un amalgame réside dans la manière de découper ou de remplir une unité de matière première. Le problème s'énonce alors: " étant donné un ensemble d'unités de matière première, combien de fois faut-il en utiliser pour satisfaire le carnet de commande?". En confection, on utilise la répartition des fréquences par taille pour déterminer la combinaison des tailles à produire et satisfaire au mieux le carnet de commande. C'est l'unité de base du carnet de commande. Le problème consiste alors à déterminer le nombre de fois à répéter ce placement [**Gilmore65**, **Prempti83**, **Dowsland95**].

I.2.2. Problème de Coupe et Contraintes de Placement

Les contraintes proviennent principalement de la technologie utilisée pour la découpe et de la stratégie de résolution du problème. Elles peuvent être classées en trois groupes : *contrainte d'admissibilité de placement*, *contraintes technologiques* et *contraintes pour l'amélioration de gain.* Les deux premières sont nommées *contraintes impératives* et la dernière *contrainte de préférence* [**Roussel94**, **Maouche95**].

I.2.2.1. Contraintes d'outils de coupe/ Optimisation de trajectoire de découpe

La manière de séparer une pièce de son support détermine le type de découpe et constitue une contrainte de placement. La découpe dite *en guillotine* est utilisée pour la découpe de formes rectangulaires. La principale contrainte est d'avoir des lignes parallèles [**Fayard95**, **Daza95**].

Lorsque les pièces sont de formes irrégulières quelconques, la découpe s'effectue par "contournage" ou par suivi de contour. L'espace entre les formes doit être suffisamment large pour faciliter le suivi de contour ainsi que la *valeur de couture* pour l'assemblage des pièces (Figure

I. 3). Cette contrainte peut être résolue par transformation morphologique ou par dilatation de formes **[Li94]**. Il est également possible d'intégrer cette contrainte dans la définition des gabarits (modèles de formes), ce qui facilite la résolution du problème.

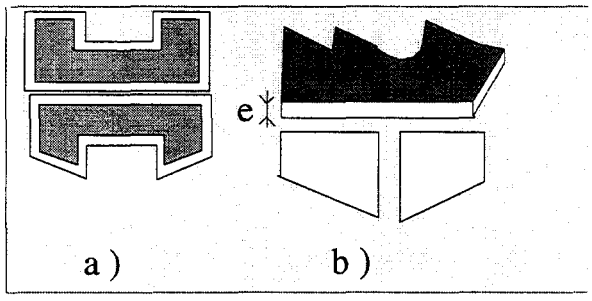


Figure I. 3 : a) Espace entre les formes pour la découpe, b) valeur de couture.

En outre, il faut veiller aussi à ce que les courbures ne touchent les droites ou les angles pointus ou que les angles ne touchent le bord de la matière **[Milenkovic91]**.

Il faut noter aussi que le suivi de contour induit aussi un problème d'optimisation de la trajectoire de l'outil de coupe. Une trajectoire est composée par un ensemble de passages obligés (découpe) et de liaisons entre les passages obligés (passage à vide). L'espace de travail de l'outil est généralement restreint, le plus souvent, il est représenté par une fenêtre de placement. La découpe d'une pièce s'effectue alors en plusieurs étapes **[Okat91]**.

I.2.2.2. Contraintes impératives

Les contraintes impératives constituent des conditions strictes pour obtenir des placements admissibles. Elles concernent le *positionnement* (non-superposition et non-débordement de formes, orientations, ou ajustement des carreaux) et la *qualité*. Dans la plupart des algorithmes d'optimisation, ces contraintes sont prises en compte par les fonctions de pénalité qui agissent sur la fonction objectif.

I.2.2.3. Contraintes de préférence ou de qualité

Les contraintes de préférence permettent de définir la qualité du placement. Leur satisfaction n'est pas toujours facile à atteindre, et un compromis est souvent nécessaire.

Ces contraintes varient selon la matière utilisée. Par exemple, pour les matières à motifs, il est préférable d'aligner les motifs selon les spécifications du styliste. Certaines matières possèdent également des caractéristiques d'aspect et de texture différents, il est alors nécessaire de limiter certaines transformations géométriques telles que les symétries, les rotations etc. Cette contrainte n'est pas toujours impérative sur les matières uniformes. Dans ce dernier cas, un écart angulaire de quelques degrés peut être toléré si celui-ci permet d'obtenir un meilleur rendement. De même, si le modèle le permet, certaines formes sont décomposables en plusieurs formes simples qui peuvent être placées à différents endroits dans la surface du support (Figure I. 4).

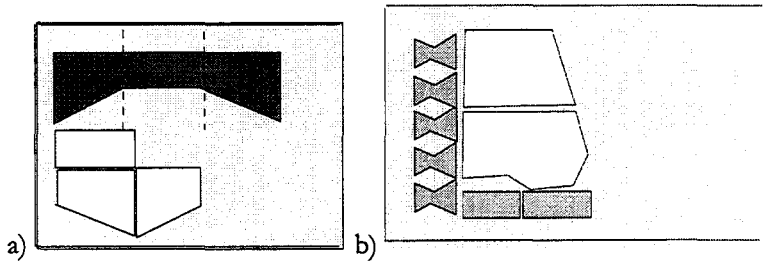


Figure I. 4 : a) Exemple de relaxation de contrainte : décomposition d'une forme en plusieurs parts pour obtenir un placement plus compact. b) Pour optimiser temps de découpe, essayer d'aligner les formes le plus possible.

Une autre contrainte de préférence est de permettre de minimiser le temps de découpe. C'est-à-dire, essayer d'aligner les bordures le plus possible afin de minimiser le nombre de détours de l'outil de coupe; les pièces doivent avoir une ligne commune de découpe (ou bien espacées selon l'épaisseur de l'outil). Le mieux est de s'approcher le plus possible de la coupe en guillotine. On peut aussi disposer des petites pièces ensemble (Figure I. 4.b).

I.2.2.4. Contrainte temporelle

Si la contrainte géométrique réduit l'espace des configurations admissibles et donc simplifie la résolution, la présence d'une contrainte temporelle ajoute une difficulté d'ordre algorithmique à la résolution. Deux comportements sont possibles vis-à-vis de la prise en compte de la contrainte temporelle:

- L'algorithme cherche une solution sous optimale de qualité moyenne correcte en un temps quasiment constant par rapport à un contexte habituel.

- L'algorithme cherche une solution évolutive et il peut être interrompu dès qu'une sollicitation extérieure apparaît. Chaque solution intermédiaire peut être améliorée ou constituer la solution finale.

Peu d'études font intervenir explicitement le temps dans les algorithmes. Il faut noter que dans les systèmes de CAO/FAO actuels, certaines contraintes ne sont pas toujours respectées à cause de la complexité de leur modélisation d'une part et d'autre part elles ralentissent la recherche. Le système nécessite alors l'intervention d'un placeur expert pour vérifier et améliorer la solution proposée.

I.3. Différents Types de Placement

On peut rencontrer dans l'industrie de confection deux types de placement: le placement interactif et le placement (semi-) automatique

I.3.1. Placement Interactif

Dans la mémoire du calculateur, sont stockées les images des parties de vêtement pour chacune des tailles des différents modèles de la collection. Le placeur a des informations sur les approvisionnements disponibles. Si les pièces de tissu ont été systématiquement inspectées et mesurées par un analyseur de laize, les mesures précises sont alors introduites. Elles deviennent l'une des contraintes du placement. Le placeur connaît le programme de fabrication et les références, les tailles du modèle qu'il peut envisager de combiner pour le placement. Par l'intermédiaire d'une tablette graphique, il affecte une position à chaque élément sur le rectangle représentant la pièce. Le système lui indique en permanence le taux d'utilisation de la surface totale. Quand il a estimé avoir atteint un taux satisfaisant, il sauvegarde en mémoire et passe au placement suivant.

I.3.2. Placement Automatique

Il existe deux types de placement automatique.

1) Le système fournit un placement définitif en un temps plus ou moins long, et avec un rendement plus ou moins bon. Le résultat est ensuite stocké dans une bibliothèque pour être exploité ultérieurement lors du lancement de la fabrication. Au cours de la

journée, on essaie d'autres placements différents afin d'enrichir la bibliothèque ou d'améliorer les placements pour une commande donnée.

2) Le système fournit un placement en temps réel pour une commande donnée. Le résultat est enregistré sur un fichier qui est transmis instantanément à la table de découpe et permet de commander directement l'outil de coupe. Ce type de fonctionnement convient essentiellement à l'organisation d'un système de production à cycle court, dans lequel les commandes sont créées et enregistrées en permanence pour des fabrications en séries très courtes. Tous les problèmes liés à cette structure ne sont pas encore bien résolus (qualité des placements, vitesse d'obtention, prise en compte des motifs et des défauts).

Il faut noter que lorsque le nombre de pièces à produire n'est pas connu à l'avance, le problème de découpe se présente comme une suite de problèmes de placement. On parlera alors de "placement continu".

I.4. Complexité du Problème de Placement

Dans sa forme la plus générale, le problème de placement est *NP-difficile*. C'est-à-dire qu'il n'existe aucun algorithme polynomial pour le résoudre. Si $NP \neq P$, l'algorithme de recherche est exponentiel. Et si $NP = P$, il peut exister un algorithme de recherche d'ordre polynomial pour trouver la solution optimale, si celle-ci existe.

L'analyse de complexité dans [**Fowler81**] montre que les problèmes de placement tels que le placement de rectangles et le problème de sac-à-dos sont *NP-complets*, même dans le cas restreint où tous les objets sont des rectangles identiques.

Dans [**Daniels95**], on montre que, si la rotation est interdite et que seule la translation est permise, le problème de placement de polygones convexes est *NP*-complet.

En général, le problème d'optimisation de placement est reconnu comme *NP-difficile*, c'est-à-dire, ses algorithmes de résolution sont d'ordre exponentiel. Dans [**Pargas93**], on montre que des algorithmes de placement par approximation de qualité moyenne, peuvent fournir une solution seulement à environ 22% de l'optimal .

I.4.1. Réduction de la Complexité

Pour réduire la complexité, plusieurs méthodes ont été utilisées. Les méthodes heuristiques souvent basées sur le savoir et l'observation d'un placeur expert humain. Ce sont par exemple, l'approche par taille de formes [**Milenkovic91, Li94, Daniels95**], et l'approche par indice d'imbrication[**Soenen77, Maouche95**]. La première consiste à placer d'abord des grandes

formes et ensuite placer des petites formes dans les espaces laissés entre les formes placées. La deuxième approche consiste à assembler les formes sur le côté où la surface d'imbrication est maximale.

Il existe également une approche par énumération implantée en graphe. A chaque noeud du graphe, toutes les combinaisons possibles sont énumérées. A cause des limitations, il n'est pas toujours possibles de les stocker et de les exploiter toutes. Dans ce cas, soit on limite les noeuds à exploiter à chaque étape par une heuristique (Recherche heuristique en arbre) soit on utilise la méthode stochastique (le recuit simulé et la méthodes évolutionniste). Ces approches seront décrites plus en détail dans la section "état de l'art" (section I.6).

Notre approche pour réduire la complexité est de limiter l'espace de placement. Cela consiste à partager le problème en sous-problèmes disjoints de taille réduite et résoudre les sous-problèmes par une méthode appropriée. L'ensemble des solutions des sous-problèmes constitue la solution au problème. La recherche se ramène alors sur un seul sous problème à la fois. De plus, les contraintes temporelles et les choix technologiques exposés dans la section décrivant la cellule flexible de coupe (section I.1.3), nous permettent de diviser l'ordre de coupe. On ne réalisera pas un seul placement U optimal, mais une succession de sous-placements U_n (eux-mêmes sous-optimaux) que l'on appellera par convention: ***bande***.

I.4.1.1. Placement en bande

Pour que la subdivision soit possible, chaque sous-problème doit répondre à certaines propriétés de finitude et de consistance **[Roussel94]**, par exemple la limite gauche d'une bande correspond au début de la matière s'il s'agit de la première bande, ou à la fin de la bande précédente pour les bandes suivantes. Comme une bande doit pouvoir contenir les plus longues formes, la longueur sera donc au moins égale au maximum des longueurs des formes non placées. D'autre part, un décalage longitudinal (un facteur α fixé par l'utilisateur) est autorisé pour permettre parfois une imbrication très rentable. Typiquement, α varie de 1 à 5% de la longueur de la plus longue des formes de la bande considérée. On observe qu'une valeur élevée peut conduire d'une part, à obtenir un front de placement très irrégulier qui éloigne la bande de la forme générale rectangulaire; d'autre part, elle diminue l'effet du placement en bande réduite et augmente potentiellement le nombre de formes incluses dans la bande et donc aussi la complexité de la recherche.

Ainsi, on peut essayer de réduire la complexité du problème en le partageant en sous-problèmes. Mais il est souhaitable que les solutions sous-optimales des ces sous-problèmes restent globalement acceptables et que des contraintes soient satisfaites de façon optimale.

I.5. Définition du Problème de Placement par Bande

Le problème du placement consiste à rechercher le meilleur amalgame au sens des objectifs du placement compte tenu des pièces à découper. Ce problème constitue la partie la plus importante du problème de découpe: on ne peut le résoudre efficacement sans résoudre efficacement le problème de placement. L'indice de performance de la découpe est déterminé par celui du placement et les contraintes spécifiques de découpe sont prises en compte au niveau du placement. On peut dire aussi que la diversité des applications et la multiplicité des contraintes n'ont pas encore permis la formalisation du problème sous une forme unique.

Pour notre cas, la définition suivante est proposée :

Etant donné un ensemble U de formes F_k, trouver une configuration optimale sans superposition de toutes les F_k dans un rectangle défini par la largeur W et de longueur L_{max} qui doit être aussi minimisée (Figure I. 5).

La feuille de matière que nous appellerons *surface de support*, est supposée non limitée en longueur. Ce qui signifie qu'on doit trouver une partition de U en N sous ensembles disjoints U_1, U_2, ... U_n, ..., U_N tels que la somme de leurs surfaces soit minimale. Typiquement, la largeur W est de 1,50m , le nombre de pièces pour un vêtement se situe entre 10 et 20. Le rendement du placement manuel pour un seul vêtement varie entre 75% et 90%.

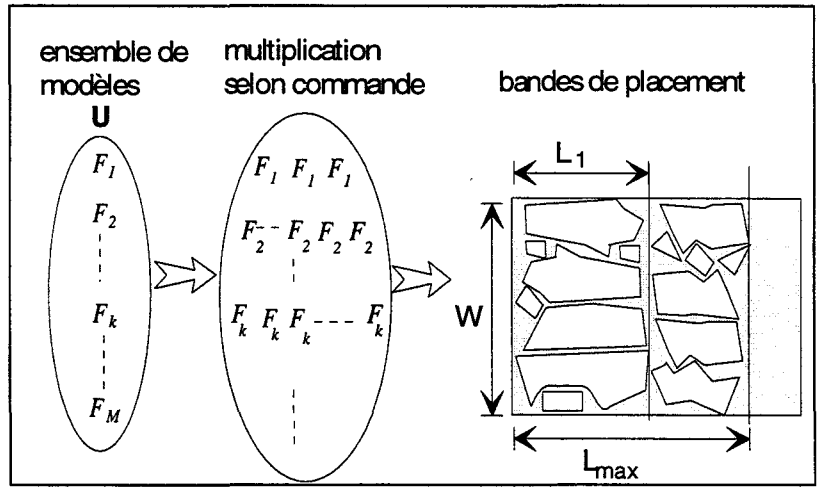


Figure I. 5 : Le problème de placement consiste à placer un ensemble de formes irrégulières, pouvant être utilisées un certain nombre de fois, dans une surface de dimension fixe, souvent rectangulaire, en optimisant l'occupation de la surface et en respectant certaines contraintes spécifiées dans les formes ou par les conditions de découpe.

I.5.1. Fonction Objectif

Soit un ensemble de formes de départ, U, dans lequel chaque forme F_k est reproduite a_k fois pour constituer un carnet de commande. Ces formes reproduites forment un ensemble noté $\mathbf{C^U}$. Une solution *s*, du problème du placement doit satisfaire le minimum de la fonction objectif :

$$\mathbf{g}(s) = W \times \sum_{i=1..N} L_i - \sum_{U} a_k . A(F_k)$$

Equation I. 1

avec : *N*, le nombre de bandes contenues dans la solution *s*. Une bande représentée par U_n est donc un sous ensemble de $\mathbf{C^U}$.

Pour des raisons de simplification, les bandes doivent satisfaire les contraintes de disjonction des ensembles:

$$\bigcap_{n=1...N} U_n = \boldsymbol{C}^U \text{ et } \forall U_m, U_n \in C^U, m \neq n, (U_m \cap U_n) = \varnothing$$

Equation I. 2

Une bande *n* notée B_n est un attribut géométrique de l'ensemble des formes U_n. C'est un rectangle de largeur W_{B_n} et de largeur L_{B_n} définie par:

$$L_{B_n} \leq (1+\alpha) \times \max_{F_k \in U_n} (L_{F_k})$$

Equation I. 3

où L_{F_k} est la longueur de la forme F_k contenue dans la bande *n*. Le coefficient α permet de relaxer la contrainte longueur pour une bande ($\alpha \leq 0.5$). Chaque forme F_k peut être utilisée a_k fois et peut être placée en contact avec une autre forme sur chaque côté, i.e. {est, sud, ouest, nord}, noté par {1, 2, 3, 4}. Donc localement, on optimise la perte dans une bande:

$$\mathbf{g}(U_n) = W \times L_{B_n} - \sum_{F_k \in U_n} \mathbf{A}(F_k)$$

Equation I. 4

avec les conditions de non-superposition :

$$\forall F_i, F_j \in U_n, i \neq j, \; F_i \cap F_j = \varnothing$$
$$\forall U_n \subset U, W_{B_n} \leq W$$

Equation I. 5

Pour calculer la perte surfacique dans une bande, on utilise la méthode de peignes qui représentent la *déficience convexe*. La surface des trapèzes formés par des peignes donne une bonne approximation de cette déficience (cf. Section I.7.2).

L'état de l'art dans les sections suivantes donne quelques exemples d'approches adoptées dans les différents problèmes de placement. Cette revue s'intéresse surtout aux méthodes de résolution les plus récentes.

I.6. Le problème de placement dans la littérature

La génération automatique de placement et de découpe est devenu un domaine de recherche actif depuis les années '60 avec le développement des ordinateurs. Le problème se rencontre dans de nombreux domaines et la recherche évolue toujours. Plus de 700 articles scientifiques publiés ont été recensés en 1992 [**Dyckhoff92**]. Malgré cela, on remarque que la communication entre les acteurs dans ce domaine reste encore faible et par conséquent les méthodes développées pour un problème pouvant être appliquées à un autre n'ont pas beaucoup servi pour progresser et des duplications des travaux sont fréquentes [**Dowsland95**]. C'est dans cette optique qu'un "groupe d'intérêt commun" sur le problème de la découpe (*SICUP*[9]) a été créé pour favoriser cette échange d'idées entre chercheurs et industriels. Aussi, récemment, les revues de recherche opérationnelle *INFORMS* [**Martello94**] et *EJOR* [**Bischoff95**] ont consacré des numéros spéciaux à ces problèmes.

Dans [**Dyckhoff92**], un classement des approches mises en œuvre dans ce domaine a été réalisé. Dans cette étude bibliographique, les auteurs distinguent trois types de problèmes de placement selon les stratégies utilisées:

1) les pièces sont toutes prises en compte en même temps et placées directement sur la surface de la matière;

2) les pièces sont assemblées en groupes puis placées sur la matière;

[9] SICUP: *Special Interest group on Cutting and Packing*, dont l'accès est disponible sur Internet depuis 1996, http://prodlog.wiwi.uni-halle.de/sicup/non_pub/

3) le placement initial non optimal (pouvant contenir des superpositions) est à optimiser progressivement.

Ces approches peuvent être appliquées de diverses façons selon les méthodes de traitement géométriques telles que la représentation des formes et les relations spatiales entre les pièces.

Dans la partie suivante, nous allons parcourir quelques stratégies utilisées dans la littérature.

Les méthodes utilisées peuvent être classées en quatre catégories : *méthode exacte* par la *programmation linéaire, recherche heuristique en arbre* et récemment les *méthodes probabilistes* telles que le *recuit simulé* et les *algorithmes évolutionnistes.*

I.6.1. Programmation Linéaire

La modélisation par la programmation linéaire a été beaucoup utilisée dans les années '60 et '70, notamment dans les travaux de Gilmore et Gomory [**Gilmore65**] et [**Benhamamouch79**].

Cette méthode consiste à représenter le problème sous forme de modèle mathématique, contenant un système d'équations linéaires sur la fonction objective et sur des contraintes linéaires. La résolution de ce système d'équation fournit des paramètres de positionnement des formes. La résolution est possible avec les nombres réels (la méthode du simplexe). Mais lorsqu'il s'agit du nombre entier, la méthode devient difficile. On procède alors par astuces. Par exemple, Stoyan *et al.* [**Stoyan96**] intègrent dans leur modèle des surfaces non utilisées dans la surface-matière. Ensuite, ils procèdent à une énumération d'une série de placements initiaux dont chacun va converger vers un optimal local différent. Si tous les points initiaux sont énumérés et optimisés, cela aboutit à une méthode à solution exacte. Mais, avec N formes à placer, le nombre de minima locaux peut dépasser $N!$. Une telle approche n'est pas toujours applicable à un problème de taille réelle.

Alors d'autres méthodes, souvent heuristiques établies à partir des critères industriels, sont utilisées.

I.6.2. Méthodes Heuristiques

Comme nous avions déjà mentionné, ces méthodes sont basées sur le savoir-faire d'un placeur expert humain. L'observation d'un placeur expert fait apparaître deux sorte d'approche: l'approche par indice d'imbrication et l'approche par taille de formes.

I.6.2.1. Heuristique: indice de voisinage

La première approche utilisée dans le Centre d'Automatique de Lille [**Soenen77**, **Vidal85**, **Maouche95**] consiste à établir un indice de voisinage entre chaque couple de formes. Cet indice est évalué hors ligne par le calcul de la surface d'imbrication sur quelques configurations intéressantes.

Plus l'indice est élevé, plus la zone d'imbrication est grande. Les indices, classés par ordre décroissant, contribuent à la construction du placement dans lequel les formes ayant les indices les plus élevés sont placées en priorité.

I.6.2.2. Heuristique: taille des formes

La première méthode de placement utilisée par le groupe de recherche de l'université de Harvard [**Milenkovic91**] consiste à observer un placeur expert et à reproduire sa méthode. L'approche consiste à placer d'abord les grandes pièces (ou les pièces les plus difficiles à placer) en colonnes de largeur égale à celle de la plus grande pièce. Les petites pièces sont réservées pour être placées en dernier, souvent dans les espaces laissés entre les grandes pièces. Le nombre de grandes pièces par colonne permet de calculer le nombre total de colonnes nécessaires, qui ensuite permet de calculer les coordonnées précises en y des pièces. Ensuite un algorithme de compression/dilatation est appliqué à ce placement initial pour améliorer l'efficience [**Li94**]. Cet algorithme consiste à calculer un ensemble de contraintes d'interactions de chaque pièce avec les pièces qui lui sont immédiatement adjacentes (haut, bas et latéral). A chaque colonne, une programmation dynamique est utilisée pour condenser toutes les contraintes des colonnes précédentes sur la dernière colonne. Ces contraintes obtenues sont de nouveau utilisées pour générer les contraintes de la prochaine colonne. Une méthode heuristique est utilisée pour déterminer un ensemble de valeurs réelles satisfaisant ces contraintes. Ensuite on essaie de placer des petites pièces. Pour éliminer les superpositions, leur solution consiste à *dilater* les espaces entre les pièces par la somme de Minkowski. Et vice versa, pour l'élimination des "vides", l'opérateur de *compression* est utilisée (ou la combinaison des deux). Cette méthode est inspirée de celle utilisée en robotique [**Avnaim89**], où la compression consiste à simuler une *force de gravitation* sur les pièces selon une direction désirée, et la dilatation simule une *force de répulsion* pour créer les espaces entre les formes. La résolution s'effectue par une séquence de modèles de programmation linéaire. Cette méthode est très lourde en calcul, mais, les résultats montrent une amélioration effective du placement de 2 à 3%, ce qui n'est pas négligeable pour une production en quantité industrielle. Par exemple, V. Milenkovic [**Milenkovic91**] rapporte qu'avec un placement de 126 formes de pantalon, de 1m de large et de 8m de long sur un matelas de six couches de tissus, de tel pourcentage permet

une réduction en longueur du placement de 0,25%, ce qui fait économiser 7 dollars par pièce sur le prix de la matière utilisée.

I.6.2.3. Heuristique: faisabilité du placement

Dans le cas où le placement de petites pièces n'est pas toujours possible avec la méthode ci-dessus, la méthode heuristique "faisabilité du placement" est utilisée. Celle-ci consiste à développer des algorithmes de placement par translation capables d'analyser si le placement est faisable ou non, à partir d'un ensemble de formes et de support **[Daniels95]**. On ne commencera à placer les formes sur la surface de support que si la réponse est positive. L'algorithme est aussi capable de détecter des situations infaisables. L'approche globale consiste à restreindre progressivement l'espace de recherche, vérifier rapidement s'il existe une configuration à rectangle circonscrit minimal pour cet espace, et si aucune solution n'est trouvée, il faut restreindre encore cet espace et réitère le processus sur ce nouveau domaine. Certaines parties de l'algorithme de vérification utilisent la programmation linéaire. Et la programmation linéaire est aussi utilisée pour produire des placements sans superpositions sur des configurations avec superpositions. Leurs algorithmes peuvent placer jusqu'à 10 polygones non-convexes. Si le nombre de formes est élevé, l'approche à deux phases est appliquée. Cette approche consiste à placer d'abord les grandes formes puis les petites. Ces dernières peuvent être d'abord assemblées séparément. Et l'ensemble peut être ensuite placé dans les espaces vides entre les grandes formes. Ces méthodes ont été implantées sur le système *UltraMark™* de *Gerber Garment Technology*[10].

I.6.3. Recherche par Enumération en Arbre

Dans cette méthode, le problème est transformé en une recherche dans un espace d'états. Le but est de trouver un chemin optimal menant d'un état initial vers un état final qui satisfait les critères d'optimalité. Une heuristique est utilisée pour choisir à chaque étape le meilleur noeud permettant d'arriver au plus vite à cet état final. La solution est obtenue en retraçant le chemin parcouru. Dans **[Arenales95, Fayard95, Daza95]**, cette stratégie de recherche est utilisée dans un graphe ET/OU pour le problème de la découpe en guillotine. Cette méthode de découpe peut s'étendre aussi au placement de formes irrégulières. Dans ce cas, on regroupe des formes par modules rectangulaires dont l'imbrication est maximale. Ensuite on place ces modules dans un schéma global. Cette approche utilisée par Adamovicz et Albano

[10] *Gerber Garment Technologies*, http://www.ggt.com/

[**Adamovicz76**], vise à réduire la complexité de l'algorithme en limitant le nombre de formes à manipuler.

Groupées dans un rectangle, les formes ont des positions relatives fixes. Cette technique semble bien fonctionner sur les formes qui sont des "image miroirs" les unes des autres.

Dans la technique de *branch and bound,* on essaie plutôt de réduire l'espace de recherche en limitant le nombre de combinaisons à exploiter à chaque étape. Dans les travaux d'Albano et Sapuppo [**Albano80**], les pièces sont considérées comme des polygones convexes qui peuvent être placées selon plusieurs orientations. Il est clair qu'on ne peut par parcourir l'espace de recherche en un temps acceptable (même avec une taille réduite du problème). Ils utilisent alors une recherche guidée par une heuristique comportant l'évaluation de la perte courante et de l'estimation de la perte future. La branche qui possède une valeur minimale de la somme de ces deux évaluations est choisie. Et ils limitent l'orientation des pièces à une seule rotation de 180° et favorisent le remplissage à gauche. Chaque pièce génère une branche à chaque noeud de l'arbre et chaque branche est soumise à la sélection pour restreindre le nombre de branche à explorer. La sélection favorisera la branche ayant l'expansion à droite la plus faible, c'est-à-dire compacter le plus près possible de l'origine du repère. L'arbre est ainsi élagué sachant que si un *backtracking*[11] se produit, il se fera sur les noeuds loin derrière dans l'arbre pour éviter de réévaluer les noeuds ignorés à l'étape précédente. Les expériences montrent des résultats satisfaisants.

L'approche de G. Roussel [**Roussel94**] consiste à élaguer l'espace de recherche en utilisant une heuristique ε-admissible. Dans le choix du chemin à poursuivre, les états de coûts situés en dessous du seuil limite sont choisis en priorité. Un autre intérêt de cette approche, vue sous l'optique de la découpe en continu, réside dans la résolution par bande du problème du placement. En effet, dans une cellule de production, les différentes tâches participant au cycle de production peuvent évoluer en parallèle comme des processus indépendants. Chaque processus travaille à son propre rythme et doit coopérer avec les processus connexes avec qui il partage des ressources communes. Si on ajoute la contrainte temporelle, l'algorithme cherche une solution évolutive et il peut être interrompu dès qu'une sollicitation extérieure apparaît. Il peut alors fournir la solution actuellement trouvée, même si celle-ci est jugée peu satisfaisante. La complexité de l'algorithme est en $O(K \times N)^R$, avec N le nombre de formes à placer, K le nombre de placements possibles par formes et R le nombre de formes pour remplir une bande. Cette éventualité implique qu'on travaille dans un environnement changeant où l'espace de recherche est périodiquement modifié.

[11] Retour en arrière

L'avantage de la méthode des graphes d'états est la possibilité d'exploiter tous les états, puisque leur nombre est fini. Mais, comme la méthode précédente, l'inconvénient est que lorsque la taille du problème dépasse une certaine valeur, il n'est plus possible d'énumérer tous les états faute de taille de mémoire suffisante. On préférera alors les méthodes stochastiques.

I.6.4. Recherche Stochastique par le Recuit Simulé

Le recuit simulé fait partie des méthodes de descente encore appelées méthodes d'amélioration itérative. Il repose sur une analogie avec la thermodynamique (voir chapitre 3) en utilisant une double dynamique : rechercher les minima à température fixée et assurer la diminution de la température.

L'application à la résolution du problème de placement **[Prempti83]** consiste en deux étapes. La première étape est une phase de pré-traitement correspondant aux états de hautes températures favorisant le désordre. Cette étape consiste à "jeter" toutes les formes sur la surface du support de façon aléatoire avec des contraintes relaxées (faible pénalisation). A la phase d'optimisation correspondant à la phase de descente en température, on cherche à éliminer les superpositions et débordement en diminuant progressivement la température, et en augmentant au fur et à mesure le degré des pénalités. L'état de minimum atteint en basse température est caractérisé par la configuration où les superpositions et débordements sont réduits et où le niveau de contraintes est élevé de sorte que les mouvements ne sont plus possibles.

Dans **[Lutfiyya91]**, les formes sont placées selon un ensemble fixe d'orientations. Leur ensemble de solutions acceptables est défini par un ensemble des coordonnées et orientations de chaque pièce dans le placement. Les solutions non acceptables (contenant des superpositions) sont aussi incluses dans l'espace de recherche mais pénalisées par un facteur de pénalité. La pénalité est appliquée en multipliant la surface estimée de superposition ce par ce facteur. La fonction objectif assure aussi que tous les éléments soient placés le plus près possible de l'origine. Les opérateurs de transformation consistent à déplacer une pièce dans un proche voisinage, changer son orientation ou échanger des positions entre deux pièces. L'utilisateur spécifie en paramètres d'entrée les probabilités d'application de chaque opérateur. Dans cet algorithme, il n'y a pas de restrictions sur les dimensions du support, ni en largeur , ni en longueur. Alors que dans un problème de découpe réel, il y a des contraintes sur au moins une dimension du support.

Des formulations plus proches des applications pratiques sont proposées dans **[Oliveira93, Heckmann95]** où la longueur est à minimiser et la largeur reste souvent fixe. Les superpositions sont permises et la fonction coût est une combinaison linéaire de la

longueur utilisée et les superpositions sont pénalisées par un terme de pénalité fixé par l'utilisateur.

La première étape de l'approche précitée [**Oliveira93, Heckmann95**] consiste à définir une approximation grossière des contours des polygones. Ceci consiste à réduire le nombre de sommets et éliminer certaines concavités "profondes" inexploitables, par approximation polygonale ou par grille de discrétisation. Les pièces sont ensuite codées avec des paramètres permettant d'avoir une forme approximative de la forme modèle initiale. La résolution par le recuit simulé commence donc avec ces approximations. Au fur et à mesure que la températures décroît, la précision des approximations des formes s'accroît pour ainsi éliminer les superpositions. La qualité de la solution obtenue dépend du degré de précision du codage des formes ainsi que de la méthode du codage elle-même. Par exemple, le codage du contour par polygone peut être plus rapide au détriment de la précision alors que le codage du contour discret permet d'obtenir des placements avec moins de superpositions mais nécessite plus de temps de calcul selon le degré de précision de la discrétisation.

En général, l'efficacité du recuit simulé réside d'une part dans la définition de la fonction coût et des différents choix des paramètres tels que température initiale, loi de décroissance de température, nombre d'itérations par palier de température pour atteindre une état d'équilibre intermédiaire) et la définition de la configuration. Cette dernière est la difficulté principale dans l'implantation de l'algorithme : une configuration est un état du placement où toutes les formes sont placées, contenant ou non des superpositions. Ce principe de placement consistant à améliorer un placement initial quelconque n'est pas toujours aisé car les superpositions sont parfois difficiles à éliminer à cause des positions connexes qui font propager l'effet de chaque mouvement. Il est toujours possible d'ajouter d'autres facteurs de pénalité, mais trouver le bon facteur de pénalité pour chaque mouvement est un autre problème.

C'est pourquoi cette méthode peut être très longue selon les configurations trouvées. Les résultats obtenus dans [**Heckmann95**] sont à 4% en moyenne en dessous de la performance humaine, mais le temps de calcul reste encore très élevé (de l'ordre de 3h34min en moyenne pour placer entre 15 à 25 formes, contre 15 à 30 min. pour l'homme).

Le recuit simulé est une recherche assez aléatoire qui ne progresse que par une seule solution à la fois, par conséquent, il ne permet pas d'explorer le domaine globale de façon effective, bien qu'à chaque fois une meilleure solution trouvée soit sauvegardée.

Les techniques évolutionnistes permettent de remédier à ce problème en maintenant et évoluant une population de solutions partielles.

I.6.5. Méthodes Evolutionnistes

L'approche évolutionniste est une métaphore de la génétique de population selon la loi de l'évolution naturelle des espèces selon Darwin. En termes d'optimisation, cette évolution consiste en trois étapes suivantes:

1) population initiale d'individus codés en chaînes dites *chromosomes*,

2) évaluation de la force (dite *fitness*) des individus,

3) génération d'une nouvelle population par les opérateurs génétiques tels que le croisement et la mutation.

Ces trois étapes sont réitérées à travers d'un certain nombre de générations décidé par l'utilisateur. L'efficacité de ces algorithmes est très dépendante de l'encodage des candidats à la solution et des opérateurs génétiques qui modifient ce code.

Pour mieux situer l'approche des différentes méthodes génétiques, on peut distinguer trois types de codage :

1. *Codage direct* : Il y a correspondance biunivoque entre la représentation de la solution et la solution elle-même. L'espace *génotypique*[12] et l'espace *phénotypique*[13] sont confondus.
2. *Codage indirect* : Le code contient des indicateurs permettant de générer la solution. L'espace génotypique est différent de l'espace phénotypique. Ce code nécessite un algorithme de décodage.
3. *Codage mixte* : c'est la combinaison des deux précédents. C'est-à-dire que le code contient une partie descriptive de la solution mais incomplète, l'autre partie contient les indicateurs permettant de générer le reste de la solution.

Dans la partie suivante, nous présentons les méthodes publiées dans la littérature pour appréhender le problème du placement en général. Jusqu'à présent, l'application de ces méthodes dans ce domaine est encore très peu répandue et les premières applications sont limitées au placement de rectangles.

Dans [**Fourman85**], le code génétique utilisé pour le placement de modules rectangulaires dans un circuit VLSI, est un code indirect. L'algorithme est appliqué pour le compactage des éléments déjà placés du circuit. L'espace de recherche est une population de positionnement que l'auteur appelle *stratégies*. Une stratégie (ou chromosome) consiste en positions relatives des éléments telles que: "*au-dessus-de*", "*en-dessous-de*", "*à-droite-de*" et "*à-gauche-de*". Ainsi, chaque

[12] Génotype : le code génétique.

[13] Phénotype : l'expression du génotype en caractère visible, i.e., la valeur correspondante.

rectangle est relié à un autre par une stratégie. Pour n rectangles, on a *a priori* 4^n chromosomes dans l'espace de recherche. Les rectangles sont placés dans le premier quadrant du plan le plus près possible de l'origine de façon à être consistant avec la liste des contraintes élémentaires. Par exemple la solution "*P au-dessus-de Q*" est consistante même si P est placé très loin de Q pourvu qu'il soit au-dessus. Ces stratégies ne sont toutefois pas écartées de la population, car ce sont des solutions partielles qui peuvent évoluer au cours des générations. Plusieurs critères de sélection ont été utilisés pour faire évoluer la population de stratégies. Les contraintes de connexion entre les rectangles sont exprimées dans la fonction objectif.

D. Smith [**Smith85**] avait proposé un système qui combine les heuristiques avec les AG pour le placement de rectangles dans un rectangle. Les AG sont utilisés pour rechercher le meilleur ordre des objets à placer dans le rectangle. Un chromosome est alors représenté sous forme de liste de rectangles à placer. Les heuristiques utilisées pour décoder le chromosome en placement sont de type "*placer-en-vertical-d'abord*" et " *placer-en-horizontal-d'abord*".

Cette représentation mixte a abouti au *croisement basé sur l'ordre des éléments* afin d'éviter de produire des configurations invalides. Deux type de mutation ont été utilisés : *scramble* consiste à faire un *mélange* aléatoire des éléments de la liste et *flip mutation* recherche toutes les orientations possibles de chaque élément lors du décodage/placement. Des résultats obtenus sont "*300 fois meilleurs qu'avec les techniques heuristiques et la programmation dynamique*" (*sic*). L'auteur propose qu'on pourrait concevoir un système adaptatif qui permet de superviser la recherche pendant que l'algorithme est en cours d'exécution. Par exemple, varier l'importance de la mutation au cours de la recherche ou changer la taille de la population à différentes stades de l'évolution. Pour cela, il faut étudier comment l'espace de recherche est exploré et faire un apprentissage de procédures pour ainsi parvenir à un système contrôlé et adaptatif.

Une autre représentation symbolique (indirect) a été proposée par Kröger *et al.* [**Kroger90, 92**] pour le placement de rectangles. Comparable à la méthode de D. Smith ci-dessus, cette méthode met en œuvre des stratégies telles que "*au-dessus-de*" "*à-droite-de*" mais en utilisant un codage en arbre binaire. Un arbre binaire contient les noeuds représentant les rectangles avec leur orientation et les arcs qui sont les liens de voisinage avec les rectangles voisins (*au-dessus*, *à droite*). Avec ce codage, un code (génotype) peut correspondre à plusieurs configurations de placement (phénotype). Donc afin d'avoir une correspondance unique entre le génotype et le phénotype, des priorités sont attribuées dans les noeuds. Un croisement spécifique à cette représentation a été développé pour ne créer que des individus valides. La méthode fonctionne pour le placement de rectangles et pour la découpe en guillotine, mais sa généralisation à des polygones à n côtés n'est pas évidente. La limitation du code (génotype) réside dans le manque d'information sur l'admissibilité et la qualité d'une solution, car il faut à

chaque fois construire le phénotype (décodage du génotype) qui demande beaucoup de temps de calcul (d'où l'utilisation de processeurs parallèles).

Pour le placement de polygones, on trouve aussi l'utilisation d'une représentation binaire proposée dans [**Petridis**]. Ce code peut être qualifié comme code indirect.

Les coordonnées (x,y) des sommets de polygones sont codés en chaîne binaire de longueur n. Ainsi, pour N polygones et $n = 10$, on a un génotype de longueur $N \times 2 \times 10$ bits. Ce qui ramène, pour un problème de 12 polygones, à $12 \times 2 \times 10 = 240$ bits à manipuler et l'espace de recherche égal à 2^{240} ou à peu près 1.77×10^{72} solutions différentes! C'est pourquoi, la représentation binaire n'est pas envisageable pour le problème de placement de formes irrégulières.

Dans [**Ismail93**], la représentation binaire est uniquement utilisée pour représenter les paramètres d'optimisation d'imbrication (coordonnées relatives et orientation). Leur approche se limite à l'imbrication de formes deux à deux. L'imbrication obtenue est ensuite recopiée par série sur la matière pour la découpe. Un autre exemple de telle approche avec les algorithmes génétiques est présentée dans [**Cook91**].

Dans [**Fujita93**], une approche hybride AG-recherche locale a été proposée pour le placement de polygones. Cette approche utilise un algorithme génétique avec un codage indirect pour la recherche combinatoire du meilleur ordre des éléments à placer. L'algorithme génétique est utilisé en association avec un algorithme de recherche locale appelé l'algorithme de Quasi-Newton. L'algorithme de Quasi-Newton assure une opération d'imbrication des éléments. Un génotype est une liste de polygones. L'ordre des ces polygones dans la liste exprime aussi leur position relative dans le placement. Le croisement basé sur l'ordre est donc utilisé. L'objectif est d'obtenir le placement le plus compact possible, en minimisant la longueur et la largeur de la zone de placement. Les superpositions sont exprimées en terme de pénalisation incluse dans la fonction objectif. Les résultats montrent que l'évaluation locale de la surface de superposition est très coûteuse en temps. Bien que cette procédure locale essaie de maintenir la relation de voisinage issu du génotype, en ajoutant un terme supplémentaire dans la fonction objectif, le génotype ne reflète pas directement les régions géométriquement distinctes. Ce qui est assez normale pour un codage indirect. Et la difficulté est la pondération des termes de pénalité dans la fonction.

Une approche similaire est décrite dans [**Pargas93**] avec une représentation assez similaire mais le code contient une information supplémentaires sur l'orientation à appliquer. Chaque forme est numérotée de 1 à N pour N formes à placer. Chaque forme peut être orientée soit de 0, 90°, 180° ou 270°. Une solution est donc une structure consistant en une liste de numéro de

formes et leur orientation respective. Chaque liste contient également la longueur de matière utilisée pour indiquer sa valeur de fitness:

$$[f_1, o_1), (f_2, o_2), ...(f_n, o_N), L]$$

Ensuite, la surface du support est aussi codée en tableau de pixel (noté A). Chaque case du tableau représente une unité d'espace de ce support. Une case peut être soit vide soit occupée par une forme, sachant que deux formes ne doivent pas avoir de cases communes dans A. Initialement, le tableau est vide, puis il se remplit au fur et à mesure que les formes sont placées. L'algorithme de placement place les formes de f_1 à f_N dans l'ordre donné par la liste. Dès qu'une forme f_i est choisie pour être placée, l'algorithme *scrute* le tableau A pour trouver les cases vides. S'il en trouve, la forme f_i est placée avec l'orientation o_i, à partir de la case vide la plus haute et la plus à gauche (remplissage en colonne). Si le placement est sans superposition, il est accepté, sinon, le processus recommence avec l'une des trois autres orientations restantes. Si les essais de placement n'ont abouti sur aucune de ces orientations, l'algorithme recommence le balayage de A pour rechercher une autre cellule vide. Les résultats sur des processeurs parallèles ont montré une bonne qualité d'imbrication, mais on remarque aussi que l'évaluation de la qualité du placement est trop coûteuse en temps de calcul, plus que le processus AG lui-même.

La représentation ordonnée (ou en liste) est simple à concevoir mais, les opérateurs peuvent produire des solutions non admissibles; souvent, ils ne permettent pas de préserver les associations "prometteuses".

Une approche assez similaire mais de façon hiérarchique est abordée par E. D. Goodman *et al.* [**Goodman94**] pour le problème de placement de modules dans un circuit intégré. Les éléments appartiennent à un groupe qui peut être lui-même inclus dans un autre groupe. Les résultats de cette étude actuellement en cours ne sont pas encore publiés.

Une autre approche du placement de polygones par codage hiérarchique est présentée dans [**Dighe96**]. Cette représentation combine représentation symbolique au niveau supérieur et représentation binaire au niveau inférieur. Le niveau supérieur traite le côté combinatoire du problème où deux types de représentation sont utilisés (en arbre binaire ou en chromosome ordonné); le niveau inférieur traite l'imbrication locale entre deux formes, où une représentation binaire est utilisée. Cette dernière est élaborée à partir d'une concaténation de paramètres de position (x, y) et d'orientation θ de tous les polygones placés, dont chaque nombre est codé en chaîne binaire. De bons résultats ont été obtenus sur les polygones réguliers, mais le temps de calcul est assez prohibitif, par exemple, les auteurs rapportent qu'un placement de 16 polygones avec, les paramètres optimaux, met 2 heures en temps d'exécution.

La définition de représentation d'un chromosome reste toujours un problème dans le placement de polygones ou de formes irrégulières. Ainsi pour appliquer les AG en tant

qu'heuristique pour résoudre notre problème de placement, il est important d'étudier d'abord l'aspect codage. Après avoir étudié le bon fonctionnement de la représentation adoptée, on peut aborder l'application à l'apprentissage.

I.7. Données pour Tester les Algorithmes

I.7.1. Carnets de Commande

Les caractéristiques des quatre carnets de commande que nous allons tester sont donnés dans les Figure I. 6 et Figure I. 7 et repris dans le tableau suivant (Tableau I.1).

carnet de commande	*ralf.cde*	*gr1.cde*	*chem3.cde*	*kd3.cde*
Nombre de formes (N)	22	48	90	112
Surface $\sum_{k=1..n} a_k \mathbf{A}(F_k)$	158 948.18	1 424 438.9	967 892.9	950 869.6
Longueur $\mathbf{Max}_{k=1..N} (L_{F_k})$	223	369	310	339

Tableau I. 1 : Caractéristiques des carnets de commande. Les unités sont en pixel.

A partir d'un carnet de commande, on peut estimer d'abord le nombre de bandes nécessaires pour une largeur de matière donnée. Si on décide que la longueur d'une bande est la longueur de la plus longue forme du carnet, la somme de surface de toutes les formes du carnet divisée par la surface de la bande estimée donne approximativement le nombre de bandes :

$$NB = \mathbf{Sup} \frac{\sum_{k=1..n} a_k \mathbf{A}(F_k)}{W \times \mathbf{Max}_{k=1..N} \mathbf{Len}(F_k)}$$

Equation I. 6

Et le rendement estimé par bande peut en être déduit :

$$\eta\,\% \approx \frac{\sum_{k=1..n} a_k \, \mathbf{A}(F_k)}{NB \times W \times \underset{k=1..N}{\mathbf{Max}} \, \mathbf{Len}(F_k)}$$

Equation I. 7

Bien sûr, ce calcul ne donne un nombre qu'à titre indicatif, car il ne tient compte ni de la disparité des formes ni des surfaces non utilisables de certaines concavités des formes.

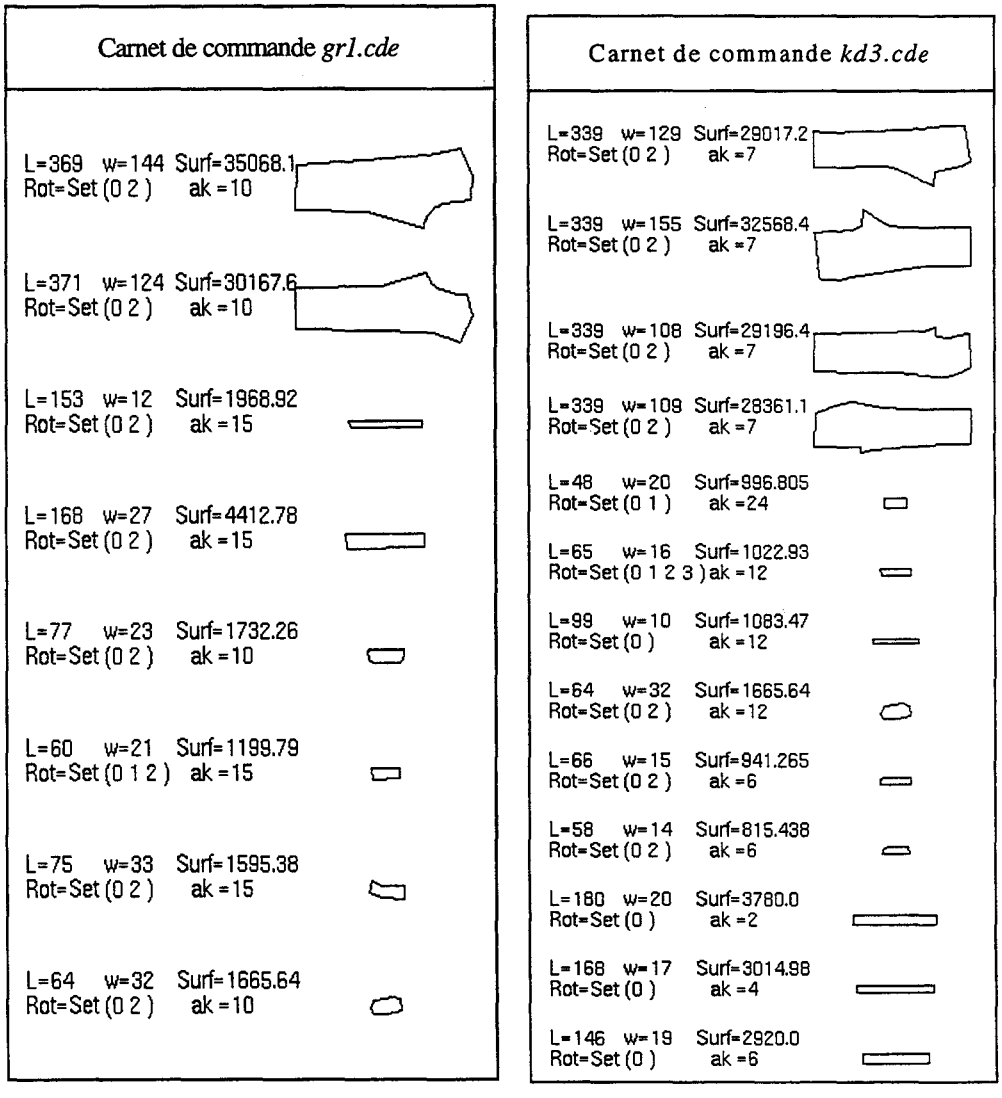


Figure I. 6 : Formes pour pantalon, carnet "gr1.cde" et carnet "kd3.cde". Le modèle de pantalon contient des formes assez "régulières" dans le sens où elles permettent parfois d'atteindre jusqu'à 90% de remplissage avec un placeur expert.

Le carnet est typiquement composé de très grandes formes et un ensemble de petites formes.

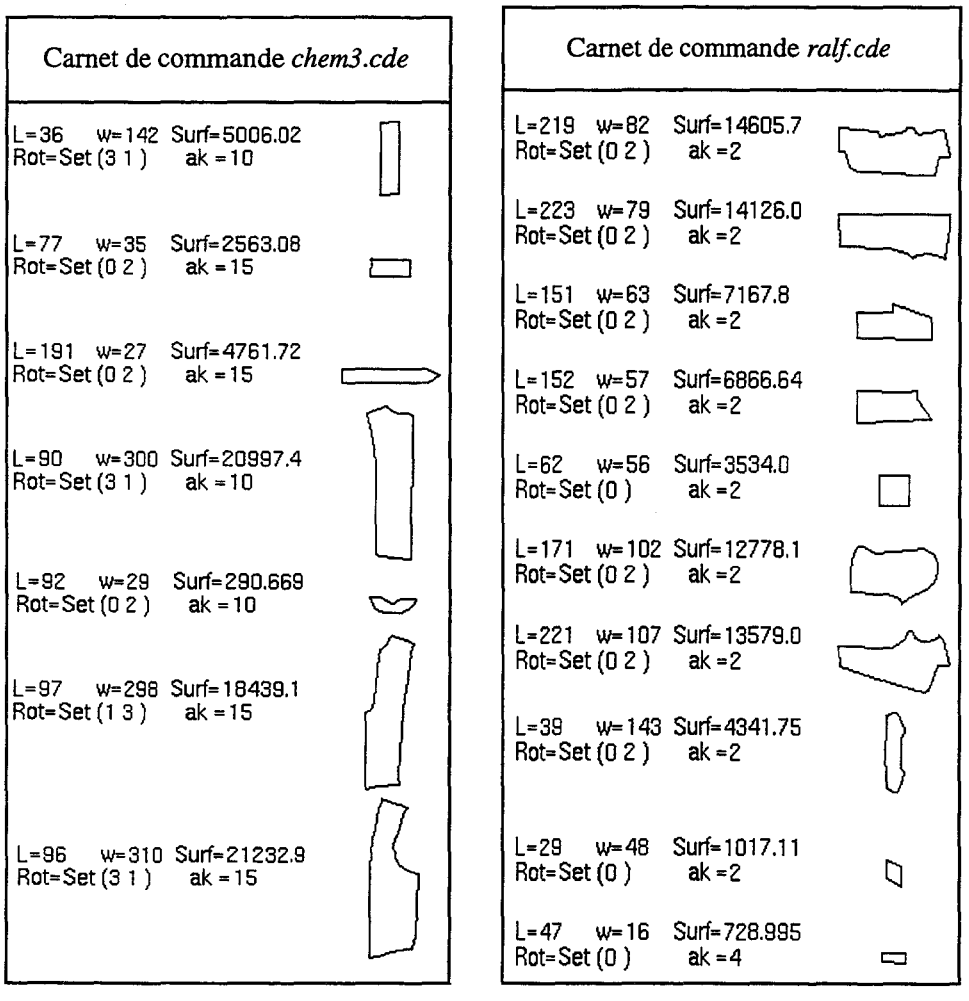


Figure I. 7 : Formes pour chemise, carnet "chem3.cde" et pour veste, carnet "ralf.cde". La différence de tailles entre les formes de veste n'est pas aussi grande que dans le cas du pantalon.

I.7.2. Format d'un contour

Les formes d'un carnet sont créées à l'aide d'un outil de dessins en Smalltalk développé par G. Roussel [**Roussel94**]. Les reproductions sont faites en respectant les proportions surfaciques. Les arcs sont extrapolés par un ensemble de segments.

Avant d'être enregistrées dans le carnet de commande, chaque forme doit être codée en peignes réduits pour être utilisable par les algorithmes d'imbrication.

I.7.2.1. Codage des contours

Une méthode originale de codage de contour, appelé "*codage par peigne de contour*", a été développée par G. Roussel [**Roussel94**]. Ce codage est basé sur le principe d'échantillonnage du contour visible de chaque côté du rectangle. Ces échantillons forment un ensemble de *dents* qui sont des segments exprimant la distance entre un point du contour et le bord du rectangle. Les dents sont générées le long du contour par une distribution de Dirac de période T dans la référence liée à chaque côté. Un peigne représente donc des échantillons du contour circonscrit (voir Figure I. 8).

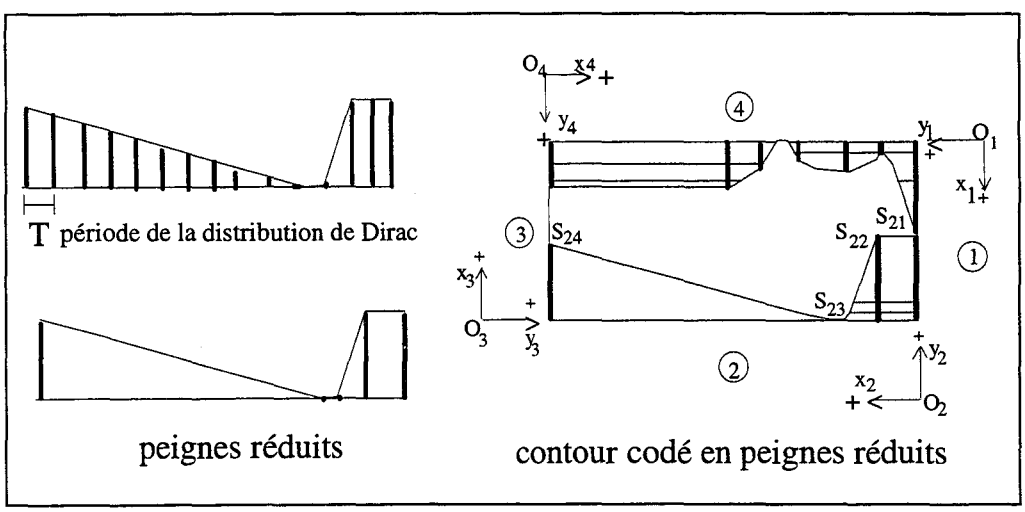


Figure I. 8 : Codage en peigne de contour. Les dents d'un peigne sont générées le long du contour par une distribution de Dirac de période T dans la référence liée à chaque côté. Les peignes réduits sont obtenus après suppression des dents redondantes.

La longueur d'une dent est une information importante pour déterminer la meilleure imbrication locale de deux formes. Cette technique de peignes est efficace pour imbriquer les formes grâce à la déficience rectangulaire définie par la longueur des dents des peignes.

- Longueur et largeur du rectangle;
- L'ensemble des orientations autorisées, {0°, 90°, 180°, 270°};
- L'ensemble des peignes réduits de chaque côté du rectangle circonscrit;
- La surface estimée de la forme.

Ces informations implantées en langage Smalltalk deviennent les *messages*, les *objets* et les *récepteurs* du modèle *Forme*.

I.8. Conclusion

Dans ce chapitre, nous avons présenté le contexte de travail. Nous avons introduit le concept d'une cellule flexible de confection dans laquelle s'inscrit la partie placement de formes. Nous avons également formulé le problème de placement avec la fonction objectif et les contraintes. Ensuite, l'état de l'art dans le domaine sur les méthodes de résolution permet de situer l'actualité et la difficulté du problème. Enfin nous avons décrit les carnets de commandes qui servent de données de tests dans les chapitres suivants. Dans ce document, nous nous intéressons aux méthodes de résolution appliquées au format existant des formes. Les chapitres suivants présentent les méthodes de résolution et les résultats de leur application aux données présentées dans ce chapitre.

I.9. Références Bibliographiques

[**Acar95**] M. Açar, editor, *Mechatronic Design in Textile Engineering*, NATO ASI series, Series E: Applied Sciences - Vol. 279, Kluwer Academic Publishers, the Netherlands, 1995.

[**Adamowicz76**] M. Adamowicz and A. Albano, "Nesting Two-Dimensional Shapes in Rectangular Modules", *Computer-Aided Design*, vol.8, pp. 27-33, 1976.

[**Albano80**] A. Albano and G. Sapupo, "Optimal Allocation of Two Dimensional Irregular Shapes Using Heuristic Search Method", *IEEE Trans. Syst. Man & Cybern.* Vol. 10, N°5, (May 1980), pp. 242-248.

[**Arenales95**] Marcos Arenales, Reinaldo Morabito, "An AND/OR-graph Approach to the Solution of Two-Dimensional Non-Guillotine Cutting Problems", *European Journal on Operations Research: Special issue, Cutting and Packing*, E.E. Bischoff, G. Wäscher Editors, vol 84, N° 3, 1995, pp. 599-617.

[**Avnaim89**] F. Avnaim, *Placement et Déplacement de Formes Rigides ou Articulées*, thèse de doctorat, Université de Franche-Comté, 1989.

[**Benhamamouch79**] D. Benhamamouch, *Un Algorithme de Découpe Optimale dans* $\mathbb{R}^2$, Thèse de doctorat en math., Université de Pierre et Marie Curie-Paris 6, France, Février 1979.

[**Bishoff95**] E. Bischoff and G. Wäscher, Editors, "Special issue on "Cutting and Packing", *European Journal of Operational Research*, vol. 84, no. 3, Aug. 3, 1995, Elsevier publishing.

[**Byrne95**] Chris Byrne, "The Industrial and Social Impact of New Technology in the Clothing Industry into the 2000s", David Rigby Associates report, originally produced for the International Labour Office (ILO), Geneva and since reprinted, in modified form, in *Textile Outlook International*, March 1995.

[**Confection91**] "Atelier de coupe: le parent pauvre de l'organisation", *Confection 2000*, N°121, Avril 1991, pp. 74-75.

[**Cook91**] Deborah F. Cook, "Genetic Algorithm Approach to a Limber Cutting optimization Problem", *Cybernetics and Systems*: An International Journal, No. 22, 1991, pp. 357-365.

[**Daniels95**] K.M. Daniels, *Containment Algorithms for Non Convex Polygons with Applications to Layout*, Harvard University, Ph.D. Dissertation, Division of Applied Science, May 1995.

[**Daza95**] V.P. Daza and A.G. de Alvarenga and J. de Diego, "Exact Solutions for Constrained Two-Dimensional Cutting problems", *EJOR: Special issue, Cutting and Packing*, vol 84, N° 3, 1995, pp. 633-644.

[**Delaporte89**] J. L. Delaporte, *Intégration des Fonctions de Conception et de Préparation de la Fabrication* pour *les Entreprises de Découpe*, Thèse de doctorat Automatique Productique, Université de Valenciennes et du Hainaut-Cambresis, Décembre 1989.

[**Delgrange88**] A. Delgrange, S. Maouche, "Real Time Automatic Assembling of Collar Parts", *Proc. Sixth International Conference on Systems Engineering*, Coventry (England), September 1988.

[**Delgrange93**] A. Delgrange, *Analyse, Conception et Réalisation d'une Cellule d'Assemblage Automatique pour l'Industrie de la Confection*, Thèse de doctorat, Université des Sciences et Technologies de Lille, 19 Mars 1993.

[**Dighe96**] R. Dighe, M.J. Jakiela, "Solving Pattern Nesting Problems with Genetic Algorithms Employing Task Decomposition and Contact Detection", *Evolutionary Computation*, Vol.32, No. 3, 239-266, 1996.

[**Downsland95**] K. Dowsland, W. B. Downsland, "Solution Approaches to Irregular Nesting Problems", invited review in E. E. Bischoff and G. Wäscher (ed.°), Special issue on "Cutting and Packing" of *European Journal of Operational Research*, vol. 84, no. 3, Aug. 3, 1995, Elsevier pub., pp. 506-521.

[**Draou86**] A. Draou, *Outils d'Aide au Placement sous Contraintes. Applications à la Confection*, Thèse de doctorat d'Automatique et Traitement du Signal, Université de Valenciennes et du Hainaut-Cambresis, mai 1986.

[**Dyckhoff92**] Harald Dyckhoff, Ute Finke, *Cutting and Packing in Production and Distribution, a Typology and Bibliography*, Physica Verlag, Heidelberg, Germany, 1992.

[**Fayard95**] D. Fayard, V. Zissimopoulos, "An Approximation Algorithm for Solving Unconstrained Two-Dimensional Knapsack Problems", in [Bishoff95], 1995, pp. 618-632.

[**Fourman85**] M. Fourman, "Compaction of Symbolic Layout using Genetic Algorithms", *Proceedings of the First International Conference on GA and their Applications*, July 24-26, Hillsdale: Lawrence Erlbaum Associates, 1985, pp. 141-153.

[**Fowler81**] R. J. Fowler, Michael S. Paterson, Steven L. Tanimoto, "Optimal Packing and Covering in the Plane Are NP-Complete", *Information Processing Letters*, vol. 12, N°. 13 , pp. 133-137, June 1981.

[**Fujita93**] Kikuo Fujita, Shinsuke Akaji, Noriyasu Hirokawa, "Hybrid Approach for Optimal Nesting Using a Genetic Algorithm and a Local Minimization Algorithm", DE-Vol. 65-1, *Advances in Design Automation*, vol. 1, ASME 1993, pp. 477-484.

[**Gilmore65**] P. Gilmore and R. Gomory, "Multistage Cutting Stock Problems of Two and More Dimensions", *Operations Research*, vol. 13, 1965, pp. 94-120.

[**Goodman94**] E.D. Goodman, A.Y. Tetelbaum, V.M. Kureichik, "A Genetic Algorithm Approach to Compaction, Bin Packing, and Nesting Problems", Technical

report of Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, No. 940702, July 1994,

[Heckmann95] R. Heckman and Th. Lengauer, "A Simulated Annealing Approach to the Nesting Problem in the Textile Manufacturing Industry", In R. E. Burkard, P. L. Hammer, T. Ibakari and M. Queyranne, Editors, *Annals of Operations Research*, J. C. Baltzar AG Science Publishers, Amsterdam, vol. No. 57, 1995, pp. 103-133.

[Hewit95] J. R.. Hewit, "Mechatronics", in [Acar95], pp. 1-26.

[Ismail92] H.S. Ismail, K.K.B. Hon, "New Approaches for the Nesting of Two-Dimensional Shapes for Press Tool Design", *International Journal of Production Research*, Vol. 30, No. 4, pp. 825-837, 1992.

[Jayaraman95] S. Jayaraman, Computer-aided design and manufacturing: a textile-apparel perspective, in M. Acar, editor, *Mechatronic design in textile Engineering*, NATO ASI series, Series E: Applied Sciences - Vol. 279, Kluwer Academic Publishers, the Netherlands, 1995, pp. 239-269.

[Kroger90] B. Kröger, P. Schwenderling and O. Vornberger, "Parallel Genetic Packing of Rectangles", in *Proceedings of Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Editors, First Workshop, Berlin: Springer Verlag, 1990, pp. 60-164.

[Laplante89] H. Laplante, "Volume Doesn't Matter with Automation", *Bobbin*, June 1989, pp. 76-79.

[Li94] Z. Li, *Compaction Algorithms for Non-convex Polygons and Their Applications*, PhD thesis in Computer Scienc, Div. of Applied Sciences, Harvard University, Cambridge, MA, May 1994.

[Lutfiyya91] H. Lutfiyya and B. McMillin, "Composite Stock Cutting Through Simulated Annealing", T.R. N° CSC 91-09 or ISC 91-04, Dept. Computer Science, University of Missouri at Rolla, 1991.

[Maouche92] Salah Maouche, "A Continuous Cutting System in the Suit Make Up Industry", *Proc. of XIth Int. Conference on Systems Science*, Wroclaw, Poland, Sept. 1992.

[Maouche95] Salah Maouche, *Contribution à l'Optimisation de Placement de Formes Irrégulières. Application à l'Industrie Textile*, Université des Sciences et Technologies de Lille, Thèse d'état, Décembre 1995.

[Martello94] Silvano Martello, Editor, "Knapsack, Packing and Cutting", part II: *Multidimensional Knapsack and Cutting Stock Problems*, *INFORS*, special issue, vol. 32, No. 4, November 1994.

[Milenkovic91] V. J. Milenkovic, K.M. Daniels and Z. Li, "Automatic Marker Making", in *Proceedings of the 3rd Canadian Conference on computational Geometry*, ed. T. Shermer, Simon Fraser University, Vancouver, B.C., August 6-10, 1991, pp. 243-246.

[Okat91] A. Okat, *Contribution à la Conduite d'une Cellule de Découpe au Laser*, Thèse de 3ème cycle, Université des Sciences et Technologies de Lille, Novembre 1991.

[Oliveira93] J. F. Oliveira and J.S. Ferreira, "Algorithms for Nesting Problem", in R. V. Vidal, Editor, *Applied Simulated Annealing*, pages 255-273, Springer Verlag, Berlin, 1993.

[Pargas93] R. P. Pargas and R. Jain, "A Parallel Stochastic Optimization Algorithm for Solving 2D Bin Packing Problems", in Proceedings, *The Ninth International Conference On Artificial Intelligence for Applications*, Orlando, FL, 1-5 March 1993, IEEE Computer Society Press, Los Alamitos, CA, pp. 18-25.

[Petridis] V. Petridis and S. Kazarlis, "Varying Quality function in Genetic Algorithms and the Cutting Problem", Dept. of Electrical Engineering, Univ. Thessaloniki (Greece). Via Univ of Edinburg, dept. Artificial intelligence, `http://www.dai.ed.ac.uk/`

[Prempti83] F. Prempti, *Méthodes Stochastiques dans les Problèmes de Placement*, thèse de doctorat, Université de Scientifique et Médicale de Grenoble, département de Recherche Opérationnelle, 1983.

[Roussel94] Gilles Roussel, *Optimisation du Placement de Formes Irrégulières sur Matières Planes. Application à l'industrie de la Confection*, Thèse de doctorat, Université des Sciences et Technologies de Lille, Janvier 1994.

[Smith85] D. Smith, "Bin Packing with Adaptive Search", *Proceedings of the First International Conference on GA and their Applications*, July 24-26, Hillsdale: Lawrence Erlbaum Associates, 1985, pp. 202-207.

[Soenen77] R. Soenen, Contribution à l'Etude des Systèmes de Conduite en Temps Réel *en Vue de la Commande d'Unités de Fabrication*, Thèse d'état, Univ. Valenciennes et du Hainaut-Cambresis, mars 1977.

[Stoyan96] Yu G. Stoyan, M.V. Novozhilova, A.V. Kartashov, "Mathematical Model and Method of Searching for a Local Extremum for the Non-Convex Oriented Polygon Allocation Problem", in *European Journal of Operations Research (EJOR)*, vol. 92, 1996, pp. 193-210.

[Taylor95] P.M. Taylor and M.B. Gunner, "Mechatronics in Automated Garment Manufacture", in **[Acar95]**, pp. 271-289.

[Vidal85] P. Vidal, Cichowlas, S. Maouche, "Flexible Cutting Frame in Real Time", 1er *C.I.N.T.T.*, Mulhouse 1985.

Chapitre II:

Algorithmes de Recherche en Arbre

TABLE DE MATIERES

II.1. Introduction 45

II.1.1. Espace de Recherche et Représentations 46

II.1.2. Arbres et Arborescence 46

II.2. Algorithmes de Recherche en Arbre 47

II.2.1. Opérateurs de Transformation 47

II.2.2. Fonction d'Evaluation 47

II.2.3. Stratégie de Contrôle 47

II.3. Algorithme A^* 49

II.3.1. Définitions 49

II.3.2. Implantation 51

II.3.3. Propriétés de A^* 52

II.3.4. Complexité de A* 52

II.4. Algorithmes Semi-Admissibles 53

II.4.1. Algorithme ε-Admissible A_ε^* 54

II.4.2. Algorithme R_δ^* : Introduction aux Principes 56

II.4.3. L'algorithme $R_{\delta\varepsilon}^*$ 62

II.5. Application au Problème de Placement 63

II.5.1. Calcul de g(u) 64

II.5.2. Calcul de $h(u)$ 65

II.5.3. Calcul de f_a et f_b 67

CHAPITRE II

Algorithmes de Recherche en Arbre

II.1. Introduction

Il s'agit ici de transformer le problème de recherche de placement optimal en une recherche d'un chemin optimal dans un espace d'états. Chaque état correspond au placement d'une pièce satisfaisant les contraintes relatives au tracé, à l'orientation possible de chaque forme, aux limites de la matière et au chevauchement des pièces déjà placées.

Dans ce chapitre, nous allons présenter les algorithmes de recherche en arbre basée sur le principe du "meilleur d'abord". La sélection d'un chemin à parcourir s'effectue dans ce cas grâce à l'estimation de son coût futur. La différence entre les algorithmes introduits réside dans cette estimation. En effet, la condition pour que la recherche aboutisse à une solution optimale est que celle-ci soit inférieure ou égale au coût optimal du chemin optimal. L'algorithme A^*_ε ne choisit que les noeuds dont le coût ne dépasse pas d'un écart de tolérance ε, pour s'assurer que le chemin choisi ait un coût toujours inférieur au coût optimal estimé. Comme le coût optimal n'est pas connu à l'avance, rien n'indique que l'estimation soit bien inférieure. Donc, dans l'algorithme $R^*_{\delta\varepsilon}$, l'estimation est basée sur une probabilité d'être proche de la valeur optimale, soit par valeur inférieure soit par valeur supérieure. La surestimation ne doit pas trop s'éloigner de l'optimale; c'est pourquoi un seuil de risque est fixé par le paramètre δ. Comme pour A^*_ε, un écart ε permet de sélectionner qu'un nombre limité de noeuds à développer.

Avant d'aborder les deux méthodes de résolution en arbre que nous avons utilisées, nous allons rappeler quelques concepts et définitions tels que la définition de l'espace d'états, la théorie de l'algorithme A^* et ses dérivés.

II.1.1. Espace de Recherche et Représentations

- Pour un problème *P* donné, un noeud *u* est appelé "sous-problème".
  - On définit un **graphe d'états** *G(U, Γ)*, un ensemble d'états ou noeuds, avec *U*, l'espace de représentation d'états et Γ, l'ensemble des opérateurs permettant la transformation d'un sous-ensemble de solutions en d'autres sous-ensembles dans cet espace [**Nilsson82**].
  - Le noeud initial est appelé la *racine* notée u_0 et le noeud terminal ou noeud but ou terminal est noté u_t.
  - Les connexions entre noeuds sont appelées *liens* ou *arcs*.
  - Si un arc est orienté d'un noeud u à un noeud *v*, on dit que : *u* est le père de *v*, on note : *père(v)* = *u* ou *v* est le fils de *u*, noté Γ*(u)* = *v*.

II.1.2. Arbres et Arborescence

1. Un arbre est un graphe *G* = (*U*, Γ) connexe et sans cycle, dans lequel chaque noeud a un seul parent (sauf la racine qui n'a pas de prédécesseur).
2. Dans un arbre *G* = (*U*, Γ), le *degré* (extérieur) ou facteur de *branchement b* d'un noeud *u* est le nombre de ses successeurs. Par définition :
   b= *Card* {*u'*∈ *U* | *u'* = Γ*(u)* } *ou* $b = d_G^+(u)$
3. La *profondeur d* d'un noeud *u* est la longueur du chemin de la racine u_0 à *u*.
4. La *taille* d'un arbre est le nombre total de noeuds de l'arbre.
5. Un noeud *terminal* est un noeud qui ne peut plus donner de noeuds fils, soit parce qu'il ne contient pas de solutions réalisables ou des solutions dont on est sûr que l'exploration ne donnera pas de solution meilleure.

II.2. Algorithmes de Recherche en Arbre

La caractéristique commune à tous les algorithmes de recherche en arbre est l'itération de base de l'algorithme d'exploration qui est constituée du cycle "*sélection-génération-évaluation-élargage-insertion*" de noeuds.

II.2.1. Opérateurs de Transformation

Pour chaque problème, on peut définir un ensemble d'opérateurs de transformation permettant de transformer l'état d'un noeud.

II.2.2. Fonction d'Evaluation

La fonction d'évaluation est nécessaire pour mesurer la *distance* entre un noeud courant et les noeuds terminaux. Un algorithme qui utilise cette information est dit "informé" ou "guidé". L'idéal est d'avoir une estimation exacte de cette distance, ainsi l'effort d'exploration de l'espace de recherche sera minimal. Cependant de telle estimation demande souvent trop de temps de calcul pour être applicable. Il faut donc trouver un compromis entre le temps de calcul et l'exploration de l'espace pour rendre un algorithme pratiquement acceptable [**Nilsson80, Pearl82**].

Une fonction d'évaluation f est introduite dans le processus de recherche pour permettre de quantifier la distance entre le noeud courant et les noeuds terminaux . Elle constitue donc une information de contrôle sur la direction que doit prendre l'algorithme de recherche. Ceci permet d'élaguer l'espace de recherche en définissant un intervalle de recherche $\left]f_{min}, f_{max}\right[$. Cet intervalle est mis à jour et affiné au fur et à mesure de l'exploration de l'espace de recherche. L'inconvénient (pour une minimisation) c'est que si un noeud a une évaluation inférieure à f_{min}, ce noeud ne sera pas exploré.

II.2.3. Stratégie de Contrôle

La stratégie de contrôle oriente l'exploration pour une recherche plus rapide. Elle comprend : la sélection, l'élagage et l'insertion de noeuds dans l'algorithme de base.

La procédure de sélection de noeud permet de choisir parmi l'ensemble des candidats, le noeud le plus prometteur. Le critère de sélection varie selon la stratégie de contrôle appliquée.

Parmi les stratégies de contrôle, on peut citer **[Winston92]**:

1. Les stratégies non informées:

   Ces stratégies n'utilisent pas de fonction d'évaluation. La stratégie "profondeur d'abord" (*Depth First* : *DF*) sélectionne d'abord les noeuds développés en dernier. Elle nécessite peu d'espace mémoire et sa complexité spatiale est en $O(db)$ si tous les noeuds fils sont engendrés à chaque itération de base (en $O(d)$ si un seul fils est engendré à chaque itération). La stratégie "largeur d'abord" privilégie les noeuds les moins profonds dans l'arborescence de recherche. Elle permet de trouver le noeud terminal, s'il en existe un, le moins profond possible. Sa complexité spatiale est en $O(b^d)$.

2. Les stratégies informées

   Ces stratégies utilisent une fonction d'évaluation pour classer les noeuds à développer. La recherche "meilleur d'abord" (*Best First* : *BF*) favorise les noeuds ayant les meilleures évaluations f selon le critère d'optimisation; les informations heuristiques globales cumulées au fur et à mesure de la recherche sont ainsi utilisées pour guider la suite de l'exploration; l'efficacité de cette stratégie dépend bien sûr de la pertinence de la fonction d'évaluation; sa complexité spatiale est au pire des cas en $O(b^d)$.

3. D'autres stratégies "hybrides" **[Ghosh91]** combinent les avantages de "profondeur d'abord" et "meilleur d'abord" pour optimiser l'espace mémoire et le temps de recherche.

La stratégie *DF* choisit le noeud qui se trouve le plus profond dans l'arbre de recherche, en général celui qui est le plus récemment créé. Rien ne peut garantir qu'il soit le meilleur parmi tous les noeuds restant à explorer.

Même si la complexité spatiale est en $O(d)$, c'est-à-dire que si un seul fils est engendré à chaque itération, la performance de cette stratégie dépend fortement de l'ordre d'examen des noeuds. Dans le cas où le noeud de meilleure évaluation est toujours exploré en premier, l'intervalle peut être réduit fortement. Il faut noter que le classement par ordre est effectué localement, c'est-à-dire sur les noeuds issus d'un même noeud père, et non globalement sur l'ensemble de tous les noeuds à explorer comme pour la stratégie meilleur d'abord. C'est pourquoi, le classement des noeuds même uniquement sur les noeuds fils, ou même effectué localement, occasionne un coût supplémentaire de tri en $O(db \log b)$ et un effort de stockage $O(db)$ des noeuds créés. Le coût du tri devient très important s'il est effectué à chaque

génération de noeuds. L'efficacité de la résolution devient médiocre puisque l'efficacité de la recherche basée sur l'ordre ne peut plus compenser le surcoût des tris.

II.3. Algorithme *A**

II.3.1. Définitions

Tous les algorithmes de la famille de *A** utilisent pour évaluer un état une fonction d'évaluation *f* résultant de la somme de deux autres fonctions *g* et *h* (Figure II. 1).

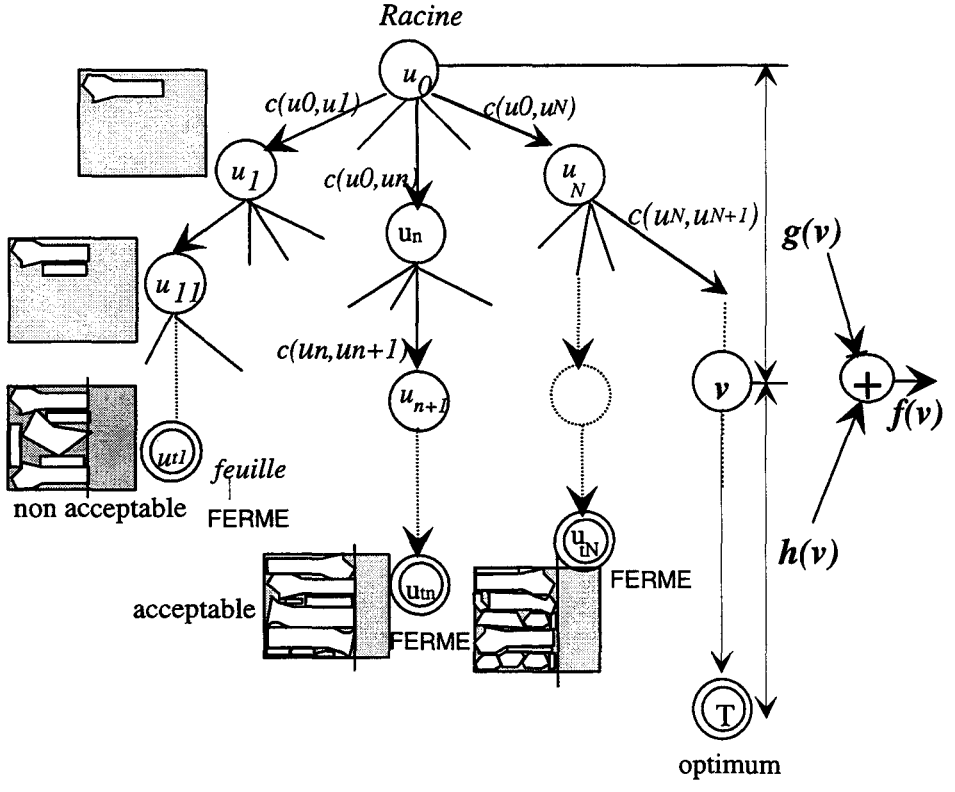


Figure II. 1: Arbre de recherche avec l'algorithme A*. A chaque noeud de l'arbre, l'algorithme est guidé par une fonction heuristique *f* est basée sur la somme du coût du chemin déjà parcouru, *g*, et l'estimation du coût du chemin futur, *h*.

Définition 1:

$$f : \mathrm{U} \rightarrow \mathbf{R}^{+},\ \forall\, u \in U\ ,\ f(u) = g(u) + h(u)$$

Equation II. 1

Définition 2: La fonction $g : U \rightarrow \boldsymbol{R}^{+}$, définit le coût du chemin courant , de l'état initial u_0 à l'état actuel u:

$$g(u) = c\ (u_0, u)\ avec\ g(u_0) = 0$$

Equation II. 2

Avec, dans un arbre orienté:

$Ch(u, v)$ l'ensemble des chemins menant de u à v ; l un chemin quelconque de u à v; et $c(u, v)$ le coût du chemin menant du noeud u au noeud v.

De même $g^*(u)$ est le coût minimal d'un chemin de l'état initial à l'état courant :

$$g^*(u) = \min_{l \in \mathrm{Ch}(u_0, u)} c\ (l)$$

Equation II. 3

Définition 3: La fonction heuristique $h : U \rightarrow \boldsymbol{R}^{+}$, détermine le coût estimé du chemin restant à parcourir pour atteindre l'état terminal u_t, à partir de l'état courant u.

$$h(u) = c(u,\ u_t)\ \text{avec}\ h(u_t) = 0$$

Equation II. 4

De même $h^*(u)$ est le coût minimal d'un chemin de l'état courant à un état u_t :

$$h^*(u) = \min_{l \in \mathrm{Ch}(u,\ u_T)} c(\ l\ )$$

Equation II. 5

Définition 4 : La fonction heuristique h est *admissible* si et seulement si :

$$\forall\, u \in U\ ,\ h(u) \leq h^*(u)$$

Equation II. 6

Il est démontré dans [Nilsson82] que si l'estimation est minorante, on pourra toujours atteindre la valeur optimale, si elle existe, par valeur inférieure.

Les conditions sur h entraînent les conditions sur f (Equation II. 1).

II.3.2. Implantation

L'algorithme A^* est un algorithme admissible de type "meilleur d'abord" gardant la trace de sa recherche en stockant les noeuds dans deux listes: **OUVERT** et **FERME**. La liste **OUVERT** contient les noeuds à développer et **FERME** contient des noeuds déjà développés.

⇒ On développe un noeud en explicitant ses descendants. Un noeud qui n'a pas encore été développé ou élargi est dit *ouvert* et celui déjà développé est dit *fermé* ; et les listes associées à ces noeuds sont respectivement **OUVERT** et **FERME**.

A chaque itération, l'algorithme A^* :

1. sélectionne dans **OUVERT** un noeud u ayant la plus faible valeur $f(u)$;
2. génère tous les fils de u, calcule leurs valeurs de g et h, puis les stocke dans **OUVERT**;
3. si un fils v de u est déjà dans **OUVERT** et si

   $g(v) > g(u) + c(u,v)$, alors $g(v) \leftarrow g(u) + c(u,v)$;
4. si un fils v de u est déjà dans **FERME** et un meilleur chemin est maintenant trouvé, alors $g(v)$ est mis à jour à ce nouveau meilleur coût et v est enlevé de **FERME** et remis dans **OUVERT** pour être exploité de nouveau.

Dans les problèmes de grande taille, A^* nécessite beaucoup d'espace mémoire pour stocker les noeuds dans **OUVERT** et **FERME**, puisque tous les noeuds admissibles pour être développés doivent être effectivement développés.

De plus, dans certains cas, il arrive que l'on remette un état de **FERME** dans **OUVERT** (étape 4). Ce cas se produit lorsque A^* développe un noeud et trouve un fils identique à un noeud quelconque de **FERME**, mais avec une meilleure évaluation. Il faudrait alors mettre à jour ce dernier et recommencer son exploration. Pour éviter ce retour, il est important que la fonction heuristique h soit monotone.

II.3.3. Propriétés de A^*

Il est démontré dans [**Nilsson82, Pearl90**] que A^* possède les propriétés suivantes :

- **Terminaison**: Pour tout graphe fini à fonction d'évaluation positive ou nulle, A^* s'arrête, soit sur un état terminal, soit parce qu'il n'y a plus de noeuds dans **OUVERT**.
- **Admissibilité**: A^* est admissible si la fonction h est admissible.
- **Consistance**: Pour tout noeud v successeur de u dans l'espace d'états U, une heuristique est consistante si et seulement si :

$$\forall u \in U, et\ v = \Gamma(u), \left| h(v) - h(u) \right| \leq c(u, v)$$

Equation II. 7

II.3.4. Complexité de A*

La complexité permet d'évaluer l'effort de calcul nécessaire pour l'exécution de l'algorithme. Pour A^*, on détermine le nombre de développements réalisés en fonction du nombre total N d'états dans l'espace d'états U du graphe. Il est démontré que A^* est exponentiel en temps de recherche (*complexité temporelle*) et en espace de mémoire (*complexité spatiale*) [**Huyn80**].

Il est aussi démontré dans [**Nilsson82, Pearl82**] que la complexité de A^* en temps de calcul est en $O(b^d)$, pour un arbre de recherche de profondeur d et de largeur b. Sa complexité en espace de mémoireest également en $O(b^d)$.

Dans [**Farreny87**], il est démontré que A^* peut effectuer dans le pire des cas, $O(2^N)$ développements d'états. Et pour une heuristique h minorante, la complexité peut être ramené à $O(N^2)$.

En pratique, cela se traduit par la taille très élevée de la liste **OUVERT**. De ce fait, il est impossible de résoudre des problèmes de grande taille sans un espace mémoire important.

Il est démontré que pour un algorithme admissible, il est rarement possible de réduire cette complexité exponentielle [**Pearl90**].

Pour appliquer A* à un problème de taille industrielle, des méthodes d'approximation ont été développées pour réduire cette complexité. C'est ainsi que sont apparus les algorithmes semi-admissibles pour essayer de trouver un bon compromis entre ces deux critères de performance. Par ailleurs l'exigence d'une solution optimale est peu réaliste si l'on considère que le problème réel est souvent représenté par un modèle approximatif. En l'occurrence une "bonne" solution suffira. Donc, il n'est pas nécessaire d'atteindre l'optimalité, mais de s'en approcher à un pourcentage près fixé par l'utilisateur. C'est par ce critère que sont conçus les algorithmes semi-admissibles. Parmi ces algorithmes, nous allons en étudier deux : A_ε^* et $R_{\delta\varepsilon}^*$.

II.4. Algorithmes Semi-Admissibles

Pour l'algorithme A^*, si les fonctions d'estimation de coûts futurs sont parfaites, cette approche nous fera rester à tout instant sur le chemin optimal. Mais les évaluations étant imparfaites, une mauvaise estimation à un endroit quelconque peut nous en faire dévier sans espoir de retour. On propose alors une relaxation de la contrainte d'admissibilité, en développant également les noeuds dont le coût présente un écart ε par rapport au coût optimal, c'est la condition d'ε-*admissibilité*. Cette condition est déjà proposé dans [**Harris74**] dans lequel l'auteur précise que le coût estimé ne doit pas dépasser le coût optimal d'une valeur ε :

$$\forall u \in U,\ h(u) - h^*(u) \leq \varepsilon$$

Equation II. 8

$h(u)$ étant une estimation, au noeud u, du coût optimal h^*. Mais on ne peut pas êtrecertain que cette estimation soit la meilleure.

C'est ainsi qu'une évaluation sous-estimée est utilisée pour être sûr d'atteindre la valeur optimale par valeur inférieure. A l'arrêt de l'algorithme, le coût du chemin ne peut être qu'inférieur ou égal au coût optimal. C'est l'idée de l'algorithme ε-*admissible* A_ε^*.

Cependant, cette contrainte d'admissibilité est trop restrictive. Car, supposons que l'on veuille se rendre d'un lieu *A* à un lieu *B*, il se peut qu'il n'y ait pas de chemin direct aboutissant à B. Mais, on estime qu'il existe deux chemins . L'un est plus court (minorant) par rapport au chemin optimal mais la distance au but est assez loin; et l'autre légèrement plus long (majorant) mais plus proche du but par valeur supérieure. Intuitivement, on choisira le deuxième chemin. Ainsi, choisir une heuristique minorante ne permet pas forcément d'obtenir la meilleure solution sous-optimale. Mais la question est de savoir comment peut-on atteindre la solution optimale par valeur supérieure.

Partant de cette idée, Pearl et *al.* **[Pearl82]** proposent une méthode qui pourra rendre cette condition encore moins restrictive. Ceci aboutit à l'algorithme R_δ^* qui fournit une procédure de recherche généralisée dans les cas où l'on veut considérer cette information d'incertitude sur *h*. Cette incertitude est exprimée en terme de *risque* de manquer la valeur optimale.

II.4.1. Algorithme ε-Admissible A_ε^*

II.4.1.1.Définitions

On dit qu'un algorithme est ε-*admissible* si et seulement si , pour tout ε donné a priori, positif ou nul, il existe un noeud u_{opt} à coût minimal dans l'espace *U* et un noeud *u* se trouvant sur le chemin optimal, tels que :

$$\forall u \in U, f(u) \leq (1+\varepsilon) f^*(u_{Opt})$$

Equation II. 9

ε représente l'écart relatif à l'optimum toléré par l'utilisateur.

Le principe est basé sur la notion d'*acceptabilité*. Donc une deuxième liste nommée **Successeurs_Acceptables** est créée pour contenir tous les noeuds acceptables tels que:

$$\forall u \in Successeurs\_Acceptables,\ f(u) \leq (1+\varepsilon) \times f^*(u_{opt})$$

Equation II. 10

Mais comme $f^*(u_{opt})$ n'est pas connue a priori, on utilise un estimateur minorant; au départ, on estime que le meilleur noeud est le noeud racine u_0. Ensuite, u_{opt} est mis a jour au fur et à mesure de la découverte de meilleurs noeuds. On appelle la quantité $(1+\varepsilon)\times f(u_0)$ le "seuil d'acceptabilité". **Tout état acceptable** est donc développé par A^*_ε.

II.4.1.2. Algorithme A^*_ε

Au départ, l'algorithme développe u_0 par une procédure **developper(*u*)** qui génère l'ensemble des **Successeurs** de *u*. Puis on sélectionne ces successeurs *v* satisfaisant la condition d'ε-admissibilité (Equation II. 9) pour créer l'ensemble des **Successeurs_Acceptables**. Pour les développements suivants, on choisit d'abord un noeud dans cette dernière liste par la procédure **Choix_Dans (*ensemble*)**; si la liste **Successeurs_Acceptables** est vide, on choisit dans **OUVERT**. La liste **SOLUTION** contient les noeuds terminaux u_t.

```
DEBUT
entrées : ε, W, L_kMax

1- INITIALISATIONS.
    1.1) OUVERT ← {u0}.
    1.2) h(u0) ← W*Lkmax.
    1.3) f(u0) ← h(u0) ; "Puisque g(u0)=0, coût du chemin déjà
parcouru"
    1.4) seuil ← (1+ε)×f(u0).
    1.5) SOLUTION ← ∅.
    1.6) developper(u0).
    1.7) Successeur_Acceptable ← { v ∈ Successeur / f(v) ≤
seuil}.

2- BOUCLE PRINCIPALE
  TantQue [ (OUVERT ≠ ∅)&(∀ut∈SOLUTION, f(ut)>seuil)] faire :
    2.1) Si Succ_Accept ≠ ∅ alors u ← Choix_Dans (Succ_Accept)
     sinon u ← Choix_Dans(OUVERT).
```

```
      2.2) developper(u).
      2.3) Successeur ← {unSuccesseur}.
      2.4) Succ_Accept ← { v ∈ Successeur / f(v) ≤ seuil}.
      2.5) ∀(u et v) ∈ OUVERT, si f(u) = f(v), alors supprimer v.
      2.6) ∀u ∈ OUVERT, si f(u) > f(u_t) alors supprimer u.;
      2.7) Si Card(OUVERT) > TailleMax, enlever l'élément queue de
OUVERT.
  FinTantQue.
 3- SORTIE
 sortir le noeud en tête de SOLUTION.
 FIN.
```

Figure II. 2 :L'algorithme de A^*_ε.

II.4.2. Algorithme R^*_δ : Introduction aux Principes

II.4.2.1.Principes

Supposons qu'à un moment de la recherche, l'algorithme vient de trouver un noeud *u* de coût *C*. Quel sera le critère de choix pour développer *u*? L'algorithme développera *u* si celui-ci présente une grande potentialité de réduire considérablement le coût futur, c'est-à-dire s'il n'est pas trop *risqué* de terminer la recherche avec ce coût *C*, sans avoir besoin d'exploiter les possibilités latentes de *u*. Ce qui signifie qu'on peut posséder une information sur le chemin passant par *u*. Cette information est basée sur l'incertitude sur l'estimation de *h** ; elle est représentée par la fonction de densité de probabilité conditionnelle $\rho_{h^*}(x|h)$, qui permet de mesurer la vraisemblance relative que *h** se trouve au voisinage de *x* sachant l'estimation *h* [**Pearl82**]≈le risque de manquer une opportunité de réduction de coût si on abandonne ce neoud *u*. Ce risque est caractérisé par une fonction *R(C)* qui dépend de $\rho_h(.)$ et de *C* pour chaque $\rho_h(.)$ donné Cette fonction de densité sera utilisée pour calculer la fonction de risque à associer à l'estimation du coût d'un noeud. La fonction de risque est définie par l'espérance mathématique d'une variable aléatoire *h* de densité de probabilité $\rho_{h^*}(x|h)$.

II.4.2.2.Fonction de risque

Dans notre cas où $g^*(u) = g(u)$, la fonction de densité de probabilité sur h^* induit la même sur $f^*(u)$ puisque la relation est linéaire.

On a donc:

$$\rho_{f^*}(y|h,g)=\rho_{h^*}(y-g|h) \quad \text{ou pour simplifier :} \quad \rho_{f^*}(y,g)=\rho_{h^*}(y-g)$$

Equation II. 11

On considère alors $f^*(u)$ comme une variable aléatoire représentant le coût du chemin optimal passant par le noeud u de **OUVERT**, et on la note par :

$$\forall u \in \text{OUVERT}, f^+(u) = g(u) + h^*(u)$$

Equation II. 12

et sa fonction de densité de probabilité s'écrit :

$$\rho_{f^+}(y,g)= \rho_{h^*}(y-g)$$

Equation II. 13

A cette fonction de densité, on associe la **fonction risque** $R(C)$ définie pour un noeud u de coût $f(u)$ noté $C(u)$, pour ne pas confondre avec celle de l' A^*_ε, ou C pour simplifier l'écriture des relations.

II.4.2.3.Définition de la fonction de risque

La fonction de risque est définie par l'espérance mathématique de $(C - f^+)$ *si* $C - f^+ > 0$. Plus la valeur de l'espérance sera forte, plus le coût futur a de chance d'être proche de f^*. Elle est définie de la manière suivante :

$$R(C) = \mathrm{E}[\mathbf{max}\,((C - f^{+}), 0)] = \int_{y=-\infty}^{C} (C - y)\rho_{f+}(y)\mathrm{d}y$$

Equation II. 14

Cette fonction mesure le risque de manquer le chemin optimal si on ne développe pas le noeud qui s'y trouve. C'est-à-dire que plus un noeud présentera une valeur de risque élevée, plus il sera choisi en priorité pour le développement.

Si l'on fixe le seuil de risque à δ, à la fin de l'algorithme, le risque de chaque noeud resté dans **OUVERT** doit être inférieur ou égal à δ. C'est la définition d'un algorithme **δ-admissible** ou risque-admissible.

Si un noeud terminal est atteint, on a :

$$\forall u \in \text{OUVERT}, R(C) \leq \delta$$

Equation II. 15

On définit alors le coût-seuil $C_\delta(u)$ relatif à ce risque-seuil δ par la solution de l'équation Equation II. 15, utilisant l'égalité $R(C) = \delta$.

$C_\delta(u)$ représente le meilleur coût acceptable avant que le risque ne tombe en dessous de δ.

II.4.2.4. Critère de choix d'un noeud

L'algorithme utilise donc un paramètre supplémentaire qui est le seuil de risque δ. Plus le risque est élevé pour un noeud, plus il est probable de trouver l'optimum à partir de ce noeud. Mais, il ne faut pas oublier que l'algorithme est dérivé de A^*, qui choisit d'abord par principe les noeuds dont le coût est le plus faible. Le seuil δ permet alors de faire un compromis entre "risque élevé et coût faible".

Chaque noeud *u* est caractérisé par sa fonction risque. Pour un niveau de risque δ donné, l'intersection de cette fonction avec δ fournit le coût $C_\delta(u)$. Par le principe du "meilleur-d'abord", on choisira de développer le noeud *u* dont $C_\delta(u)$ est le plus faible. Dans la liste **OUVERT**, on classera les noeuds par ordre croissant de $C_\delta(u)$.

La **Figure** II. 3 ci-dessous illustre comment ce choix est effectué.

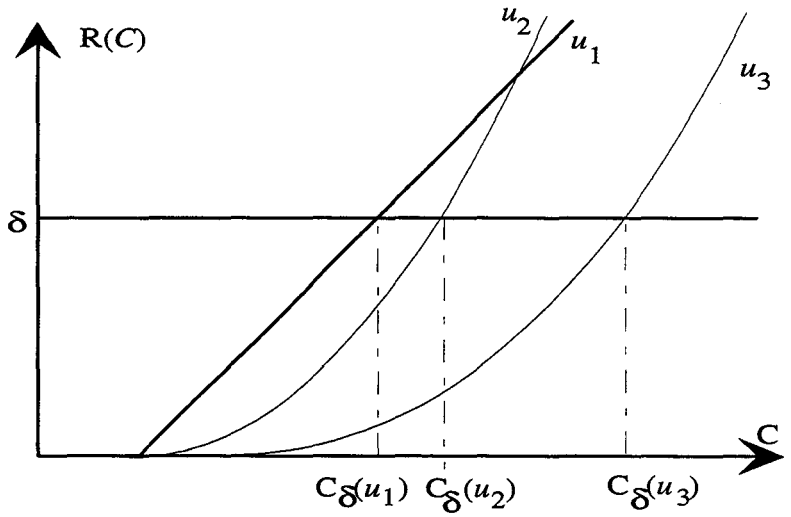


Figure II. 3 : Exemple de choix d'un noeud en fonction R(C). Le noeud u_1 présente le coût le plus faible pour une même valeur de risque, il sera sélectionné. Ce choix n'est pas pénalisant, puisqu'à ce coût, le risque correspondant à u_2 ou u_3 reste faible par rapport au seuil-risque δ.

II.4.2.5. Choix du seuil δ

En fait, il est assez difficile de fixer une valeur à δ. Aussi, on le définit plutôt comme un pourcentage de risque par rapport au coût associé et le coût seuil sera déterminé par l'équation:

$$\frac{R(C)}{C} = \delta \text{ avec } 0 \leq \delta \leq 1$$

Equation II. 16

L'étape la plus délicate est de trouver la fonction qui mesure ce risque, en l'occurrence la fonction de densité de probabilité sur les estimations.

II.4.2.6. Choix de la fonction de densité

Plusieurs types de fonction de distribution, continues ou discrètes, peuvent être utilisés. Dans notre cas d'optimisation combinatoire, il est très courant de rencontrer des fonctions de type discret. Si on utilise l'occupation de surface de matière comme fonction objectif, cette dernière varie plutôt par "sauts" d'une quantité de valeur. Les fonctions discrètes sont mathématiquement difficiles à manipuler même s'il est possible de les formuler autrement de façon à les rendre continues, en pratique, on doit se rappeler que ce sont des *représentations idéalisées.* L'ensemble des points de "sauts" est alors équivalent à l'ensemble des variables aléatoires. Si p représente la probabilité qu'un événement E se produise, et p_k la probabilité d'apparitions successives de cet événement E à l'abscisse k, la fonction de distribution s'écrit alors:

$$F(x) = \sum_{k \le x} p_k \quad (k \text{ entier})$$

Equation II. 17

Si on se ramène au problème d'estimation de h^*, cela représente la probabilité d'obtenir exactement la valeur h^* . Mais comme on ne connaît pas sa valeur exacte, la probabilité p ne peut être connue avec exactitude. Donc, si on suppose pouvoir encadrer la valeur optimale $h^*(u)$ par deux estimations, on peut représenter la densité $\rho_{h^*(x)}$ par une fonction uniforme. De plus, les performances de l'algorithme R_δ^* sont exprimées en termes probabilistes, les simplifications peu éloignées de l'exactitude n'auront que des effets de second ordre sur les résultats fournis par l'algorithme. C'est pourquoi, on décide de prendre une fonction de densité uniforme.

Cette distribution exprime une probabilité de trouver un événement certain à l'intérieur de l'intervalle $[a, b]$. Ses fonctions de densité $f(x)$ et de distribution $F(x)$ sont :

$$f(x) = \begin{cases} \dfrac{1}{b-a} & \text{pour } x \in [a,b] \\ 0 & \text{ailleurs} \end{cases} \qquad \text{avec } F(x) = \int_a^b f(t)\,dt = 1$$

Equation II. 18

L'application au calcul de la fonction de risque nécessite alors des estimations des bornes inférieure et supérieure de la fonction normale.

Soient f_a, une estimation optimiste de $f^*(u)$ et f_b, son estimation pessimiste. C'est-à-dire que pour le choix d'un noeud de coût f_a, le risque de ne pas développer les autres noeuds est nul ; et avec le coût f_b, on atteint la limite du risque-seuil. En supposant que les estimations de coût de chaque noeud *u* sont distribuées uniformément entre f_a et f_b, la fonction de densité s'écrit :

$$\rho_{f^+}(y) = \begin{cases} \dfrac{1}{f_a - f_b} & \text{si } f_a \le y \le f_b \\ 0 & \text{ailleurs} \end{cases}$$

Equation II. 19

Les valeurs de f_a et f_b sont à déterminer selon le problème et la fonction d'estimation utilisée (Voir le paragraphe "Application au placement", p.63).

Pour un noeud *u* de coût *C(u)*, le risque vaut :

$$R(C) = \begin{cases} 0 & \text{pour } C \le f_a \\ \dfrac{(C - f_a)}{2(f_b - f_a)} & \text{si } f_a \le C \le f_b \\ C - \dfrac{f_a + f_b}{2} & \text{si } C \ge f_b \end{cases}$$

Equation II. 20

On résout l'équation :

$$\frac{R(C)}{C} = \delta \quad \text{avec } 0 \le \delta \le 1$$

Equation II. 21

Et en posant $\delta' = 2\delta/(f_b - f_a)$, on obtient le coût seuil :

$$C_\delta(u) = \begin{cases} f_a + \sqrt{2\delta(f_b - f_a)} & \text{pour } 0 \le \delta < \dfrac{f_b - f_a}{2} \\ \delta + \dfrac{f_a + f_b}{2} & \text{pour } \delta \ge \dfrac{f_b - f_a}{2} \end{cases}$$

Equation II. 22

Ainsi, lorsque $\delta = 0$, le coût seuil vaut f_a ; et lorsque $\delta = 1$, le coût vaut f_b. C'est-à-dire qu'on se déplace dans l'intervalle estimé.

La liste **OUVERT** doit être organisée dans l'ordre croissant des $C_\delta(u)$. La recherche se fait alors en "dépilant" de cette liste par le sommet. Et en fin de la recherche, le coût d'un noeud terminal u_t satisfait la condition :

$$g(u_t) \le C_\delta(u_0)$$

Equation II. 23

Puisqu'à l'arrivée, $h(u_t) = 0$, on a donc :

$$C_\delta(u_t) = g(u_t)$$

Equation II. 24

C'est la **condition de δ-admissibilité** de l'algorithme.

Comme l'algorithme A_ε^*, au lieu de passer du temps à sélectionner le meilleur chemin parmi ceux qui sont de coûts assez identiques, il est souhaitable de fixer un écart de tolérance sur la solution sous optimale à trouver. Ainsi permettra-t-on de terminer l'algorithme plus vite avec une solution acceptable dans la limite d'écart ε autorisé. C'est sur quoi est basé l'algorithme $R_{\delta\varepsilon}^*$ présentée dans la section suivante.

II.4.3. L'algorithme $R_{\delta\varepsilon}^*$

Le principe est assez identique à celui de A_ε^*. On détermine d'abord, par les relations de risque, le coût seuil par rapport au risque δ -seuil. Ensuite, seuls les noeuds satisfaisant ce

niveau seuil seront à considérer. Puis, on classe les noeuds dans l'ordre croissant des coûts $C_\delta(u)$ dans **OUVERT**. Une sous-liste appelée **FOCAL** est créée pour contenir les noeuds acceptables par rapport à l'écart ε . C'est-à-dire :

$$\varepsilon = \frac{C_\delta(u) - C_\delta(u_{opt})}{C_\delta(u_{opt})} \qquad \text{avec } 0 \leq \varepsilon \leq 1$$

Equation II. 25

Ainsi, l'algorithme garde la même structure que celle de A^*_ε avec quelques modifications apportées au niveau du calcul du coût et des critères de comparaison.

Pour le noeud racine u_0, l'évaluation des estimations optimistes f_a et pessimistes f_b permet de fixer le coût seuil maximal acceptable. Car, pour les itérations, on cherchera à développer les noeuds dont les coûts $C_\delta(u)$ sont inférieurs ou égal à $(1+\varepsilon)\, C_\delta(u_0)$.

Dans ce choix, l'information sur le risque n'est pas très explicite, car elle est contenue dans la fonction de coût: on ne sélectionne que les noeuds ayant un risque au moins égal à δ , c'est-à-dire ceux dont le coût est le plus faible pour un seuil de risque donné.

II.5. Application au Problème de Placement

Pour la mise en oeuvre des algorithmes, on considère la matière comme ayant une longueur infinie, et sans défauts. Donc, la longueur du tissu sera définie comme une succession de "bandes" dont la longueur de chacune est donnée par celle de la plus grande forme restant à placer. On essaie de remplir au mieux la première bande sur toute la largeur ; ensuite, on passe à la bande suivante, jusqu'à ce qu'il n'y ait plus de pièces à placer.

Pour les deux algorithmes, on doit calculer *g(u), h (u)* et f_a *et* f_b.

Appelons l'état initial u_0, correspondant au placement d'aucune pièce, et un état final qui représente le placement de toutes les pièces possibles sur la bande. Chaque chemin entre l'état u_0 à un état final quelconque est un placement possible. Le coût de l'arc c_{ij} reliant un état u_i à un état u_j, représente la perte produite par le placement de la pièce F_{k+1} associée au placement u_j, (Figure II. 4) :

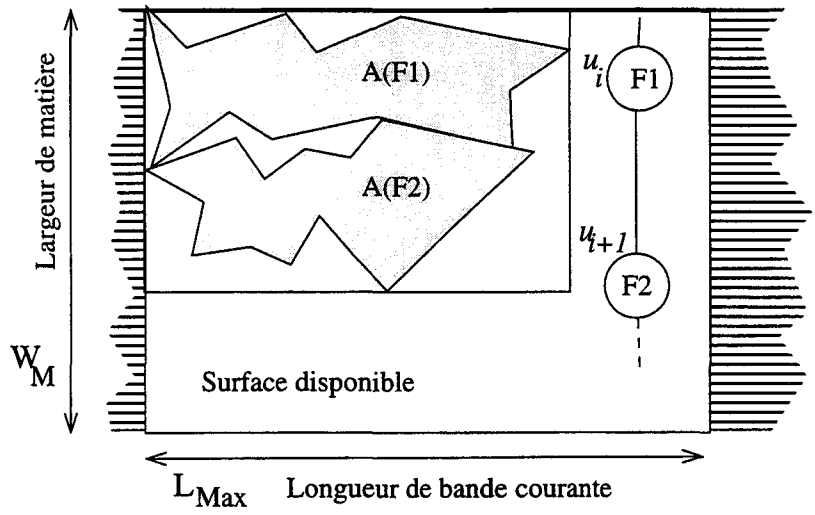


Figure II. 4 : L'état du placement u_j correspond au placement de F_2, générant le coût en terme de surface de perte de matière $c_{i,j} = L_k .W_M - A(F_1) - A(F_2)$.

II.5.1. Calcul de g(u)

La fonction *g(u)* comptabilise les pertes générées jusqu'à l'état actuel. Comme on raisonne en occupation de l'espace, cette perte est décroissante. On l'évalue au fur et à mesure du placement des pièces.

Au départ, aucune pièce n'étant pas placée, la perte est maximale; équivalente à la surface disponible elle vaut la surface de la bande. La perte peut alors être évaluée à chaque instant de manière récursive par la relation:

$$g(u_0) = L_k \times W$$

Equation II. 26

$g(u_i) = g(u_{i-1}) + g_{suppl}(u_i)$, évaluation intermédiaire

Equation II. 27

$$g(u_i) = L_{k^u} \times W - \sum_{\cup F_i \in U_n} A(F_i)$$, évaluation finale

Equation II. 28

$i = 1, 2, 3.. k$, k le nombre total de formes placées sur la bande U_n.

En fin de bande, l'estimation d'un $g(u_{i+1})$ supérieur au $g(u_i)$ permet de détecter un noeud terminal (dépassement de la largeur de bande). Car lorsque la perte engendrée par le noeud fils devient supérieure à celle du noeud père, cela signifie qu'on recommence une nouvelle bande.

La valeur de $g(u)$ à un noeud u intermédiaire étant déterminée, il faut maintenant estimer le coût supplémentaire $h(u)$ pour aller de u à un noeud terminal.

II.5.2. Calcul de $h(u)$

La valeur $h(u)$ doit être choisie de façon à respecter l'admissibilité de l'algorithme A^*_ε. Pour respecter la définition de $f(u)$, on doit avoir $h(u) \leq 0$. Puisque la fonction $f(u)$ est définie comme la perte de matière en fin de bande, $h(u)$ équivaut donc à l'occupation de la surface restante.

On estime que l'espace que l'on va occuper sur la surface disponible équivaut à la surface autour des formes concaténées plus la bande latérale restante (**Figure** II. 5) :

$$h(u) = -[(W - W_u) \times L_{k^u} + A(F_{k^u}, c)]$$

Equation II. 29

avec $A(F_{k^u}, c)$, la surface du peigne côté c de la forme F_{k^u} concaténée avec la forme précédente.

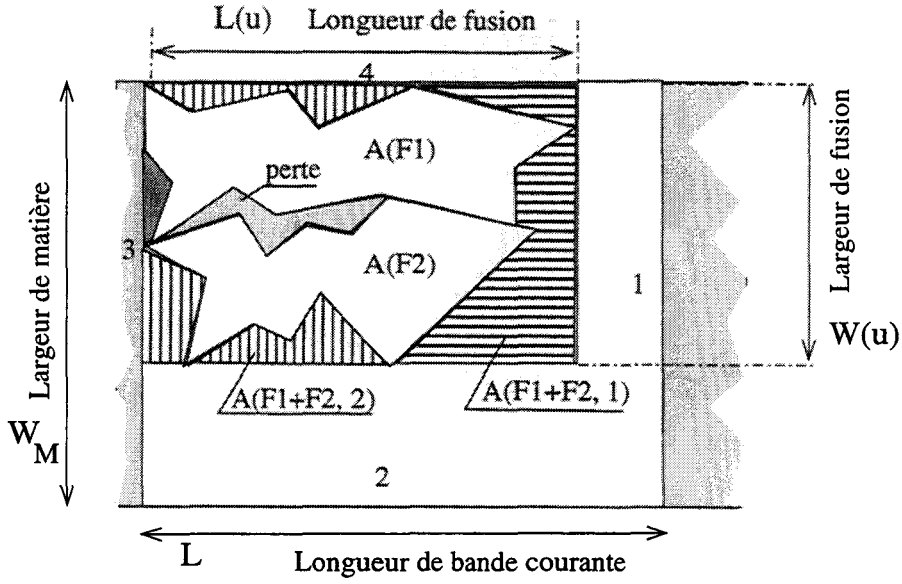


Figure II. 5 : L'estimation de la valeur de h(u) à l'Equation II. 29 est obtenue en enlevant de la surface matière la surface du rectangle circonscrit des formes imbriquées, et la surface des peignes utilisable sur les quatre côtés autour de la forme imbriquée. Comme la valeur de h(u) doit être minorante, on admet que la surface des peignes sur deux côtés est suffisant.

On constate bien sur le dessin que si on considère les quatre $A(F_{k^u}, côté)$, on aurait compté deux fois la surface qui est commune à deux côtés consécutifs. Afin d'être sûr d'avoir un $h(u)$ minorant, c'est-à-dire $|h(u)|$ majorant, pour l'algorithme A^*_ε, soit on ne considère que les côtés 2 et 4, on aura ainsi $h(u)$ minorant ; soit on ajuste la relation $h(u)$ précédente en fonction des types de formes à placer (majorité de grandes formes, ou petites formes), par les paramètre constant *opt*, et une fonction *correcteur(u)*. On obtient la forme définitive de $h(u)$:

$$h(u) = -[(W - W_u) \times L_{k^u} + A(F_{k^u}, c)] \times opt \times \text{correcteur}(u)$$

Equation II. 30

II.5.3. Calcul de f_a et f_b

Pour l'algorithme $R^*_{\delta\varepsilon}$, on doit faire des estimations f_a et f_b qui représentent respectivement les estimations optimiste et pessimiste du coût à l'arrêt de l'algorithme. La meilleure estimation f_a de la perte finale optimiste est de prendre $g(u)$ plus $h(u)$, où $h(u)$ n'est plus nécessairement minorante. Pour estimer f_b, on peut prendre la même considération, mais, afin d'avoir $f_b > f_a$, on ne prendra pas les surfaces $A(F_i, c)$.

$$f_a(u) = g(u) - [(W - W_u) \times L_{k^u} + A(F_{k^u}, c)]$$

Equation II. 31

$$f_b(u) = g(u) - (W - W_u) \times L_{k^u}$$

Equation II. 32

Il faut noter que l'intervalle entre les deux valeurs ne soit pas trop étroit. A cause de la variation discrète des valeurs des fonctions de coût, il est possible de ne trouver aucun noeud dans l'intervalle précisé. De plus les paramètres de sous optimalité δ et ε restreignent encore plus le nombre de noeuds à choisir (**Figure** II. 6).

Une possibilité pour estimer les coûts f_a et f_b serait d'évaluer durant la phase de "pré-traitement" tous les coûts de façon statistique et en extraire le coût moyen. Ensuite on peut fixer un écart-type pour déterminer l'intervalle entre f_a et f_b. Ce type d'approche est souvent utilisé dans les algorithmes de recuit simulé [**Kirkpatrick83, Oliveira93**].

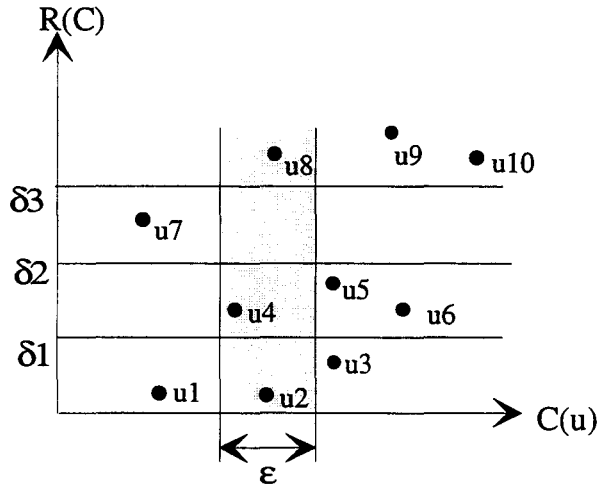


Figure II. 6 : Dans un problème discret, la variation des paramètres de sous-optimalité (ε ou δ) à l'intérieur d'un certain intervalle n'implique pas toujours la variation de l'ensemble de noeuds à choisir.

II.7. Résultats Avec l'Algorithme A_ε^*

Dans les expériences suivantes, les paramètres sont choisis d'abord de façon empirique. Les exemples de placements sont donnés en fin de ce chapitre.

II.7.1. Rendement

Dans le tableau 1 ci-dessous, on observe que pour certains carnet de commande, le rendement augmente globalement lorsque l'écart ε diminue. Cette tendance n'est pas toujours vraie pour d'autres (Figure II. 7-1).

carnet de commande	*ralf.cde*	*chem3.cde*	*gr1.cde*	*kd3.cde*
rendement estimé [1]	**71,2%**	**89,2%**	**88,83%**	**93,5%**
ε = 0	58,6%	**70,6%**	**75,5%**	**79,0%**
ε = 5	**70,4%**	70,5%	75,2%	78,0%
ε = 10	53,0%	70,0%	74,9%	69,0%
ε = 15	53,1%	69,0%	71,6%	69,1%
ε = 20	63,6%	68,7%	71,2%	77,6%
ε = 30	63,6%	68,5%	75,3%	68,7%
ε = 40	63,4%	68,5%	74,1%	78,0%
Ecart moyen	15,71%	22,20%	16,73%	0,64%

Tableau II. 7-1: Résultats avec A_ε^* en fonction de ε.. L'écart moyen est calculé à partir de rendements expérimentaux.

[1] Le rendement est estimé à l'aide de Equation 1.1 et Equation 1.2 du chapitre 1.

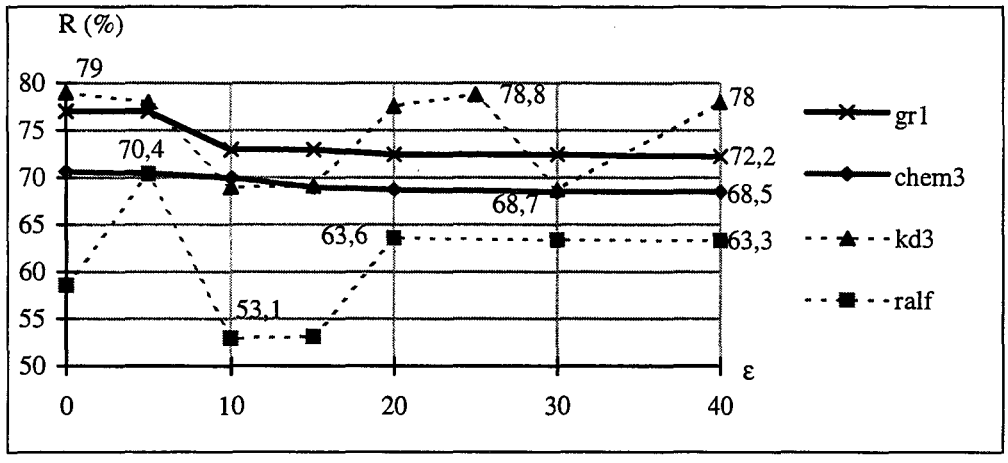


Figure II. 7-2: Evolution du rendement en fonction d'ε des différents carnets.

II.7.2. Complexité

L'augmentation du nombre de noeuds générés avec ε décroissant est observée pour les carnets "*gr1.cde*" et "*ralf.cde*", mais, pour les carnet "*chem3.cde*" et "*kd3.cde*", le nombre de noeuds semble augmenter pour certaines valeurs d'ε (**Figure** II. 7-3), la durée de la recherche en fonction d'ε est suit la même évolution que celle du nombre de noeuds visités (**Figure** II. 7-4).

Cette évolution peut être expliquée par le caractère discret du problème. Il arrive qu'à certain stade de la recherche, l'algorithme peut passer beaucoup de temps à trouver une solution sous-optimale, parce que le reste du stock des formes ne peut pas satisfaire le critère de sous-optimalité ε souhaité.

Pour chaque exécution, la durée de la recherche peut varier légèrement à cause de l'environnement multitâche de Smalltalk sous Unix-Windows™. Le nombre de noeuds visités donne l'image de la durée d'exécution (**Figure** II. 7-5).

L'évolution exponentielle est due au caractère combinatoire du problème de placement. Aussi, la taille du problème ajoute encore plus à la complexité de la recherche car une faible variation en taille du problème accroît très vite le nombre de noeuds visités (**Figure** II. 7-6).

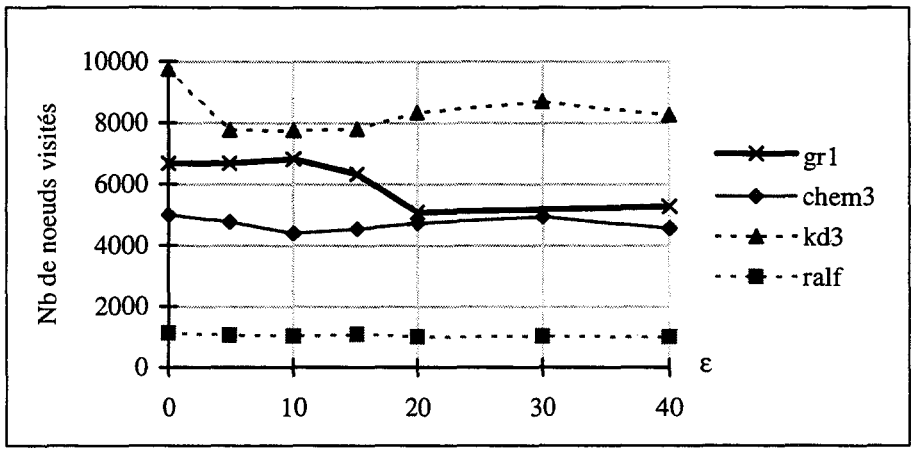


Figure II. 7-3: Nombre de noeuds visités en fonction d'epsilon.

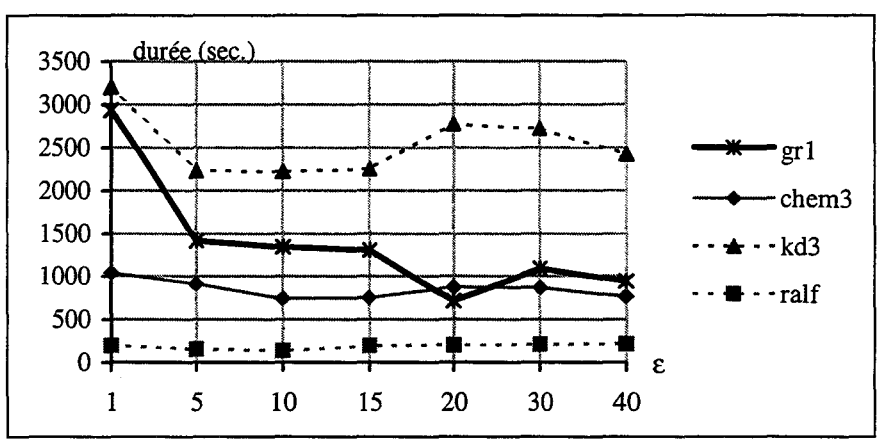


Figure II. 7-4 : Temps d'exécution en fonction d'ε.

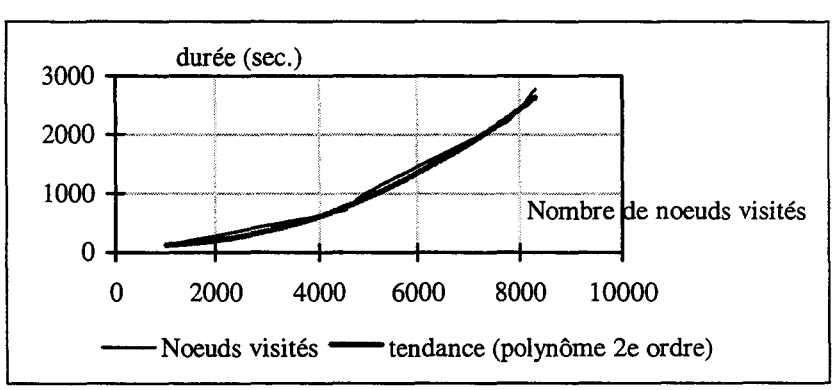


Figure II. 7-5 : Evolution de la durée de calcul en fonction du nombre de noeuds visités.

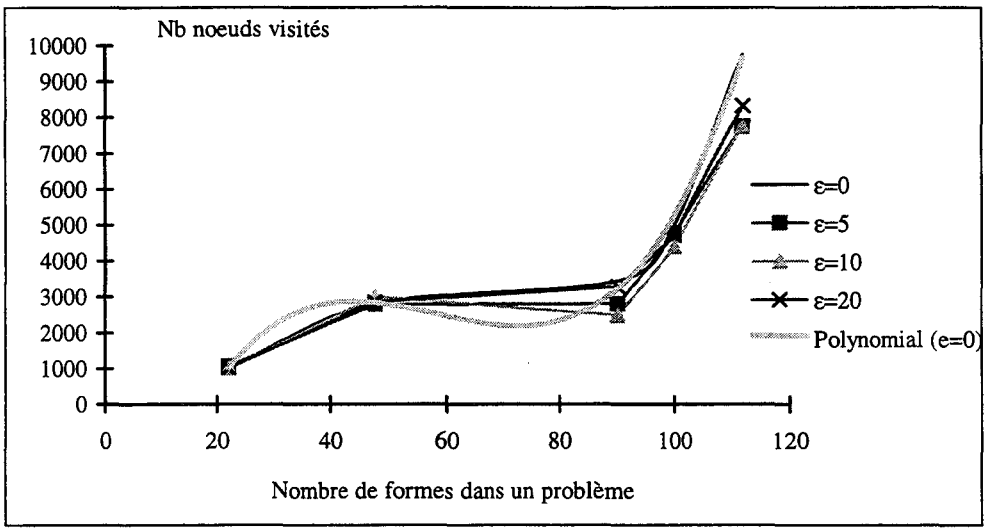


Figure II. 7-6: Evolution du temps de calcul en fonction de la taille du problème.

II.8. Résultats Obtenus avec l'Algorithme $R^*_{\delta\varepsilon}$

Dans cet algorithme, on doit ajuster deux paramètres. La démarche sera d'en fixer un et faire varier l'autre. En fixant le taux de risque δ, et en faisant varier ε, on recherche une solution à $(1+\varepsilon)$ près du coût optimal. Pour chaque carnet, on peut étudier l'influence des paramètre sur le rendement et sur la complexité de la recherche.

Les exemples de placements sont donnés à la fin du chapitre.

II.8.1. Rendement

Pour les trois carnets testés, les courbes (**Figure** II. 8-1, **Figure** II. 8-2, **Figure** II. 8-3) montrent que le rendement devient meilleur pour des ε plus élevés. L'influence du paramètre δ ne semble intervenir sur le rendement que pour certaines valeurs d'ε (**Figure** II. 8-4). Ce phénomène peut être dû au le caractère discret du problème comme nous avions expliqué sur le schéma à la **Figure** II.6.

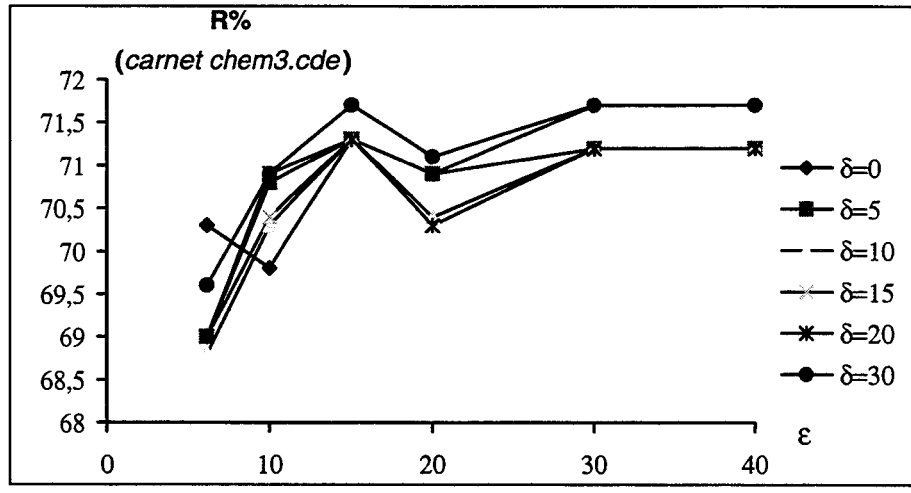


Figure II. 8-1 : Rendement en fonction d'ε et de δ avec le carnet chem3.cde.

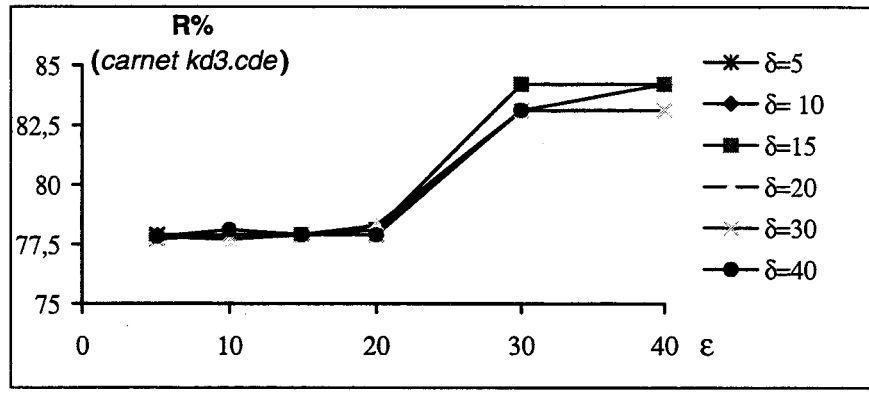


Figure II. 8-2 : Evolution du rendement en fonction de ε et de δ avec le carnet kd3.cde.

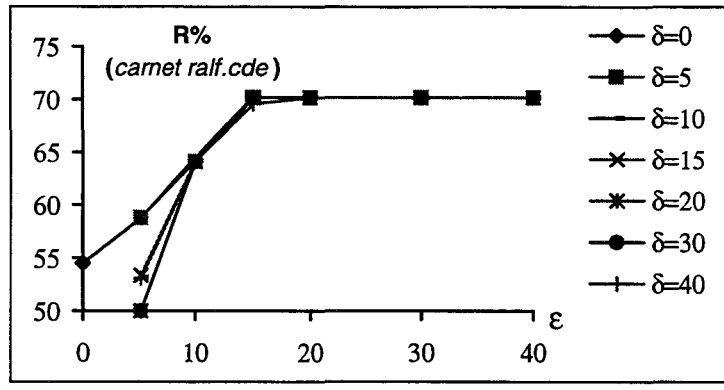


Figure II. 8-3: Evolution du rendement en fonction de ε et de δ avec le carnet ralf.cde.

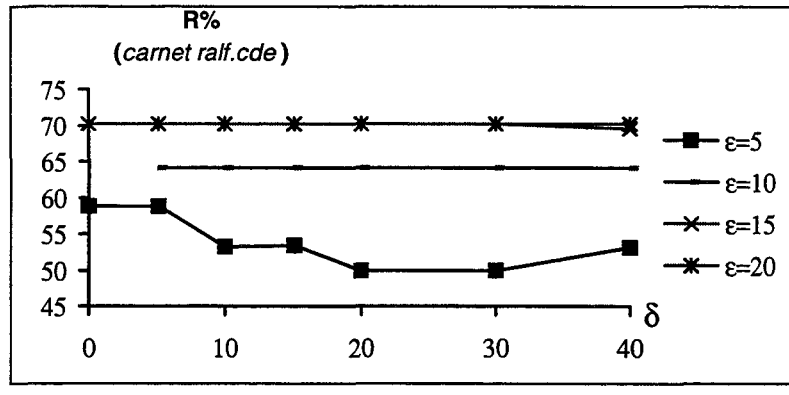


Figure II. 8-4: Evolution du rendement en fonction de ε et de δ avec le carnet ralf.cde..

II.8.2. Complexité

Si on fait tendre l'écart ε vers zéro, le nombre de noeuds générés devient très important (**Figure** II. 8-5, **Figure** II. 8-6). Mais, il faut remarquer que pour certaines valeurs d'ε, la variation du paramètre δ semble beaucoup influencer le rendement (**Figure** II. 8-7). Pourtant, dans certains cas, l'algorithme ne parvient pas à atteindre un noeud terminal. Ceci peut être expliqué par le fait que la taille de la liste OUVERT est limitée. Durant la recherche, on procède à des suppressions de certains noeuds au fur et mesure de l'élagage du graphe. Il arrive qu'un même noeud peut être ouvert plusieurs fois, d'où un nombre croissant de noeuds visités. Alors l'algorithme passe beaucoup de temps à évaluer les noeuds dont le coût pourraient satisfaire les critères d'optimalité demandés. Ce temps est proportionnel au nombre de noeuds évalués

(**Figure** II. 8-8), en plus de l'explosion combinatoire relative à la taille du problème (**Figure** II. 8-9). Il faut noter aussi que l'algorithme $R^*_{\delta\varepsilon}$ passe étrangement beaucoup de temps sur la troisième dernière bande (voir la section II.11.2 Exemples de placements). Ce comportement de l'algorithme nous est inexplicable. On pourrait tenter une explication par le fait que l'espace de recherche réduit à l'ensemble de formes restantes comporte beaucoup de minima locaux et les fonctions d'estimation utilisées ne sont plus adaptées pour la nouvelle forme de cet espace. Pour ce genre de comportement, il faudrait introduire à l'algorithme une notion d'adaptation.

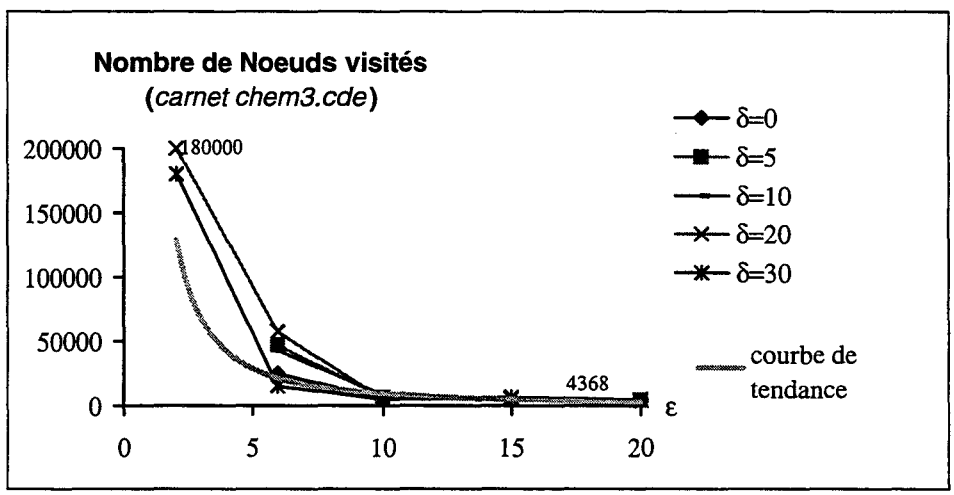


Figure II. 8-5 : Evolution de nombre de noeuds visités en fonction de ε .

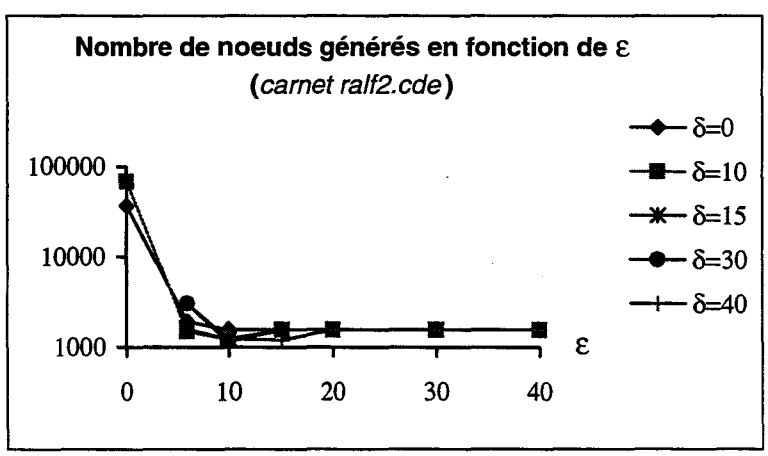


Figure II. 8-6 : Nombre de noeuds visités en fonctions de ε et δ avec le carnet ralf.cde.

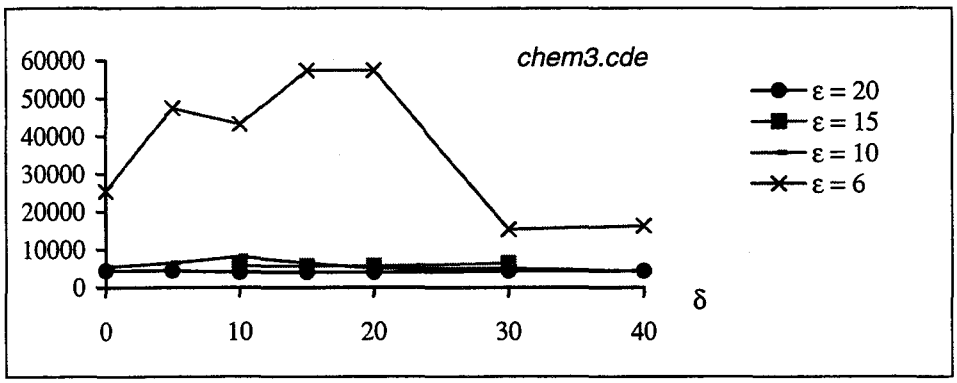


Figure II. 8-7 : Evolution de nombre de noeuds visités en fonction de δ.

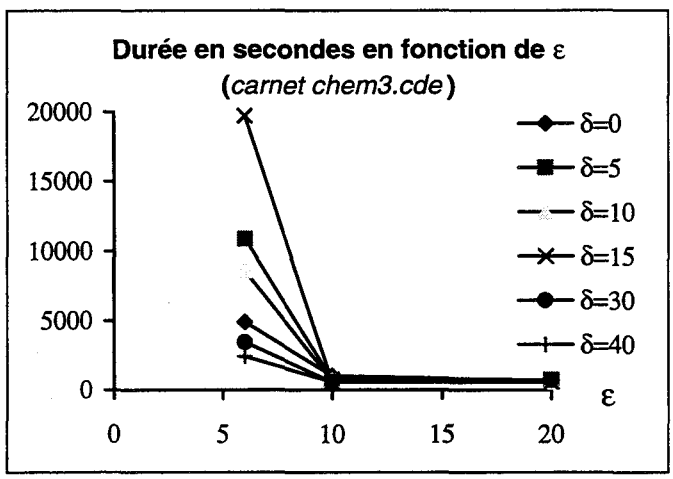


Figure II. 8-8 : Durée de la recherche en fonction des facteur ε et δ.

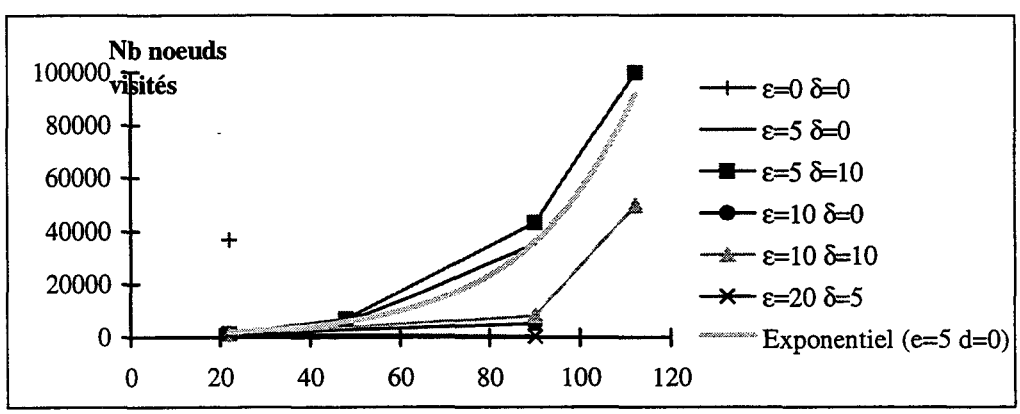


Figure II. 8-9: Evolution exponentielle du nombre de noeuds visités en fonction du nombre de formes dans un carnet.

II.8.3. Commentaire

Les graphiques montrent que si on fait tendre l'écart ε vers zéro, le nombre de noeuds visités devient très important, sans pour autant fournir de bon rendement, puis qu'on a vu que le rendement est meilleur avec des ε relativement élevés. Pour certains cas, l'algorithme ne parvient pas à atteindre un noeud terminal alors que le temps de recherche devient de plus en plus important. Deux explications possibles sont la limitation de la taille de la liste des noeuds exploités et le caractère discret du problème de placement.

Les listes OUVERT et FERME sont limitées à un nombre maximal de noeuds. Lorsque vers la fin de la recherche, l'algorithme stagne parce qu'il ne parvient pas à trouver une solution satisfaisant les critères d'optimalité demandés, le nombre de noeuds testés augmente et donc la taille des listes des noeuds. Certains noeuds excédents supprimés des listes peuvent réapparaître et réévalués. La gestion des listes FERME et OUVERT est un problème général pour les algorithmes de recherche en arbre **[Cung94]**.

La variation discrète des entrées peut causer la stagnation de la recherche. Lorsque qu'il n'existe aucune forme qui pourrait remplir la surface restante et que critères demandés ne sont pas encore atteints, l'algorithme peut également stagner.

Mais, également les définitions de nos fonctions f_a et f_b ne sont pas parfaites ainsi que l'utilisation de la densité normale pour h. La fonction h représente les "gains" futurs qui viennent réduire les pertes actuelles représentées par g dans la fonction totale f. Mais le gain futur, qui est égal à la surface réelle de la forme ajoutée, s'accompagne de perte supplémentaire, qui correspond aux surfaces inutilisables générées par la configuration du placement. Donc choisir une forme à placer F_k de surface $A(F_k)$ n'implique pas forcément une réduction de pertes de quantité $A(F_k)$ La surestimation de ces pertes permet d'être sûr de pouvoir placer une forme et la forme choisie sera sûrement incluse dans l'espace libre estimé. Mais si la surface libre permet de contenir exactement cette forme, mais son heuristique est légèrement supérieure à la valeur admissible, l'algorithme admissible peut manquer cette opportunité. Alors que si l'on s'approche par valeur supérieure, cette configuration serait acceptable. Accepter les surcoûts implique aussi l'augmentation l'espace de configurations. Enfin, les paramètres des algorithmes étaient choisis de façon empirique, alors qu'il serait plus judicieux de trouver des valeurs spécifiques au problème. Ceci n'est pas toujours facile à déterminer.

II.9. Comparaison des Deux Techniques en Arbre

Dans cette partie nous récapitulons quelques résultats pour la comparaison. Il est assez difficile de comparer les deux algorithmes car l'ajustage des paramètres influence aussi la qualité du résultats. Toutefois, on peut extraire des résultats précédents les meilleurs rendements

obtenus et comparer les coûts en temps de calcul et en espace mémoire. Le tableau ci-dessous (Tableau II. 9-1) récapitule le meilleur rendement obtenu pour chaque carnet. Les mêmes données sont représentées graphiquement à la **Figure** II. 9-1.

	Algorithme A^*_ε		Algorithme $R^*_{\delta\varepsilon}$	
Carnet	meilleur rendement obtenu(%)	temps	meilleur rendement obtenu (%)	temps
ralf.cde	70.4 ($\varepsilon = 5$)	2'53"	70.2 ($\varepsilon = 20, \delta = 0$)	6'27"
chem3.cde	70.6 ($\varepsilon = 0$)	17'17"	71.1 ($\varepsilon = 20, \delta = 30$)	11'44"
gr1.cde	77.0 ($\varepsilon = 5$)	48'7"	76.3 ($\varepsilon = 15, \delta = 20$)	1h35'37"
kd3.cde	79 ($\varepsilon = 0$)	54'28	78.3 ($\varepsilon = 20, \delta = 20$)	2h03'11"

Tableau II. 9-1: Récapitulatif de quelques résultats.

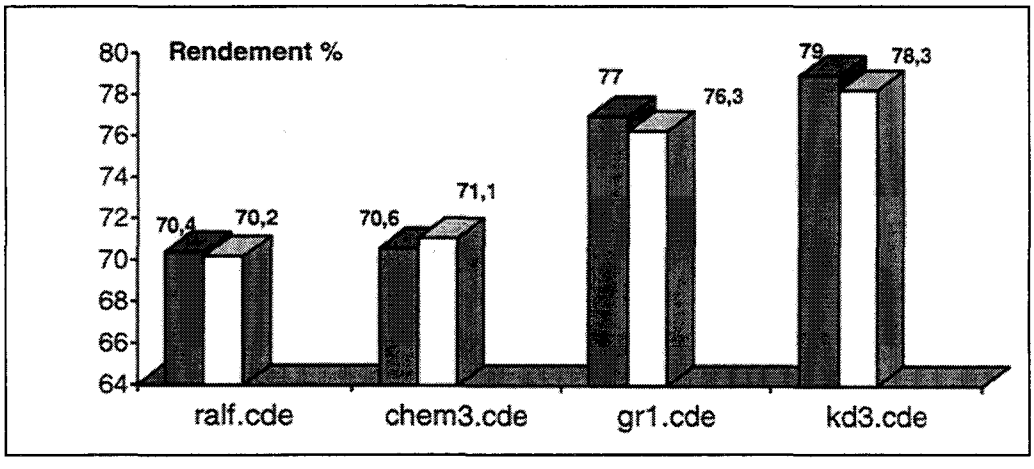


Figure II. 9-1: Comparaison du meilleur rendement obtenu pour chaque carnet.Les barres à gauche représentent les résultats par A^*_ε et à droite ceux de $R^*_{\delta\varepsilon}$.

Les deux algorithmes sont très compétitifs. L'algorithme A^*_ε fournit de meilleurs résultats, sauf sur le carnet chem3.cde où l'algorithme $R^*_{\delta\varepsilon}$ semble être légèrement meilleur en rendement et en temps de calcul (**Figure II.** 9-2).

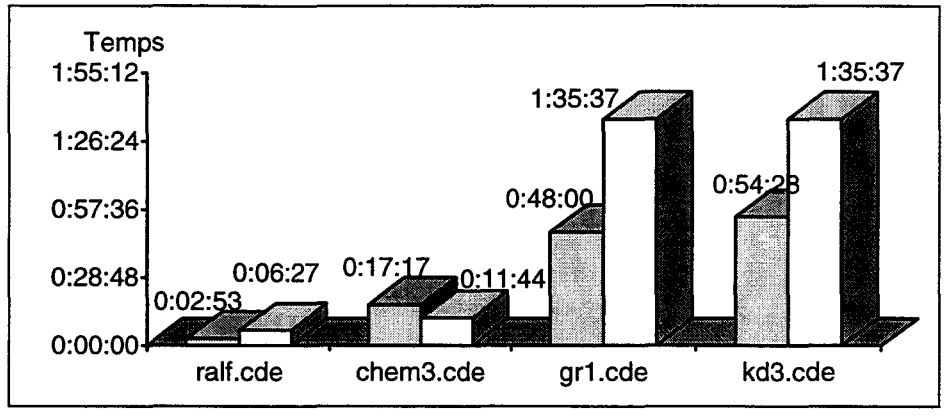


Figure II. 9-2: Durée globale de recherche pour les rendements précédents.

Il faut cependant mentionner que l'importance du temps de calcul global de l'algorithme $R^*_{\delta\varepsilon}$ (surtout pour les carnet *gr1.cde* et *kd3.cde*) est surtout due à l'anomalie montrée sur les exemples de placement à la **section II.11.** Cette anomalie concerne l'étrange comportement de l'algorithme $R^*_{\delta\varepsilon}$ qui passe énormément de temps sur les troisième et quatrième bande, alors que la taille de l'espace de recherche est déjà beaucoup réduite. Il nous est difficile d'expliquer ce comportement inattendu, mais nous suspectons l'inadéquation des fonctions d'estimation. En effet, il est possible qu'à ce stade, l'espace de recherche comporte des minima locaux assez similaires et la fonction d'évaluation ne permet de distinguer explicitement les noeuds.

Quant à la complexité par rapport à la taille du problème, l'algorithme $R^*_{\delta\varepsilon}$ est plus exponentiel que A^*_{ε} (**Figure** II. 9-3).

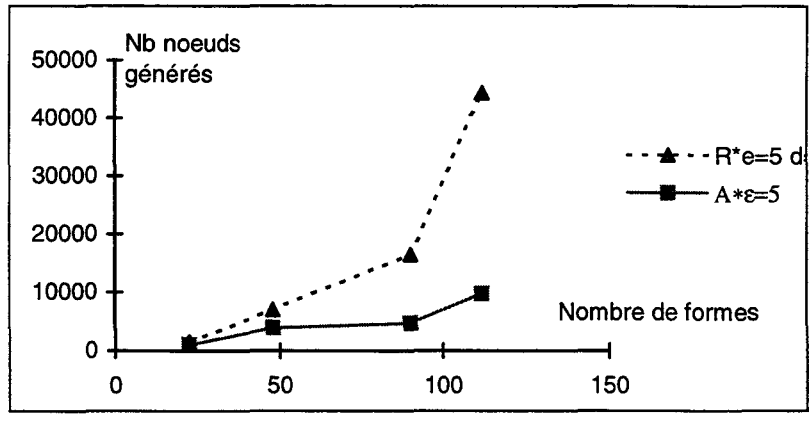


Figure II. 9-3 Comparaison du nombre de noeuds développés par des deux algorithmes pour les rendements du tableau **Erreur! Source du renvoi introuvable.**.

Ces résultats globaux ne donnent pas vraiment le comportement en détail des deux algorithmes. Par exemple, on remarque que $\mathbf{A}^*_\varepsilon$ a tendance à placer plus de petites formes que $R^*_{\delta\varepsilon}$ place plus de grandes formes (Figure II. 9-4, Figure II. 9-5). Parfois, il n'est pas nécessaire d'obtenir le meilleur rendement sur la première bande, car la répartition des formes peut être déséquilbrée dans le placement global. Ceci montre que l'estimation de h par valeur inférieure favorise la progression de la recherche par petites valeurs (ici la diminution de la perte) et vice versa, la surestimation favorise le placement de grandes formes. Toutefois, cette surestimation ne permet pas toujours d'aboutir à une meilleure solution . Dans notre cas, ceci s'explique par le fait que la bande est limitée en longueur et en largeur, si bien que l'algorithme génère beaucoup de solution invalides par rapport à ces contraintes de dimensions et l'algorithme effectue beaucoup de backtracking. Il serait intéressant de tester l'algorithme sur un placement non limité en dimension de la matière (c'est le cas du placement de composantes dans un circuit imprimé où le but est de compacter au maximum).

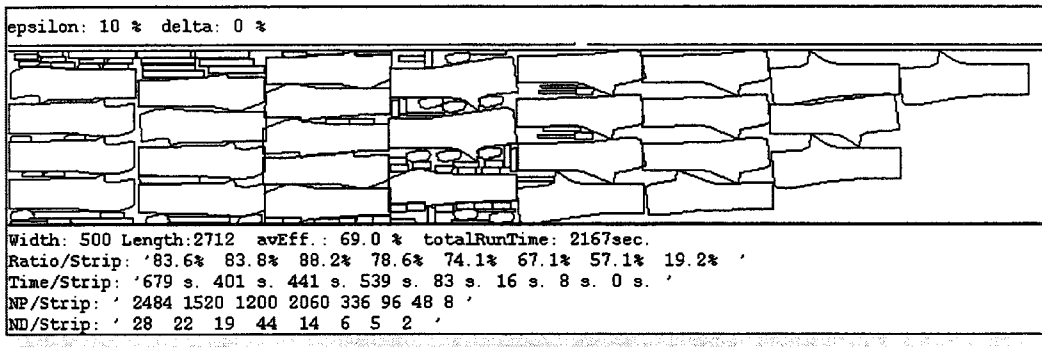


Figure II. 9-4: Exemple de placement avec $\mathbf{A}^*_\varepsilon$. On remarque dans les premières bandes que l'algorithme favorise le placement des petites formes.

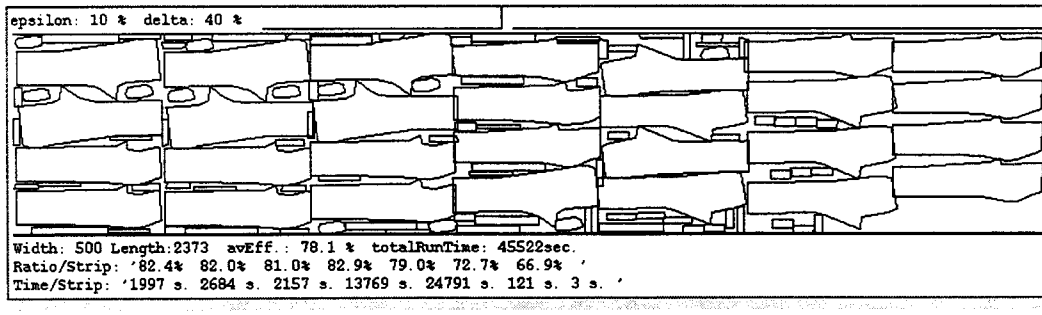


Figure II. 9-5: Exemple de placement avec $R^*_{\delta\varepsilon}$. On remarque dans les premières bandes que l'algorithme favorise le placement des grandes formes.

II.10. Conclusion sur les recherches heuristiques en arbre

Globalement, on observe que l'algorithme $R^*_{\delta\varepsilon}$ examine plus de noeuds que A^*_ε puisque les noeuds non admissibles sont également considérés. L'efficacité est très dépendante de l'ajustage des paramètres peut ajuster les paramètres. Avec deux paramètres à ajuster, on restreint encore plus les noeuds à développer à chaque étape. Si la liste des noeuds classés par C_δ est vide, on prend dans OUVERT, ce qui revient à utiliser A^*_ε.

Avec ces remarque, on pourrait nous poser la question "pourquoi utiliser $R^*_{\delta\varepsilon}$?". La première réponse est que nous voulions relaxer la contrainte d'admissibilité de A^*_ε et permettre d'accepter aussi les noeuds non admissibles, mais dont le coût dépasse légèrement la valeur optimale. Appliqué au problème de placement, il s'avère qu'il n'est pas facile d'estimer cette valeur optimale et qu'il n'est pas non plus facile de prévoir le comportement de l'algorithme à cause du caractère discret du problème.

Toutefois, nous pensons que la notion du risque est à considérer dans le cas où l'algorithme A^*_ε ne trouve plus aucune solution admissible et qu'il faut alors "regarder" au-delà des limites de l'admissibilité. La précision de $R^*_{\delta\varepsilon}$ est également fortement dépendante de la fonction de densité de probabilité et des intervalles définis par f_a et f_b. Il serait plus profitable de définir ces fonctions avec des données statistiques au cours d'une phase de pré-traitement avec différents types de carnets.

En effet, il est probable que, arrivé à ce stade de recherche avec une nouvelle forme de l'espace de recherche, l'algorithme a la difficulté de distinguer entre les " bons" et les moins bons noeuds, si bien que tous les noeuds pourraient être développés, et vraisemblablement, à cause de la taille fixe des listes OUVERT, FOCAL, FERME, certains noeuds pourraient être évalués plusieurs fois. Pour remédier à ce problème, on pourrait utiliser la stratégie de tabou. Dans cette dernière méthode, les noeuds menant aux échecs sont enregistrés dans une liste "taboue", et lors de la recherche si le même type de noeud se présente, il ne sera pas développé. Mais nous pensons que cette stratégie de taboue fait encore intervenir une liste supplémentaire et la gestion de cette liste ajoute un temps de traitement supplémentaire. La technique du recuit simulé pourrait apporter une solution dans le sens où la faible température permet déjà de limiter l'acceptation d'un certains nombre de configurations.

En résumé, les algorithmes dérivés A^* sont nombreux et reste encore populaire, car son principe est simple et efficace pour des problème de petite taille. Mais dans les problèmes à grande taille, A^* nécessite beaucoup d'espace mémoire pour stocker les noeuds dans OUVERT

et FERME, puisque TOUS les noeuds admissibles pour être développés doivent être effectivement développés.

Les améliorations actuelles visent surtout à diminuer la complexité tant spatiale que temporelle, soit gérant mieux l'espace mémoire [**Ghosh91**, **Cung94**], soit en utilisant une machine parallèle [**Arenales95**, **Daza95**], soit encore, on se tourne vers les algorithmes stochastiques tels que le recuit simulé que nous verrons dans le chapitre suivant.

II.10.1. A^*_ε

II.10.1.1. Carnet gr1.cde

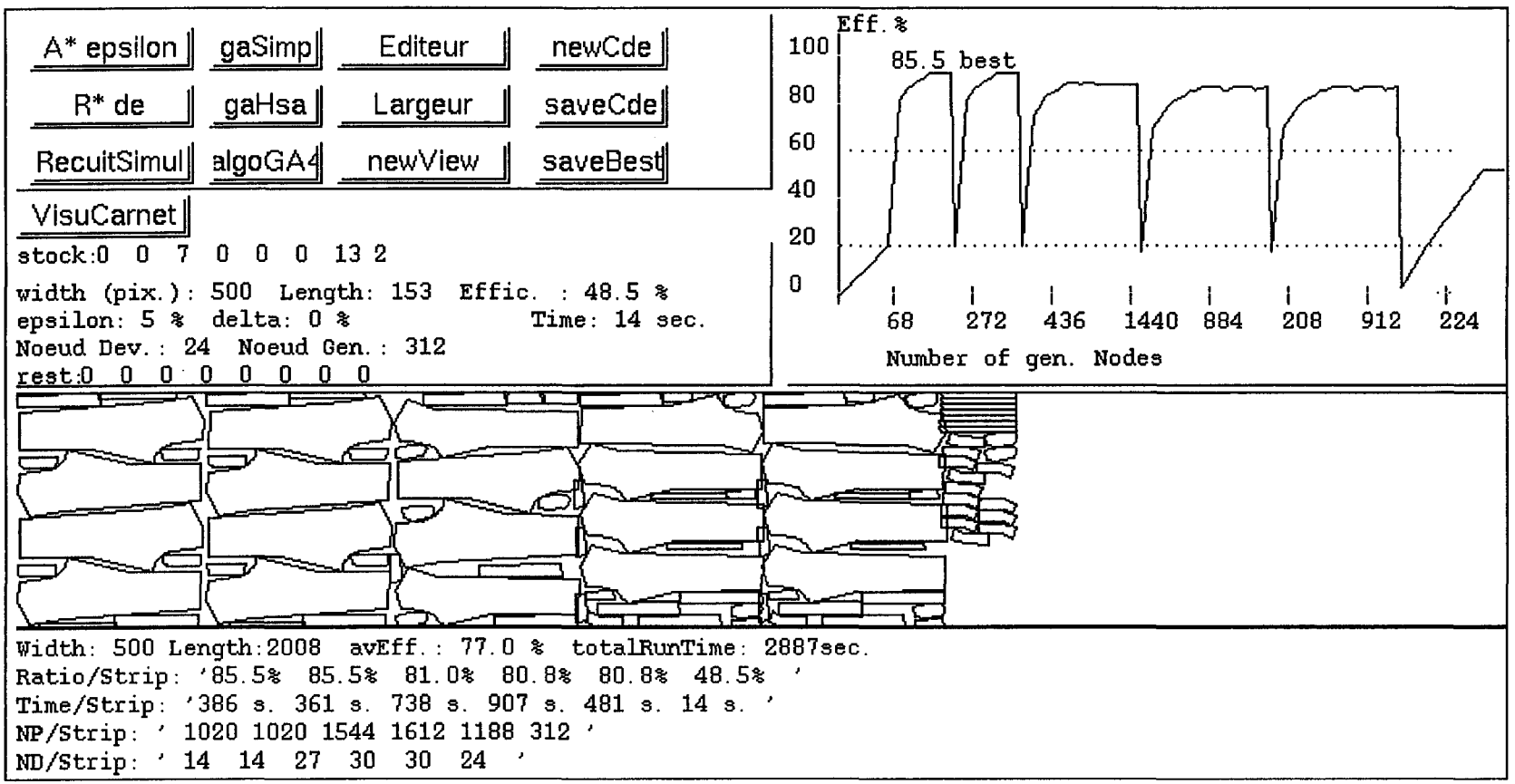
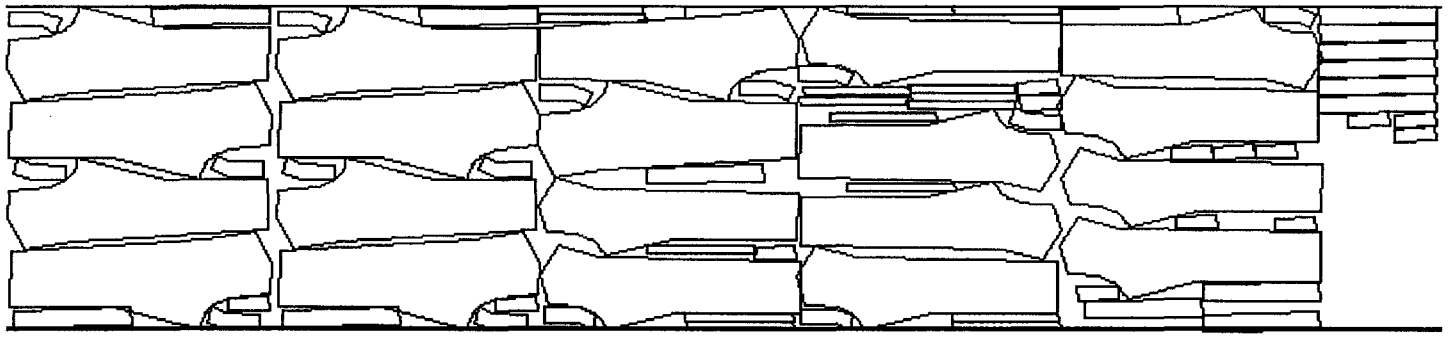
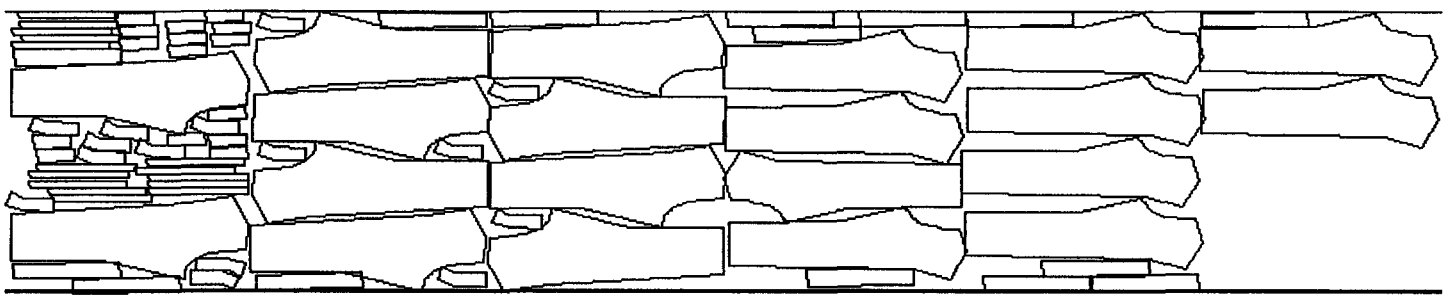


Figure II. 10-1: Interface visuelle de placement. La figure montre ici le placement de carnet gr1.cde avec $\varepsilon = 5$.



```
Width: 490 Length:2023  avEff.: 74.9 %  totalRunTime: 1342sec.
Ratio/Strip: '85.7%  85.7%  81.4%  80.9%  79.6%  36.1%  '
Time/Strip: '270 s. 239 s. 437 s. 288 s. 105 s. 3 s. '
```

Figure II. 10-2: Carnet gr1.cde avec $\varepsilon = 10$.



```
Width: 490 Length:2226  avEff.: 71.2 %  totalRunTime: 713sec.
Ratio/Strip: '78.0%  84.4%  80.9%  74.7%  74.3%  35.1%  '
Time/Strip: '359 s. 159 s. 64 s. 75 s. 54 s. 2 s. '
```

Figure II. 10-3: Carnet gr1.cde avec $\varepsilon = 20$.

II.10.1.2.Carnet *chem3.cde*

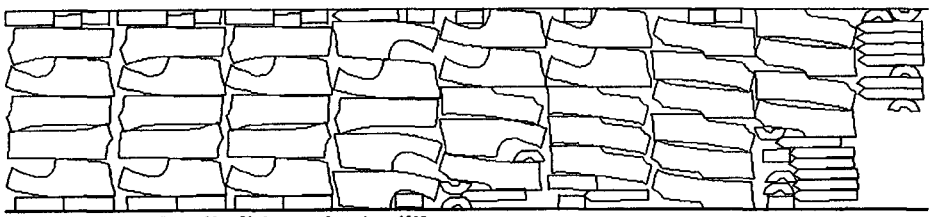


Figure II. 10-4 : Carnet chem3.cde avec $\varepsilon = 0$.

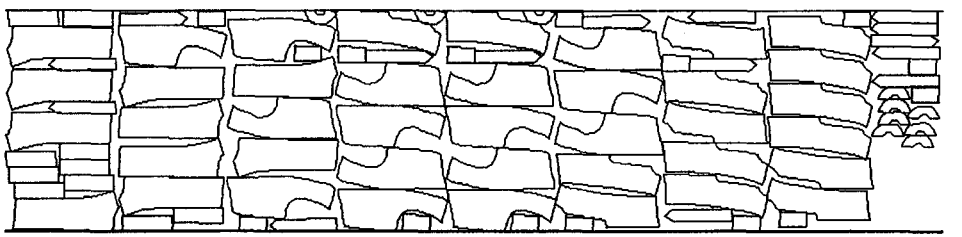


Figure II. 10-5: Carnet chem3.cde avec $\varepsilon = 10$.

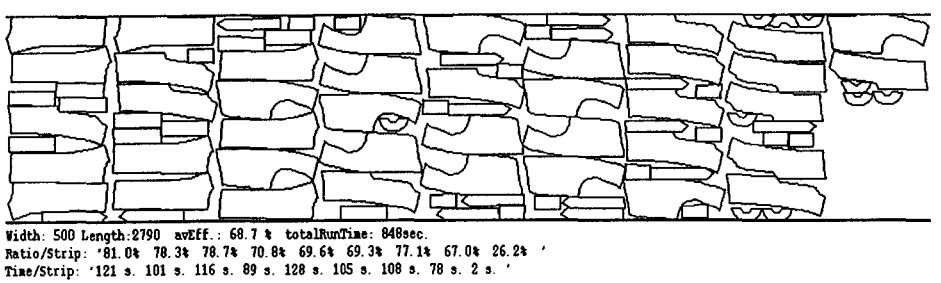


Figure II. 10-6: Carnet chem3.cde avec $\varepsilon = 20$.

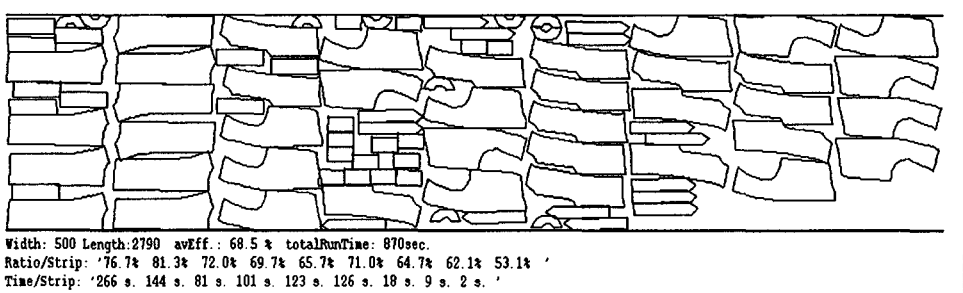
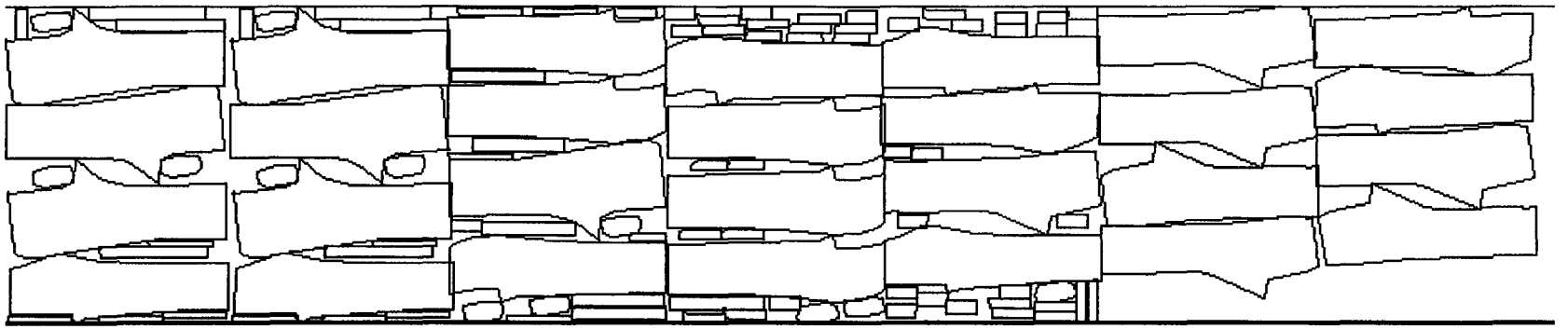


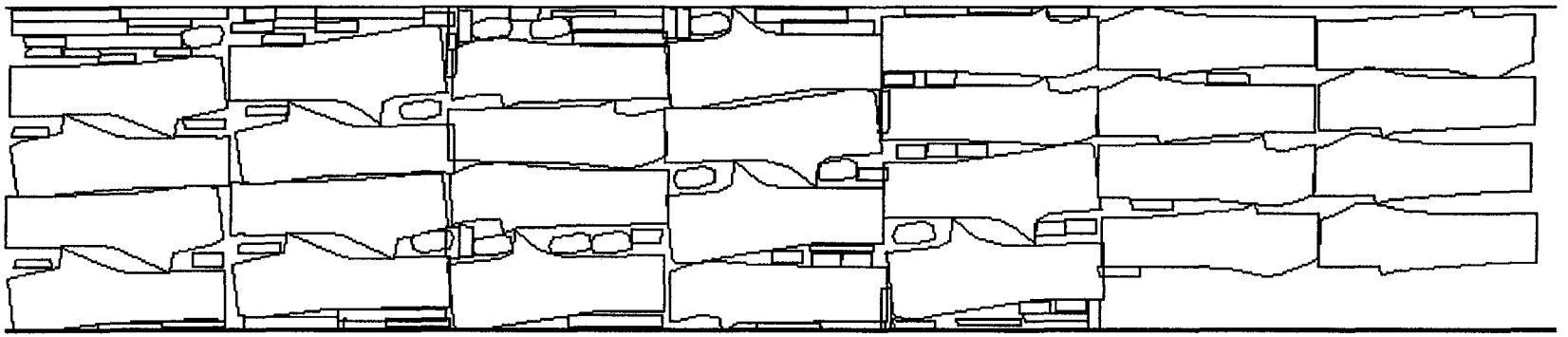
Figure II. 10-7: Carnet chem3.cde avec $\varepsilon = 30$.

II.10.1.3.Carnet *kd3.cde*



```
Width: 500 Length:2373  avEff.: 79.0 %  totalRunTime: 3268sec.
Ratio/Strip: '84.8%  84.8%  86.2%  82.5%  80.7%  67.5%  66.5%  '
Time/Strip: '814 s. 671 s. 596 s. 763 s. 395 s. 26 s. 3 s. '
```

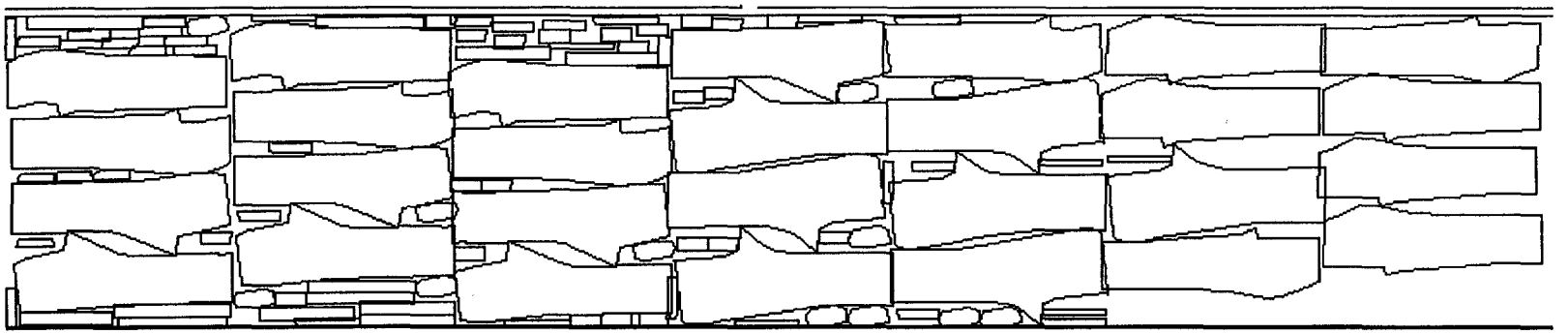
Figure II. 10-8: Carnet kd3.cde avec $\varepsilon = 0$.



```
Width: 500 Length:2373  avEff.: 78.0 %  totalRunTime: 2822sec.
Ratio/Strip: '82.4%  81.3%  83.4%  84.1%  81.9%  67.6%  65.2%  '
Time/Strip: '682 s. 488 s. 547 s. 687 s. 382 s. 34 s. 2 s. '
```

Figure II. 10-9: Carnet kd3.cde avec $\varepsilon = 10$.

epsilon: 20 % delta: 0 %

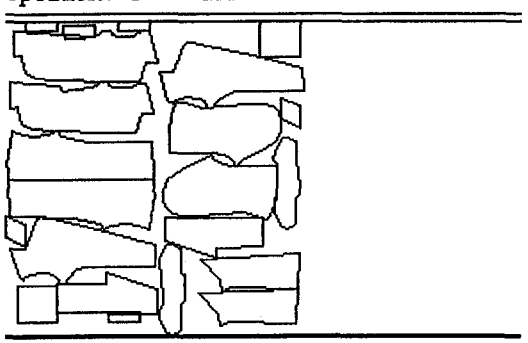


Width: 500 Length:2373 avEff.: 77.6 % totalRunTime: 3182sec.
Ratio/Strip: '83.5% 82.9% 81.6% 82.7% 79.2% 68.2% 65.2% '
Time/Strip: '929 s. 689 s. 782 s. 514 s. 242 s. 24 s. 2 s. '
NP/Strip: ' 2484 1764 1996 1308 644 104 32 '
ND/Strip: ' 28 23 28 22 20 8 6

Figure II. 10-10: Carnet kd3.cde avec $\varepsilon = 20$.

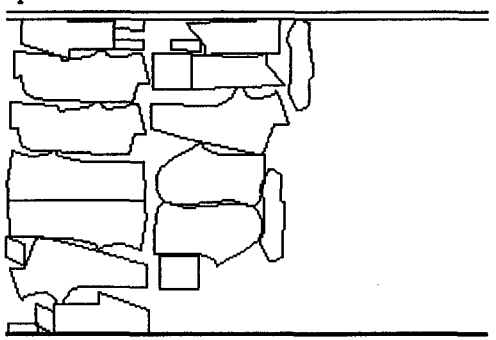
II.10.1.4.Carnet *ralf.cde*

epsilon: 5 % delta: 0 %

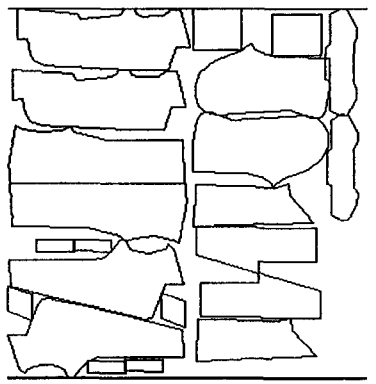


Width: 500 Length:444 avEff.: 70.4 %
Ratio/Strip: '74.5% 66.3% '
Time/Strip: '158 s. 15 s. '
NP/Strip: ' 816 256 '
ND/Strip: ' 14 13 '

epsilon: 10 % delta: 0 %



Width: 500 Length:483 avEff.: 53.0 %
Ratio/Strip: '77.9% 58.8% 22.3% '
Time/Strip: '120 s. 13 s. 0 s. '
NP/Strip: ' 708 296 8 '
ND/Strip: ' 13 14 2 '



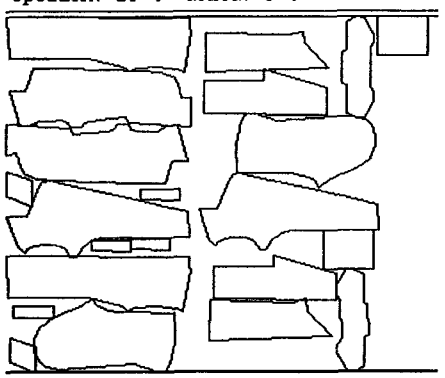
Width: 500 Length:433 avEff.: 63.6 %
Ratio/Strip: '77.9% 69.4% 43.4% '
Time/Strip: '195 s. 13 s. 0 s. '

Figure II. 10-11: Carnet ralf.cde avec resp. $\varepsilon = 5, 10, 20$.

II.10.2. $R^{*}_{\delta\epsilon}$

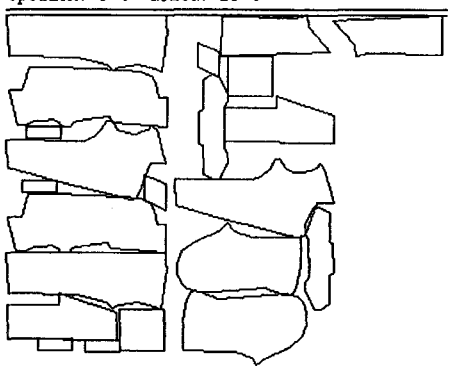
*II.10.2.1.*Carnet *ralf.cde*

epsilon: 10 % delta: 0 %



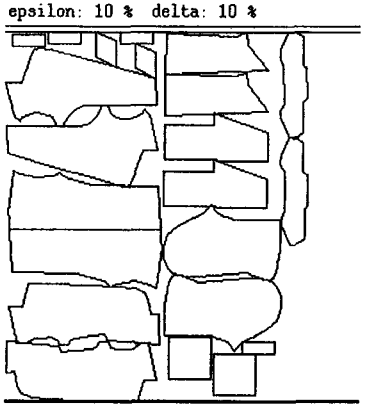
Width: 500 Length:506 avEff.: 50.3 %
Ratio/Strip: '79.3% 60.3% 11.4% '
Time/Strip: '301 s. 11 s. 0 s. '
NP/Strip: ' 1300 264 4 '
ND/Strip: ' 24 11 2 '

epsilon: 0 % delta: 10 %



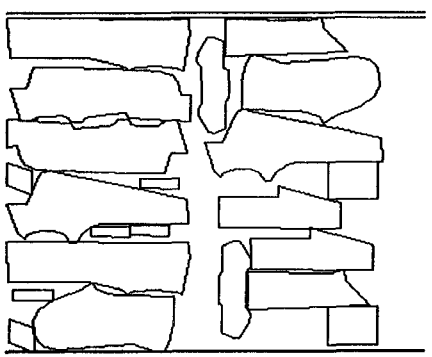
Width: 500 Length:596 avEff.: 48.6 %
Ratio/Strip: '76.8% 60.1% 9.0% '
Time/Strip: '2701 s. 2767 s. 0 s. '
NP/Strip: ' 11628 56940 8 '
ND/Strip: ' 226 2223 3 '

epsilon: 10 % delta: 10 %



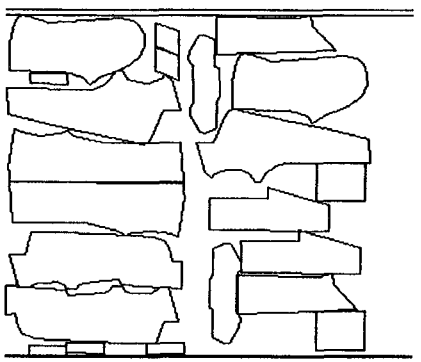
Width: 500 Length:433 avEff.: 64.1 %
Ratio/Strip: '77.6% 71.4% 43.4% '
Time/Strip: '183 s. 31 s. 1 s. '
NP/Strip: ' 808 416 16 '
ND/Strip: ' 16 23 4 '

epsilon: 20 % delta: 0 %



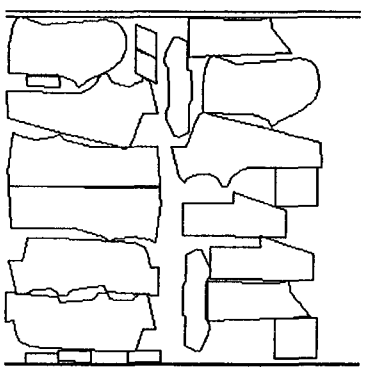
Width: 500 Length:444 avEff.: 70.2 %
Ratio/Strip: '79.3% 61.0% '
Time/Strip: '373 s. 14 s. '
NP/Strip: ' 1300 264 '
ND/Strip: ' 24 13 '

epsilon: 15 % delta: 40 %



Width: 500 Length:444 avEff.: 69.6 %
Ratio/Strip: '78.2% 61.0% '
Time/Strip: '272 s. 14 s. '
NP/Strip: ' 940 264 '
ND/Strip: ' 17 13 '

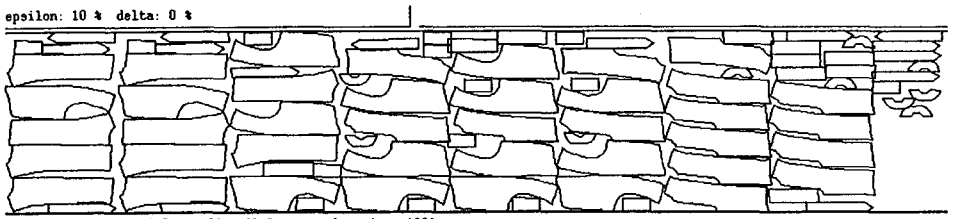
epsilon: 15 % delta: 50 %



Width: 500 Length:444 avEff.: 69.6 %
Ratio/Strip: '78.2% 61.0% '
Time/Strip: '230 s. 13 s. '
NP/Strip: ' 940 264 '
ND/Strip: ' 17 13 '

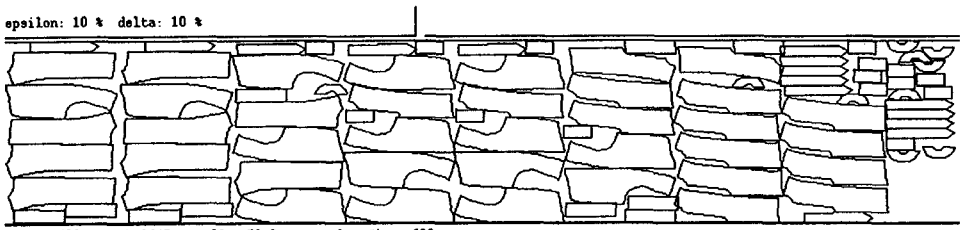
*II.10.2.2.*Carnet *chem3.cde*

epsilon: 10 % delta: 0 %



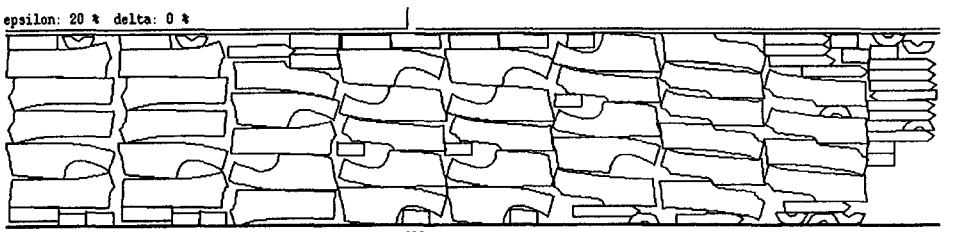
Width: 500 Length:2647 avEff.: 69.8 % totalRunTime: 1038sec.
Ratio/Strip: '78.8% 78.8% 78.4% 71.2% 75.9% 72.6% 75.0% 74.0% 23.1% '
Time/Strip: '106 s. 120 s. 112 s. 347 s. 124 s. 138 s. 39 s. 45 s. 7 s. '

epsilon: 10 % delta: 10 %



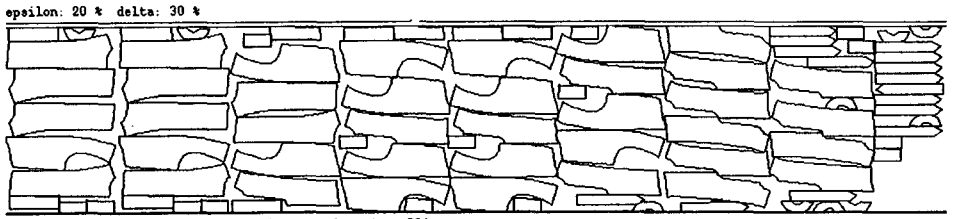
Width: 500 Length:2647 avEff.: 70.3 % totalRunTime: 699sec.
Ratio/Strip: '77.4% 77.4% 74.2% 72.8% 72.8% 74.8% 75.0% 73.3% 34.8% '
Time/Strip: '94 s. 87 s. 81 s. 61 s. 62 s. 77 s. 40 s. 50 s. 147 s. '

epsilon: 20 % delta: 0 %



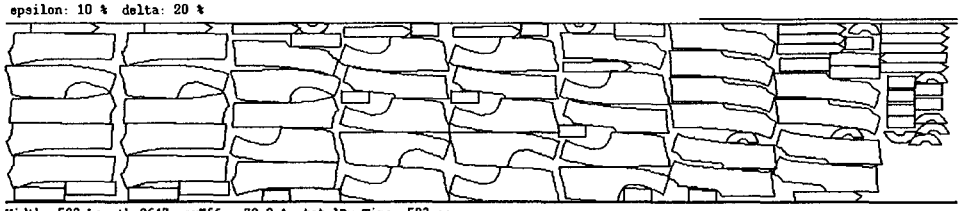
Width: 500 Length:2647 avEff.: 70.9 % totalRunTime: 699sec.
Ratio/Strip: '77.8% 77.8% 74.2% 75.2% 75.2% 71.2% 75.9% 68.6% 42.5% '
Time/Strip: '108 s. 106 s. 62 s. 83 s. 75 s. 187 s. 26 s. 45 s. 7 s. '

epsilon: 20 % delta: 30 %



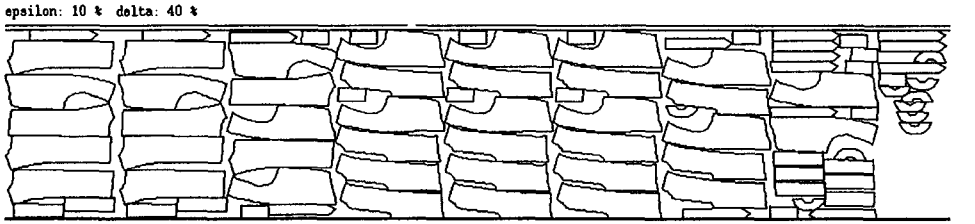
Width: 500 Length:2647 avEff.: 71.1 % totalRunTime: 704sec.
Ratio/Strip: '77.8% 77.8% 73.6% 75.2% 75.2% 71.2% 75.9% 68.6% 44.5% '
Time/Strip: '108 s. 107 s. 57 s. 85 s. 77 s. 190 s. 27 s. 46 s. 7 s. '

epsilon: 10 % delta: 20 %



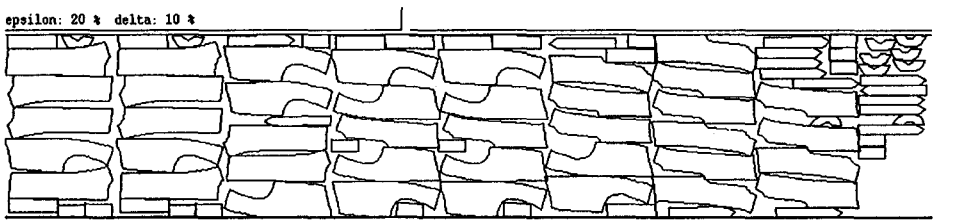
Width: 500 Length:2647 avEff.: 70.9 % totalRunTime: 583sec.
Ratio/Strip: '77.4% 77.4% 76.3% 72.8% 72.8% 74.0% 73.9% 72.8% 41.1% '
Time/Strip: '91 s. 85 s. 101 s. 61 s. 64 s. 73 s. 37 s. 39 s. 32 s. '

epsilon: 10 % delta: 40 %



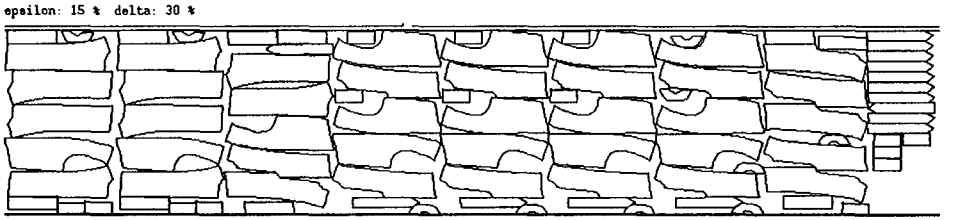
Width: 500 Length:2671 avEff.: 69.9 % totalRunTime: 591sec.
Ratio/Strip: '77.4% 77.4% 76.2% 78.0% 78.0% 78.0% 70.8% 73.9% 19.6% '
Time/Strip: '83 s. 81 s. 88 s. 77 s. 71 s. 70 s. 51 s. 67 s. 3 s. '

epsilon: 20 % delta: 10 %



Width: 500 Length:2647 avEff.: 70.4 % totalRunTime: 602sec.
Ratio/Strip: '77.8% 77.8% 75.3% 75.2% 75.2% 72.2% 75.9% 72.8% 31.4% '
Time/Strip: '107 s. 110 s. 58 s. 90 s. 91 s. 76 s. 28 s. 35 s. 7 s. '

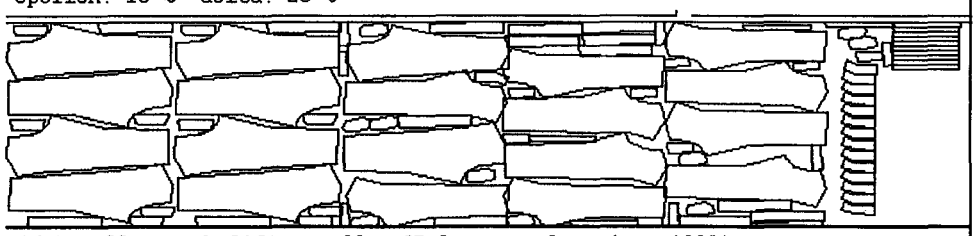
epsilon: 15 % delta: 30 %



Width: 500 Length:2659 avEff.: 71.7 % totalRunTime: 1526sec.
Ratio/Strip: '77.8% 77.8% 76.6% 71.2% 71.2% 71.2% 67.4% 72.4% 59.4% '
Time/Strip: '109 s. 113 s. 80 s. 333 s. 317 s. 346 s. 193 s. 23 s. 12 s. '

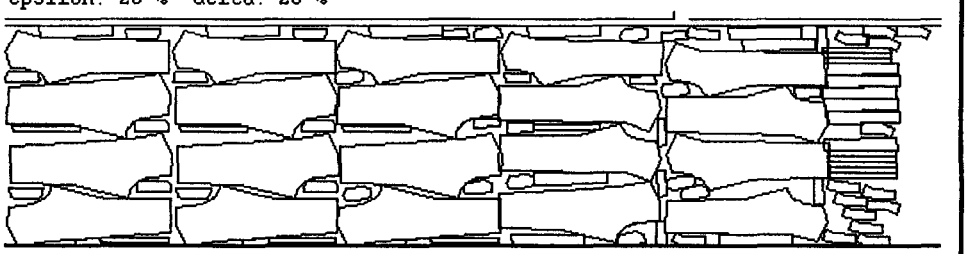
II.10.2.3.Carnet *gr1.cde*

epsilon: 10 % delta: 20 %



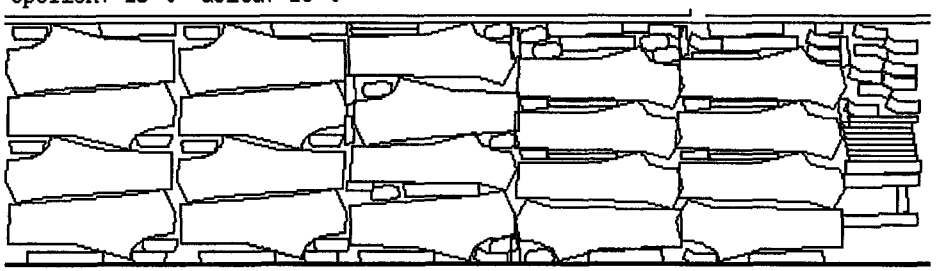
Width: 500 Length:2161 avEff.: 67.3 % totalRunTime: 12081sec.
Ratio/Strip: '83.0% 83.0% 81.0% 81.2% 78.5% 39.3% 25.4% '
Time/Strip: '979 s. 897 s. 9209 s. 653 s. 316 s. 25 s. 2 s. '
NP/Strip: ' 2344 2328 13804 1508 1016 456 80 '
ND/Strip: ' 37 37 264 28 25 28 12 '

epsilon: 20 % delta: 20 %



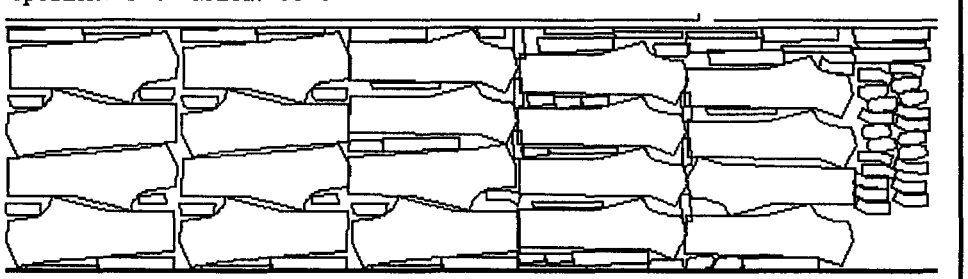
Width: 500 Length:2098 avEff.: 67.6 % totalRunTime: 8965sec.
Ratio/Strip: '81.0% 81.0% 81.3% 79.5% 76.7% 69.7% 4.3% '
Time/Strip: '890 s. 876 s. 2748 s. 544 s. 1100 s. 2807 s. 0 s. '
NP/Strip: ' 1836 1780 4920 1420 2712 23112 8 '
ND/Strip: ' 26 26 81 26 70 2301 3 '

epsilon: 15 % delta: 20 %



Width: 500 Length:2023 avEff.: 76.3 % totalRunTime: 7697sec.
Ratio/Strip: '83.0% 83.0% 80.8% 79.9% 79.5% 51.5% '
Time/Strip: '1179 s. 1067 s. 1752 s. 3018 s. 664 s. 17 s. '
NP/Strip: ' 2344 2328 3456 6272 1796 308 '
ND/Strip: ' 37 37 65 139 48 23 '

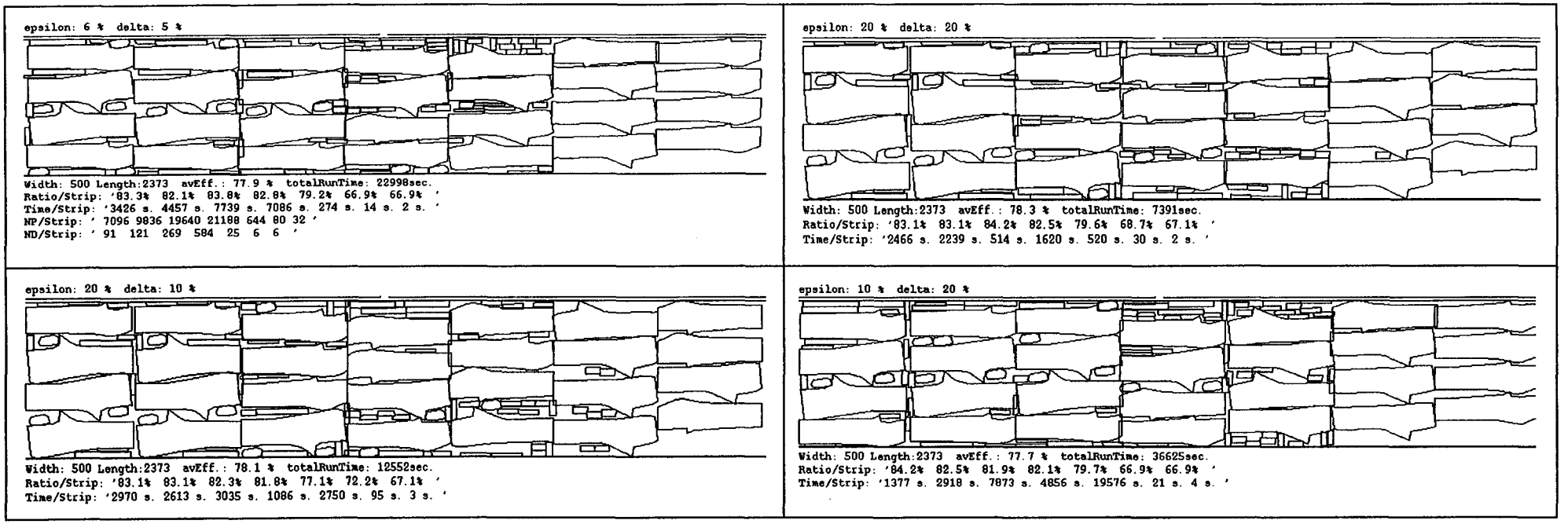
epsilon: 5 % delta: 20 %



Width: 500 Length:2023 avEff.: 77.3 % totalRunTime: 23325sec.
Ratio/Strip: '84.8% 84.8% 81.6% 79.6% 79.3% 53.4% '
Time/Strip: '1514 s. 1393 s. 5275 s. 14941 s. 159 s. 43 s. '
NP/Strip: ' 3808 3808 11120 33968 816 1032 '
ND/Strip: ' 69 69 185 646 23 68 '

Remarque: Il est intéressant de noter que l'algorithme passe énormément de temps sur les troisième et quatrième bandes. Le temps de recherche ainsi que le nombre de neouds générés augmentent fortement, plus que ceux des premières bandes. Nous ne pouvons pas expliquer ce comportement mais on pourrait penser que les fonctions d'estimation ne sont plus adaptées à la nouvelle forme de l'espace de recherche réduit.

*II.10.2.4.*Carnet *kd3.cde*



Le comportement de l'algorithme avec le carnet *kd3.cde* ressemble à celui avec le carnet *gr1.cde*. L'algorithme perd énormément de temps sur la cinquième bande. Ce comportement identique pour les deux carnets assez identiques semble indiquer que les formes utilisées doivent impliquer le type de fonctions d'estimation. Aussi, il serait intéressant de doter l'algorithme d'une certaine capacité d'apprentissage puisque dans ces exemples, les deux premières bandes sont exactement identiques. Il suffit de tester si les formes restantes permettent de reproduire un placement identique et il suffit de recopier ce placement sur la deuxième bande. Le processus de placement de la deuxième bande peut être évitée résultant sur un gain de temps considérable.

II.11 Références Bibliographiques

[**Arenales95**] Marcos Arenales, Reinaldo Morabito, "An AND/OR-graph Approach to the Solution of Two-Dimensional Non-Guillotine Cutting Problems", *European Journal on Operations Research: Special issue, Cutting and Packing*, E.E. Bischoff, G. Wäscher Editors, vol 84, N° 3, 1995, pp. 599-617.

[**Cung94**] Van-Dat Cung, *Contribution à l'Algorithmique Non Numérique Parallèle: Exploration d'Espaces de Recherche*, thèse de doctorat ès informatique, Université Paris VI, avril 1994.

[**Daza95**], V.P. Daza and A.G. de Alvarenga and J. de Diego, "Exact Solutions for Constrained Two-Dimensional Cutting problems", *EJOR: Special issue, Cutting and Packing*, E.E. Bischoff, G. Wäscher Editors, vol. 84, N° 3, 1995, pp. 633-644.

[**Farreny87**] H. Farreny and M. Ghallab, *Elément d'Intelligence Artificielle*, Hermès 1987.

[**Ghosh91**] Ghosh Subrata, Mahanti Ambuj, "Improving the Efficiency of Limited-Memory Heuristic Search", Tech. Report, University of Maryland, Dept. Computer Science, 1990-1991.

[**Harris74**] L.R. Harris, "The Heuristic Search Under Conditions of Error", *Artificial intelligence*, vol.5, 1974, pp. 217-234.

[**Hart68**] P.E. Hart, N.J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, 1968, pp.100-107.

[**Huyn80**] N. Huyn, R. Dechter, and J. Pearl, "Probabilistic Analysis of the Complexity of A*", *Artificial Intelligence*, vol. 15, 1980, pp. 241-254.

[**Kirkpatrick83**] S. Kirkpatrick et al., "Optimization by Simulated Annealing ", *Science* 220, 671-680, 1983.

[**Nilsson82**] N.J. Nilsson, *Principles of Artificial Intelligence, Symbolic Computation*, Ed. Springer-Verlag, Berlin Heidelberg, New York, 1982.

[**Oliveira93**] J. F. Oliveira and J.S. Ferreira, "Algorithms for Nesting Problem", in R. V. Vidal, Editor, *Applied Simulated Annealing*, , Springer Verlag, Berlin, 1993, pp. 255-273.

[**Pearl82**] J. Pearl, and J.H. Kim, "Studies in Semi-Admissible Heuristics", *IEEE Trans. PAMI-4*, July 1982, pp. 392-400.

[**Pearl90**] J. Pearl, *Heuristique, Stratégie de Recherche Intelligente pour la Résolution de Problème par Ordinateur.* Coll. Intelligence Artificielle, Cepaduès, 1990.

[**Winston92**] Patrick Henry Winston, *Artificial Intelligence*, 3ème édition, series: "The Art of Computer Programming", Addison-Wesley, Reading, Massachusetts, 1992.

Chapitre III:

Algorithme de Recuit Simulé

TABLE DE MATIÈRES

I.1. Introduction 97

I.2. Recuit Simulé 98

I.2.1. Critère de Metropolis 99

I.3. Caractéristiques du Recuit Simulé 100

I.3.1. Température 100

I.3.1.1. Température initiale T_0 100

I.3.1.2. Décroissance de la température 101

I.3.2. Chaîne de Markov 102

I.4. Application au Placement 104

I.4.1. Placement par "Jet" Aléatoire 104

I.4.2. Placement par Bandes 106

I.4.3. Algorithme Hybride du Recuit Simulé Arborescent 108

I.4.3.1. Principe de Transformation 108

I.4.3.2. Fonction de coût 110

I.4.3.3. Tirage des formes 110

I.4.3.3.1. Tirage aléatoire 110

I.4.3.3.2. Tirage proportionnel 110

I.4.3.4. Redondance des formes 111

I.4.3.5. Température 111

I.4.3.6. Chaîne de Markov et équilibre statistique 112

I.4.3.7. Le Recuit 113

I.4.3.8. Critère d'arrêt 113

I.4.3.9. L'algorithme implanté 114

I.4.4. Résultats 114

I.4.4.1. Comportement par palier de température 115

I.4.4.2. Recuit....116

I.4.4.3. Complexité du recuit simulé....118

I.5. Comparaison avec les Algorithmes en Arbres....119

I.6. Conclusion....120

I.6.1. Exemple de Placement 1....122

I.6.2. Exemple de Placement 2....123

I.6.3. Exemple de Placement 3....124

I.7. Références Bibliographiques....125

CHAPITRE III

Algorithme de Recuit Simulé

III.1. Introduction

Une méthode brute pour résoudre un problème difficile est la recherche aléatoire ou énumérative dans laquelle les points sont pris au hasard ou de façon systématique dans l'espace de recherche et leur valeur est déterminée. Si celle-ci est bonne, on la sauvegarde, sinon elle est rejetée et un autre point est évalué. C'est une stratégie peu intelligente et peu utilisée.

Pour les fonctions continues, on utilise la méthode du gradient, appelé aussi *hill-climbing,* dans laquelle la recherche est guidée par le gradient de la fonction. Mais cette technique est souvent limitée aux fonctions unimodales (ayant un seul minimum ou maximum) car l'algorithme s'arrête sur le premier minimal local trouvé. Pour des fonctions multimodales, l'idée est de combiner la recherche aléatoire et le gradient. Ce dernier commence par trouver un minimum. Puis une fois que celui-ci est localisé, la recherche recommence avec un autre point choisi au hasard. L'algorithme s'arrête après un certain nombre d'itérations défini par l'utilisateur, ou si un certain nombre de minima locaux trouvés n'apportent aucune amélioration. Cette méthode est appelée méthode d'amélioration itérative; elle est simple à implanter et peut donner de bons résultats s'il n'y a pas trop d'optima locaux. Lorsqu'il y a plusieurs optima locaux, ce qui est souvent le cas dans les problèmes combinatoires, on peut recommencer l'algorithme plusieurs fois avec un autre point, mais rien n'assure l'obtention d'un optimum global [**Renders95**]. D'où l'idée de bien guider le choix des configurations à "visiter".

Dans cette partie, l'algorithme utilisée est un hybride du recuit simulé et de la recherche en arbre du chapitre précédent. Le caractère déterministe de la recherche en arbre est maintenant remplacé par le caractère stochastique du recuit. L'algorithme est de type construction.

III.2. Recuit Simulé

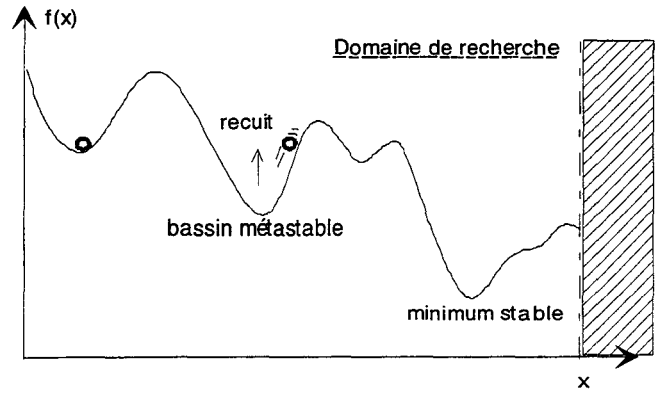
Le recuit simulé est une version améliorée de la méthode d'amélioration itérative. Il est proposé en 1983 par Kirkpatrick et *al.* **[Kirpatrick83]** pour la résolution d'un problème de placement en VLSI (Very Large Scale Integration). La méthode est inspirée du principe thermodynamique dans lequel les déplacements dans l'espace de recherche sont basés sur la distribution de Boltzmann. La probabilité de Boltzmann, notée π_i mesure la probabilité de trouver un système dans une configuration C_i d'énergie $E(C_i)$, à une certaine température T donnée, dans l'espace de configurations U et elle est définie par :

$$\underset{i \in U}{\pi_i} = \frac{\exp \frac{-E(C_i)}{k.T}}{\sum_{j \in U} \exp \frac{-E(C_j)}{k.T}}$$

Equation III. 1: k *est appelé la constante de Boltzmann*

Dans cette expression, le facteur kT montre que lorsque la température est très élevée, tous les états sont à peu près équiprobables, c'est-à-dire un grand nombre de configurations sont accessibles. Au contraire quand la température est basse, les états à haute énergie deviennent peu probables par rapport à ceux de faible énergie.

Pour appliquer ce principe au problème de minimisation de coût, le processus de recherche peut être assimilé à un processus de *recuit* comme en métallurgie. Quand on chauffe un métal à une température très élevée, le métal devient liquide et peut occuper toute configuration. Quand la température décroît, le métal va se figer peu à peu dans une configuration qu'il est de plus en plus difficile à déformer (on dit qu'il est *refroidi*). A moins de le réchauffer (*recuit*), le métal peut être retravaillée de nouveau pour lui donner la forme désirée. L'algorithme de Kirkpatrick *simule* ce processus en combinant dans l'algorithme les mécanismes de refroidissement et de recuit (**Figure** III. 1).



BU LILLE 1

Figure III. 1 : Parcours de l'espace de recherche avec le recuit simulé. Le principe de "recuit" qui se traduit par une augmentation du niveau d'énergie, permet de sortir des minima locaux.

Cependant, le concept de température d'un système physique n'a pas d'équivalent direct avec le problème à optimiser. Le paramètre température T doit être simplement un paramètre de contrôle, indiquant le contexte dans lequel se trouve le système (ex: stade de la recherche). En fait, le paramètre T contrôle les déplacements vers les points voisins les moins bons pour échapper aux optima locaux, sans pour autant trop s'écarter du chemin vers le vrai minimum. L'équivalent de l'énergie sera la valeur de la fonction de coût g.

Ainsi, dans l'algorithme de recuit simulé, la probabilité de Boltzmann n'est pas directement appliquée, mais le critère de Metropolis est utilisé. Le critère de Metropolis permet de décider si une nouvelle configuration générée présente une variation de coût acceptable. Il permet de décider aussi de sortir des minima locaux quand le critère d'arrêt n'est pas encore atteint.

III.2.1. Critère de Metropolis

Après chaque passage d'une configuration u à une configuration v, on calcule la variation de la fonction de coût $\Delta g = g(v) - g(u)$. La transformation est acceptée selon la probabilité $p(u,v)$ telle que:

$$\mathrm{p}(u,v) = e^{\frac{-\Delta g}{T}}$$

Equation III. 2

Lorsque la variation $\Delta g \leq 0$, l'exponentielle est supérieure ou égale à 1, la nouvelle configuration doit être acceptée, on lui affecte alors la probabilité maximale de 1.

Si $\Delta g > 0$, on compare $p(u,v)$ à un nombre aléatoire r, [0,1[:

→ *Si* $r < p(u,v)$ la configuration v est acceptée;

→ sinon elle est rejetée et on essaie une autre configuration.

Les configurations ayant une forte augmentation en Δg sont donc moins probables pour une température donnée, d'autant moins que la température est faible.

Au début de l'algorithme, le facteur T est élevé, la probabilité $p(u, v)$ est proche de 1 et presque toutes les variations Δg sont acceptables. Au contraire, quand T diminue, les remontées sont de plus en plus difficiles et seules de très faibles variations peuvent être acceptées. Si une configuration est rejetée, le système essaie d'en trouver une autre, sinon elle est acceptée et la recherche continue avec celle-ci jusqu'à ce que le critère d'arrêt soit atteint.

Dans les sections suivantes, nous présentons comment le calcul du facteur température et les autres paramètres qui influencent l'efficacité de l'algorithme.

III.3. Caractéristiques du Recuit Simulé

III.3.1. Température

La température est un paramètre de contrôle. Le but est d'avoir une température assez haute pour sauter les barrières et suffisamment basse pour être malgré tout attiré vers le minimum le plus profond. En diminuant lentement la température, on permet au système de rechercher les "bassins d'attraction" dont la préférence est toujours donnée à celui dont le coût est minimal. La loi selon laquelle la température décroît est également importante pour l'efficacité de l'algorithme, puisqu'elle doit aussi laisser le temps au système de tester le maximum de configurations pour être sûr d'obtenir le minimum global. Aussi, la température initiale doit-elle être suffisamment élevée pour que la descente en température soit aussi lente que possible.

III.3.1.1. Température initiale T_0

Sachant que les températures élevées favorisent le désordre, la valeur de la température initiale T_0 doit être choisie élevée. Elle est déterminée, sinon fixée arbitrairement, lors d'une phase de pré-traitement avec une exploration initiale partielle de l'espace de configurations. D'après Kirkpatrick et *al.* [**Kirpatrick83**], T_0 doit être choisie de sorte que la probabilité d'acceptation de la plus mauvaise solution soit environ 80% (*i.e.* $P_r = 0.8$). Ensuite, dans

l'expression de *p(u, v)*, l'utilisateur doit fixer l'accroissement maximal acceptable de la fonction objectif, noté Δg^+, pour en déduire la valeur de T_0. On obtient alors l'expression suivante :

$$T_0 = \frac{\Delta g^+}{\ln(P_r)}$$

Equation III. 3

Cette relation peut être déduite de la probabilité de Boltzmann selon l'application. Par exemple, dans [**Heckmann95**], on génère aléatoirement un certain nombre de configurations initiales. Puis on calcule la température initiale de façon à pouvoir accepter dans la suite de l'algorithme, les configurations dont le coût est 3σ fois moins bon que la valeur moyenne des coûts connus, avec σ l'écart type du nombre de valeurs initiales de la fonction coût. La probabilité d'acceptation P_r est réglée à 0.01 pour avoir une température initiale suffisamment élevée. Ainsi, la température initiale vaut:

$$T_0 = \frac{-3\sigma}{\ln(P_r)}$$

Equation III. 4

Dans [**Lutfiyya91**], la définition de T_0 est basée sur le taux de configurations acceptées par rapport aux configurations générées.

III.3.1.2.Décroissance de la température

Le changement de température T_k vers T_{k+1} est déterminé par le moment où l'on a détecté l'équilibre statistique (ou l'état de *quasi-équilibre*) à la température T_k. La recherche de cet état s'effectue en réitérant la *chaîne de Markov* (voir *section 3.2*), qui correspond à peu près au nombre de configurations testées. La variation de température se fait donc par "palier" suivant la fonction de décroissance utilisée. Les fonctions les plus couramment rencontrées dans la littérature sont les fonctions linéaires, discrètes ou exponentielles (Voir Tableau III. 1).

Type	Fonctions	Paramètres	Commentaires
Linéaire	$T_{k+1} = \alpha \times T_k$	$\alpha \leq 1$ (typ. 0.95 à 0.8)	[Oliveira93]
Discrète	$T_{k+1} = T_k - \Delta T$	$\Delta T > 0$, pour la descente $\Delta T < 0$, pour le recuit	[Randelman86]
Exponentielle	$T_{k+1} = T_k . exp(\frac{-\lambda . T_k}{\sigma_k})$ avec $0 \leq \frac{T_{k+1}}{T_k} \leq 1$	σ_k, l'écart-type des coûts des configurations acceptées sous la température T_k, λ paramètre de réglage fixé par l'utilisateur	[Heckmann95]

Tableau III. 1: Les lois de décroissance de température les plus utilisées.

La plus utilisée est la loi linéaire, qui permet d'avoir une décroissance ni trop rapide (discrète) ni trop lente (exponentielle).

La décroissance exponentielle permet de tenir compte de l'état précédent par l'utilisation de l'écart type des coûts obtenus sous le palier de température précédent. Au début de la recherche, presque toutes les configurations sont acceptées. Comme ces configurations peuvent être très dispersées dans l'espace de recherche, l'écart type peut être alors relativement grand et donc la température décroît plus ou moins lentement selon la valeur donnée au paramètre λ. La décroissance est donc dynamique et adaptative.

Pour la fonction discrète, la décroissance est indépendante de la valeur de l'état précédent du système et dépend uniquement de la valeur de la différence ΔT. Cette fonction peut être utilisée lorsque le nombre d'itérations nécessaire pour atteindre un état d'équilibre peut être évalué avec une assez bonne précision. Le nombre d'itérations à chaque étape de température est déterminé par la longueur de la "chaîne de Markov".

III.3.2. Chaîne de Markov

La chaîne de Markov est *l'ensemble des états finis aléatoires* constitué d'une suite des probabilités associées à chaque configuration visitée à la température T_k. Lorsque T_k est constante, la probabilité est homogène. Et si le nombre de transitions tend vers l'infini, l'état le plus probable apparaît plus souvent et on obtient alors l'équilibre statistique à cette température. Théoriquement, on montre que si l'on fait tendre la longueur de la chaîne de Markov vers l'infini, on peut obtenir la convergence asymptotique de l'algorithme [Lin93].

La fonction π_i (Equation III-3) montre que la probabilité d'obtenir un état de basse énergie peut être atteint après un certain nombre d'itérations à température T constante:

$$\lim_{T \to 0, T = T_k} [\lim_{MK \to \infty} p(X_{MK} \in U^*)] = \lim_{T \to 0} [\sum_{i \in U^*} \pi_i] = 1$$

Equation III. 5

*Avec $X_{MK, MK=1,2,...}$ la chaîne de Markov pouvant prendre des valeurs dans U et **U*** l'ensemble des minima de **U**.*

En pratique, il n'est pas toujours possible de connaître *a priori* le nombre exact d'itérations nécessaire et le temps de calcul est limité, si bien que la convergence asymptotique ne peut être qu'approchée. A cause de cette approximation, l'algorithme de recuit simulé ne peut garantir d'atteindre le minimum global avec une probabilité égale à 1. Ainsi, pour augmenter la chance d'obtenir une solution la plus proche du minimum global, il est important trouver les valeurs de compromis entre les paramètres de contrôle de l'algorithme.

En résumé, le recuit simulé utilise une double dynamique : 1°) recherche de minima à température fixée avec la chaîne de Markov et 2°) diminution par étape de la température (**Figure** III. 2).

```
"Algorithme de recuit simulé"
1- Initialisation
1_1 - valeurs initiales des paramètres.
1_2 - Définir: u ← Configuration de départ
1_3 - k ← 1
2- TANT_QUE (refroidi=FAUX) faire :
  2_1- TANT_QUE (equilibre=FAUX) faire :
    2_11: v := transformation (u, T_k )
    2_12: Δg ← coût (v) - coût(u)
    2_13: p(u,v) ← min {1, exp (-Δg / T_k )}
    2_14: Si p(u,v) > random[0; 1) alors u ← v; "critère de
Metropolis"
  2_2- JUSQU'A équilibre à T_k ;
  2_3- T_{k+1}← LoiDecroissance( T_k );
  2_4- k ← k +1
3- JUSQU'A système refroidi
4- resultat ← u.
5- FIN.
```

Figure III. 2 : Le recuit simulé peut être vu comme un algorithme progressant par palier de température à l'intérieur duquel un ensemble de configurations est généré et testé.

III.4. Application au Placement

Cette partie décrit quelques travaux sur l'application du recuit simulé au problème de placement. Deux types d'approches sont utilisés: l'approche la plus fréquente consiste à commencer par un premier "jet" aléatoire de formes sur le support et essayer d'arranger globalement les formes par élimination des recouvrements. Cette méthode permet d'obtenir une solution globale. La deuxième approche intègre le principe du recuit simulé dans la recherche en arbre; l'espace de recherche réduit au remplissage d'une seule bande à la fois; la solution globale est un ensemble de solutions sous-optimales.

III.4.1. Placement par "Jet" Aléatoire

Les pièces sont "jetées" aléatoirement sur la matière. Ensuite sur chacune sont effectuées un certain nombre d'essais (translations, symétries, rotations) et on garde la meilleure position

selon des critères locaux. Les pièces sont répertoriées selon leur ordre de priorité (selon leur surface et leur niveau de difficulté pour l'imbrication). Les pièces dont le niveau de priorité est plus élevé sont traitées en premier et le placement des formes irrégulières se ramène alors au placement de rectangles englobant une ou plusieurs pièces [**Prempti83**].

Dans [**Oliveira93**, **Heckmann95**], le processus de placement se déroule en quatre étapes:

1. *Placement rapide de toutes les formes avec acceptation sans contrainte de recouvrement en utilisant un codage approximatif des formes;*
2. *Elimination des recouvrements par de légères rotations ou translations;*
3. *Affinement du placement par un algorithme de compactage en utilisant une représentation approchée des formes;*
4. *Affinement du placement avec le codage exact des formes.*

L'algorithme s'arrête quand le coût moyen reste à peu près constant pour un nombre constant d'états d'équilibre consécutifs. La fonction de coût est la somme de trois termes pondérés chacun par un coefficient de pénalité (pénalité de recouvrement, pénalité de dépassement en largeur et pénalité de dépassement en longueur).

Ici, une configuration est définie par un état où toutes les formes sont placées avec ou sans recouvrement. A chaque passage d'une configuration à une autre, on choisit la forme et le mode de transformation à lui affecter. Le mode de transformation, numéroté de 1 à M , comprennent la translation, la rotation ou l'échange de position avec une autre pièce. La sélection du mode de transformation i est soumise à une fonction de probabilité dépendant des coûts des configurations connues et de l'amélioration de coût estimée ΔC_i induite par ce choix. La probabilité est donnée par la relation suivante:

$$P_i = \frac{\Delta C_i}{\sum_{j=1..M} \Delta C_j}.$$

La température initiale est calculée lors de l'étape de pré-traitement (Equation III-4) et elle décroît selon la loi exponentielle donnée dans le Tableau III. *1*. Les résultats rapportés sont relativement bons par rapport à l'opérateur humain (qui est d'environ 4% supérieur), avec un temps de calcul de l'ordre de 200 minutes pour placer entre 15 et 25 formes (contre 15 à 30 minutes pour l'opérateur humain). Ce temps de calcul est en grande partie dû à la définition des configurations où toutes les formes sont "jetées" sur la bande. Et à cause des positions connexes des formes, l'élimination des recouvrements est très difficile parce que changer la position d'une forme peut remettre en cause celles des autres formes. Des pénalités

pénalités supplémentaires peuvent être ajoutées, mais il n'est pas non plus facile de trouver la valeur optimale du facteur de pénalité pour chaque changement de configuration.

Pour éviter ce problème, G. Roussel **[Roussel94]** propose le placement en *bande*. Cette stratégie permet de subdiviser le problème en sous-problèmes (*bandes*) de taille maniable. Et les recouvrements sont traités durant l'opération de transformation des configurations.

III.4.2. Placement par Bandes

L'approche de G. Roussel **[Roussel94]** est basée sur le raisonnement suivant. Comme le recuit simulé est une recherche progressant point par point, on peut l'appliquer à une recherche en arbre dans laquelle un successeur à chaque noeud est généré à chaque fois. La recherche d'une solution optimale devient la recherche d'une *solution locale optimale.*

A chaque itération, une forme est tirée et placée au hasard. Si le placement est accepté on continue le placement à partir de ce noeud sinon on en tire une autre. Mais, si le dernier placement n'est pas bon, mais acceptable selon le critère de Metropolis, on poursuit quand même la recherche. La température initiale T_0 fixée à une valeur arbitrairement élevée permet de rechercher une première solution, pas forcément bonne, par des explorations rapides de l'espace de recherche. Puis après avoir obtenu une première bande, la température est réinitialisée à T_1 à partir de laquelle la décroissance lente va commencer. Elle est définie par la surface de la plus grande forme multipliée par un facteur μ_1. Les valeurs de μ_1 et de μ_2 sont déterminées de façon expérimentale selon l'ensemble des formes à placer (ex. valeurs pour les formes de pantalon : $\mu_1 = 1$ et $\mu_2 = 0.91$). La décroissance est effectuée par la suite géométrique de raison μ_2 : $T_k = \mu_2 \times T_{k+1}$.

Le changement de palier k est indiqué par la fin de la chaîne de Markov. Considérant que chacune des N formes peut subir 32 modes de transformation (4 types de rotation, 2 de symétrie et 4 positionnements connexes à une autre forme), la longueur minimale de MK est égale à $32 \times N$. A chaque palier de température, l'algorithme teste $32 \times N$ configurations avant de passer au palier suivant, jusqu'à ce que le critère d'arrêt soit atteint.

L'algorithme s'arrête lorsque le rendement maximal espéré est atteint ou lorsque la *vitesse de décroissance* relative de la fonction de coût est inférieure à la vitesse minimale fixée par l'utilisateur. La vitesse de décroissance relative de la fonction de coût est basée sur la différentielle de la variation de coût sur la variation de nombre de noeuds exploités. Si la variation de la fonction de coût reste constante, la variation en nombre de noeuds développés e_n va croître, et donc la vitesse de l'évolution de coût est négligeable:

$$\frac{g_{n-1} - g_n}{g_{n-1} \times (\text{nbNoeudsGeneres} - e_n)} \leq V_{\min}$$

Equation III. 6

avec e_n le nombre de noeuds exploités pour obtenir la dernière bonne solution de coût g_n. V_{min} est empiriquement fixé à 10^{-5}, ce qui équivaut à une diminution de 1% du coût pour 1000 transitions. Mais l'expérience montre que ce critère est très difficilement atteint (**Figure** III. 3).

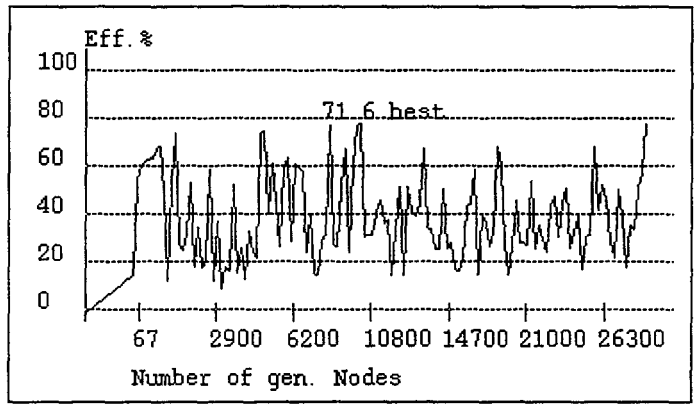


Figure III. 3 : Evolution de la recherche en pourcentage d'occupation de surface avec l'algorithme proposé dans *[**Roussel94**]*, sur le carnet *gr2.cde*. La recherche est très lente à converger. Si le rendement maximal espéré n'est pas encore atteint alors qu'un critère d'arrêt est satisfait, la solution en sortie est le meilleur noeud terminal sauvegardé jusqu'à présent, qui n'est pas forcément l'optimal.

Cette lenteur de convergence peut être expliquée par le fait que la descente ou la remontée dans l'arbre est régie par un nombre aléatoire. Avant chaque transformation d'un noeud *u*, un nombre aléatoire *NA1* compris entre 0 et 1 est généré. Si *NA1* est inférieur à 0.5, on génère un successeur de *u*, sinon le noeud *u* est supprimé du chemin et on remonte vers son prédécesseur. Cette stratégie est à l'origine des remontées fréquentes et aussi du fait qu'à chaque palier de température, l'information acquise depuis le début de la recherche peut ne pas être enregistrée, si à la fin de la chaîne de Markov, aucune configuration stable[1] meilleure n'est trouvée. De

[1] Une configuration stable ici correspond à un état où un noeud terminal quelconque est atteint.

même, le critère d'arrêt par la vitesse de décroissance peut être satisfait avant que le rendement maximal espéré soit atteint (cas où le même minimal local est rencontré plusieurs fois à la suite).

Il est difficile de satisfaire le critère de vitesse de croissance car la recherche est trop aléatoire. Nous proposons de réduire ce caractère aléatoire en éliminant le paramètre *NA1*. L'algorithme proposé ci-après reprend le principe de placement en arbre, en éliminant l'aspect purement aléatoire de la descente dans l'arbre et la stratégie reste similaire à une recherche en profondeur. L'algorithme de G. Roussel [**Roussel94**] constituera l'algorithme de référence.

III.4.3. Algorithme Hybride du Recuit Simulé Arborescent

Dans cet algorithme inspiré du précédent, chaque noeud est partiellement développé. A chaque noeud u, une seule forme est tirée et le nombre de successeurs possibles de u est égal au nombre de mode de transformations possibles à partir de cette forme. A ce stade, un critère de sélection est utilisé pour choisir parmi ces successeurs un noeud pour poursuivre la recherche. Cette stratégie peut être vue comme une recherche en profondeur restreinte à des développements limités à chaque palier, à une seule forme au lieu de toutes les formes du carnet (**Figure** III. 4).

III.4.3.1.Principe de Transformation

La transformation élémentaire consiste à donner une nouvelle configuration à u pour arriver à une configuration v (successeurs ou prédécesseur de u) voisine de u au sens de la topologie définie dans l'espace des configurations. Les configurations à tester à chaque étape sont limitées aux configurations dont l'énergie est proche de la configuration courante. Le coût décroît globalement par palier (**Figure** III. 5).

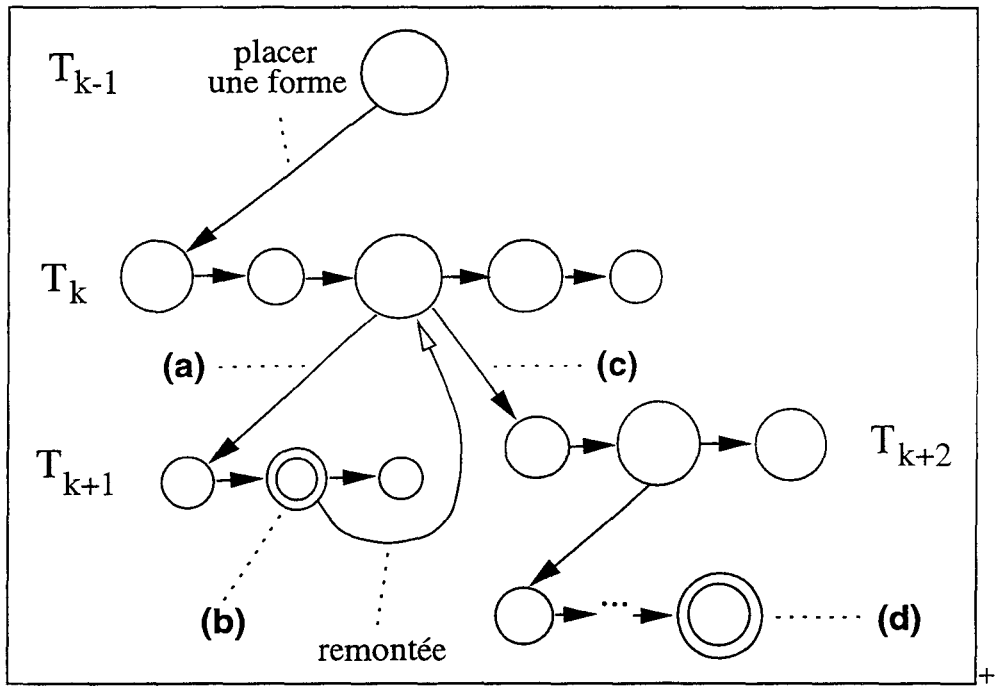


Figure III. 4: Principe du recuit simulé hybride arborescent. Cette stratégie peut être vue comme une recherche en profondeur restreinte à des développements limités à chaque palier, à une seule forme choisie de façon aléatoire au lieu des générations systématiques sur toutes les formes du carnet.

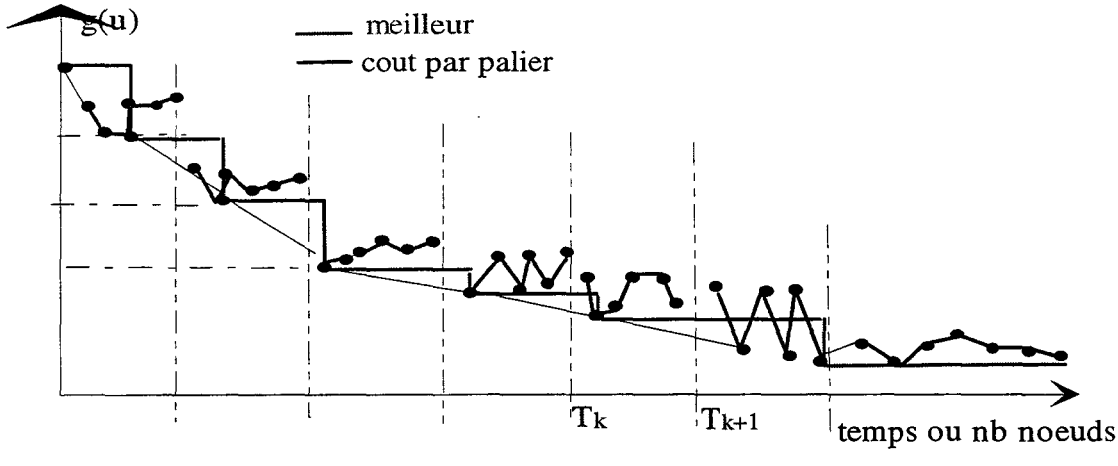


Figure III. 5: L'évolution du coût par palier de température. Les configurations testées sont celles dont l'énergie est voisine de celle de la configuration courante.

Les remontées sont acceptées si elles ne génèrent pas trop de grandes variations de coût. Il existe alors deux possibilités de parcours :

1) On engendre un successeur de u en tirant aléatoirement une forme dans le stock courant;

2) On remonte vers le prédécesseur de u si la descente n'est pas possible (noeud terminal).

La descente est donc plus fréquente, puisque pour une forme placée, la surface non occupée diminue et la variation de coût est négative, donc le noeud est systématiquement accepté. A l'inverse, la remontée supprime la dernière forme placée et provoque un accroissement des chutes et l'acceptation du prédécesseur dépend du critère de Metropolis.

III.4.3.2.Fonction de coût

Ici, le coût d'une configuration est exprimé en surface non occupée considérée comme une perte dans la bande (Voir chap. 1, section 1.4.1, Equation 1.1).

III.4.3.3.Tirage des formes

III.4.3.3.1. Tirage aléatoire

Le tirage aléatoire sélectionne une forme disponible dans le stock courant et des configurations sont générées à partir de cette forme. Ce tirage présente l'inconvénient de ne pas mémoriser les formes déjà testées. En effet, les formes n'ayant pas généré les configurations acceptables peuvent réapparaître à l'étape suivante. Alors, on propose le tirage proportionnel.

III.4.3.3.2. Tirage proportionnel

Ce tirage tient compte de l'espace disponible restant dans la bande, mais aussi favorise le placement des grandes formes par un choix proportionnel à leur surface (Equation III. 7).

$$p(F_i) = \frac{(W_M - W_u) \times (1+\alpha) \times L_{\max} - A(F_i)}{(W_M - W_u) \times (1+\alpha) \times L_{\max}}$$

Equation III. 7

A cause de la contrainte de sur passement, seules les formes dont la surface satisfait la condition suivante participeront au tirage (Equation III. 8):

$$(W_M - W_u) \times (1 + \alpha) \times L_{max} - A(F_i) \geq 0$$

Equation III. 8

S'il n'existe aucune forme satisfaisant cette condition, la configuration correspond à un noeud terminal.

Une fois qu'une forme est choisie, on génère un nombre aléatoire r entre 0 et 1 que l'on compare à $p(F_i)$:

- *si* $r \leq p(F_i)$, prendre F_i;
- *si* $r > p(F_i)$, tirer une forme autre que F_i.

Il peut arriver que la forme choisie satisfasse les conditions de surface mais échoue après la phase d'imbrication à cause de la contrainte de dépassement. Dans ce cas, la configuration est également considérée comme un état terminal.

III.4.3.4.Redondance des formes

Dans certaines approches [**Daniels95**], on autorise la redondance des formes pour mieux améliorer le rendement ou faciliter la découpe (par exemple, la coupe en guillotine). Ces formes sont de préférence de petites surfaces qui peuvent s'insérer dans les espaces laissés entre les grandes formes (ex., grande concavité).

Dans notre cas, cette possibilité n'est pas utilisée. Pour éviter la redondance des formes, on attribue à chaque bande une liste de formes présentes. L'état des listes de toutes les bandes permet de contrôler l'absence de redondance.

III.4.3.5.Température

Ce paramètre influence l'acceptation des configurations, aussi on préconise de prendre pour T_0 la surface maximale disponible.

$T_0 = W_M \times L_{max}$ avec L_{max} la plus longue forme dans le stock disponible.

Comme on a défini la température initiale par la surface totale non occupée sur la bande, on peut diminuer la température, de la surface de la forme qui vient d'être ajoutée au placement:

$T_k = T_{k-1}$ - A(F_i), tout en veillant à avoir $T_k > 0$

III.4.3.6.Chaîne de Markov et équilibre statistique

Si l'objectif est d'obtenir tout de suite une bande acceptable, il faut favoriser plus la descente que la remontée vers le prédécesseur. A chaque température donnée, une forme est placée. On considère l'équilibre comme étant une "bonne" dispositions de cette forme par rapport à l'état du placement actuel. Donc pour un état de placement donné, ou une température donnée, le nombre d'itérations correspond au nombre de permutations possibles de cette forme. La longueur de la chaîne de Markov n'est plus fixe, mais elle est liée aux à chaque forme F_i.

$$MK(F_i) = Card\{orientationsPossibles(F_i)\} \times Card\{connexitesPossibles(F_i)\} \times Card\{symetrie(F_i)\}$$

$Card\{orientationsPossibles(F_i)\}$ est la cardinalité de l'ensemble de rotations permises pour la forme F_i. Dans notre cas, l'ensemble de rotations est limité à quatre multiples de 90°, i.e. {0°; 90°; 180°; 270°}.

$Card\{connexitesPossibles(F_i)\}$ correspond au nombre de côtés sur lesquels la forme F_i peut être adjacente avec les autres formes. Concrètement cela correspond aux 4 côtés du rectangle circonscrit.

$Card\{symetrie(F_i)\}$ est le nombre de symétries admises pour la forme F_i. La symétrie peut être par rapport à la ligne verticale ou horizontale.

Ainsi, le nombre de noeuds générés par itération peut être fortement réduit et par conséquent la rapidité de convergence.

La condition de quasi-équilibre pour une température donnée est obtenue lorsque l'on obtient un rectangle circonscrit minimal entre les deux formes assemblées, après *MK* essais. Si ces essais s'avèrent tous négatifs, on autorise une remontée vers le plus récent prédécesseur en augmentant la température ("réchauffe" du système) pour repartir vers un autre chemin. Ici, il peut être utile de mémoriser temporairement le noeud que l'on vient d'abandonner afin d'éviter de revenir sur le même état, si le tirage fournit la même forme que le tirage précédent.

L'état d'équilibre est obtenu lorsqu'on obtient la surface minimale d'imbrication. Cette dernière est définie par le produit de la longueur de fusion par la largeur de fusion, c'est-à-dire la surface de la zone d'intersection des formes imbriquées:

$$\text{Min}\ (Larg eurFusion \times LongueurFusion)$$

Equation III. 9

En diminuant la température, la configuration sera progressivement "figée", avec cette condition, on est sûr d'avoir la forme finale la plus compacte possible, puisque la superposition des formes est évitée lors de la procédure de concaténation. Cette stratégie présente un risque d'emprisonner un espace vide si au début, il n'y a pas de tirage de petites formes pour combler les espaces entre les grandes formes.

Dans l'implémentation de l'algorithme, on créera une liste ordonnée de noeuds successeurs acceptables classées dans un ordre croissant de leur surface de fusion. Cette liste contiendra au plus *MK* noeuds et sera vidée après chaque palier de température.

III.4.3.7.Le Recuit

La recuit (remontée en température) consiste à enlever la dernière forme placée et refaire un autre tirage pour placer une nouvelle forme.

III.4.3.8.Critère d'arrêt

Si la contrainte temporelle existe, l'algorithme peut être arrêtée sans que celui-ci atteigne l'état stationnaire et la solution (sous-optimale) est la meilleure solution sauvegardée jusqu'à présent. Les critères d'arrêt que nous avons adoptés :

1) Le rendement sous-optimal espéré, ρ_{max} est atteint à ε près:

$(\rho_k \geq \rho_{max})$ & $(g(u) < surfaceFormeMin)$

- $avec \quad \rho_{\max} = (1-\varepsilon).\rho$
- $avec \quad \rho = \gamma.\, Round\,(\sum_{F_k \in EP} a_k.\mathcal{A}(F_k)/NB)$
- $avec \; NB = Sup\,(\sum_{F_k \in EP} a_k.\mathcal{A}(F_k)/W_M \times L_{\max})$

NB est le nombre de bandes pour contenir tous les effectifs de tous les modèles du carnet de commande ; γ est un paramètre à régler sur l'estimation d'occupation maximale de la bande (typiquement $\gamma = 1$). Une valeur trop élevée de e peut provoquer un arrêt prématuré alors qu'une valeur trop faible peut demander un temps de recherche beaucoup plus long.

2) Le second critère est le nombre maximal de recuits: (*NbRecuits* > *NbRecuitMax*).

III.4.3.9.L'algorithme implanté

L'algorithme commence par une phase de pré-traitement consistant à générer une configuration initiale dans laquelle une forme est placée aléatoirement. Ensuite, la recherche progresse à partir de cette configuration initiale comme dans un parcours d'arbre en utilisant les principes du recuit simulé décrit auparavant (**Figure** III. 6).

```
"ALGORITHME DE RECUIT SIMULÉ APPLIQUÉ À UN ARBRE"
1- Initialisation
1_1 : T0 , u0, chaineMarkov, n=0
1_2 : u ← u0, Tk ← T0
2 - TANT_QUE (refroidi=FAUX) faire :
2_1 : forme ← tirageDans (Fk),
2_2 : n ← 0
2_3 :                                   MK                                  ←
card{OrientationsPermises(forme)}xcard{connexitésPermises(forme)}
2_4 : config ← {orientation, symétrie, connexité}
2_5 - TANT_QUE (n< MK) faire :
25_1: n ← n+1;
25_2 : v := transformation(u, Tk, config)
25_3 : n := n+1
25_4 : Dg ← coût(v) - coût(u)
25_5 : p(u,v) = Min (e^(-Δg/T) ;1)
25_6: si p(u,v ) > Random[0;1) alors listeSucc ← v ; "accepté"
25_7 : si coût (v) < coût(meilleur) alors meilleur ← v ;
25_8 : config ← {orientation;connexité;symétrie} de forme
2_6 : u ← { ∀u , g(u) = Min_{u_i ∈ listeSucc} g(u_i) } ;
2_7 : Tk+1← Tk *α
2_8 : si refroidi = VRAI resultat ← meilleur sinon Aller à 2_1.
4 - FIN.
```

Figure III. 6: Algorithme de recuit simulé adapté à la recherche en arbre.

III.4.4. Résultats

Les exemples de placement sont donnés en fin du chapitre.

III.4.4.1. Comportement par palier de température

La courbe à la **Figure** III. 7 montre que les variations de coûts progressent par étape. L'état d'équilibre obtenu au palier précédent permet de progresser vers le minimum et favorise la convergence. Mais l'obtention de l'optimal global n'est pas toujours garantie, puisque la température peut déjà atteindre le stade du refroidissement avant d'atteindre le rendement espéré. Dans ce cas, il faudrait augmenter la longueur de la chaîne de Markov.

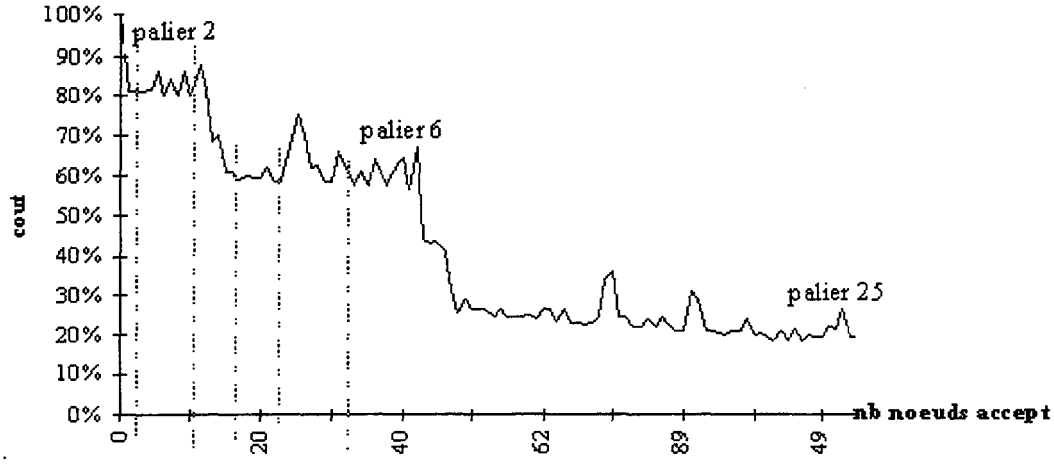


Figure III-7 : Variation de coût pendant la recherche.

Figure III. 7: Evolution par palier de la recherche en variation de coût.

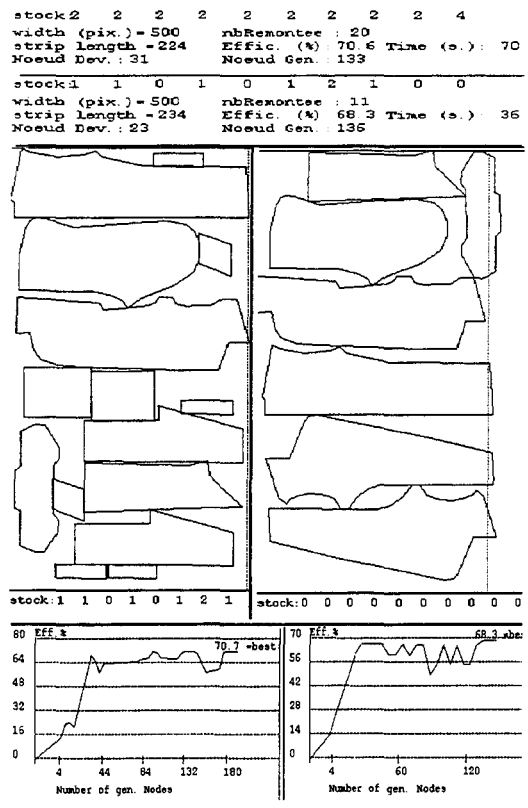


Figure III. 8: Exemple de placement par bande. Les courbes illustrent l'évolution du pourcentage de l'occupation de surface par bande.

III.4.4.2.Recuit

On observe en pratique que les recuits apparaissent plus vers la fin de la bande. Les remontées sont plus fréquentes vers la fin de la bande car de plus en plus de configurations sont rejetées, soit par violation de contraintes (dépassement de largeur de bande) ou augmentation du coût (dépassement en longueur).

Tant que l'algorithme ne rencontre pas de minimal local, la recherche évolue toujours jusqu'à ce que la température atteigne la température de refroidissement (**Figure** III. 9).

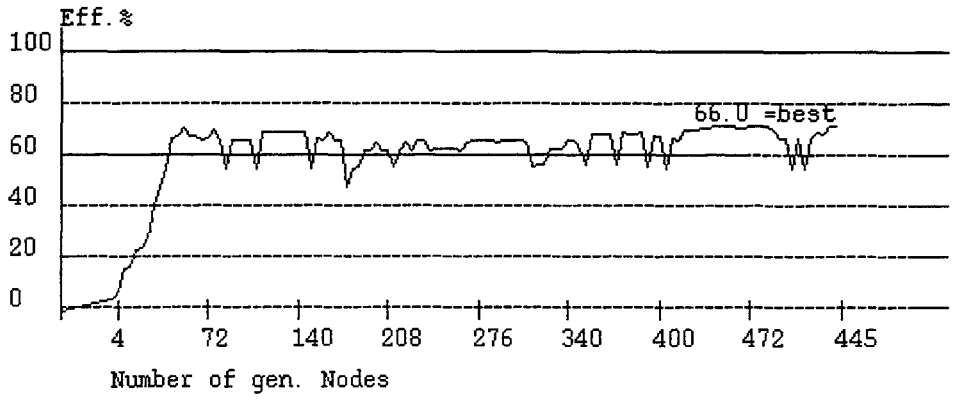


Figure III. 9: Cas où la recherche rencontre beaucoup de minima locaux.

La méthode de recuit utilisée ici n'est pas très efficace. Car si l'association des formes en début de la recherche n'étaient pas elles-mêmes très compactes et si les tirages successifs fournissent la même forme à placer, la recherche peut stagner. Pour résoudre ce problème, trois possibilités sont offertes:

- Soit il faudra modifier le mode de tirage pour que la forme choisie convienne bien (ex. Placement des grandes d'abord). Soit mémoriser la forme du tirage précédent et essayer une forme différente jusqu'à épuisement du tirage. Mais ceci ne résout pas le problème du mauvais placement initial.
- Remonter non pas vers le prédécesseur immédiat mais vers un des noeuds acceptables n'appartenant pas au chemin actuel. Il faudra alors prévoir une liste de mémorisation de ces noeuds dont le nombre peut être important.
- Utiliser le principe de population des algorithmes génétiques. Un noeud jugé "moins bon" au niveau d'énergie actuel peut devenir "meilleur" dans le futur.

III.4.4.3.Complexité du recuit simulé

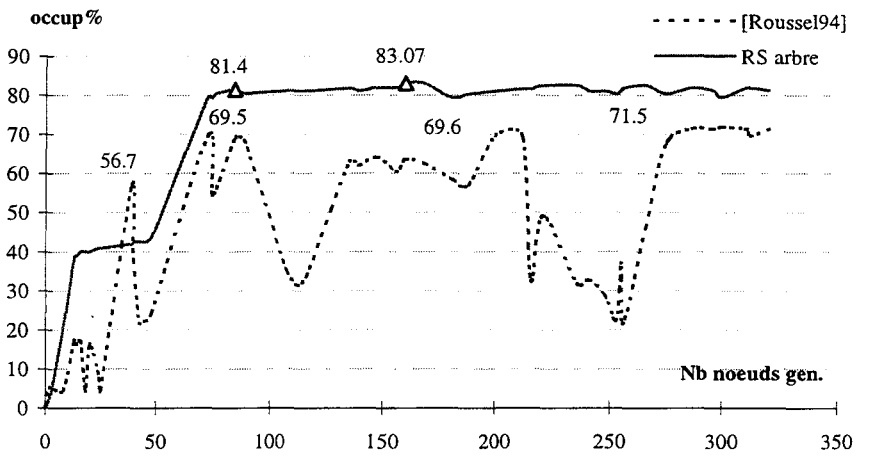


Figure III. 10: Comparaison de l'évolution de l'occupation en surface en fonction du nombre de noeuds générés. La courbe en trait plein montre que notre algorithme adapté converge plus vite par rapport à l'algorithme de référence de *[**Roussel94**]*. Dans ce dernier, les remontées et les descentes dans l'arbre ont les mêmes probabilité de se produire. Rien ne force l'algorithme à descendre (bien que la température continue à décroître). Si bien qu'à la température de gel, l'algorithme ne converge toujours pas.

Les courbes (**Figure** III. 10) montrent que le nombre de noeuds générés est en général deux ou trois fois moins par rapport aux recherches combinatoires A_{ε}^{*} et $R_{\delta\varepsilon}^{*}$ est plus faible. Toutefois, l'obtention de l'optimal n'est pas toujours garantie si la température devient trop faible et bloque trop tôt les possibilités de recuit.

Le résultat final est assez dépendant des placements initiaux. En effet si, les premières formes placées étaient essentiellement des grandes formes, la bande sera vite remplie et les formes placées vers la fin seront essentiellement petites servant à améliorer le rendement en remplissant les petites surfaces restantes. Les remontées finales ne libèrent que de petites surfaces et donc les variations en coût restent faibles. Cependant, les remontées ne peuvent s'effectuer que par niveau immédiatement supérieur à cause de la notion d'antécédence de la concaténation des formes. Ceci implique que le temps pour remonter jusqu'à une noeud assez élevé pour sortir de ce minimum local peut être très long.

III.5. Comparaison avec les Algorithmes en Arbres

Le tableau ci-dessous récapitule les données de comparaison du recuit simulé (RS) avec l'algorithme A^*_ε. Le nombre de remontées (recuits) pour le RS est fixé à 20. Et pour A^*_ε, ε est fixé à 5%.

	Rendement (%)		Temps (20 remontées)		Nombre de noeuds générés	
Carnet	**RS**	A^*_ε	**RS**	A^*_ε	**RS**	A^*_ε
ralf.cde	**68.7**	70.4	**1'19**	2'53"	**130**	1072
chem3.cde	**68.6**	70.5	**13'47"**	11'40"	**864**	4776
gr1.cde	**72.6**	77.0	**2h09'10"**	48'7"	**2316**	6693

Tableau III. 2: Récapitualtifs de quelques résultats comparant le recuit simulé et A^*_ε.

Les rendements obtenus par le RS sont moins bons que les algorithmes de recherche en arbre (**Figure** III. 11). Mais le temps de recherche peut être très compétitif (**Figure** III. 12), par exemple pour un carnet de petite taille (*ralf.cde*), la méthode du RS peut être plus rapide (faible complexité, voir **Figure** III. 13). Pour un problème de plus grande taille (*gr1.cde*), la forme de l'espace de recherche est très irrégulière et multimodale, l'algorithme peut prendre beaucoup de temps pour trouver un état stationnaire dont le coût satisfait le critère d'acceptabilité.

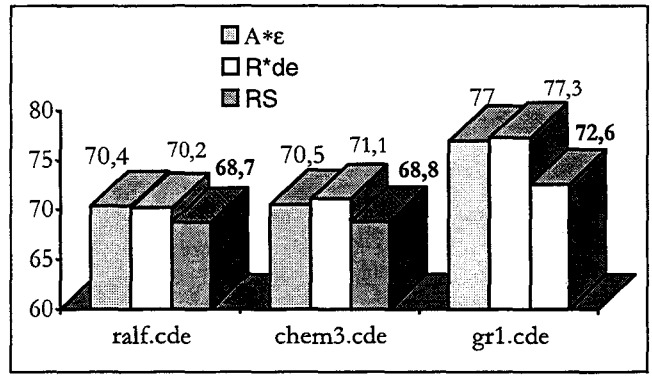


Figure III. 11: Comparaison du recuit simulé avec les algorithmes A^*_ε et $R^*_{\delta\varepsilon}$.

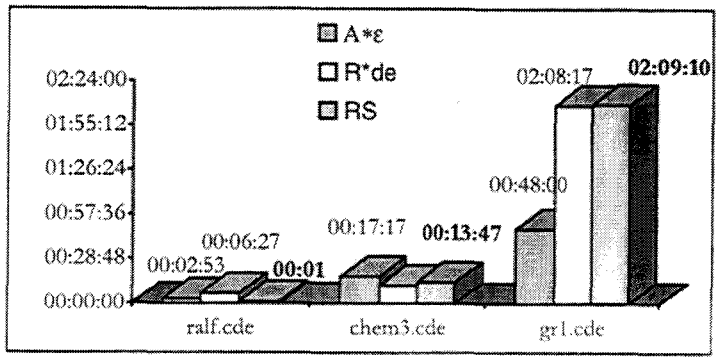


Figure III. 12: Comparaison du temps de recherche.

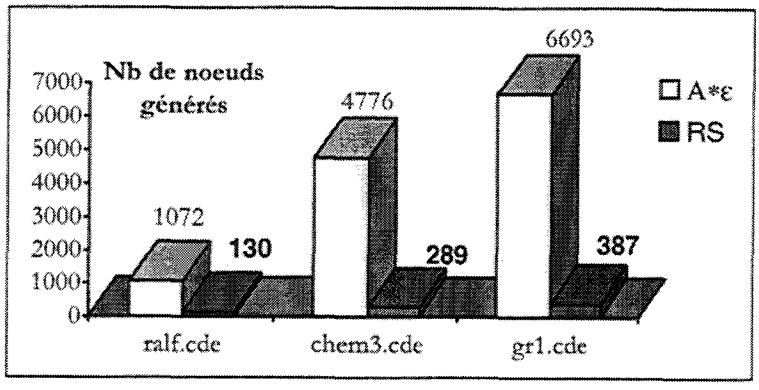


Figure III. 13: Comparaison de complexité, nombre de noeuds générés. Le recuit simulé est beaucoup moins complexe que les recherches énumératives A_{ε}^{*} et $R_{\delta\varepsilon}^{*}$. Le temps de recherche peut être long à cause du processus de "décroissance de température" pour trouver un état stationnaire minimal.

III.6. Conclusion

Dans ce chapitre, nous avons décrit la méthode de recherche stochastique du recuit simulé et son application au problème de placement. L'algorithme implanté est adapté de l'algorithme existant de G. Roussel [**Roussel94**], qui sert de référence dans la comparaison.

La modification de l'algorithme en une recherche similaire à la recherche en arbre, a permis à l'algorithme de mieux converger.

Mais le problème de minima local n'est pas pour autant éliminé. Le recuit simulé est très dépendant de la structure de voisinage et le nombre de paramètres qui gère la recherche. Pour plus d'efficacité, on pourrait envisager le cas où les paramètres sont obtenus par une analyse du carnet de commande et ajustables de façon dynamique au cours de la recherche. Comme le

problème des minima locaux est inhérent au principe même du recuit simulé. Des techniques d'amélioration ont été proposées, notamment la technique du "*recuit très rapide*" (*very fast simulated re- annealing*) **[Ingberg89,93]**.

Comme la plupart de la recherche aléatoire, le recuit simulé ne considère qu'une seule solution à la fois et ne construit pas la forme général du domaine de recherche. Il n'y a pas de sauvegarde des transformations précédentes pour guider la recherche du prochain point. Une solution possible serait de maintenir plusieurs états de recherche à chaque palier de température similaire au principe évolutionniste (voir chapitre suivant). Par exemple, on peut considérer les états comme membre d'une population et chaque palier de température sera considéré comme un stade d'évolution ou génération. Cette technique hybride a été beaucoup appliquée dans de nombreux problèmes **[Davis87, Koakutsu93, Lin93, Yip94]**. Cette hybridation consisterait à tirer profit des deux principes de recherche séquentiel et évolutif. La méthode séquentielle permet d'explorer en profondeur une région donnée alors que la méthode évolutive introduit les règles de conduite générale pour guider la recherche au travers de l'espace de recherche.

III.6.1. Exemple de Placement 1

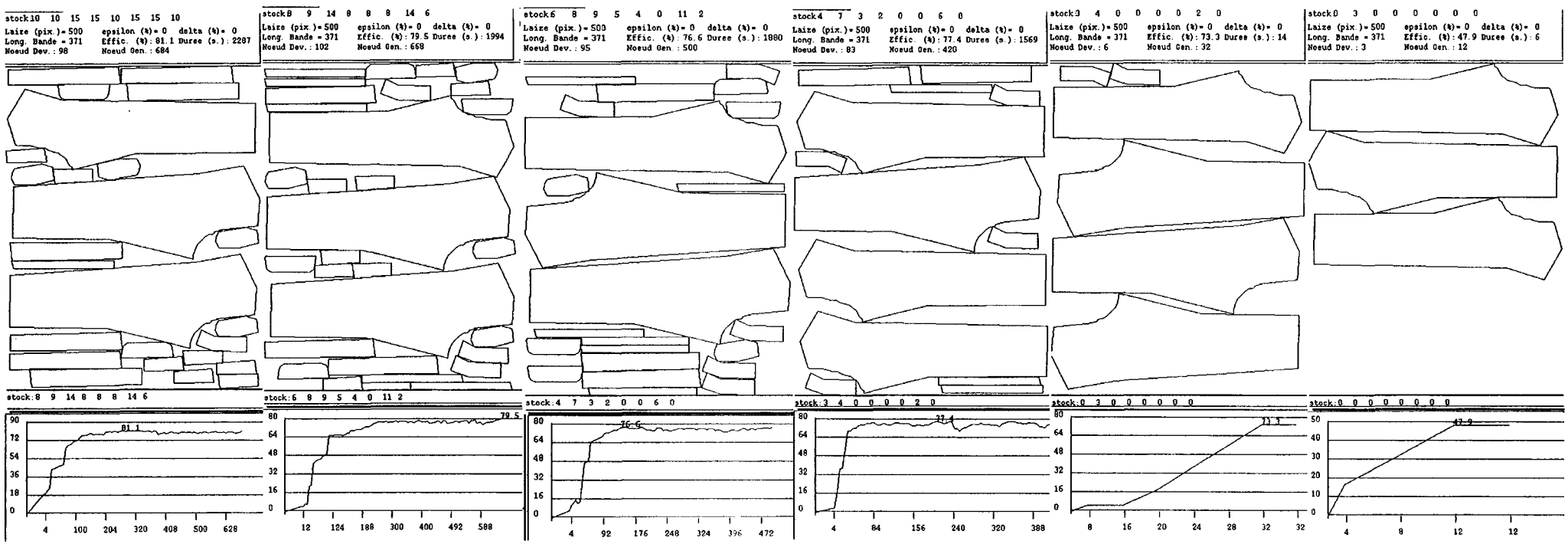


Figure III. 14 :Exemples de placement avec les formes de pantalon (carnet gr1.cde)

III.6.2. Exemple de Placement 2

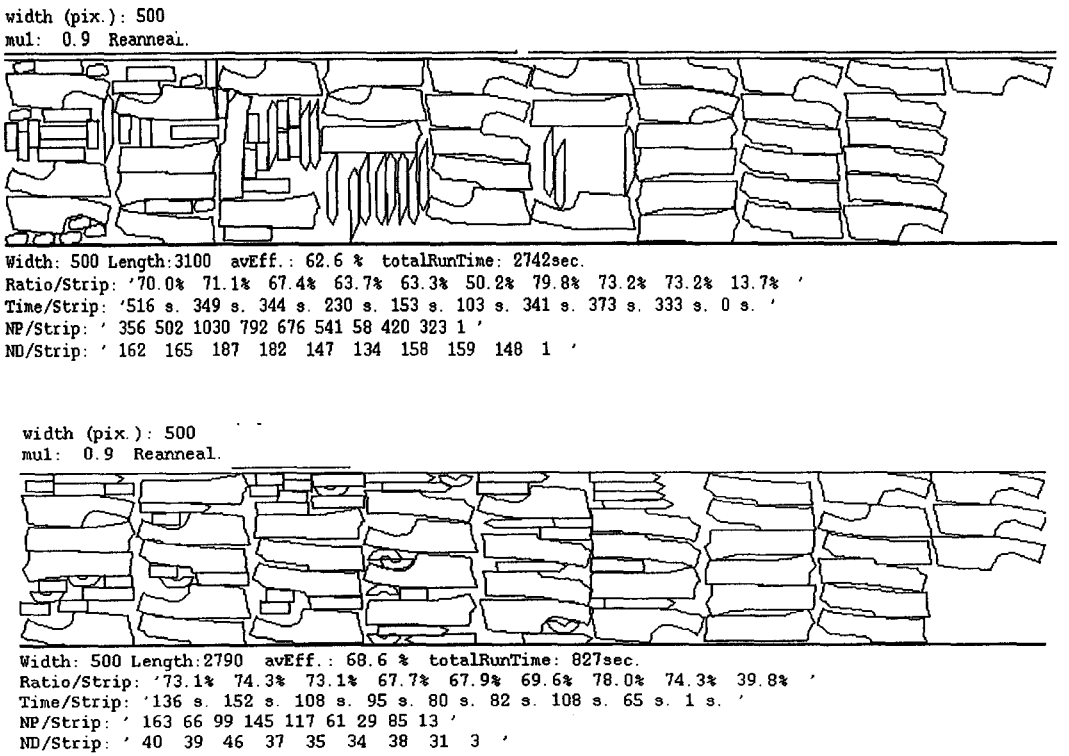


Figure III. 15 : Exemples de placement avec les formes de chemise (carnet chem3.cde).

III.6.3. Exemple de Placement 3

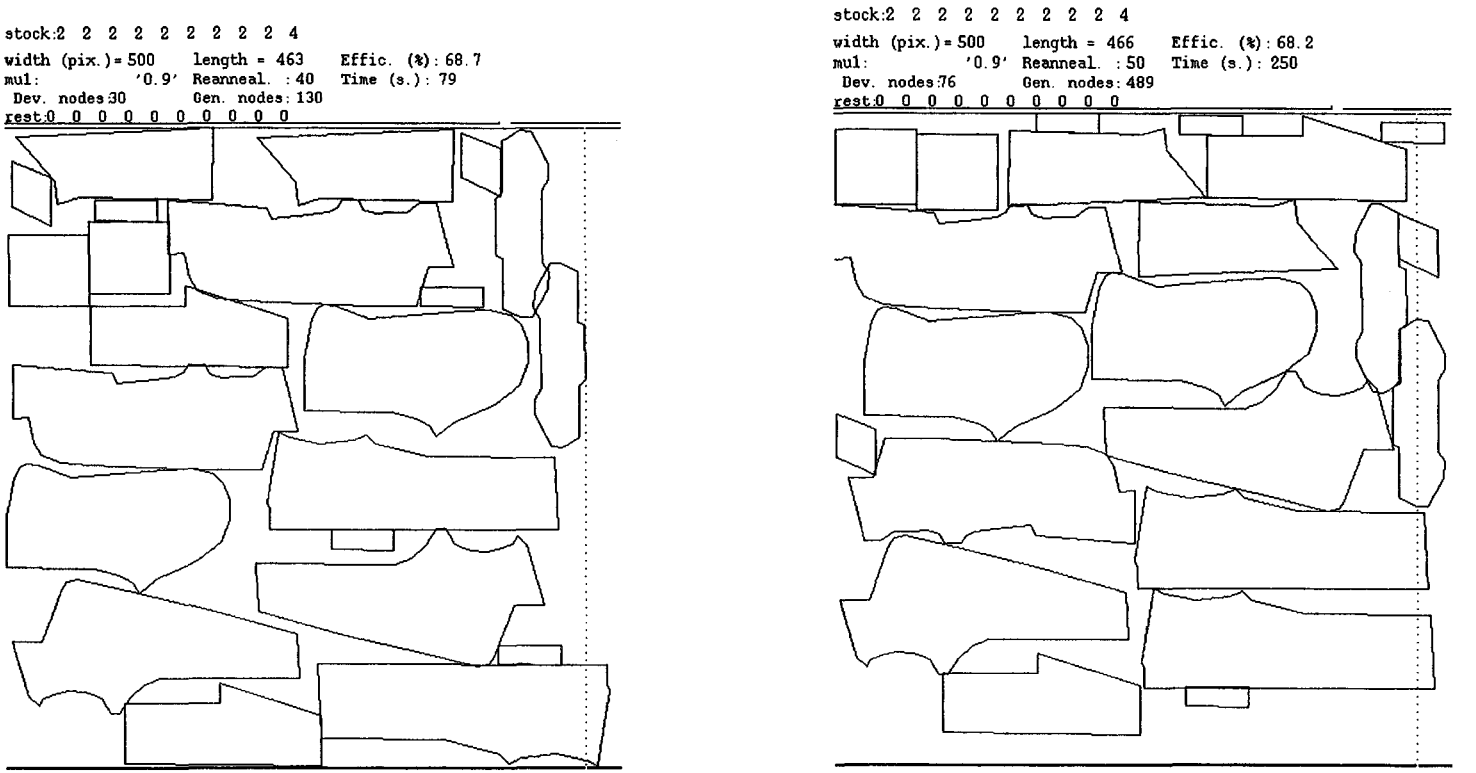


Figure III. 16: Exemples de placement avec les formes de veste (carnet ralf.cde). Ces exemples de placement ont été obtenus sans la méthode par bande.

III.7. Références Bibliographiques

[Heckmann95] R. Heckman and Th. Lengauer, "A simulated Annealing Approach to the Nesting Problem in the Textile Manufacturing Industry ", In R. E. Burkard, P. L. Hammer, T. Ibakari and M. Queyranne, editors, *Annals of Operations Research*, vol. No. 57, pp. 103-133. J. C. Baltzar AG Science Publishers, Amsterdam, 1995.

[Ingberg89] Lester Ingberg, "Very Fast Simulated Re-Annealing", *Journal of Mathematical Computer Modelling*, Vol. 12, No. 8, pp. 967-973,1989.

[Ingberg93] Lester Ingberg, "Simulated Annealing: Practice versus Theory", *Journal of Mathematical Computer Modelling*, vol. 18, no. 11, pp. 29-57, 1993.

[Kirkpatrick83] S. Kirkpatrick et al., "Optimization by Simulated Annealing ", *Science* 220, 671-680, 1983.

[Lin93] F-T Lin, C-Y Kao, C-C Hsu, Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems, *IEEE Trans. On Sys. Man & Cybern.*, Vol. 23, n° 6, nov-dec 1993, pp. 1752-1767.

[Lutfiyya91] H. Lutfiyya and B. McMillin, "Composite stock cutting through simulated annealing", T.R. N° CSC 91-09 or ISC 91-04, dept. Computer Science, Univ. Missouri at Rolla, 1991.

[Oliveira93] J. F. Oliveira and J.S. Ferreira, "Algorithms for Nesting problem", in R. V. Vidal, editor, *Applied Simulated Annealing*, Springer Verlag, Berlin, pp. 255-273,1993.

[Prempti83] F. Prempti, *Méthodes stochastiques dans les problèmes de placement*, thèse de doctorat, Univ. Scientifique et Médicale de Grenoble, département de recherche opérationnelle

[Randelman86] R. E. Randelman, and G.S. Grest, N-City Traveling Salesman Problem - Optimization by Simulated Annealing, *Journal of Statistic Physics*, Vol. 45, pp. 885-890, 1986.

[Renders95] J. M. Renders, *Algorithmes Génétiques et Réseaux de Neurones*, Hermès, Paris 1995.

[Roussel94] Gilles Roussel, *Optimisation du Placement de Formes Irrégulières sur Matières Planes. Application à l'industrie de la Confection*, Thèse de doctorat, Université des Sciences et Technologies de Lille 1, Janv 1994.

[Yip94] P.P.C. Yip, Y.H. Pao, "A Guided Evolutionary Simulated Annealing Approach to the Quadratic Assignment Problem", *IEEE Trans on SMC*, vol. 24, n°9, Sept. 1994.

Chapitre IV:

Algorithmes Evolutionnistes

TABLE DE MATIÈRES

IV.1. Introduction 131

IV.1.1. Historique des Algorithmes Evolutionnistes 132

IV.2. Principe des Algorithmes Evolutionnistes 133

IV.2.1. Codage pour les AG 134

IV.2.1.1. Codage binaire 135

IV.2.1.2. Codage non-binaire 136

IV.2.2. Décodage 136

IV.2.3. Construction de la Population Initiale 137

IV.2.4. Fonction de Fitness 137

IV.2.5. Fitness Scaling 138

IV.2.6. Sélection 139

IV.2.6.1. Sélection par la roue de la fortune (Roulette Wheel Selection) 139

IV.2.6.2. Sélection par rang de classement 140

IV.2.6.3. Sélection par tournoi 141

IV.2.7. Modèle de Reproduction 142

IV.2.8. Opérateurs Génétiques 142

IV.2.8.1. Croisement 143

IV.2.8.1.1. Croisement simple (à un point) 143

IV.2.8.1.2. Croisement multiple 143

IV.2.8.2. Mutation 144

IV.2.9. Critère d'Arrêt 145

IV.2.9.1. Performance 145

IV.2.9.2. Performance en-ligne 146

IV.2.9.3. Performance hors-ligne 146

IV.2.9.4. Convergence 146

IV.2.9.4.1. Convergence d'un code binaire 147

IV.2.9.4.2. Convergence d'un code non-binaire 147

IV.2.10. Compromis entre l'Exploration et l'Exploitation 148

IV.2.10.1. Dérive génétique 148

IV.2.11. Choix des Valeurs des Paramètres 149

IV.2.11.1. Taille de la population 149

IV.2.11.2. Taux de croisement 149

IV.2.11.3. Taux de mutation 149

IV.2.12. Théorème des schémas 150

IV.2.12.1. Définitions 150

IV.2.12.2. Cas non binaire 150

IV.2.12.3. Schémas et hyperplans 151

IV.2.12.4. Hypothèse de "building blocks" 152

IV.3. Aspects Pratiques des AG 153

IV.3.1. Concept de Sous-Population 153

IV.3.2. AG Parallèles 154

IV.4. Application au Problème de Placement 157

IV.4.1. Problème de Codage 157

IV.4.1.1. Codage en chaîne 157

IV.4.1.2. Codage hiérarchique 158

IV.4.2. Première Approche : Codage en Peignes Réduits 161

IV.4.2.1. Utilisation de peignes 162

IV.4.2.2. Codage en peigne pour l'algorithme génétique 163

IV.4.2.3. Opérateurs génétiques 163

IV.4.2.4. Exemple de croisement 164

IV.4.2.5. Remarque sur le codage en peignes 165

IV.4.3. Deuxième Approche : Codage Arborescent 166

IV.4.3.1. Description 166

IV.4.3.2. Structure linéaire associée à un arbre 167

IV.4.3.3. Décodage en phénotype 168

IV.4.3.4. Propriétés 169

IV.4.3.5. Contrôle de redondance 169

IV.4.3.6. Arbre initial 170

IV.4.3.7. Population initiale 170

IV.4.3.8. Sélection pour former le bassin de reproduction 171

IV.4.3.9. Croisement de sous-arbres 171

IV.4.3.10. Mutation d'arbre 173

IV.4.4. Implantation 174

IV.4.4.1. Quelques composantes de l'algorithme en langage Smalltalk 175

IV.4. Résultats 177

IV.5. Discussion 183

IV.6. Conclusion 183

IV.7 Références bibliographiques 185

CHAPITRE IV

Algorithmes Evolutionnistes

IV.1. Introduction

Les algorithmes génétiques (AG), plus généralement regroupés sous le nom d'algorithmes évolutionnistes (AE), sont des méthodes adaptatives généralement utilisées dans les problèmes d'optimisation. Ils sont basés sur le processus d'évolution génétique des organismes biologiques à travers des générations selon la théorie de l'évolution de Darwin. Ces organismes vivent ensemble dans un environnement où ils se reproduisent et se partagent les mêmes ressources telles que nourriture et abris contre les prédateurs. Ainsi, les individus les plus forts ont plus de chance de survivre vis-à-vis de la nourriture et des prédateurs ou bien trouver un partenaire pour se reproduire. Ceux qui ont réussi à survivre et à trouver des partenaires vont générer un nombre relativement plus important de progénitures. Les plus faibles vont avoir peu ou pas de descendants. Par ce fait, les gènes des individus les plus adaptés vont se transmettre dans plusieurs individus des générations suivantes. La combinaison des meilleurs gènes des différents ancêtres peut parfois produire des "super-individus" qui s'adaptent encore mieux que leurs parents. Ainsi, les espèces évoluent et deviennent de plus en plus adaptés à leur environnement.

En imitant ce principe, les algorithmes évolutionnistes appliqués à un problème d'optimisation font évoluer un ensemble de solutions candidates, appelé *population* d'individus. Un individu représente une solution possible du problème donné. A chaque individu est attribué un "*fitness*" qui mesure la qualité de la solution qu'il représente, souvent c'est la valeur de la fonction à optimiser. Ensuite, une nouvelle population des solutions possibles est produite en sélectionnant les parents parmi les meilleurs de la "*génération*" actuelle pour effectuer

des *croisements* et des *mutations* . La nouvelle population contient une plus grande proportion de caractéristiques des meilleurs individus de la génération précédente.

De cette façon, de génération en génération, les meilleurs gènes se propagent dans la population, en se combinant ou échangeant les meilleurs traits. En favorisant les meilleurs individus, les régions les plus prometteuses de l'espace de recherche sont explorées. Si l'algorithme est bien conçu (*codage*, fonction d'évaluation des individus et d'autres paramètres judicieusement choisis), la population convergera vers un état stationnaire minimal.

IV.1.1. Historique des Algorithmes Evolutionnistes

Les algorithmes évolutionnistes regroupent différentes méthodes d'optimisation stochastiques basées sur le même paradigme (voir *Annexe 1: Classification des méthodes de recherche*). Il s'agit de faire évoluer une *population* d'*individus* en utilisant des *opérateurs stochastiques (croisement, mutation)* et le principe de *survie du meilleur.*

Les méthodes les plus répandues sont les algorithmes génétiques (***AG***). Les AG furent développés à l'Université de Michigan (USA) par J. Holland [**Holland75**] puis par d'autres chercheurs tels que J. De Jong [DeJong75], J. J. Goldberg [**Goldberg89**], L. Davis [**Davis91**] et Z. Michalewicz [**Michalewicz92**].

La programmation évolutionniste (***PE***) fut développée aux USA [**Fogel64, Fogel95**].

Les stratégies évolutionnistes (***SE***) pour l'optimisation numérique sont développées en Allemagne par H. P. Schwefel et I. Rechenberg[**Rechenberg94**].

Récemment, les extensions apportées par J. Koza [**Koza92**], ont permis le développement de la programmation génétique (***PG***). La PG permet de générer des fonctions informatiques à partir des principes évolutionnistes, la population est un ensemble de codes de base de programmes informatiques de type LISP.

Globalement, les différences entre ces méthodes résident dans la stratégie de résolution [**Bäck93, Park94, Fogel94**]. Les AG sont considérés comme des méthodes de résolution "ascendantes", c'est-à-dire que la solution optimale peut être obtenue progressivement en assemblant des parties optimales des solutions partielles, les opérateurs de recombinaison jouent alors un rôle essentiel. Les stratégies évolutionnistes et la programmation évolutionniste sont vues comme des méthodes "descendantes", dans lesquelles l'environnement agit comme une pression pour faire émerger la solution optimale. Les opérateurs de recombinaison y ont un rôle secondaire [**Fogel94**]. Une analyse plus fine fait apparaître une autre différence qui réside dans la représentation des individus. Dans la communauté AE, il est

unanime que le succès ou l'échec de la méthode réside dans la définition de l'espace *génotypique*[1] (l'espace d'action des opérateurs) et l'espace *phénotypique* (l'espace de calcul de fitness ou d'observation).

Les premières applications étaient marquées par l'utilisation de chaînes binaires de longueur fixe pour les AG, des vecteurs de réels pour les stratégies évolutionnistes (*cf. Annexe 2*), un automate à état fini pour la programmation évolutionniste et des codes de programme pour la programmation génétique (*cf. Annexe 3*). Ceci jusqu'au début des années 80 où les recherches étaient principalement théoriques avec peu d'applications réelles [**Goldberg89**]. Ensuite, après de nombreuses investigations dans diverses disciplines, chaque nouvelle application apportait une nouvelle perspective à l'amélioration de la théorie. Les tendances actuelles sont plus pragmatiques et acceptent les représentations spécifiques au domaine, par exemple en traitement d'images, dans les problèmes d'ordonnancement, le codage des formes ou le placement de composants [**Cohoon87**, **Cohoon91**, **Lutton93**, **Caux95**, **Schoenauer96**]. La flexibilité est valable aussi pour la programmation évolutionniste où divers types de codes sont traités ainsi que pour la stratégie évolutionniste où l'on introduit en plus des opérateurs standards des opérateurs spécifiques [**Rosca94**, **Oppacher95**]. Il est maintenant difficile de trouver des frontières réelles entre les méthodes. En l'occurrence, on utilisera indifféremment dans la suite les termes algorithme évolutionniste ou algorithme génétique.

Dans l'état actuel des recherches, les algorithmes évolutionnistes sont encore à l'état de développement sur les fondements théoriques. Mais, toujours est-il qu'ils connaissent un succès grandissant en recherche opérationnelle et en intelligence artificielle, notamment dans la résolution de problèmes d'optimisation et d'apprentissage [**Golberg89**, **Wah92**, **Gathercole94**].

IV.2. Principe des Algorithmes Evolutionnistes

Les algorithmes évolutionnistes partagent un paradigme commun [**Goldberg89**, **Michalewicz92**, **Bäck93**, **Beasley93a**, **Beasley93b**, **Alliot94**, **Renders95**]. Leurs caractéristiques principales sont les suivantes (**Figure** IV. 1) :

[1] En terme génétique, l'ensemble de paramètres représenté par un *chromosome* particulier est appelé *génotype*. Le génotype contient les informations nécessaires pour construire un organisme - qui est appelé *phénotype*. Le phénotype est défini comme caractéristiques visibles ou exprimées du génotype. Les mêmes termes sont utilisés en AG. Le phénotype est déduit du génotype.

- Une représentation génétique du problème, c'est-à-dire un codage approprié des solutions sous forme d'individus (ou génotype);
- Une fonction d'évaluation pour sélectionner les individus selon leur "force" *(fitness)*;
- Un mode de sélection des individus à reproduire;
- Des opérateurs génétiques pour modifier les représentations associées aux individus;
- Des paramètres internes de l'algorithme (ex: taille de la population, probabilité d'application de chaque opérateur...).

```
Début
1°) Initialisation
2°) Evaluation
3°) TANT que (critère_d_arrêt=FAUX)
4°) Evolution
   41°) Sélection
   42°) Recombinaison(croisement, mutation)
   43°) Evaluation
5°) Fin Tant que;
Fin.
```

Figure IV. 1: Forme générale d'un algorithme évolutionniste. La recherche s'effectue à partir d'une *population* de points; la progression d'un état à un autre s'effectue par des *opérateurs* stochastiques tels que la *sélection*, le *croisement* et la *mutation*.

IV.2.1. Codage pour les AG

Pour appliquer un AG, la première chose à se demander est: "quel génotype doit-on utiliser pour le problème?" C'est à dire, comment les paramètres peuvent se mettre sous formes de chaînes binaires (si le code binaire est absolument nécessaire)? En l'occurrence, combien de bits par paramètres, dans quel domaine de valeurs décimales décodées?. Est-ce que le code doit avoir une longueur constante ou dynamiquement changeante? etc.

La littérature insiste sur le fait que les AG ne "voient" pas le domaine du problème parce que ce dernier est occulté par la représentation ; si celle-ci est mal conçue, on risque d'exclure certaines solutions de l'espace de recherche. Donc la question n'est pas "*quel type de problèmes peut être résolu avec un AG?*" mais plutôt "*Est-ce que la représentation utilisée peut conduire à l'efficacité de l'AG ?*". C'est-à-dire, quelle représentation contiendrait le mieux les caractéristiques essentielles de

l'espace de recherche. Il est difficile de savoir à l'avance quelles seront les éventuelles similarités que l'on trouvera dans cet espace.

Pour J. Holland [**Holland75**], la meilleure représentation est le codage binaire. Car, selon lui, plus une représentation est détaillée plus on peut en extraire des informations sur la similarité entre les chaînes. Cette représentation est pédagogiquement la plus facile à comprendre et à expliquer.

IV.2.1.1.Codage binaire

Les AG primitifs utilisaient un codage binaire de longueur fixe. Un élément de la population est codé en chaîne de longueur n de caractères pris dans l'alphabet fini $\{0, 1\}$.

Par exemple, prenons un problème de partition [**Khuri94**, **Chu95**] où l'ensemble **E** doit être divisé en sous-ensembles **V** comportant des éléments de **E** dont la somme ne dépasse pas un certain nombre fixé. Un candidat **V** est codé par un vecteur **x** tel que :

$\mathbf{x} = \{x_1 x_2 ... x_n\}$ avec $x_i = 1$ lorsque le $i^{\text{ème}}$ élément de **E** est présent dans **V** sinon $x_i = 0$.

Pour un ensemble **E**={1,2,3,4,5,6,7,8,9} et un sous-ensemble **V**={2,3,6,8}, le code correspondant à **V** est le vecteur **x** = 011001010.

Ce principe de codage est applicable à beaucoup de problèmes faisant intervenir la présence ou l'absence d'un élément dans un ensemble fini d'éléments ou lorsque chacun des paramètres peut être représenté par un seul bit (gène ou *allèle*[2]). Mais lorsque les paramètres sont codés par plusieurs bits (gène *multi-allèle*), chacun d'eux est codé avec une longueur de chaîne et intervalle [*min*, *max*] propres. Ensuite, on les concatène de façon à former une chaîne unique.

Exemple : [0001 0101 ... 100 11111] [$param_1$ $param_2$ ... $param_N$]

Dans une optimisation de fonction réelle avec un codage binaire, la longueur du chromosome dépend de la précision que l'on désire obtenir. Par exemple, si une fonction g doit être optimisée sur l'intervalle [-1 ; 2] avec une précision de six chiffres après la virgule, il faudra alors diviser l'intervalle [-1 ; 2] en 3×1000000 intervalles plus petits. Ce qui donne un chromosome de 22 bits ($2^{21}<3000000<2^{22}$).

Il est certain que tous les problèmes ne peuvent être facilement codés en binaire ou alors il est pratiquement prohibitif par l'encombrement de mémoire. Il apparaît alors d'autres formes de codage non binaire plus spécifique au problème traité (dans ce cas, on parle plutôt d'algorithme évolutionniste [**Michalewicz92**]).

---

[2] Un allèle est un élément d'un gène

IV.2.1.2.Codage non-binaire

Avec les développements récents, d'autres types de représentation plus sophistiquée sont apparus. Ainsi, un chromosome peut être une liste des villes à parcourir pour le problème du voyageur de commerce [**Grefenstette87a**, **Lin93**], une liste d'objets à manipuler pour un problème du bin-packing [**Khuri95**, **Reeves95**, **Faulkenauer96**], un chromosome formé de deux listes pour le placement des composant en VLSI [**Cohoon87**], une matrice [**Caux95**, **Schoenauer96**] ou les arbres de symboles.

Dans un génotype en arbre, un individu peut-être un arbre de symboles (comme pour la programmation génétique [**Koza92**], *Annexe 3*). L'individu peut être aussi une combinaison de chaînes et d'arbre dont certaines parties seulement évoluent et pas d'autres. Dans d'autres applications, le génotype peut être une liste de symboles qui indiquent des opérations à effectuer par un algorithme qui interprète les symboles et qui attribue un fitness selon le résultat de ces interprétations. Nous verrons plus en détail ce type de représentation dans la deuxième partie de ce chapitre.

IV.2.2. Décodage

Le problème du décodage (appelé aussi *expression*) pour convertir le génotype en phénotype, concerne surtout les codes indirects et mixtes (**Figure** IV. 2). Le décodage convertit une chaîne en paramètres du problème afin de pouvoir évaluer la qualité de la chaîne (**Figure** IV. 3) . Le phénotype est ensuite fourni à la fonction de fitness qui retourne la valeur du fitness permettant de classer la chaîne dans la population.

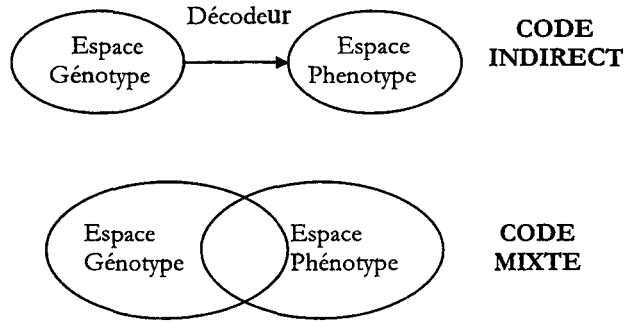


Figure IV. 2: Nécessité d'un algorithme de décodage pour les codes de génotype indirects et mixtes.

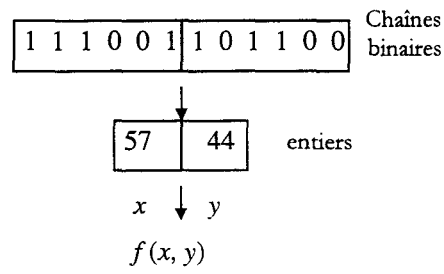


Figure IV. 3: Exemple de décodage d'une chaîne binaire à deux paramètres.

Les étapes suivant la définition du codage consistent en la génération de la population initiale et la définition du mode de sélection selon la fonction d'évaluation ainsi que les opérateurs génétiques permettant les modifications et recombinaisons du code.

IV.2.3. Construction de la Population Initiale

La population initiale peut être obtenue en générant aléatoirement les chaînes de l'espace de recherche. Cette méthode a l'avantage de commencer la recherche à partir de diverses solutions de l'espace de recherche.

Pour privilégier une direction de recherche ou lorsque le problème est fortement contraint, les individus peuvent être également initialisés de façon heuristique ou directement introduits par l'utilisateur [**Caux95**]. Mais cette méthode peut faire converger trop rapidement la recherche vers un optimum local.

IV.2.4. Fonction de Fitness

La construction d'une fonction de fitness appropriée est très importante pour obtenir une bon fonctionnement de l'AG. Cette fonction représente l'environnement du problème, c'est-à-dire qu'elle permet de décider combien la solution présentée par le génotype peut résoudre le problème. Les critères à prendre ne compte pour construire une fonction de fitness sont les suivants[3](**Tableau** IV. 1):

La fonction de fitness:

[3] The flying_Circus d'EvoNet: http://www.wi.leidenuniv.nl/~gusz/Flying_Circus/1.Reading/Tutorial/

• Dépend des critères que l'on veut maximiser ou minimiser.
• Est une boîte noire dont l'entrée est le phénotype et la sortie est la valeur du fitness.
• Peut changer de façon dynamique pendant le processus de recherche.
• Peut être si compliquée qu'on ne peut calculer que sa valeur approchée.
• Devrait attribuer des valeurs très différentes aux individus afin de faciliter la sélection.
• Doit considérer les contraints du problème. S'il peut apparaître des solutions invalides, la fonction de fitness doit pouvoir attribuer une valeur proportionnelle à la violation des contraintes.
• L'environnement peut présenter du bruit dans les évaluations (partielles).
• La valeur de la fonction de fitness peut être aussi attribuée par l'utilisateur (ex. valeur esthétique [**Gritz95**]).

Tableau IV. 1: Considérations pour choisir la fonction de fitness.

IV.2.5. Fitness Scaling

Comme la recherche progresse, les valeurs de fitness peuvent devenir de plus en proches et à pleine échelle, il peut être difficile de distinguer les individus à partir de leur fitness. Il existe alors certains opérateurs pour changer d'échelle de la fonction pour faciliter le mécanisme de la sélection. Les plus communs sont les suivants [**Grefenstette86, Michalewicz92**]:

- Echelle linéaire (linear scaling) :

  $f' = a + b \times f$

  les constantes a et b doivent être calculées selon la fenêtre du "zoom".

- Echelle selon la loi de puissance (*Power Law Scaling*) :

  $f' = f^k$

- Tronquage par une fonction sigma

  $f' = \mathbf{max}[0, f - (\bar{f} - c\sigma)]$, *avec* $\bar{f}$ (resp. σ) la moyenne de fitness value (resp. écart-type) de la population.

La **Figure** IV. 4 donne un exemple de fitness scaling linéaire. Dans cet exemple, la recherche a atteint le stade où les individus ont le fitness compris entre 105 et 110. Pour une

échelle ayant $f_{min} = 0$, la distinction entre les individus est moins précise que si on réajuste f_{min} à f'_{min} qui vaudrait ici 100.

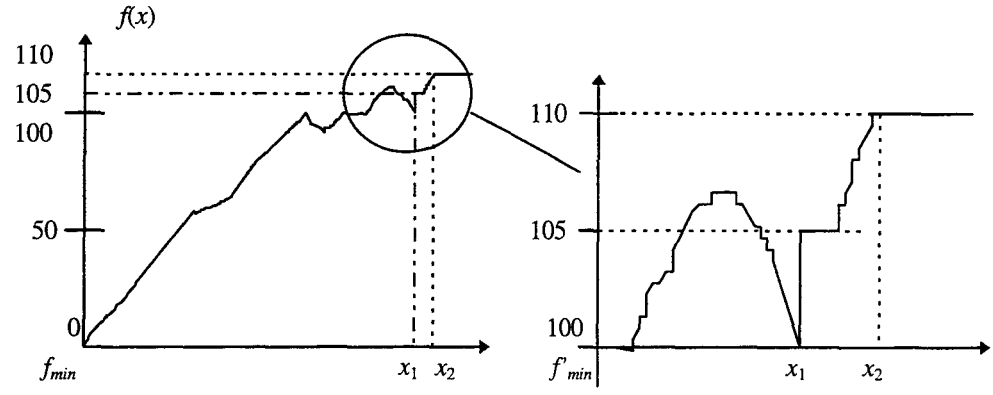


Figure IV. 4: Le fitness scaling permet de dilater l'échelle dans la région de recherche actuelle. Dans cet exemple, le premier graphique montre qu'à pleine échelle ($f_{min} = 0$), la distinction entre les individus de fitness 100 et 105 est moins précise qu'avec l'échelle réajustée dans le deuxième graphique ($f_{min} = 100$).

La valeur de f'_{min} est remise à jour quand le fitness de tous les individus d'une population devient supérieur à f'_{min}. Elle est ajustée à la plus petite valeur de $f(x)$ apparue dans la fenêtre des W dernières générations. J.J. Grefenstette [**Grefenstette86**] préconise la valeur du paramètre W entre 0 et 7.

IV.2.6. Sélection

Comme dans la sélection naturelle, un caractère stochastique est introduit dans la probabilité de sélection qui est souvent basée sur la fonction de *fitness*. Les individus sélectionnés sont placés dans un bassin de reproduction dans lequel auront lieu des opérations de croisement et de mutation. Plusieurs procédures de sélections existent. On peut citer la sélection proportionnelle par la roue de la fortune (*roulette wheel*), par rang de classement dans la population (*rank*), par tournoi (*tournament*).

IV.2.6.1. Sélection par la roue de la fortune (Roulette Wheel Selection)

Dans cette méthode de sélection, les chromosomes sont choisis avec une probabilité proportionnelle à leur fitness, par exemple un individu avec un fitness de 2 aura deux fois plus de chance que celui qui n'en a que 1.

La métaphore de la roulette vient de la procédure informatique pour appliquer cette méthode de *sélection proportionnelle.* Chaque case de la roue est pondérée proportionnellement à la valeur de fitness de chaque individu. Ainsi, les individus ayant la plus grande valeur de fitness auront plus de chance d'être choisis [**Goldberg89**]. Dans une population de N individus, la fonction de sélection est la suivante:

$$\forall i \in N,\ p(x_i) = \frac{fitness(x_i)}{\sum_{j=1..N} fitness(x_j)}$$

Avec cette sélection, chaque individu x_i peut être choisi $N \times p_s(x_i)$ fois pour être mis dans le "*bassin de reproduction*". S'il existe un individu dont le fitness est largement plus élevé que les autres, ses chromosomes sont prépondérants dans ce bassin et ses descendants domineront dans la population suivante.

En terme d'optimisation, l'algorithme converge plus vite vers la direction représentée par ce point, c'est le phénomène de la *convergence prématurée.* De plus, la population devient homogène, la recherche stagne et se comportera comme une promenade aléatoire.

Pour éviter qu'un même individu soit sélectionné trop souvent, Z. Michalewicz a proposé une méthode de probabilité cumulée [**Michalewicz92**].

IV.2.6.2.Sélection par rang de classement

Dans ce schéma de sélection, les individus d'une population sont classés dans une liste selon l'ordre croissant de leur fitness, ensuite la sélection est proportionnelle à leur rang dans la liste de la population [**Davis91**]. Cette méthode est utilisée pour une fonction de fitness mal définie ou quand les valeurs de fitness sont très proches.

Le classement permet de calculer le nouveau fitness basé sur le rang :

$$F' = MAX - (rang - 1) * (\frac{MAX - MIN}{N - 1})$$

avec $N = taille\_de\_population,$

$1.0 < MAX \leq 2.0$ *et* $MIN = 2 - MAX$

et $rang \in \{1,2,3,\ldots,N\}$

Cette nouvelle fonction de fitness sert ensuite dans la sélection par "échantillonnage stochastique universel" ("Stochastic Universal sampling" [**Bäcker87**]), qui ressemble un peu au critère de Metroplis. Cette sélection est décrite par l'algorithme en langage C suivant:

```
/*Algorithme d'Échantillonnages stochastique Universel */
    temp = random dans [ 0 , 1 ];
somme = 0.0;
for ( i = 0; i < N; i = i + 1 ) /* N = taille de la population */
      for ( somme = somme + ExpVal[ i ]; /* ExpVal[ i ] contient la probabilité
   qu'un individu i soit sélectionné, c-à-d. F_i / F' */
          somme > temp;
          temp = temp + 1.0 )
SelectInd( i ); /* sélectionner individu i */
```

Avec cette nouvelle fonction de fitness, les meilleurs ont toujours plus de chance d'être choisis, mais moins souvent que la RWS et les moins bons individus auront aussi plus de chance de participer au bassin de reproduction.

IV.2.6.3.Sélection par tournoi

Cette méthode consiste à prélever au hasard un échantillon de n individus (2 minimum) à chaque *tournoi*, c-à-d, à chaque fois qu'on a besoin d'un parent de plus dans le bassin de reproduction. Ensuite, le meilleur de cet échantillon est choisi pour être parent. La taille de l'échantillon détermine le degré de compétition. Cette méthode est utilisée dans le cas d'une grande taille de la population.

La sélection d'un individu n par tournoi binaire du $n^{ième}$ individu se fait de la façon suivante:

$$select_n := \begin{cases} ind_i & si\ fitness(ind_i) > fitness(ind_j) \\ ind_j & \text{sinon} \end{cases}$$

pour $n=\{1,2,3,...,N\}$, et nombres aléatoires $i,j \in \{1,2,3,...,N\}$, $i \neq j$.

Le remplacement binaire d'un individu n par deux individus créés se fait alors de la façon suivante:

$$remplace_n := \begin{cases} ind_i & si\ fitness(ind_i) < fitness(ind_j) \\ ind_j & sinon \end{cases}$$

pour $n = \{1,2\}$, et nombres aléatoires $i,j \in \{1,2,3,...,N\}$, $i \neq j$.

Un remplacement déterministe remplace systématiquement le plus mauvais individu, stratégie utilisée par D. Whitley dans le logiciel GENITOR [4].

IV.2.7. Modèle de Reproduction

Pendant la phase de reproduction, les chromosomes sont sélectionnés et leurs structures sont modifiées pour générer de nouveaux individus qui vont former la génération suivante. Différentes stratégies existent pour maintenir la population constante. Les stratégies pour décider le "sort" des autres individus qui ne sont pas sélectionnés sont les suivantes:

- La stratégie *élitiste* favorise le meilleur individu. Ce dernier reçoit un traitement d'élite et n'est jamais efface de la population.

- La stratégie de *renouvellement de la population* (*generational replacement*) remplace la population entière par des individus nouvellement créés. Cette stratégie est facile à implanter mais elle peut faire perdre les traces de la recherche par élimination systématique des individus.

- La stratégie *un-seul-à-la-fois* (*steady-state*) consiste à sélectionner dans la population initiale deux parents, les croiser entre eux pour générer un (ou deux) enfant(s). L'enfant est muté puis réinséré dans la population, en utilisant par exemple le remplacement binaire. Cette stratégie favorise la progression s'opère dans le voisinage. Aussi, parents et enfants cohabitent dans la même population pendant plusieurs générations. Il arrive qu'un individu n'est jamais choisi pour être parent. Une solution consiste à affecter une durée de vie à chaque individu, appelée aussi "taux de dégénérescence" **[Michalewicz92, Chung93]**. La durée de vie des individus est un paramètre introduit pour éviter qu'un même individu apparaisse trop souvent d'une génération à l'autre. Lorsque la présence d'un individu dans la population dépasse la durée maximale affectée en nombre de générations, il est éliminé de la population.

L'autre alternative est la *recherche aléatoire*. La création des enfants est faite de façon aléatoire et sans être basée sur le matériel génétique des parents. Cette stratégie ne permet de d'obtenir l'évolution.

IV.2.8. Opérateurs Génétiques

Les parents sélectionnés sont introduits dans le bassin de reproduction (*mating pool*) où ils seront de nouveaux choisis aléatoirement pour voir leurs chromosomes subir des transformations par les *opérateurs génétiques*. Les deux principaux opérateurs de base sont le *croisement* et la *mutation*. Le croisement réalise une opération binaire (ou sexué), et nécessite deux

[4] http://www.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/genitor

parents. La mutation est une opération unaire (ou asexuée), utilisée pour introduire une faible variation dans la solution ou changer la direction de la recherche.

IV.2.8.1.Croisement

Le croisement est un échange par *blocs* d'éléments entre deux chaînes pour en générer un ou deux autres. Il existe plusieurs façons de procéder à cet échange selon les contraintes du problème traité. Ci-après nous décrivons les méthodes les plus courantes avec les chromosomes en chaîne.

IV.2.8.1.1.Croisement simple (à un point)

Un site de croisement est choisi au hasard sur la longueur de chaque chromosome parent et une coupe du chromosome est réalisée (appelé *site* de croisement). Cette coupe produit deux morceaux que l'on permute. Les chaînes enfants résultantes, contiennent chacune un morceau hérité de chacun des parents (**Figure** IV. 5).

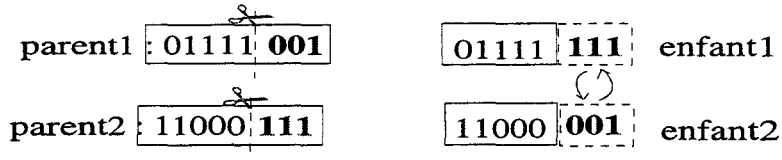


Figure IV. 5: Croisement simple avec un site de croisement.

Pour les chromosomes de faible longueur, ce type de croisement est suffisant. Cependant, si la stratégie de sélection élitiste est utilisée, après plusieurs générations lorsque la population devient uniforme, ce croisement risque de ne plus créer de diversité.

IV.2.8.1.2.Croisement multiple

Le croisement précédent peut être généralisé en croisement en k-points de coupe, générant k+1 sous-chaînes pour chaque chromosome. Les deux chromosomes-fils sont obtenus par concaténation de ces sous-chaînes en alternant les parties venant de chaque parent.

Pour éviter ce caractère systématique, certains auteurs introduisent un nombre aléatoire pour décider de quel parent sera pris chaque élément. Par exemple, pour une chaîne de longueur m, le croisement est effectué en m étapes. A l'étape k, le $k^{ème}$ gène du parent_1 est recopié avec une probabilité de 0.5, à la position correspondante sur le fils_1. Le $k^{ème}$ gène du

parent_2 est recopié sur le fils_2, à la même position. Ce croisement, appelé *uniforme*, est souvent utilisé pour les chromosomes de longueur élevée et lorsqu'on désire créer un effet "cassure" plus important [**Goldberg89, Davis91**].

En général, les méthodes du croisement sont diverses et fortement dépendantes du codage et les contraintes du problème. Pour les chromosomes non binaires basés sur l'ordre des éléments comme dans le problème du voyageur de commerce, il existe aussi des croisements standards [**Falkenauer94a, Goodman94, Reeves95, *Annexe 4***].

Il faut remarquer que le résultat du croisement ne produit pas toujours des chromosomes valides dans le domaine de recherche. Il existe des croisements heuristiques ou des algorithmes de *réparation* pour rester dans le domaine de recherche [**Michalewicz92**]. La réparation des chromosomes invalides peut être aidée par le phénomène de mutation. Dans beaucoup d'applications, la mutation est appliquée immédiatement après l'opération de croisement.

IV.2.8.2.Mutation

La mutation est une forme de l'adaptation génétique de l'individu à son environnement. Elle se manifeste par une modification brutale d'un gène. Le phénomène se produit très rarement dans la nature (de 0,1% à 5% de la population).

L'opérateur de mutation est un opérateur unaire générant un seul individu à partir d'un seul individu. L'opérateur consiste à modifier la valeur d'un élément de la chaîne. Pour le codage binaire, il suffit de changer un bit choisi aléatoirement, de 0 en 1 ou vice versa.

Le mode de mutation varie d'une application à l'autre. Pour obtenir une variation plus uniforme, L. Davis [**Davis91**] propose la mutation par *balayage*. Dans cette mutation, chacun des *n* bits de la chaîne est testé pour la mutation. Pour cela, on génère *n* nombres aléatoires que l'on compare un par un au taux de mutation. Si le nombre aléatoire est inférieur à ce taux, le bit correspondant change d'état. Ce dernier est généralement généré de manière aléatoire.

Exemple : Taux de mutation : 0.008.

chaîne$_1$: (1 0 1 0) , Test : (0.8 ; 0.10 ; 0.28 ; 0.37)

→ Tous les nombres de Test sont supérieurs à 0.008. La chaîne$_1$ reste inchangée.

chaîne$_2$: (1 1 0 0) , Test : (0.12 ; 0.09 ; 0.005 ; 0.8)

→ Le bit 3 subit une mutation. On génère un bit aléatoire. Si c'est 0, la chaîne$_2$ ne change pas, si c'est un 1, la chaîne$_2$ devient : (1110).

Cette mutation permet de changer plusieurs bits à chaque opération mais nécessite un peu plus de temps de calcul que la mutation standard.

La mutation peut être appliquée à certains individus immédiatement après le croisement. Si le nombre aléatoire généré pour le nouvel individu est inférieur au taux de mutation, cet individu sera muté.

La volonté de limiter les occurrences de mutation vient de l'idée fondamentale de l'AG. La considération traditionnelle est que le croisement doit être plus important pour explorer rapidement l'espace de recherche. La mutation sert d'opérateur d'exploitation permettant d'effectuer un petit déplacement dans l'espace de recherche, une sorte de *recherche dans le voisinage* dont la progression s'opère point par point [**Goldberg89**]. Ce qui n'est pas vraiment la motivation principale des AG, c'est pourquoi le taux de mutation dans les algorithmes reste très faible.

Certains individus non sélectionnés ni pour le croisement ni pour la mutation peuvent réapparaître intacts dans la nouvelle population. Ceci permet à ces individus de préserver leurs structures (on appelle aussi reproduction par *copie conforme*). Le paramètre pour contrôler ce phénomène est aussi appelé l'écart de génération **G** (*generation gap*). Il est choisi entre 0.3 et 1. Donc si G = 0.5, 50% des individus de la population actuelle passent intacts à la génération suivante.

IV.2.9. Critère d'Arrêt

Le critère d'arrêt peut être arbitraire (par exemple, le nombre maximal de générations) ou il peut être basé sur le critère de convergence.

IV.2.9.1.Performance

Quand on exécute l'AG, on ne peut pas suivre la dynamique de la population en regardant seulement la population à chaque étape de l'exécution. Le problème c'est qu'il y a différentes statistiques à observer et établir les relations entre elles pour comprendre les informations extraites.

Les mesures statistiques sont importantes pour contrôler et réajuster les paramètres en_ligne, car l'ajustage des paramètres, souvent inter-dépendants, n'est pas toujours facile. Ces mesures statistiques étaient proposées par K. De Jong [**De Jong75**], et utilisées dans [**Greffenstette86**]. Ci-après, deux de ces mesures.

IV.2.9.2.*Performance en-ligne*

La performance en cours de la recherche est mesurée par la *performance moyenne* de l'algorithme basée sur la fonction de fitness:

$$Performance-en-ligne(T)=\frac{1}{T}*\sum_{t=1}^{t=T}F(t)$$

Avec T, le nombre total d'évaluations de fitness jusqu'à l'instant t, $F(t)$ est la $t^{ième}$ évaluation de fitness. Lors de l'exécution de l'AG, *Performance-en-ligne(T)* converge de plus en plus vers une valeur stable (état stationnaire), et comme *Performamnce-en-ligne(T)* converge, les solutions trouvées par l'algorithme deviennent de plus en plus stables.

IV.2.9.3.*Performance hors-ligne*

La mesure de performance *hors-ligne*, est similaire à celle *en-ligne* sachant que la performance *hors-ligne* accorde plus l'importance aux *meilleures* performances obtenues:

$$Performance-hors-ligne(T)=\frac{1}{T}*\sum_{t=1}^{t=T}F_{\max}(t)$$

$$avec \quad F_{\max}(t)=\sup_{t=1}^{t=T}(F(t))$$

Dans l'équation ci-dessus, ce qui change par rapport à la performance *en-ligne* est la fonction $F_{max}(t)$, qui enregistre le meilleur fitness jusqu'à présent et ignore toute autre évaluation.

Ces mesures de performance peuvent être utilisées comme critère d'arrêt. Puisque si *Performance-hors-ligne(T)* converge vers une valeur stable durant l'évolution, la chance d'obtenir une solution encore meilleure diminue fortement et l'on peut déjà arrêter la simulation, car si *meilleur* signifie seulement une légère amélioration, continuer l'algorithme serait une perte de temps.

IV.2.9.4.Convergence

Au lieu de juger la convergence par la performance *en-ligne* ou *hors-ligne*, on peut aussi l'observer directement en mesurant comment les individus changent à chaque position de gène. Un gène est dit avoir convergé lorsque 95% de la population partage la même valeur **[Beasley93a]** et la population est qualifiée d'avoir convergé si tous les gènes ont convergé.

IV.2.9.4.1.Convergence d'un code binaire

Le calcul de la convergence d'un code binaire est basé sur les valeurs des gènes à chaque position. La convergence d'un gène à la position l est exprimée par les relations suivantes:

$$C(l)=\frac{1}{N}*\max\left(\sum_{n=1}^{n=N} h(n,l,1), \sum_{n=1}^{n=N} h(n,l,0)\right)$$

$$\text{avec } h(n,l,v)=\begin{cases} 1 & \text{si } individu_n \text{ a } gene_l = v \\ 0 & \text{sinon} \end{cases}$$

$C(l) \in [0.5, 1.0]$ définit la convergence du gène à la position l, N est la taille de la population et $h(n,l,v)$ est une fonction pour identifier si $individu_n$ possède la valeur voulue v du gène à la position l. Plus $C(l)$ est grand, plus la population aura convergé pour le gène à la position l.

Donc, pour que la population entière converge, il faut qu'il y ait convergence à chaque position des gènes et pour tous les individus:

$$C=\frac{1}{L}*\sum_{l=1}^{l=L} C(l),$$

Avec L la longueur du chromosome (le nombre de gènes); et $C \in [0.5, 1.0]$, définit la convergence de la population entière à toutes les positions des gènes et pour tous les individus. Comme pour $C(l)$, plus C est grand plus la population aura convergé. Donc, si C converge vers 1.0 (100%), la population a convergé, en l'occurrence, il y a de moins en moins de diversité et de plus en plus de solutions stables.

Pour tester la convergence, il suffit de compter le nombre de gènes où l'on a: $C(l) > 0.95$.

$$C_p = \sum_{l=1}^{l=L} \begin{cases} 1 & si\ C(l) > 0.95 \\ 0 & sinon \end{cases}$$

$C_p \in [0, L]$. Quand $C_p = L$, la population a effectivement convergé et une action possible est de recommencer l'algorithme ou l'arrêter au lieu de persister au risque de perdre du temps.

IV.2.9.4.2.Convergence d'un code non-binaire

Et pour le calcul de la convergence d'un code non-binaire, il faut définir une mesure de convergence spécifique au code.

IV.2.10. Compromis entre l'Exploration et l'Exploitation

Tout algorithme d'optimisation doit utiliser ces deux stratégies pour trouver l'optimal global : l'*exploration* pour la recherche de régions nouvelles et inconnues de l'espace de recherche, et l'*exploitation* pour utiliser la connaissance acquise aux points déjà visités pour aider à trouver des points meilleurs. Ces deux exigences peuvent paraître contradictoires mais un bon algorithme de recherche doit trouver le compromis entre les deux.

Une recherche purement aléatoire est bonne pour l'exploration mais pas l'exploitation alors que la recherche dans le voisinage est une bonne méthode d'exploitation mais pas d'exploration [**Beasley93a**]. La combinaison des deux peut être efficace mais il est difficile de trouver le bon compromis, c'est-à-dire combien d'exploration faut-il effectuer avant de continuer sur l'exploitation ?

J. Holland et D. Goldberg [**Holland75, Goldberg89**] ont montré qu'un algorithme génétique pouvait associer les deux aspects en même temps de façon judicieuse, pour cela, il faut ajuster les taux d'application des opérateurs génétiques, sachant que le croisement favorise plus l'exploration tandis que la mutation favorise plus l'exploitation.

IV.2.10.1.Dérive génétique

Théoriquement, il est possible de contrôler la convergence d'un AG grâce aux propriétés d'exploration du croisement et d'exploitation de la mutation [**Goldberg89**]. Mais cette étude est basée sur des hypothèses simplificatrices comme une taille infinie de la population. En pratique, la taille de la population est finie. Alors, l'algorithme sera toujours susceptible à des erreurs stochastiques (*genetic drift*), et peut converger vers un minimal local.

A cause de ces erreurs stochastiques, même en l'absence de la sélection (fitness constant), des membres de la population vont tout de même converger vers un point quelconque de l'espace des solutions. Un gène prédominant à la génération présente peut très bien devenir moins prédominant dans les générations futures. Si par chance cette prédominance est maintenue sur plusieurs générations successives, et que la population est finie, alors tous les membres de la population peut hériter de ce gène. Une fois qu'un gène a convergé de cette façon, il est alors fixé et dans ce cas même le croisement ne peut plus introduire de nouvelles valeurs. Cette dérive peut être réduite en augmentant le taux de mutation. Cependant si la mutation est trop élevée, la recherche devient plus aléatoire et l'information sur le gradient de la fonction n'est pas exploitée.

IV.2.11. Choix des Valeurs des Paramètres

Les valeurs de paramètres doivent être choisies avec les considérations suivantes [**Grefenstette86**]:

IV.2.11.1.Taille de la population

Elle doit être judicieusement choisi en fonction de la taille du problème et du code:

- *Trop faible* : l'AG n'a pas assez d'échantillons de l'espace de recherche;
- *Elevée* : l'AG est plus informé; Une taille élevée prévient contre la convergence prématurée;
- *Trop élevée* : le nombre élevé d'évaluations de la fonction fitness par génération ralentit la convergence.

IV.2.11.2.Taux de croisement

Plus le taux de croisement p_{cros} est élevé, plus il y aura de nouvelles structures qui apparaissent dans la population.

- *Trop élevé* : les "bonnes" structures risquent d'être cassées trop vite par rapport à l'amélioration que peut apporter la sélection;
- *Trop faible* : la recherche risque de stagner à cause du faible taux d'exploration.

Le taux habituel est choisi entre 60% et 100%.

IV.2.11.3.Taux de mutation

La mutation est un opérateur secondaire pour introduire la diversité dans la population. Son taux d'application p_m est choisi entre 0.1% et 5%.

- *Trop élevé*, le taux de mutation rend la recherche trop aléatoire;
- *Trop faible,* la recherche risque de stagner à cause du faible taux d'exploration.

Si la taille de la population est faible, un taux de croisement faible doit être combiné avec un taux de mutation élevé. Les observations sont valables pour les fonctions continues sans contraintes avec codage binaire.

Mais sur cette étude de l'influence des paramètres, Grefenstette [**Grefenstette86**] suggère que la performance d'un AG dépend plus du codage et de la fonction fitness que de l'optimalité de ses paramètres.

IV.2.12.Théorème des schémas

Le plus souvent, l'étude des AG se concentre sur les lois empiriques pour les rendre plus performants. Il n'y a pas vraiment de "théorie" générale qui explique pourquoi les AG résolvent bien certains problèmes et pas d'autres. Néanmoins, plusieurs hypothèses ont été avancées pour expliquer partiellement leur performance. Celles-ci peuvent être utiles pour une bonne application de l'algorithme. Le théorème sur les schémas de J. Holland [**Holland75**] constitue la première explication rigoureuse.

IV.2.12.1.Définitions

◆Un *schéma* H est une chaîne de nombres pris dans un alphabet {0, 1, *} où * peut prendre la valeur 0 ou 1. On note parfois $H \in \{0, 1, *\}^L$, avec L la longueur de la chaîne.

◆On appelle *instances* de H toutes les chaînes $a \in \{0,1\}^\ell$ identiques à H à toutes les positions où H contient un 1 ou un 0.

◆On définit l'*ordre* de H, noté $\sigma(H)$, le nombre de positions fixes (0 ou 1) dans H, par exemple : σ (0**1*) = 2 .

◆On définit la *longueur* de H, notée $\delta(H)$, la distance entre la première position fixée et la dernière fixée, par exemple : δ (0**1*) = 3 .

◆On dit qu'un schéma H_1 est plus *général* que H_2, lorsque toutes les instances de H_2 sont incluses dans l'ensemble des instances de H_1. Et inversement, H_2 est plus *spécifique* que H_1.

Par exemple, **0* est plus général que **01. On dit aussi que les schémas ***1 et **0* sont redondants, car ils peuvent se résumer à **01.

Puisqu'un individu contient plusieurs schémas, le nombre de schémas qui sont réellement traités dans chaque génération est d'ordre N^3, où N est la taille de la population. Cette propriété est appelée "*parallélisme implicite*" qui est l'une des explications de la performance d'un AG.

IV.2.12.2.Cas non binaire

Le concept de schéma pour les codages non-binaires est basé soit sur les similarités entre chaînes (ou blocs élémentaires) soit sur l'ordre des éléments de la chaîne [**Davis91**].

Par exemple, la chaîne (5 1 6 4 2 3) contient entre autres, des schémas :

(5, 1), (5, 6), (5 4),..., (5 6 3), (1 4 2 3), (5 1 6 4 2 3), qui sont en fait des combinaisons possibles des éléments basées sur leur ordre d'apparition dans la chaîne.

Ainsi, pour les codes non binaires, il n'y a pas vraiment de règles sur les schémas. Généralement, on définit un alphabet et les valeurs que peut prendre le signe "*". Il existe différentes façons d'interpréter la signification du signe "*" et les positions fixes permettent de maintenir les bonnes structures acquises.

IV.2.12.3.Schémas et hyperplans

L'intérêt des schémas réside donc dans l'exploitation massive de variables. Ils permettent ainsi de partager le domaine de recherche en plusieurs régions. Par exemple, si x est codé en binaire naturel, les schémas H_1=0**** et H_2=1**** partagent le domaine de recherche en deux. Les schémas H_{11}=10*** et H_{12}=11*** subdivisent encore le domaine recouvert par H_1 en deux, et ainsi de suite, on essaie de converger de proche en proche vers le minimal global (**Figure** IV. 6). Pour plus de détail sur les schémas, se référer à [**Holland75**].

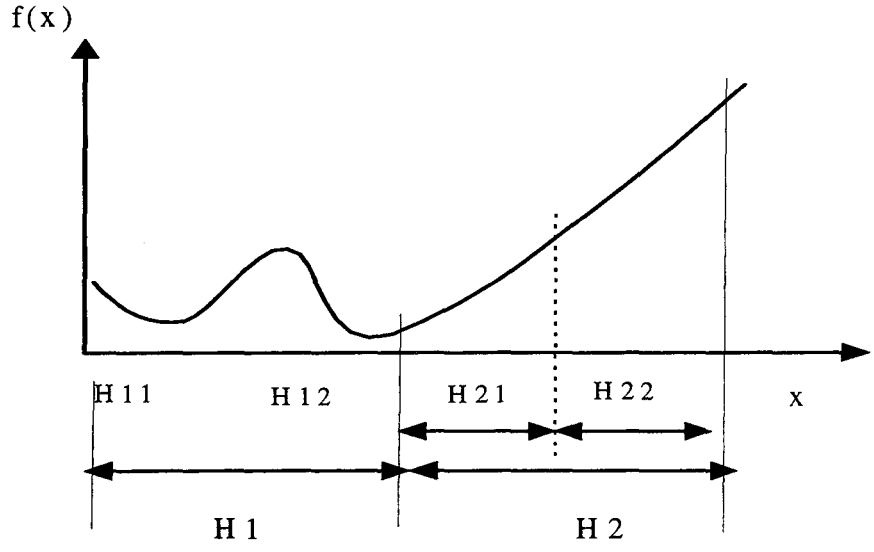


Figure IV. 6: Propriété d'exploration des schémas. Plus un schéma est général (d'ordre faible), plus la région de l'espace qu'il recouvre est large.

Cette structure comprend plusieurs chaînes, on l'appelle aussi "brique élémentaire" (*building block*). Ainsi, au lieu de traiter des individus un par un, c'est l'ensemble d'individus présentant ces similarités qui sont traités. Les schémas constituent le début d'une solution optimale, c'est

de cette brique élémentaire que naîtront théoriquement les meilleurs individus de la génération future.

IV.2.12.4.Hypothèse de "building blocks"

Selon D. Goldberg **[Goldberg89]**, la puissance d'un AG réside dans sa capacité de trouver des briques élémentaires (*building blocks*). Les briques élémentaires sont des schémas de longueur minimale consistant en éléments qui fonctionnent bien ensemble et qui permettent d'améliorer la performance de l'individu auquel ils sont incorporés. Un codage approprié à un problème est celui qui favorise la formation des briques élémentaires. Ce codage doit assurer à la fois que ces gènes relatifs ont des loci[5] proches entre eux dans le chromosome et que leur interaction est très faible **[Goldberg89]**.

Les interactions (appelées aussi *épistasies*) entre les gènes signifient que la contribution d'un gène au fitness dépend de la valeur des autres gènes dans le chromosome. En fait il y a toujours une certaine interaction entre les gènes dans les fonctions fitness multimodales. Celle-ci conviennent bien aux AG, puisque les fonctions unimodales peuvent être résolues avec des méthodes classiques plus simples. Si ces règles sont respectées, alors un AG va être aussi efficace que prévu par le théorème des schéma.

Malheureusement, la condition d'interaction est une pré-condition pour la condition de proximité des gènes. Si la contribution au fitness global de chaque gène est indépendante des autres gènes, il sera alors possible de résoudre le problème par le *hill-climbing* en faisant évoluer les gènes un par un. Si on peut assurer qu'un gène inter-agit très peu avec d'autres gènes *et* que ceux-ci peuvent être placés ensemble dans le chromosome, alors les conditions d'interaction et de proximité peuvent être satisfaites. Mais s'il y a trop d'interactions, aucune condition ne peut être satisfaite **[Davidor91]**.

En bref, on devrait concevoir un codage qui soit conforme à ces recommandations puisque cela assure la performance de l'algorithme mais soulève deux questions intéressantes :

- Est-il possible de trouver un codage qui satisfasse les conditions des hypothèses des building blocks ?
- Si cela n'est pas possible, peut-on modifier l'AG pour améliorer sa performance ?

[5] locus position d'un gène dans un chromosome

IV.3. Aspects Pratiques des AG

Malheureusement, les fondements théoriques de J. Holland ne peuvent pas se généraliser à tous les problèmes. Les méthodes pratiques et l'évolution grandissante de la biologie viennent enrichir la bibliothèque des AG au fur et à mesure des applications dans divers domaines. De nouveaux modes de sélection et opérateurs génétiques sont également proposés (Voir *Annexe 4* et **[Goodman94]**).

La programmation génétique fait partie de cette nécessité de s'adapter à un problème particulier. Elle a été proposée par J. Koza **[Koza92]** pour élaborer des programmes de type LISP avec les algorithmes génétiques utilisant une population de codes de programmes. Le paradigme reste le même. Cette méthode est particulièrement efficace pour trouver des fonctions de calcul complexes (Voir *Annexe 3*).

Lorsque le codage est trop difficile à concevoir ou l'algorithme génétique seul est trop lent, la méthode AG-hybride est souvent utilisée. L'hybridation avec les méthodes locales telles que le "branch-and-bound" **[Cotta95]** ou le recuit simulé permet de faire abstraction de la représentation des individus. Avec le recuit simulé, l'algorithme génétique fait évoluer un individu comme avec la méthode du recuit simulé normal. Lorsque le critère de Metropolis n'est pas satisfait, un autre individu est choisi pour évoluer. Tous les individus de la population à une température donnée auront subi une évolution. A la température suivante, le processus recommence jusqu'à ce qu'on obtienne un état stationnaire final **[Davis87, Lin93, Koakutsu93, Yip94]**.

Une autre alternative pour résoudre le problème de multimodalité est d'utiliser le concept de sous-populations au lieu d'une seule population. Chaque sous-population, appelée *niche écologique* **[Goldberg89, Beasley93b, Schmidt96]**, peut évoluer séparément pour préserver la diversité et peut-être aussi pour la meilleure exploitation de la diversité.

IV.3.1. Concept de Sous-Population

Dans l'écosystème naturel, différents espèces évoluent en parallèles et forment une "espèce". Chaque espèce occupe ce qu'on appelle *niche écologique* **[Beasley93b]**.

Dans un AG, des niches sont représentées par des maxima (ou minima) de la fonction de fitness. Avec une fonction multimodale, un AG classique ne permet pas de localiser tous les maxima (resp. minima), puisque la population entière va converger vers un seul maximum (ou minimum). Dans le meilleur des cas, on pourrait espérer que l'algorithme converge vers le plus haut pic, mais s'il existe plusieurs pics de fitness égal, l'AG va quand même s'arrêter sur un seul.

Le concept de sous population permet de résoudre ce problème. Deux approches ont été utilisées:

1.*Crowding scheme* [**De Jong75**]: les enfants remplacent les individus existants s'ils ont la même ressemblance chromosomique. Ceci se traduit par une "colonisation" d'une région prometteuse (**Figure** IV. 7).

2. *Sharing scheme* [**Goldberg89, Lutton93**]: les individus ayant un fitness élevé se partagent leur fitness, comme les individus se partagent les ressources dans la nature. Ceci évite une convergence prématurée. Cette stratégie a été utilisée avec succès dans [**Lutton93**] pour le traitement d'image. Dans le cas du palcement, cela peut se traduire par une combinaison entre les chaînes qui ont des formes complémentaires à se partager.

La notion de l'éloignement géographique a été utilisée aussi pour ne permettre aux individus de se reproduire qu'avec ses voisins. Ceci a été implanté dans les AG parallèles.

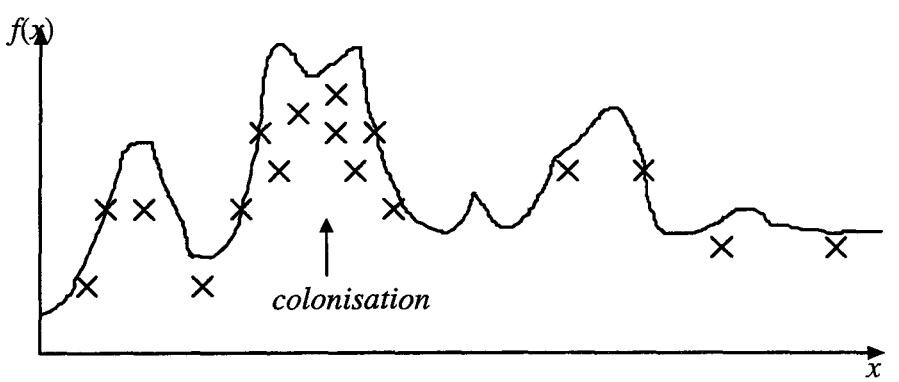


Figure IV. 7 : Exemple de "crowding scheme" ou colonisation. Lorsqu'une région apparaît prometteuse, on maintient plus d'individus de cette région sans pour autant délaisser les individus représentant d'autres régions.

IV.3.2. AG Parallèles

Il existe plusieurs types d'implantation d'AG parallèles: distribué, cellulaires, massivement parallèles [**Talbi94, Kröger90,92**].

Dans les premiers temps, les AG consistait en un processeur maître qui manipule les valeurs de fitness et la sélection; les processeurs esclaves s'occupent des opérateurs génétiques et l'évaluation de fitness (voir **Figure** IV. 8).

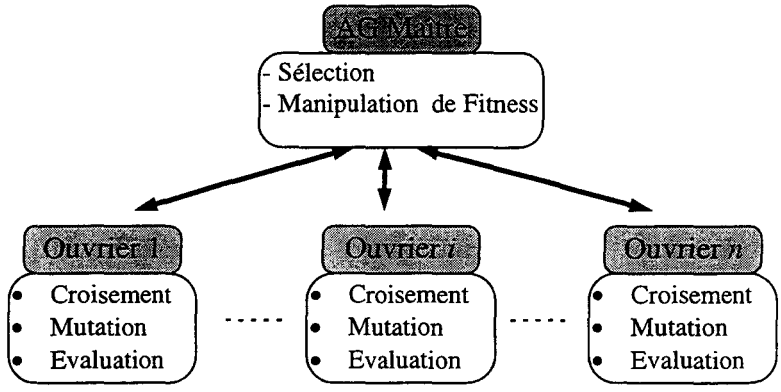


Figure IV. 8: Premiers AG parallèles consistant en un processeur maître et des processeurs esclaves qui exécutent exactement les mêmes tâches en ce qui concerne les opérateurs génétiques et les évaluations.

Avec les développement des réseaux locaux, les protocoles de communication sont devenus beaucoup plus rapides et le matériel moins cher. Maintenant, on distingue les architectures "grain fin" (*fine grain PGA*) ou "gros grain" (*Coarse grain PGA*) [6].

AG Parallèle "grain fin":

Cette implantation modélise les individus distribués dans la nature te la notion de voisinage est importante puisque les individus ne peuvent se reproduire qu'avec des individus situés à une distance qui lui est accessible. Cela implique une disposition d'individus selon une *topologie*: en anneau (*ring*), en tore (*torus*). Dans la topologie en anneau, chaque individu possède deux voisins, un à gauche et un à droite. Dans la deuxième topologie, un individu peut avoir huit voisins.

AG parallèle "gros grain":

Ce modèle est aussi appelé *modèle en îlot, modèle de migration* ou encore modèle *distribué*. Dans ce modèle les individus d'une même population vivent dans différents îlots et évoluent en parallèle, avec un AG différent du voisin (ex. différents génotype, différents opérateurs, sélection etc.). De temps en temps, il y a échange : certains individus *émigrent* vers un autre îlot pour se reproduire (**Figure** IV. 9).

[6] Online GA Tutorial, http://www.wi.leidenuniv.nl/~gusz/Flying_Circus/1.Reading/Tutorial/04/index.html#C41, "needs for parallelisation"

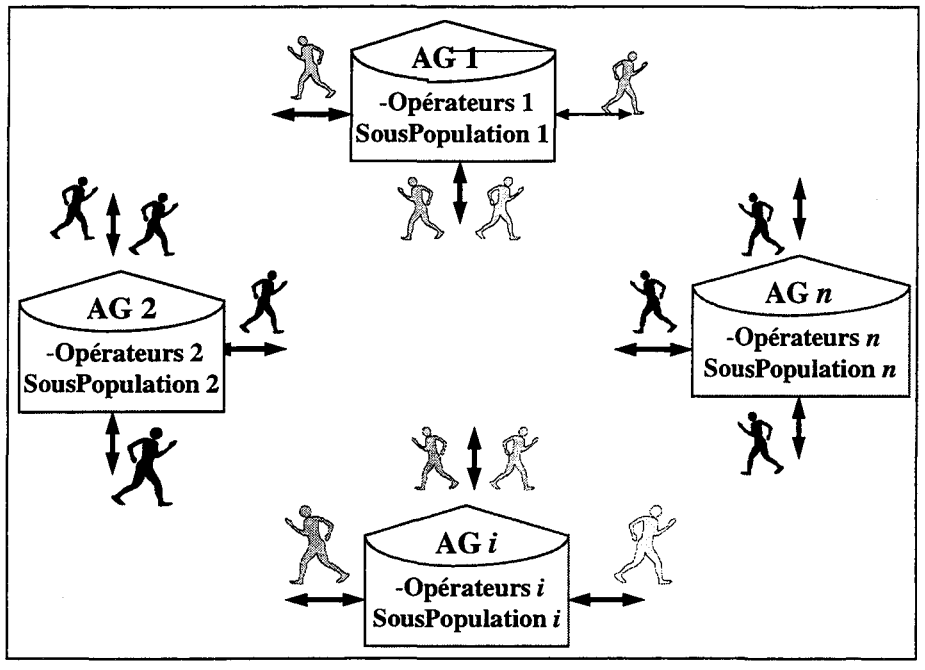


Figure IV. 9: Exemple d'un AG distribué. Dans ce modèle, les individus sont localisés sur des îlots, et de temps en temps, un individu d'un îlot s'échappe sur l'îlot voisin pour se reproduire. Ceci introduit le paramètre, fréquence d'échange, probabilité d'escapade etc. Le meilleur de la population est le meilleurs de tous les meilleurs des sous population.

Il est intéressant de noter que dans ce modèle distribué, les algorithmes des îlots peuvent être d'un autre type que génétique. Ceci permet des compétitions ou coopérations avec d'autres algorithmes locaux (Recuit simulé, Recherche tabou ou réseaux de neurones...).

IV.4. Application au Problème de Placement

On a vu que la performance d'un AG dépend beaucoup du codage et des opérateurs d'évolution utilisés. Dans les premières applications des AG, il était nécessaire d'utiliser des chaînes binaires. C'est pourquoi les auteurs se concentrent sur certains aspects du problème de placement dans lesquels il était possible d'appliquer le codage binaire. Mais plus tard, on s'aperçoit qu'il est difficile d'utiliser cette représentation, plusieurs formes de codages non binaires sont alors apparues. Dans les sections ci-après, quelques codages utilisés pour les problèmes relatifs au placement vont être décrits. Ensuite, nous décrivons nos deux approches évolutionnistes et quelques résultats sont montrés à la fin de ce chapitre.

IV.4.1. Problème de Codage

IV.4.1.1.Codage en chaîne

Dans le domaine relatif au placement, le sujet le plus étudié concerne le problème du "*bin packing*". Pour ce problème, le codage peut se mettre facilement sous forme de chaîne. Par exemple, si le problème consiste à remplir m containers (*bins*), indexé par i, avec n objets, indexé par j, on peut représenter un individu par une chaîne binaire de longueur n dans laquelle chaque élément x_{ij} de la chaîne vaut 1 si l'objet i appartient au container j sinon 0. Pour le coder en chaîne non binaire, chaque objet i prend le numéro du container j auquel il appartient dans la chaîne [**Reeves95, Smith85, Khuri95**].

L'utilisation de la chaîne binaire dans placement de polygones a été proposée dans [**Petridis94**]. Dans cet article, un placement est considéré comme un ensemble de positions des différents polygones placés. Chaque position (X_{ij}, Y_{ij}) d'un polygone dans le repère global est codée en binaire sur n bits. Un polygone P_i, $i = 1..N$, est repéré par son premier sommet, noté (X_{i1}, Y_{i1}), avec X_{i1} = MIN $\{X_{ij}\}$ et Y_{i1} = MIN $\{Y_{ij}\}$, ses autres sommets (X_{ij}, Y_{ij}), avec $j > 1$, sont définis relativement à (X_{i1}, Y_{i1}), lus dans le sens trigonométrique. La position du polygone P_i est alors repérée seulement par les coordonnées de son premier sommet (X_{i1}, Y_{i1}). Ainsi, pour N polygones et une longueur de chaîne $n = 10$, on a un génotype de longueur $N \times 2 \times 10$ bits. Ce qui limite le nombre de polygones à traiter avec cette méthode.

Une représentation basé sur l'ordre des éléments a été utilisée dans [**Fujita93, Pargas93**]. Cette représentation utilise une liste de numéros de chaque forme en tant que chaîne comme celle appliquée au problème du voyageur de commerce [**Grefenstette87**]. Par exemple, la

chaîne de caractères "*hecgabfd*" représente une solution à un problème dans lequel 8 polygones différents sont à placer dans l'ordre indiquée dans la chaîne. L'avantage de ce codage est la possibilité d'utiliser le principe du *building block* ou de schéma avec la relation de voisinage. Par exemple, la chaîne "*j a b d f*" peut être divisée en quatre couples "*j a*" "*a b*" "*b d*" et "*d f*". Dans ce cas les couples ne signifient pas seulement une liste de deux formes mais ils incluent aussi la relation spatiale comme le sens de contact par exemple. Ce codage permet de manipuler le côté combinatoire du problème, il ne donne pas directement les positions et sens des pièces, qui sont représentés par un grand nombre de variables spatiales continues. C'est pourquoi un algorithme d'optimisation locale est souvent utilisée avec.

Dans **[Ismail92]**, au contraire, l'algorithme génétique sert à optimiser la fonction objectif locale de l'imbrication. L'imbrication elle-même utilise un algorithme heuristique sur les formes codées en quatre listes issues d'une grille de *discrétisation*. La grille de *discrétisation* est le rectangle minimal de chaque forme. Une case de la grille contient un 1 si elle intercepte la surface sinon 0. Le nombre des 0 "vus" de chaque côté de la grille est récapitulé ligne par ligne dans une liste de même longueur que le côté correspondant de la grille. On obtient ainsi quatre listes pour décrire chaque forme. Ce type de codage se retrouve dans notre codage de formes par un ensemble de 4 peignes **[Bounsaythip95]**.

IV.4.1.2.Codage hiérarchique

La représentation hiérarchique de chromosomes est utilisée dans la résolution du problème de la découpe en guillotine **[Koakutsu93, Kado95a,b]**. Cette technique est basée sur la notation polonaise inverse utilisée pour décrire la position d'un rectangle par rapport à un autre. L'ensemble des opérateurs est {* ; + }, et les modules sont pris comme opérandes. Par exemple, la notation "*AB+*" signifie "*A en dessous de B*" et "*AB**" signifie "*A à gauche de B*". Un exemple est donné à la ***Figure IV.*** 10.

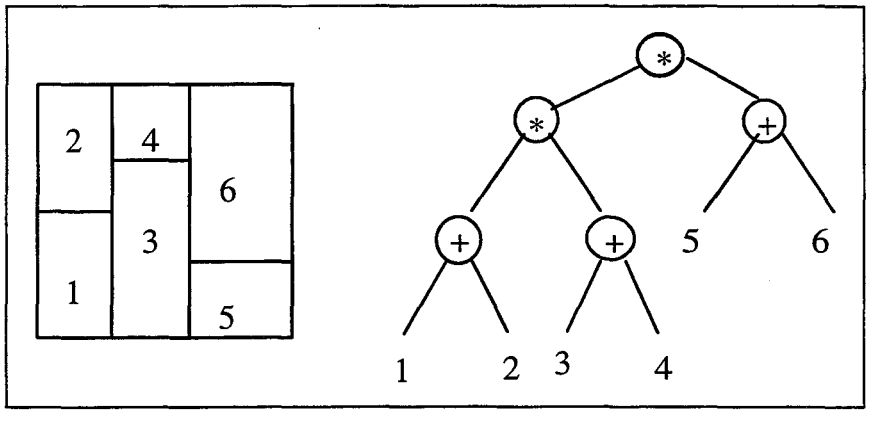


Figure IV. 10 : Placement en guillotine avec la notation en polonaise inverse, le code correspondant est : *12+34+*56+**.

Pour le même type de problème, le codage en arbre binaire orienté a été utilisé dans **[Kroger90]**. Les noeuds de l'arbre représentent les rectangles placés dont les adjacences sont indiquées par la nature des arcs. L'arc entre les noeuds r_i et r_j est noté *arc-r* si r_j est adjacent à droite (*right*) de r_i et *arc-t* si r_j est placé au dessus (*top*) de r_i. Chaque noeud a donc deux arcs sortants de type *r* ou de type *t* (sauf pour la racine). Ainsi, chaque individu est représenté par le quadruple $G = (V, E, o, p)$. Avec V l'ensemble des rectangles, E l'ensemble des arcs (*edges*), o opérateur d'orientation, et p le degré de priorité pour éviter des conflits lorsque deux rectangles satisfont les mêmes critères d'adjacence par rapport à un même rectangle. La transformation du génotype en phénotype consiste alors à placer successivement les rectangles selon les données contenues dans les noeuds et les arcs.

Ces techniques de placement de rectangles peuvent être étendues au placement de formes irrégulières si ces dernières peuvent se ramener au cas des rectangles. Ceci peut se faire dans la phase de pré-traitement, pendant lequel des modules rectangulaires comprenant un ou plusieurs formes imbriquées sont créés. Cette approche est envisagée pour le placement de composants en VLSI par Goodman et *al.* **[Goodman94]**. Dans cette étude, la représentation hiérarchique des chromosomes est basée sur des groupes en tant que gènes. Chaque groupe d'éléments ou sous-groupe est un placement partiel à une certaine position. Un chromosome est donc constitué d'un ensemble de groupes associés à un ensemble de positions de chaque groupe. Chaque groupe peut être lui même constitué de sous groupe de même structure (***Figure** IV.* 11).

Exemple d'un code à 3 niveaux:

chr1 :

< { {[2]}(x_{G1}, y_{G1}), {[1(x_1,y_1)], [4(x_4,y_4)],

[7(x_7,y_7)]}(x_{G2},y_{G2})} } (x_{G5},y_{G5})

{ {[3(x_3,y_3)], [5(x_5,y_5)]}(x_{G3},y_{G3}), {[6(x_6,y_6)],

[8(x_8,y_8)],[9(x_9,y_9)]}(x_{G4},y_{G4}) } (x_{G6},y_{G6})

{[9]} (x_{G7},y_{G7}) > (0,0)

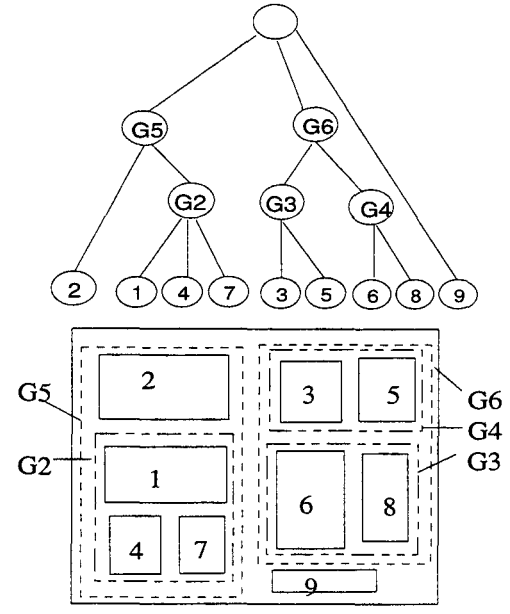


Figure IV. 11 : Représentation du chromosome hiérarchique dont la structure linéaire est de la forme : **Chromosome : {Groupes}(position du groupe)** avec {Groupes}= {{groupe1}(position1}, {groupe2}(position2), ...}

A l'état initial, chaque groupe contient un seul élément. Durant le processus d'évolution, certains groupes fusionnent pour former un groupe plus grand. Mais, à l'heure actuelle, les résultats relatifs à l'application de ce code ne sont pas encore publiés.

Une autre approche hiérarchique a été proposée dans [**Dighe96**]. Cette approche consiste en deux niveaux d'optimisation. Etant donné un ensemble de polygones, le AG de niveau 1 essaie de trouver les meilleurs arrangements symboliques des polygones. Le AG de niveau 2 décode ces arrangements pour trouver la disposition réelle des polygones. Le niveau 2 traite donc de l'imbrication de polygones deux à deux en utilisant le chromosome binaire consistant en une concaténation des variables de position (x, y) et d'orientation θ de tous les polygones placés. Un exemple de tel codage est donné dans le tableau suivant (***Figure IV.*** 12):

x	y	θ	x	y	θ	x	y	θ	...
10001	10001	10111	00101	01011	00111	10000	01000	11111	...

Figure IV. 12 : Codage du placement de polygones par concaténation des chaînes de paramètres de positionnement et de rotation de chaque polygone.

Un arbre binaire est utilisé pour le niveau 1. Chaque noeud de l'arbre est un AG de niveau 2 qui recherche le rectangle circonscrit minimal de deux polygones à assembler. Une fois que deux polygones ou deux groupes de polygones sont assemblés, le groupe résultant maintient les positions relatives et orientations pour toutes les opérations subséquentes. Une telle représentation en arbre ne permet d'assembler qu'un ensemble de 2^n polygones. Cette limitation a été résolue en insérant des polygones vides (surface = 0) dans le chromosome.

Les exemples de placement des polygones convexes ont montré de bons résultats. Mais, dans certains cas, la représentation en arbre ne peut pas atteindre le minimal global à cause de l'assemblage par étape qui ne permet pas toujours de réarranger des petits défauts tels que les petits chevauchements par exemple. Dans notre approche, les chevauchements sont éliminés dès la phase de l'imbrication par l'utilisation du code en peigne.

IV.4.2. Première Approche : Codage en Peignes Réduits

Le code en peignes réduits est proposé par G. Roussel [**Roussel94**], issu d'une étude sur l'imbrication des formes. La recherche d'une imbrication optimale entre les formes F_j et F_k consiste à les approcher l'une vers l'autre jusqu'à ce qu'il y ait un point de contact qui représente une profondeur d'imbrication maximale sur un côté donné. La profondeur de l'imbrication est mesurée sur deux dimensions, une profondeur par rapport à l'axe des X et l'autre par rapport à l'axe des Y (***Figure IV.*** 13). Le but est d'obtenir le rectangle minimal circonscrit aux deux formes.

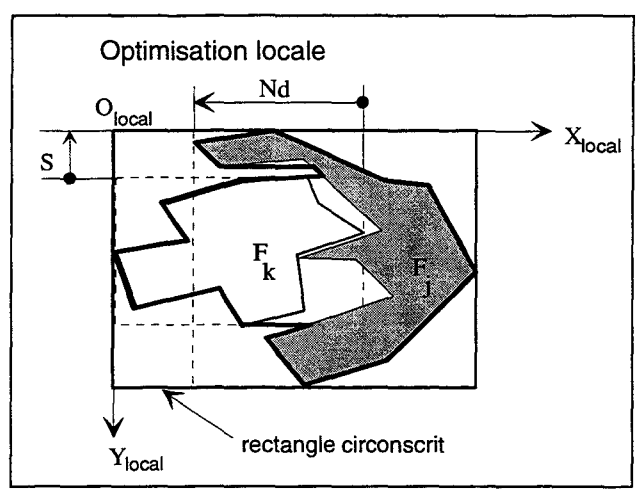


Figure IV. 13 : Imbrication de deux formes. N_d et S indiquent la profondeur maximale de l'imbrication sur l'axe des X et sur l'axe des Y respectivement.

IV.4.2.1.Utilisation de peignes

Un peigne représente des échantillons du contour visible de chaque côté du rectangle circonscrit. Ces échantillons forment un ensemble de *dents* qui sont des segments exprimant la distance entre un point du contour et le bord du rectangle. Les dents sont générées le long du contour par une distribution de Dirac de période T dans la référence liée à chaque côté. Le peigne réduit est obtenu en éliminant les dents redondantes (de même longueur) (***Figure*** *IV.* 14).

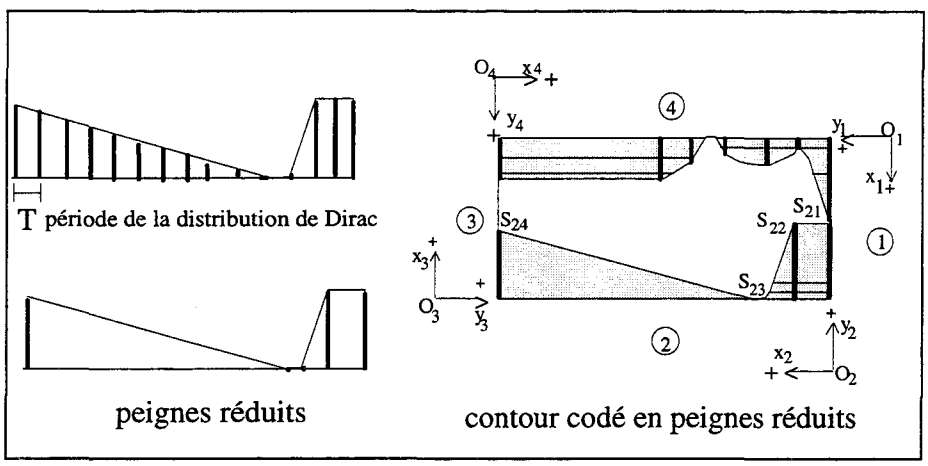


Figure IV. 14 : Codage en peigne. Les dents sont générées le long du contour par une distribution de Dirac de période T dans la référence liée à chaque côté. Un peigne réduit est obtenu en éliminant les dents redondantes d'un peigne complet.

La longueur d'une dent est une information importante pour déterminer la meilleure imbrication locale de deux formes grâce à la déficience rectangulaire définie par la longueur des dents des peignes.

Notre première approche pour appliquer le paradigme évolutionniste considère ce code en peigne pour représenter un chromosome. Chaque chromosome est donc un ensemble de quatre peignes qui sont eux-mêmes constitués d'ensembles de segments. L'utilisation du code pour l'algorithme génétique est décrite dans la section suivante.

IV.4.2.2.Codage en peigne pour l'algorithme génétique

Dans cette partie, nous décrivons comment appliquer le code en peigne pour coder un individu de l'algorithme génétique. Un individu représente une forme et il est codé par un ensemble de quatre peignes (**Figure** IV. 15).

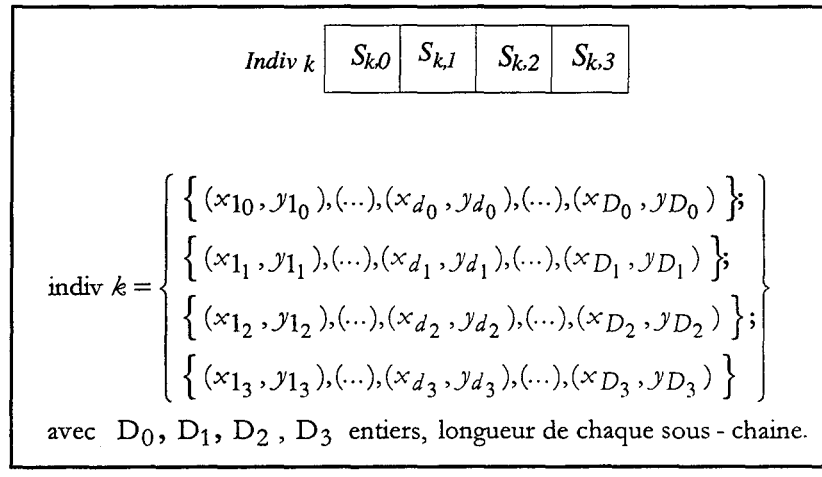


Figure IV. 15 : Codage d'un individu en peigne. Chaque peigne est constitué d'un ensemble de coordonnées des segments représentant la longueur de ses dents.

Ce code étant fortement épistatique, les opérateurs génétiques ne peuvent pas être aléatoires au risque de changer complètement la forme initiale. Les croisements heuristiques permettent d'éviter l'utilisation d'un algorithme de réparation de chromosome pour satisfaire les contraintes comme dans beaucoup de problèmes d'ordonnancement [**Caux95**].

Dans la section suivante nous décrivons les trois types de croisements utilisés. L'opérateur de mutation n'est pas utilisée.

IV.4.2.3.Opérateurs génétiques

Les croisements utilisés sont de type arithmétique, c'est-à-dire que les enfants sont les résultats des opérations arithmétiques sur des éléments pris sur les parents. Avec ce type d'opérateur, les sites de croisements ne peuvent pas être choisis aléatoirement et l'opération de croisement n'est pas basée sur l'échange de gènes entre les chromosomes parents. Dans nos croisements, une partie des gènes (peignes) d'un chromosome est modifiée par combinaison avec une partie des gènes de l'autre chromosome. La recombinaison est un algorithme de fusionnement de dents qui sont mises en contact.

IV.4.2.4.Exemple de croisement

L'exemple ci-dessous montre le croisement entre la forme F_1 (**Figure** IV. 16) et la forme F_2 (**Figure** IV. 17).

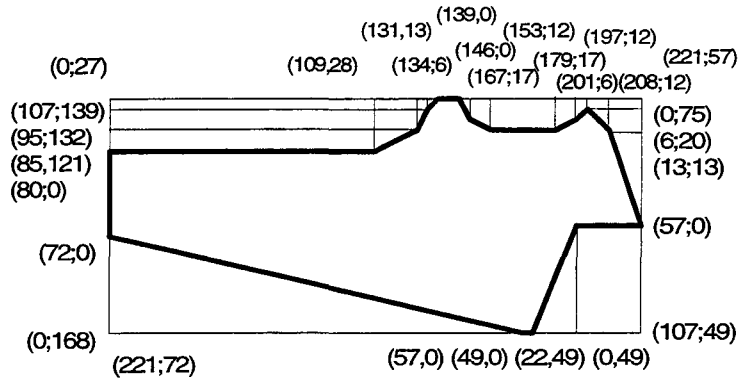


Figure IV. 16 : Coordonnées de la forme F_1.

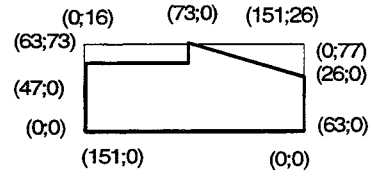


Figure IV. 17 : coordonnées de la forme F_2.

Les coordonnées de F_1 et F_2 sont respectivement les suivantes:

$$F_1 = \begin{cases} S_{1,0}:\{(0;75)\ (6;20)\ (13;13)\ (57;0)\ (107;49)\}\ ; \\ S_{1,1}:\{(0;49)\ (22;49)\ (49;0)\ (57;0)\ (221;72)\}; \\ S_{1,2}:\{(0;168)\ (72;0)\ (80;0)\ (85;121)\ (95;132)\ (107;139)\}\ ; \\ S_{1,3}:\{(0;27)\ (109;28)\ (131;13)\ (134;6)\ (139;0)\ (146;0)\ (153;12)\ldots \\ \ldots(167;17)\ (179;17)\ (197;12)\ (201;6)\ (208;12)\ (221;57)\} \end{cases}$$

$$F_2 = \begin{cases} S_{2,0}:\{(0;77)\ (26;0)\ (63;0)\}; \\ S_{2,1}:\{(0;0)\ (151;0)\}; \\ S_{2,2}:\{(0;0)\ (47;0)\ (63;73)\}\ ; \\ S_{2,3}:\{(0;16)\ (73;0)\ (151;26)\} \end{cases}$$

Après croisement, le premier enfant aura la forme suivante (**Figure** IV. 18):

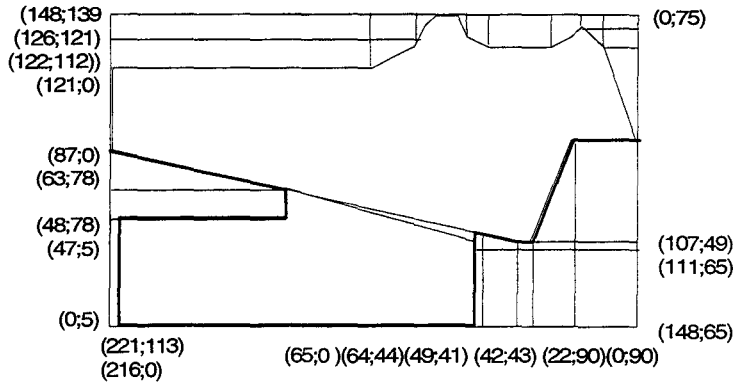


Figure IV. 18 : Résultat de l'imbrication des formes F_1 et F_2 précédentes.

Les nouvelles coordonnées de la forme composée sont calculées par rapport au nouveau rectangle circonscrit qui sert de référence locale. Les positions relatives entre les deux formes peuvent être aussi générées durant cette opération pour les placer dans la référence globale qui est la référence matière. Les coordonnées de la nouvelle forme après imbrication sont :

$$F_{1+2} = \begin{Bmatrix} S_{1+2,0}:\{(0;75)\ (6;20)\ (13;13)\ (57;0)\ (107;49)\ (111;65)\ (148;65)\ \}; \\ (S_{1,1}) \cup (S_{2,1}):\{\ (0;90)\ (22;90)\ (42;43)\ (49;41)\ (64;44)\ (65;0)\ (216;0)\ (221;113)\ \}; \\ S_{1+2,2}:\{(0;5)\ (47;5)\ (48;78)\ (63;78)\ (87;0)\ (121;0)\ (122;112)\ (126;121)\ (148;139)\}; \\ S_{1,3}:\{(0;27)\ (109;28)\ (131;13)\ (134;6)\ (139;0)\ (146;0)\ (153;12)\ (167;17)\ (179;17)\ldots \\ \ldots(197;12)\ (201;6)\ (208;12)\ (221;57)\} \end{Bmatrix}$$

L'ensemble $S_{1,3}$ de la forme F_1 est inchangé tandis que les autres ensembles ont été modifiés.

IV.4.2.5.Remarque sur le codage en peignes

On peut noter que l'opération d'imbrication est assez complexe. Surtout si on a une forme très irrégulière, les peignes peuvent contenir un ensemble de dents très important, c-à-d, si la période de la distribution de Dirac est petite. Si la recherche de l'imbrication optimale doit s'effectuer sur chaque côté, le temps de calcul peut être important. De plus, l'imbrication locale optimale n'induit pas forcement une imbrication globale optimale. Ainsi, il est nécessaire d'utiliser un algorithme de recherche de niveau plus élevé.

Ceci est effectué avec la méthode génétique utilisant une représentation de type arborescent. Le code en arbre doit fournir à l'algorithme local les paramètres tels que rotation et côté d'imbrication.

IV.4.3. Deuxième Approche : Codage Arborescent

Notre approche évolutionniste décrite ci-après utilise aussi une structure hiérarchique. L'algorithme utilise une population de chromosome-arbres et fonctionne avec l'algorithme d'imbrication de formes décrite précédemment.

Un chromosome-arbre représente un placement possible de toutes les formes. Chaque arbre représente une bande de placement. Un arbre est lui même composé de sous-arbres binaires qui sont composés de formes. Une forme est reliée à une autre par un opérateur qui indique le type de voisinage dans le placement. La structure est similaire à celle utilisée en programmation génétique [**Koza92**, ***Annexe 3***]. Comme la PG est un paradigme similaire à AG, certaines techniques utilisées peuvent être appliquées à notre code en arbre. Les formes sont considérées comme des symboles (génotype) constituant les noeuds de l'arbre et le code en peignes réduits est utilisé pour réaliser le placement (phénotype) et évaluer le fitness.

IV.4.3.1.Description

Dans notre représentation en arbre, les formes sont reliées ensemble par un opérateur de placement (connexité γ, rotation θ) et peuvent être vues comme des opérandes. Les opérateurs de placement contiennent les paramètres nécessaires à l'imbrication. Un chromosome représente un placement possible de toutes les formes et est lui-même constitué d'un ensemble d'arbres. Chaque arbre représente une bande de placement et est lui-même composé de sous-arbes binaires (***Figure IV.*** 19).

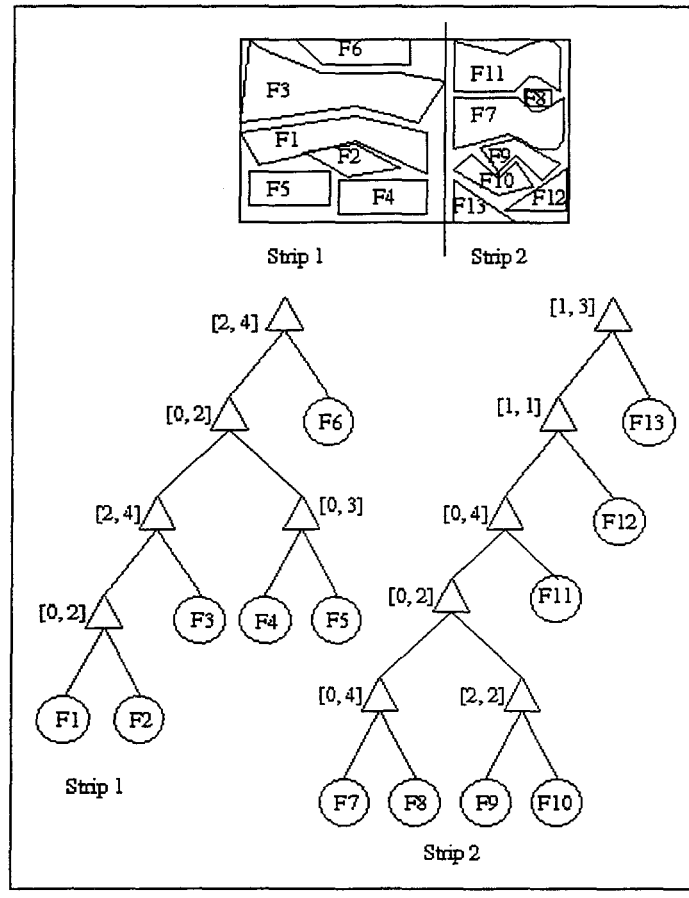


Figure IV. 19 : Exemple de placement et sa représentation en arbre.

Le paramètre θ indique l'orientation de la seconde pièce à placer. Le centre de rotation d'une forme est le point d'intersection des diagonales de son rectangle circonscrit. La variable γ est le numéro du côté adjacent à la pièce à placer. Cette numérotation est faite dans le repère global. De cette manière, la recherche d'imbrication peut être effectuée plus rapidement si les paramètres fournis par les arbres sont optimaux.

IV.4.3.2.Structure linéaire associée à un arbre

La représentation en arbre est associée à un code linéaire (le génotype). Ce code transcrit la structure d'un arbre en listes de formes et d'opérateurs. Comme un arbre binaire est codé de gauche à droite avec toujours deux feuilles émanant de chaque opérateur (arbre binaire), la structure linéaire peut s'écrire de la façon suivante:

$$T = (\ldots ((F_1, F_2) \Delta [\theta_2, \gamma_2]), \ldots; \ldots ((F_j, F_k) \Delta [\theta_k, \gamma_k]) \ldots) \Delta [\theta_{end}, \gamma_{end}])$$

La rotation θ_k est choisie dans {0; 1; 2; 3} et γ_k dans {1; 2; 3; 4}. Pour une forme composée, la rotation est choisie dans {0; 2}. L'opérateur [θ_{End}, γ_{end}] représente le dernier opérateur au sommet de l'arbre. Un exemple concret de cette représentation est donné à la ***Figure*** *IV.* 20.

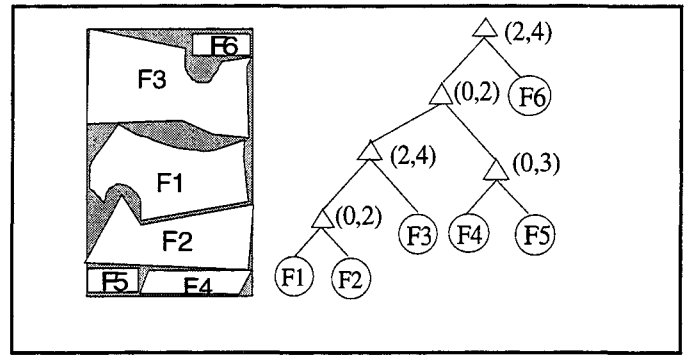


Figure IV. 20 : Le code linéaire associée à cet arbre est: T= *(((((F_1, F_2) Δ [0, 2])*, F_3) Δ *[2, 4])*, *((F_4, F_5) Δ [0, 3])* Δ *[0,2])*, F_6) Δ *[2,4]*.

IV.4.3.3.Décodage en phénotype

Le phénotype du code précédent correspond au placement. Le décodage du génotype n'indique pas explicitement la manière exacte par laquelle les formes doivent être imbriquées. Il nécessite donc l'algorithme du deuxième niveau. Car, dans le cas où il existe plusieurs possibilités pour une même configuration indiquée dans le génotype (***Figure*** *IV.* 21), l'algorithme du deuxième niveau doit fournir l'imbrication optimale.

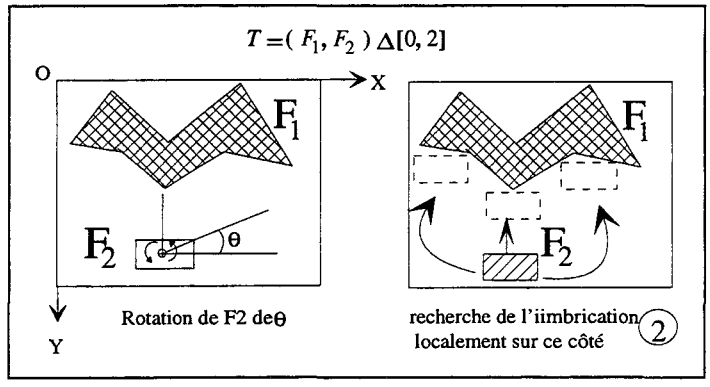


Figure IV. 21: Réalisation d'un placement à partir du code arbre. *(F_1, F_2)* Δ *[θ_2, γ_2]* signifie que F_1 est fixé et F_2 est tourné de $\theta_2 \times \pi$ /2 puis placée à côté de F_1 sur le côté γ_2 of F_1

IV.4.3.4.Propriétés

Par cette méthode, au lieu de faire le placement en ajoutant une seule forme à la fois à chaque étape, un bloc de formes composées peut s'associer à un autre aussi. Non seulement ceci permet de faire progresser le placement plus vite, mais aussi de préserver ou d'échanger les meilleurs blocs de formes composées. Un individu-arbre est donc une solution partielle contenant des éléments qui peuvent faire évoluer la recherche vers une solution optimale. Ceci est crucial pour le paradigme AG puisqu'il s'agit d'explorer l'espace de recherche afin d'identifier les régions prometteuses, en même temps que d'exploiter l'information ainsi rassemblée par un effort croissant sur ces régions.

En outre, les interactions entre gènes (*épistasies*) sont réduites : les positions relatives d'un gène par rapport à un autre n'affecte pas la valeur du fitness du chromosome bien qu'il y ait encore des interactions entre allèles, mais celles-ci sont locales au gène. Dans le courant actuel de la recherche, il est préconisé de prendre un degré d'épistasie moyen [**Beasley93c**]. Car si une représentation contient très peu ou pas du tout d'épistasie, cela signifie que la valeur d'un élément n'affecte pas celle des autres restants et l'optimisation revient à faire une minimisation point par point. Dans ce cas, la recherche par voisinage est plus efficace. D'un autre côté, si une représentation est hautement épistatique, trop d'éléments sont alors interdépendants et l'amélioration du fitness devient difficile à déceler. Par conséquence, la complexité devient telle que l'espace de mesure de performance ne contient aucune régularité significative. Dans ce cas, la recherche aléatoire se révélera plus efficace [**Goldberg89**, **Davidor91**, **Beasley95a,b**].

IV.4.3.5.Contrôle de redondance

Dans un chromosome, chaque forme apparaît a_k fois, on doit donc faire une vérification de redondance de formes à chaque fois qu'il y a échange de sous-arbres . Pour chaque individu i , le nombre d'exemplaires a_i d'une forme F_k doit être consistant dans l'ensemble des N arbres de la population (**Figure** IV. 22).

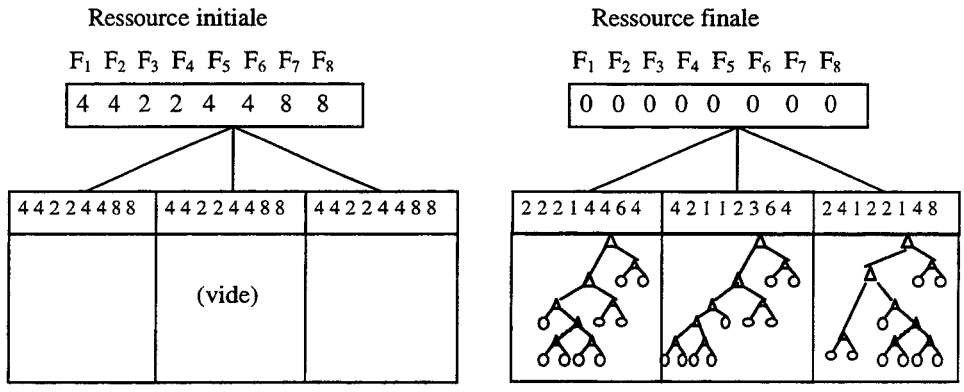


Figure IV. 22 : Contrôle des ressources dans le stock.

IV.4.3.6.Arbre initial

L'arbre initial contenant une seule feuille F_k correspond au placement de la première forme dans une bande. Cet arbre peut être interprété comme un arbre composé de la forme F_k associé à la forme contenante qui est la matière. Ceci est équivalent à positionner F_k sur la surface matière avec une rotation d'angle θ_k par rapport à son centre dans le référentiel global, en respectant évidemment son ensemble d'orientations permises. La forme est ensuite placée avec une connexité par défaut ($\gamma_k = 1$), en haut à gauche qui est l'origine du repère global. Donc un arbre minimal est un arbre contenant une seule forme mais ayant deux feuilles, la deuxième feuille étant la matière qui constitue la partie fixe.

IV.4.3.7.Population initiale

La population est constituée de placements partiels (arbres à deux feuilles). Les feuilles sont choisies au hasard. Pour la première forme il n'y a pas de connexité.

La population est initialisée avec des arbres contenant une seule forme. Cette initialisation est similaire à la méthode heuristique décrite dans [**Soenen77**]. Dans ce dernier, on initialise la solutions en disposant les formes dans autant de bandes que de pièces. Ensuite, on cherche à diminuer le nombre de bandes.

L'arbre à une feuille est construit en faisant subir à cette première forme une rotation. Les arbres deviennent de plus en plus importants au fur et à mesure que la recherche progresse, par mutation ou par croisement (***Figure** IV.* 23).

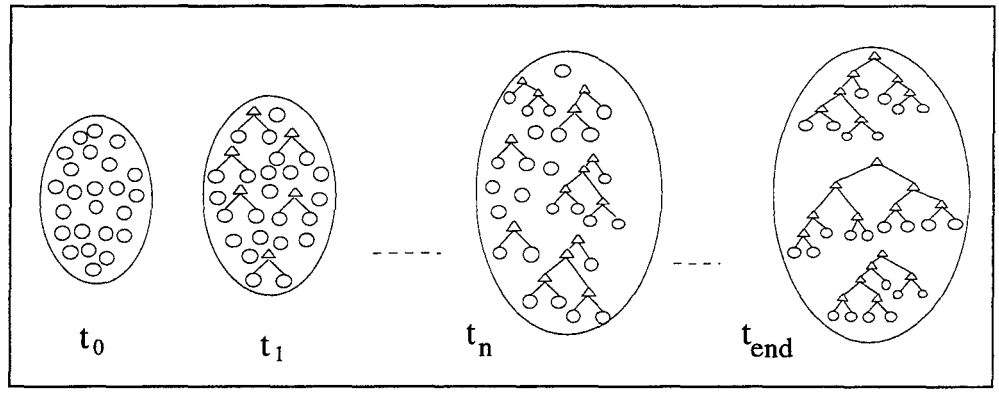


Figure IV. 23 : Evolution d'un chromosome à travers des générations.

IV.4.3.8.Sélection pour former le bassin de reproduction

Cette méthode de sélection est de type $(\mu + \lambda)$ utilisée dans la méthode de stratégie à évolution [**Bäck94**, ***Annexe 2***]. Elle est effectuée de la manière suivante: Les λ individus nouvellement générés viennent s'ajouter aux μ autres individus de la population actuelle et la sélection s'effectue ainsi dans une population à $(\mu + \lambda)$ individus pour choisir μ individus parents. La priorité de sélection se porte d'abord sur les individus nouvellement générés. Les individus de faible fitness sont éliminés lorsque la taille de la population dépasse μ.

Cette méthode permet aux parents et aux enfants de se concourir. Ceci est vrai aussi dans la nature où certaines espèces de longévité plus élevée, parents et enfants se trouvent longtemps dans la même population.

IV.4.3.9.Croisement de sous-arbres

Dans un chromosome, chaque forme F_k ne peut apparaître plus de a_k fois, il est donc nécessaire de vérifier la redondance des formes à chaque échange de sous-arbres. Pour notre cas, le site de croisement est choisi au hasard, mais le croisement n'est effectué qu'après avoir vérifié la redondance. Ce croisement doit pouvoir transmettre des bons blocs aux générations suivantes.

On échange les sous-arbre entre arbres (***Figure*** *IV.* 24). Le croisement est permis si les arbres sont composés de plus de trois formes, sinon il sera soumis à la mutation.

La procédure est la suivante:

- Sélectionner aléatoirement deux arbres, soit T_1 and T_2;
- Dans chaque arbre, sélectionner au hasard un site de croisement, soit X_1 et X_2 resp.;
- Tester les contraintes;
- Echanger le sous-arbres si les contraintes satisfaites; le premier sous-arbre (t_1, X_1), de racine X_1, est pris dans T_1 sur la droite de X_1 et le second (t_2, X_2) est pris dans T_2 sur la droite de X_2;
- Générer les arbres-enfants:
- soit en changeant les valeurs de rotation et de connexité,
  - soit en prenant l'opérateur en X_1,
  - soit en prenant l'opérateur en X_2.

Avant le croisement, la vérification de contraintes telles que le nombre de formes, la surface occupée etc. doit être effectuée. Les éléments échangés peuvent garder leurs orientation et connexité relatives sinon ils peuvent être mutés.

La vérification de redondance et de débordement est faite de la façon suivante:

- vérifier si (t_2, X_2) ne contient pas de forme redondante par rapport à l'arbre hôte (T_1 - (t_1, X_1));
- Si oui, enlever celui qui est de trop dans (t_2, X_2);
- tester si la somme des surfaces des formes dans (t_2, X_2)peut être contenue dans la surface de bande laissée par l'arbre (T_1 - (t_1, X_1));
- si non, le croisement échoue; il n'y a pas d'enfant, l'arbre parent reste intact;
- même opération pour (t_1, X_1) sur (T_2 - (t_2, X_2)).

La redondance des formes est une contrainte qui restreint la liberté des opérateurs. On pourrait la relaxer en ajoutant un terme de pénalité à la fonction de fitness.

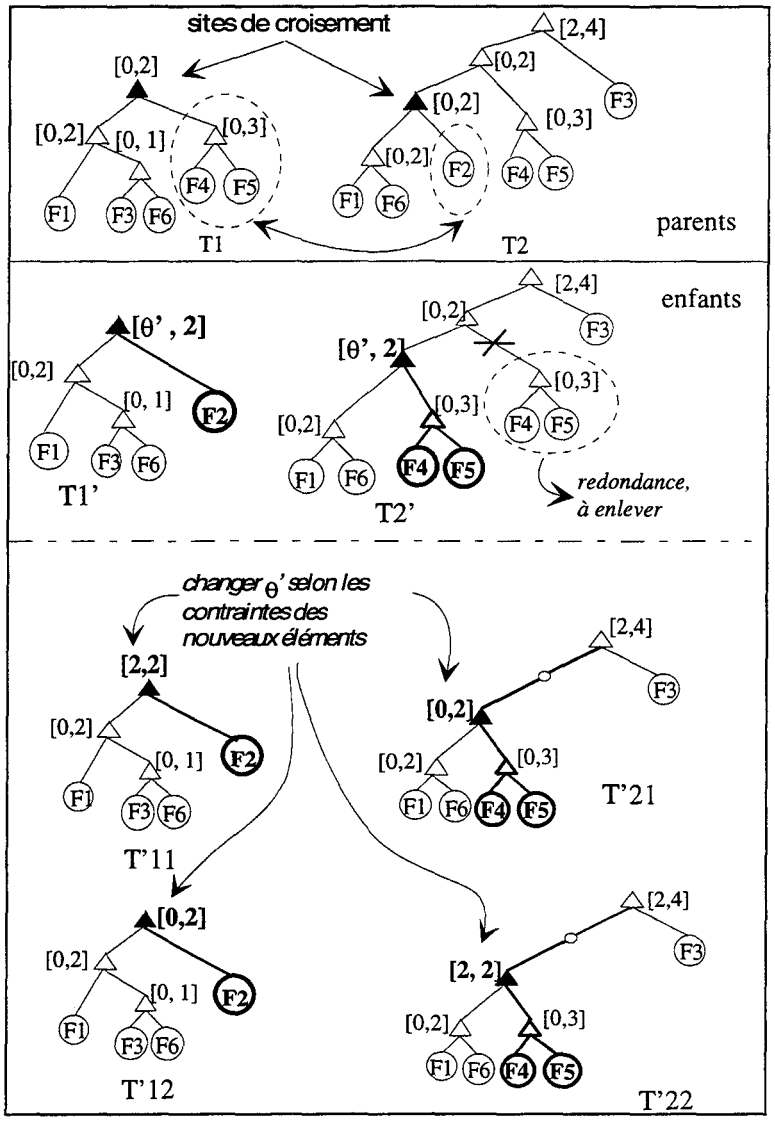


Figure IV. 24 : Croisement entre deux arbres.

IV.4.3.10.Mutation d'arbre

Les trois types de mutation qui peuvent être utilisés sont les suivants (***Figure*** *IV.* 25):

- changer un opérateur dans un arbre choisi au hasard;
- permuter au hasard deux formes dans un même chromosome;
- enlever une forme d'un arbre, ceci peut être utile pour faire de la place à une forme plus grande en enlevant une petite forme ou pour permettre le croisement.

La mutation est un opérateur local générant un seul successeur. Chaque sous-arbre du chromosome est testé aléatoirement s'il doit être muté ou pas. Dans le cas échéant,. l'opération

de mutation consiste soit à enlever un noeud du sous-arbre et le réinsérer dans un autre sous-arbre du même individu; soit à modifier seulement l'orientation ou la connexité en prenant soin de respecter les contraintes de dépassement en largeur ou de superposition. Si une de ces contraintes n'est pas satisfaite, l'individu reste inchangé.

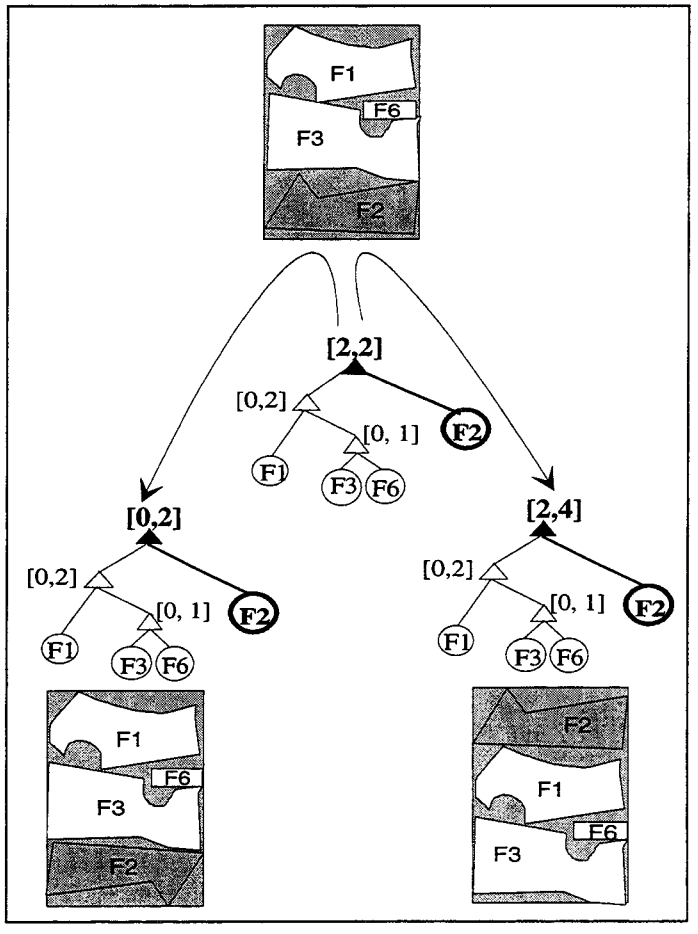


Figure IV. 25: Mutation d'un arbre. Le site de mutation est choisi parmi les noeuds "opérateur" dont l'élu peut être prendre de nouvelles valeurs dans les valeurs permises.

IV.4.4. Implantation

Cet algorithme résout le côté combinatoire du placement pour trouver la meilleure combinaison de formes et leurs paramètres de placement pour envoyer à l'algorithme du deuxième niveau qui optimise les imbrications. Les principales composantes de l'algorithme sont présentées dans les sous-sections suivantes.

IV.4.4.1. Quelques composantes de l'algorithme en langage Smalltalk

```
4.    " Initialisation "
      4.1.  nbGene = 0, population new, popSelect new,
            meilleurIndividu new.
      4.2.  pC, pM, nbGeneMax, popSize.
      4.3.  maxPop = nbBande*popSize. "si résolution bande par
            bande"
      4.4.  population: self popInit.
      4.5.  popSuivante := population.
5.    " Evolution "
      5.1.  whileTrue:[nbGene < nbGeneMax]
      5.2.  [if popSelect size < maxPop
            5.2.1.  ifTrue: [popSelect add: (self selectionIn:
                  popSuivante).
            5.2.2.  ifFalse:[popSelect add: (self selectionIn:
                  population).
      5.3.  population := popSuivante.
      5.4.  popSuivante := self generer: popSelect.
      5.5.  popSelect add: (self selectionIn: popSuivante).
      5.6.  self supprimeFaibleDe: population.
      5.7.  nbGene := nbGene +1] "fin whileTrue"
^meilleurIndividu.
```

```
      "population Initiale"
4.    whileTrue: [population size < maxPop]
      4.1.  [racine := self creerRacine. "definir la surface de la
            matiere"
      4.2.  forme1 := self tirageAleatoireDans: stock.
      4.3.  connex1:=2.
      4.4.  orientation1 := randomIn:(rotationSetOf: forme1).
      4.5.  noeud1 := self placeA:racine avec:forme1 cnx:connex1
            rot:orientation1.
      4.6.  forme2 := tirageAleatoireDans: stock.
      4.7.  connex2 := randomIn: Set(1, 2, 3, 4).
```

```
    4.8.  orientation2 = randomIn:(rotationSetOf: forme2).
    4.9.  noeud2 := self placeA:noeud1 avec:forme2 cnx:connex2 rot:
          orientation2.
    4.10. population add: noeud2]. "fin whileTrue"
^population
```

```
" SelectionIn: aPopulation "
    |r pi qi moy pool|
  1) r := Random new.
  2) pi := 0, qi := 0.
  3) moy := (self moyPopulation: aPopulation.
  4) whileTrue [pool size < maxPop]
      4.1.  [:i| pi := fitness (indiv_i)/ moy.
      4.2.  qi := qi + pi.
      4.3.  r next < qi
      4.4.  ifTrue: [pool add: indivi]
      4.5.  i := i +1 mod: aPopulation size.
        ]. "fin whileTrue"
^pool
```

Dans le bassin de reproduction (*pool*), des individus sont tirés au sort aléatoirement pour le croisement ou pour la mutation.

L'algorithme utilisant le chromosome en arbre cherche les combinaisons de formes et la tâche de l'algorithme utilisant le code par peignes cherche la meilleure imbrication locale de ces combinaisons. Cette opération consiste à trouver le rectangle de surface minimale circonscrit aux deux formes imbriquées.

IV.4. RESULTATS

Le simulations sont effectuées en Smalltalk sur une *SparcStation20* avec un ensemble de 22 formes (carnet *ralf.cdé*). L'interface de placement (***Figure IV.26***) permet de voir le comportement des algorithmes en fonction des paramètres. Dans un premier temps, on va étudier la convergence de l'algorithme en recherchant l'écart final des résultats sur plusieurs exécutions. Ensuite en faisant varier des paramètres caractéristiques (population, taux de croisement, etc.), on observera la performance en terme de qualité des solutions fournies.

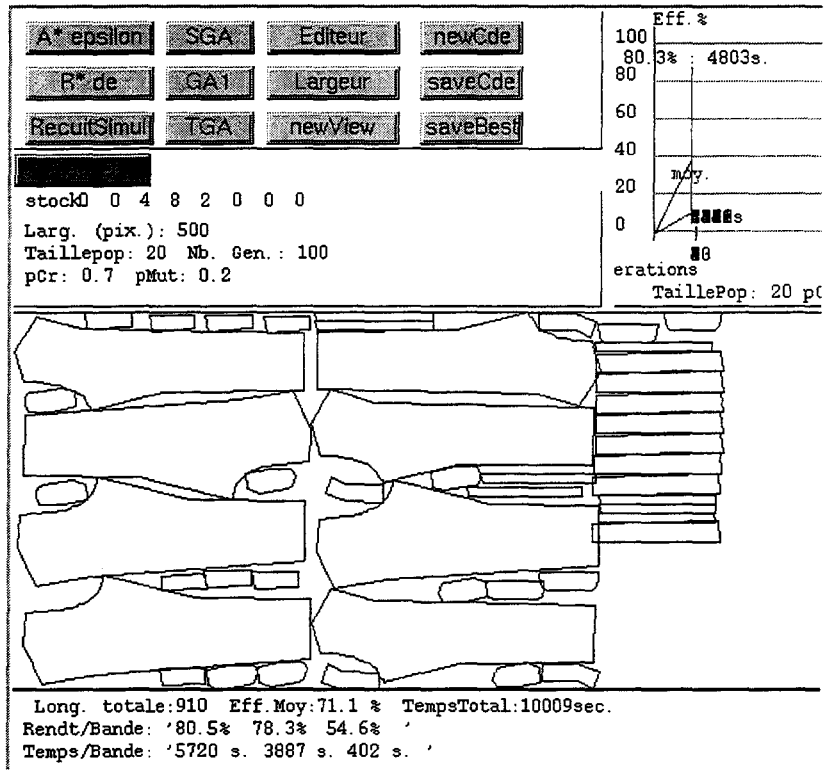


Figure IV.26: Interface de placement.

Dans la section suivante, nous étudions l'influence des paramètres génétiques sur le comportement de la recherche. Dans un premier temps, la taille de la population est fixe et les taux de croisement et mutation varient. Ensuite, c'est la taille de la population qui varie.

Le choix de la taille de la population dépend aussi de l'importance de l'ensemble des formes à placer. Selon E. Falkenauer [**Falkenauer94a,b**], pour un problème de bin-packing, la taille de la population doit être choisie selon le principe de *redondance minimale*. C'est-à-dire que chaque membre de l'espace de recherche doit être représenté aussi peu que possible par des

chromosomes distincts (une seule fois dans le cas idéal). Ainsi, cela permet de réduire la taille de l'espace dans lequel l'algorithme doit parcourir.

Cependant, ce minimum doit être juste assez pour avoir quand même une diversité dans la population, et la redondance dans une structure de chromosome peut être utile pour sortir des minima locaux. Le même chromosome n'évolue pas de la même manière s'il est exposé à un environnement différent **[Collard94]**.

Dans notre cas, nous choisissons une taille de population de façon à ce que la structure des individus recouvre assez bien le nombre de bandes que l'ensemble des formes peut fournir. Pour l'étude, nous nous concentrons sur le carnet "*ralf.cde*" (*cf. chapitre 1*) et la population minimale pour ce carnet est de 3.

IV.4.1. Population = 5

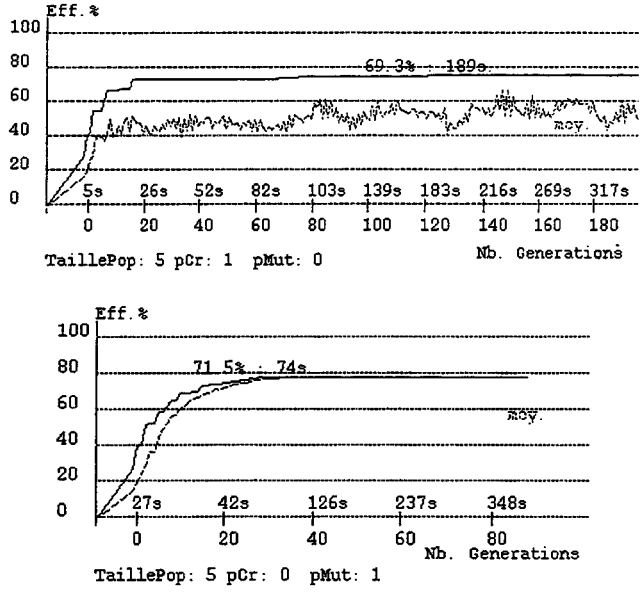


Figure IV. 17: L'influence des taux de croisement et de mutation sur la convergence. Le croisement favorise plus l'exploration et la convergence est moins rapide tandis que la mutation favorise l'exploitation qui fait converger rapidement l'algorithme.

Sur la **Figure IV. 17**, on peut noter que le croisement permet surtout l'exploration, l'algorithme est lent à se converger. Tandis que la mutation permet surtout l'exploitation et l'algorithme converge plus vite. Il faut donc trouver le balance dans l'application de ces opérateurs. Les

courbes à la **Figure IV. 28** montrent que la recherche converge avec plus ou moins de régularité selon l'importance du taux de croisement.

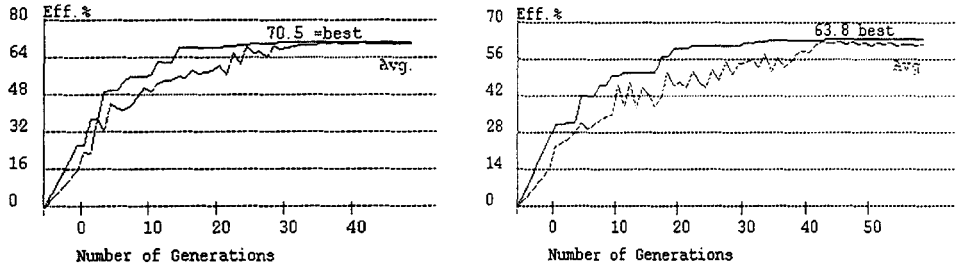


Figure IV.28: Evolution de la recherche pour une taille de population de 5 avec pCross = pMut = 50% (à gauche) et pCross = 80% pMut = 20% (à droite).

On remarque que le fitness moyen de la population évolue plus ou moins comme le fitness du meilleur individu. Donc, si la convergence est trop rapide, et si la taille de la population est trop faible, il n'y a pas assez de diversité pour permettre à l'algorithme de s'en sortir. On peut essayer d'accroître la taille de la population.

IV.4.2. Population = 10

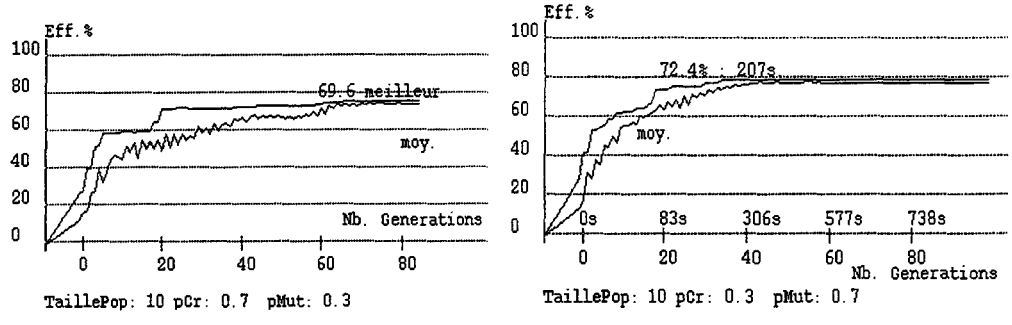


Figure IV.29 : pC = 70% et pMut = 30% pC = 30% et pMut = 70%

IV.4.3. Population = 20

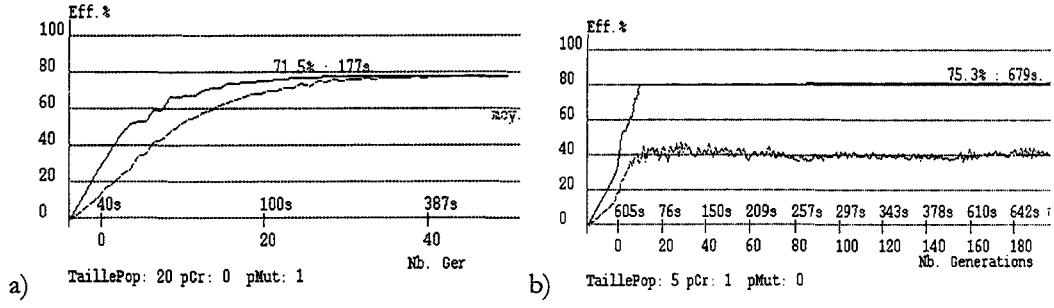


Figure4.30 : a) pC = 0% et pMut = 100% b) pC = 100% et pMut = 0%

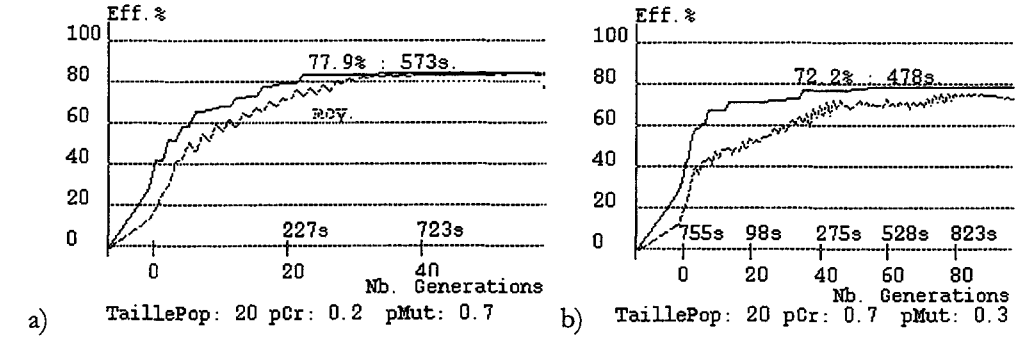


Figure IV.31: *a*) pC= 20% et pMut = 70% *b*) pC= 70% et pMut = 30%

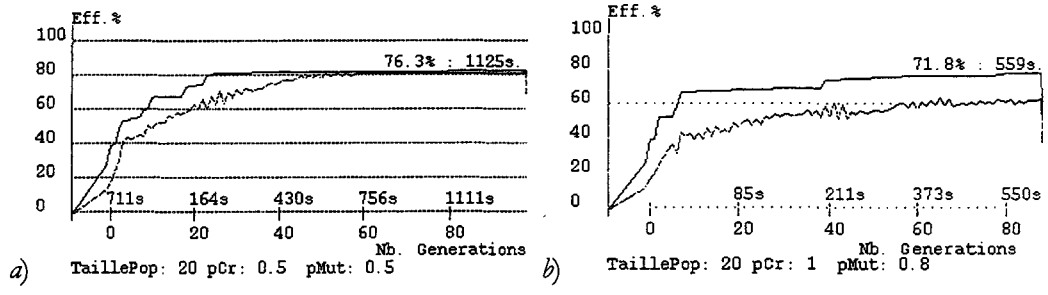


Figure IV.32: *a*) pC=50% et pMut = 50% *b*) pC= 100% et pMut = 80%

IV.4.4. Exemples de placement

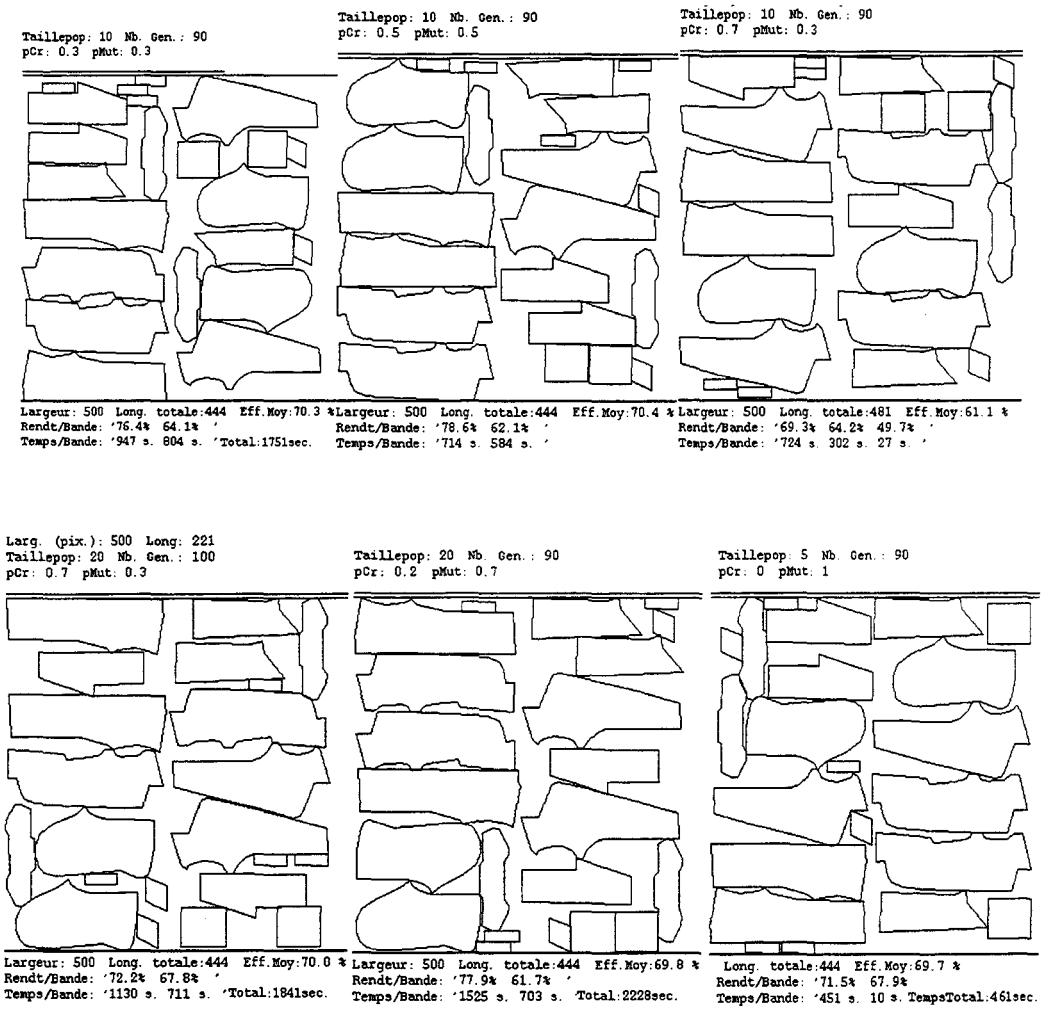


Figure IV.33: Exemples de placements d'un ensemble de 22 formes.

IV.4.5. Autres exemples de placement

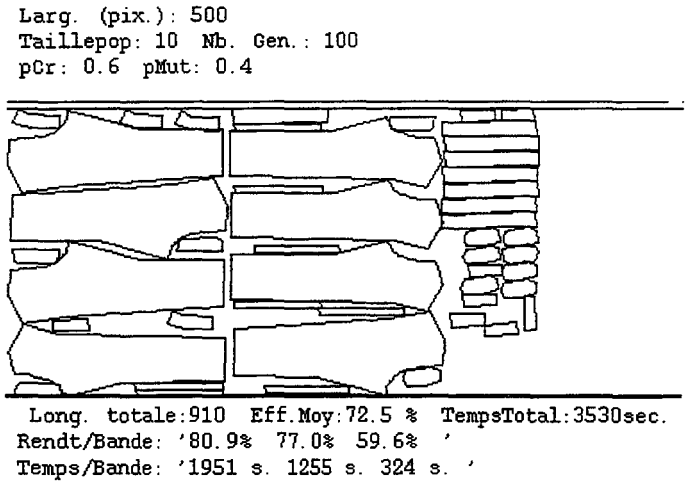


Figure IV.34 : Exemple de placement de 48 formes de pantalon.

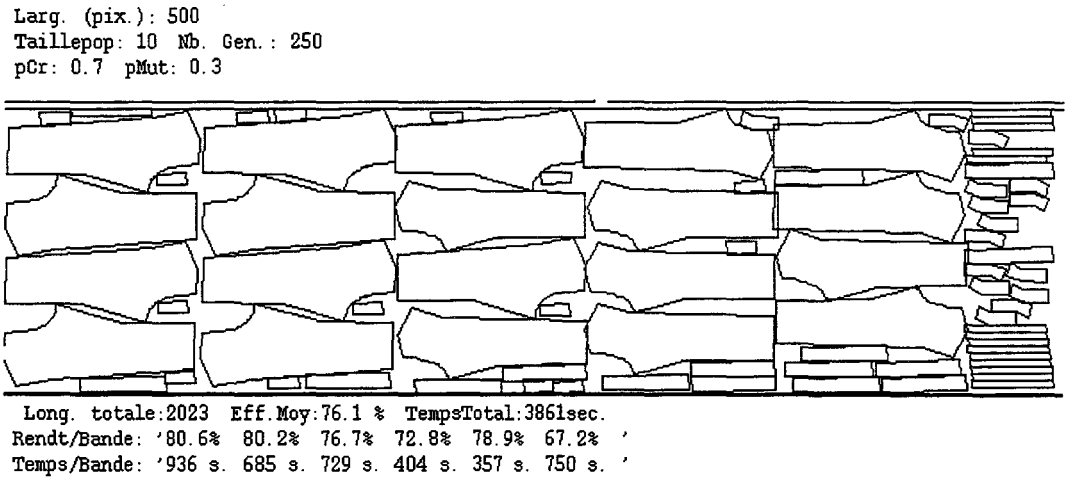


Figure IV.35 : Exemple de placement de 84 formes de pantalon.

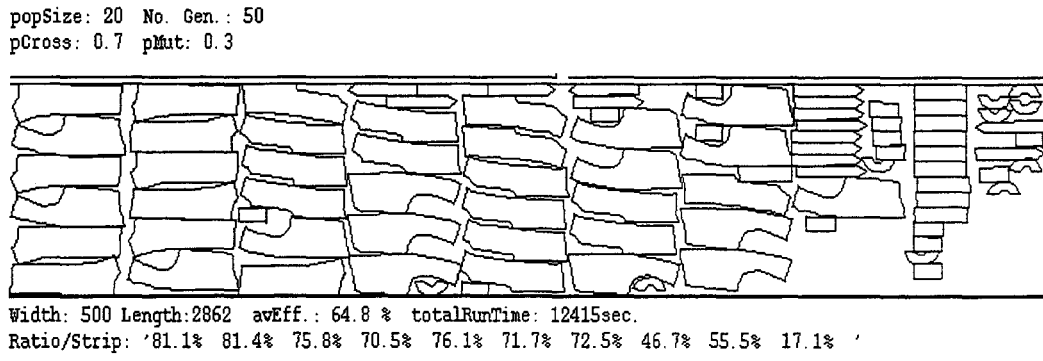


Figure IV.36 : Exemple de placement avec l'ensemble chem3.cde.

IV.5. Discussion

Le temps d'exécution est assez long. L'AG débute la recherche à partir de plusieurs solutions candidates et il peut obtenir plusieurs bonnes solutions mais nécessite des exploitations locales de ces bonnes solutions. L'utiliser d'un taux de mutation dynamique pourrait aider, vers la fin de la recherche, de préserver le plus possible les structures acquises; c'est-à-dire, permuter deux formes simples sera plus fréquent dans notre cas. Donc, supprimer ou échanger des formes est une recherche locale avec exploitation de la région immédiate.

La fonction fitness n'est pas très efficace pour déterminer le meilleur placement final en fonction des contraintes. Le meilleur fitness de la génération actuelle peut ne plus l'être dans le futur. Donc, la fonction fitness doit inclure l'information qui estime le mieux placement final (une sorte d'heuristique) sans avoir à décoder l'arbre entier.

Les AG peuvent être utilisés pour une recherche hors-ligne, si une solution globalement optimale est réellement nécessaire (par exemple, avec le matériau de coût élevé). D'autres expériences restent encore à faire ainsi que l'implantation de l'algorithme avec un autre langage que le Smalltalk, afin de bénéficier les librairies existantes et apprécier mieux l'efficacité de notre approche par rapport aux autres approches de la littérature.

IV.6. Conclusion

En résumé, l'approche par le codage hiérarchique est meilleure que l'approche par peigne réduit. La combinaison des deux permet de partager le processus de recherche en deux niveaux: le côté combinatoire pris en charge par l'approche en arbre et le côté d'imbrication locale pris en charge par le codage en peigne.

L'efficacité des algorithmes sur un petit ensemble de formes est compétitive avec les autres méthodes développées. Comme AG doit maintenir une population de solutions et doit maintenir toutes les formes dans le chromosome pour les faire évoluer en parallèle, la solution qu'il fournit est une solution globale avec le placement de tout l'ensemble. Selon la taille de la population, il peut exiger un temps plus long avant qu'une solution acceptable soit obtenue. Mais il est capable de trouver une solution globale. Cependant, il ne peut pas fournir de sous-solution en cours de la recherche au préjudice d'altérer la structure du chromosome. Toutefois, ceci ne devrait pas constituer de réel problème puisque les interaction entre gènes est faible; en enlevant un gène du chromosome, la longueur du chromosome change. On peut alors faire une recherche avec les chromosome à taille évolutive. On peut considérer le cas où AG doit s'adapter à un environnement changeant [**Collard94**].

IV.7. Références Bibliographiques

[Alliot94] J. M. Alliot, T.Schiex, *Intelligence Artificielle & Informatique Théorique,* Cepaduès, Toulouse1994, pp. 441-460.

[Baker87] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm", Vanderbilt University, *Proceedings of the Second International Conference on Genetic Algorithms*, ICGA'87, Lawrence Erlbaum Associates Publishers, 1987, pp. 14-21.

[Bäck93a] Th. Bäck, G. Rudolph, H.-P. Schwefel, "Evolutionary Programming and Evolution Strategies: Similarities and Differences ", D.B. Fogel and W. Atmar, editors: *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 11-22, Evolutionary Programming Society, San Diego CA 1993.

[Bäck94] Th. Bäck and H.-P. Schwefel, "Evolution Strategies I: Variants and their computational implementation", and "Evolution Strategies II: Theoretical aspects implementation "G. Winter, J . Périeaux, M. Galá and P. Cuesta, editors*: Genetic Algorithms in Engineering and Computer Science*, , Wiley, Chichester, 1995, pp. 111-126 and pp. 127-140.

[Bäck95] Th. Bäck, "Generalized Convergence Models for Tournament- and (μ , l)-Selection", L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 2-8, Morgan Kaufmann, San Francisco CA, 1995.

[Bäck96a] Th. Bäck, M. Schütz and S. Khuri, "Evolution Strategies: An Alternative Evolution Computation Method", J. M. Alliot, E. Lutton, E. Ronald, M. Schoenhauer and D. Snyers, editors: *Artificial Evolution*, Springer, Berlin, 1996, pp. 3-20.

[Beasley93a] D. Beasley, D.R.Bull, R.R.Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals", Techn. Report of Univ. of Cardiff, UK, *University Computing*, 15(2), 1993, pp. 58-69.

[Beasley93b] D. Beasley, D.R.Bull, R.R.Martin, "An Overview of Genetic Algorithms: Part 2, Research Topics", Techn. Report of Univ. of Cardiff, UK, *University Computing*, 15(4), 1993, pp. 170-181.

[Beasley93c] D. Beasley, D.R.Bull, R.R.Martin, "Reducing Epistasis in Combinatorial Problems by Expansive Coding", *Proc. 5th Int; Conf. GA*, Univ. Illinois at Urbana-Champaign, Jul. 1993, S. Forrest, Editor, Morgan Kaufmann Pub. Inc.

[Booker89] L.B. Booker, D.E. Goldberg, J.H. Holland, "Classifier Systems and Genetic Algorithms", *Artificial Intelligence,* Vol. 40, 1989, pp235-282.

[Bounsaythip95] C. Bounsaythip, S. Maouche, M. Neus "Evolutionary Search Techniques, Application to Automated Lay-Planning Optimization Problem", in *Proc. IEEE Systems Man and Cybernetics (SMC'95)*, Vancouver, Canada, Oct. 22-25, 1995, vol. 5, pp. 4497-4503.

[Caux95] C. Caux, H. Pierreval, M.C. Portman, "Les Algorithmes Génétiques et Leur Application d'Ordonnancement", *Automatique, Productique Informatique Industrielle,* Vol 29-n°4-5/1995, pp. 409-443.

[Chu95] P.C. Chu, J. Beasley, "A Genetic Algorithm for the Set Partitioning Problem", Technical report, April 1995.

[Chung93] Meng Quing Chung, *Application des Algorithmes Génétiques à la Résolution de Problèmes et à la Commande de Systèmes*, Thèse Université de Paris XII, octobre 1993.

[Cohoon87] J. P. Cohoon, W.D. Paris, "Genetic Placement", *IEEE Trans. CAD*, Vol. CAD-6, N° 6, Nov. 1987, pp. 956-964.

[Cohoon91] J. P. Cohoon, S.U. Hegde, W.N. Martin, D.S. Richards, "Distributed Genetic Algorithms for the Floor Plan Design problem", *IEEE Trans. CAD*, Vol. CAD-10, N° 4, Apr. 1991, pp. 483-491.

[Collard94] Ph. Collard and J-Ph. Aurand and J. Biondi, "De l'Intérêt des Sosies dans un Système Autonome", *Actes des journées de Rochebrune*, Jan. 17-21, 1994.

[Corne95] D. Corne and P. Ross, "Some Combinatorial Landscapes on which Genetic Algorithm Outperforms Other Stochastic Iterative Methods", *Evolutionary Computing; AISB Workshop, Sheffield 1995, Selected paper, Editor T. Fogarty, Springer-Verlag Lecture Notes in Computer Science*, 1995.

[Cotta95] C. Cotta, J.F. Aldana, A.J. Nebro, J.M. Troya, "Hybridizing Genetic Algorithms with Branch and Bound Techniques For the Resolution of The TSP", in *Proc. of the Int. Conf. on Artif. N.Net. and GAs*, April 1995, in Alès, France, pp. 277-280.

[Davidor91] Y. Davidor, *Genetic Algorithms and Robotics : a heuristic strategy for optimization,* World Scientific series in Robotics and Automated Systems, Vol. 1, Ed. T.M. Husband.

[Davis87] L. Davis, Editor, *Genetic Algorithms and Simulated Annealing - (Research notes in Artificial Intelligence)*, Morgan Kaufman publishers, Inc., 1987.

[Davis91] L. Davis, *The Genetic Algorithm HandBook*, Ed. New York: Van Nostrand Reinhold, ch. 17, 1991.

[De Jong75] K. De Jong, *"An Analysis of the Behaviour of a Class of Genetic Adaptive Systems"*, Doctoral dissertation, Dept. Computer and Communication Sciences, University of Michigan, Ann Arbor.

[DeJong80] K. De Jong, "Adaptative System Design : a Genetic Approach", *IEEE Trans. SMC*, Vol. 10, N° 9, sept. 1980, pp. 566-574.

[Dighe96] R. Dighe, M.J. Jakiela, "Solving Pattern Nesting Problems with Genetic Algorithms Employing Task Decomposition and Contact Detection", *Evolutionary Computation*, Vol.32, No. 3, 1996, pp. 239-266.

[Falkenauer94a] E. Falkenauer, "A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems", *Evolutionary Computation,* vol. 2, No. 2, MIT 1994, pp. 123-144.

[Falkenauer94b] E. Falkenauer, "Setting New Limits in Bin Packing with a Grouping GA using Reduction", *Tech. Report Ro108*, March 1994, CRIF, Research Center for Belgian Metal Working Industry, dept. Industrial Automation, 14 pages.

[Fogel93] D.B. Fogel, Applying Evolutionary Programming to Selected Travelling Salesman Problem, *Cybernetics and Systems* : An International Journal, no. 24, 1993,
pp. 7-36.

[Fogel94] L.J. Fogel, "Evolutionary Programming in Perspective: the Top-Down View", in [Zurada94], pp. 135-146.

[Fogel95] D.B. Fogel, *Evolutionary Computation : Toward a new Philosophy of Machine Intelligence*, IEEE Press, Ed. J.B. Anderson, NY 1995.

[Fourman85] M. Fourman, "Compaction of Symbolic Layout using Genetic Algorithms", *Proceedings of the First International Conference on GA and their Applications*, July 24-26, Hillsdale: Lawrence Erlbaum Associates, 1985, pp. 141-153.

[Fujita93] Kikuo Fujita, Shinsuke Akaji, Noriyasu Hirokawa, "Hybrid Approach for Optimal Nesting Using a Genetic Algorithm and a Local Minimization Algorithm", DE-Vol. 65-1, *Advances in Design Automation*, vol.1, ASME 1993, pp. 477-484.

[Gathercole94] C. Gathercole, P. Ross,"Dynamic Training Subset Selection for Supervised Learning in Genetic Programming", to be published in *PPSN-III*, Springer Verlag, 1994.

[Ghosh91] Ghosh Subrata, Mahanti Ambuj, "Improving the Efficiency of Limited-Memory Heuristic Search", Tech. Report, University of Maryland, Dept. Computer Science, 1990-1991.

[Goldberg89] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Jan.1989.

[Goodman94] E.D. Goodman, A.Y. Tetelbaum, V.M. Kureichik, "A Genetic Algorithm Approach to Compaction, Bin Packing, and Nesting Problems", Technical report of Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, No. 940702, July 1994.

[Grecu96] D. L. Grecu, D. C. Brown, "Dimensions of Learning in Agent-Based Design", *AID'96 workshop on Machine Learning in Design*, Internet *http://cs.wpi.edu/~dgecu*

[Grefenstette86] J. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", *IEEE Trans. on SMC*, Vol. 16, n°1, Jan/Feb. 1986, pp. 122-128.

[Grefenstette87a] J. J. Grefenstette, *Genetic Algorithm and Their Applications : Proceedings of the 2nd Int. Conf. On Genetic Algorithm*, Hillsdale, NJ: Lawrence Erlbaum Associates, 1987.

[Gritz95] L. Gritz and J. K. Hahn, Genetic Programming Evolution of Controllers for a 3-D Character Animation, *Proc. of Genetic Programming'97 Conference*, Jul. 1997.

[Holland75] J.H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1975.

[Holsapple93] C.W. Holsapple, V.S. Jacob, "A Genetic Based Hybrid Scheduler for Generating Schedules in Flexible Manufacturing Contexts", *IEEE Trans. on Systems Man & Cybernetic*, Vol. 23, N° 4, Jul/Aug. 1993.

[Ismail92] H.S. Ismail, K.K.B. Hon, "New Approaches for the Nesting of Two-Dimensional Shapes for Press Tool Design", *International Journal of Production Research*, Vol. 30, No. 4, 1992, pp. 825-837.

[Kado95b] Kazuhir Kado, *An investigation of Genetic Algorithms for Facility Layout Problem*, Thesis of Univ. of Edinburg, 1995.

[Khuri94] S. Khuri, T. Bäck, J. Heitköter, "An Evolutionary Approach to Combinatorial Optimization Problems", *Proc. of CSC'94*, Phoenix Arizona, March 8-10, 1994.

[Khuri95] S. Khuri, M. Schütz, "Evolutionary Heuristics for the Bin Packing Problem", *Proc. of ICANNGA'95*, Alès, France, April 18-21, 1995.

[Koakutsu93] S. Koakutsu, H. Hirata, "Genetic Simulated Annealing for Floor Design", *Proc. IFIP 16th Conference on System Modeling and Optimization*, July 1993, Compiègne, France, pp. 268-277.

[Koza92] J. R. Koza, *Genetic Programming*, Cambridge, MA, MIT Press,1992.

[Kroger90] B. Kröger, P. Schwenderling and O. Vornberger, "Parallel Genetic Packing of Rectangles", in *Proceedings of Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Editors, First Workshop, Berlin: Springer Verlag, 1990, pp. 60-164.

[Kroger92] B. Kroger, P. Schwenderling and O. Vornberger, "Massive Parallel Genetic Packing", in G.L. Reijns and Jian Luo, Editors, *Transputing in Neural Network Applications*, IOS Press, Amsterdam, 1992, pp. 214-230.

[Lin93] F-T Lin, C-Y Kao, C-C Hsu, Applying the Genetic Approach to Simulated Annealing in Solving Some NP-Hard Problems, *IEEE Trans. on Syst. Man & Cybernetic*, Vol. 23, n° 6, nov-dec 1993, pp. 1752-1767.

[Lutton93] Evelyne Lutton, "Rapport d'Expertise SGDN: Etat de l'Art des Algorithmes Génétiques", Rapport INRIA - Roquencourt, 8 décembre 1993.

[Maher96] M. L. Maher, J. Poon and S. Boulanger, "Formalising Design Exploration as Co-Evolution. A Combined Gene Approach", *Advances in Formal Design Methods for CAD*, John S. Gero and Chapman&Hall, Editors, 1996.

[Michalewicz92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag, collection *Artificial Intelligence*, Berlin Heidelberg 1992.

[Oppacher95] F. Oppacher, D. Deugo, "Automatic Change of Representation in Genetic Algorithms", in *Proc. of the Int. Conf. on Artif. Neural Network and GA (ICANNGA)*, in Alès, France, April 1995, pp. 218-222.

[Pargas93] R. P. Pargas and R. Jain, "A Parallel Stochastic Optimization Algorithm for Solving 2D Bin Packing Problems", in Proceedings, *The Ninth International Conference On Artificial Intelligence for Applications*, Orlando, FL, 1-5 March 1993, IEEE Computer Society Press, Los Alamitos, CA, pp. 18-25.

[Park94] K. Park, B. Carter, "On the Effectiveness of Genetic Search in Combinatorial Optimization", BU-CS-94-010, novembre 1994.

[Pearson95] D.W. Pearson, N.C. Steele, R.F. Albrecht, Editors, *Proc. Of the Int. Conf. on Artificial Neural Nets and Genetic Algorithms*, Alès, France, Springer Verlag, 1995.

[Petridis] V. Petridis and S. Kazarlis, "Varying Quality function in Genetic Algorithms and the Cutting Problem", Dept. of Electrical Engineering, University of Thessaloniki (Greece), 1994. Via Univ of Edinburg, dept. Artificial intelligence, http://www.dai.ed.ac.uk/.

[Preux94] Philippe Preux, "Les Algorithmes Génétiques", rapport Technique AS-145 LIFL, ou TR n°1 au Laboratoire d'informatique du Littoral, février 1994.

[Randelman86] R. E. Randelman, and G.S. Grest, "N-City Traveling Salesman Problem - Optimization by Simulated Annealings", *Journal of. Statistical Physics*, No. 45, pp. 885-890, 1986.

[Rechenberg94] Ingo Rechenberg, "Evolution strategy", in [ZMR94], pp. 147-159.

[Reeves95] Colin R. Reeves, "A Genetic Approach to Bin-Packing", in *Proc of the 1NWGA*, Vaasa, 9-12 January 1995, pp. 35-49.

[Renders95] J. M. Renders, *Algorithmes Génétiques et Réseaux de Neurones*, Hermès, Paris 1995, ISBN 2-86601-467-7, ISSN 1258-1828.

[Rosca94] J. P. Rosca, D. H. Ballard, "Genetic Programming with Adaptive Representations", Techn. Report 489, Univ. Rochester, NY, Feb. 1994.

[Ross94] P. Ross, D. Corne and H.-L. Fang, "Successful Lecture Time Tabling with Evolutionary Algorithms", *ECAI'94, Workshop W17: Applied Genetic and Other Evolutionary Algorithms*, 1994.

[Roussel94] Gilles Roussel, *Optimisation du Placement de Formes Irrégulières sur Matières Planes. Application à l'industrie de la Confection*, Thèse de doctorat, Université des Sciences et Technologies de Lille 1, Janv 1994.

[Schmidt96] Martin Schmidt, Genetic Algorithms, Neural Networks, and Fuzzy Logic", Master Thesis, Univ. Aarhus, 1996,
`ftp://ftp.daimi.aau.dk/pub/empl/marsh/www.zip`

[Schoenauer96] Marc Schoenauer, "Shape Representation for Evolutionary Optimization and Identification in Structural Mechanics", *EUROGEN'96*, 1996.

[Smith85] D. Smith, "Bin Packing with Adaptive Search", *Proceedings of the First International Conference on GA and their Applications*, July 24-26, Hillsdale: Lawrence Erlbaum Associates, 1985, pp. 202-207.

[Soenen77] R. Soenen, *Contribution à l'Etude des Systèmes de Conduite en Temps Réel en Vue de la Commande d'Unités de Fabrication*, Thèse d'état, Univ. Valenciennes et du Hainaut-Cambresis, mars 1977.

[Talbi94] El-Ghazali Talbi, P. Bessière, J.M. Ahuactzin, E. Mazer, "Algorithmes Génétiques Parallèles et leurs Applications", LIFL, USTL.

[Wah92] B. W. Wah, "Population-Based Learning: a New Method for Learning from Examples under Resource Constraints", *IEEE Trans. on Knowledge and Data Engineering*, vol. 4, no. 5, Oct. 1992, pp. 454-474.

[Whitley91] D. Whitley, T. Starweather, D. Shaver, "The Traveling Salesman Problem and Sequence Scheduling Quality Solutions Using Genetic Edge Recombination", *in* [Davis 91].

[Yip94] P.P.C. Yip, Y.H. Pao, "A Guided Evolutionary Simulated Annealing Approach to the Quadratic Assignment Problem", *IEEE Trans on SMC*, vol. 24, n°9, Sept. 1994.

[Zulawinski95] B.W. Zulawinski, W.F. Punch, E.D. Goodman, "The Grouping Algorithm (GGA) Applied to the Bin Balancing Problem", Tech. Report of Michigan State University, 1995

[Zurada94] J. M. Zurada, R. J. Marks II, C. J. Robinson, Editors. *Computational Intelligence Imitating Life*. IEEE Press, June 1994.

Chapitre V :

Conclusion et Perspectives

CHAPITRE V

Conclusion et Perspectives

V.1. Comparaison des Quatre Méthodes

Dans les graphiques (Figure V. 1, Figure V. 2), nous avons rassemblé quelques résultats permettant de comparer les méthodes développées.

On remarque que pour le carnet *ralf.cde*, l'algorithme évolutionniste permet d'atteindre le plus haut rendement[1]. Autrement, les rendements sont assez compétitifs entre les différentes méthodes. Les raisons pour lesquelles les algorithmes de recuit simulé et évolutionniste ne trouvent pas de maximal dans cet exemple, c'est en partie dû au critère d'arrêt non-basé sur l'optimalité de la solution mais plutôt sur le nombre de générations ou sur le nombre de recuits. Lorsqu'il n'y a pas trop d'extrema locaux (carnet *ralf.cde*), l'algorithme évolutionniste permet d'atteindre le point maximal. Dans le cas contraire, l'algorithme s'arrête sur une solution non-optimale. Bien sûr, on pourrait laisser l'algorithme fonctionner jusqu'à ce qu'il trouve le maximal, mais souvent ce n'est toujours pas possible (dérive génétique, convergence prématurée etc.). De plus, on a pu observer dans les chapitres précédents que cette approche par bande crée un espace de recherche changeant qui nécessiterait un paramétrage dynamique.

[1] (avec une population de 20, taux de croisement de 2% et taux de mutation de 70%, et 50 générations).

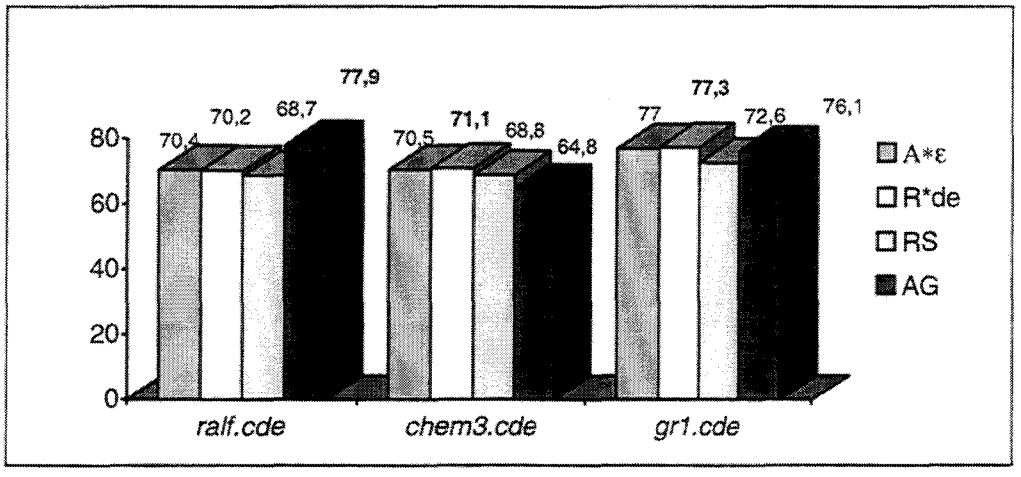


Figure V. 1: Exemples de rendement en % obtenu par chaque méthode. Les rendements sont les meilleurs rendements obtenus (pour AG, ralf.cde,

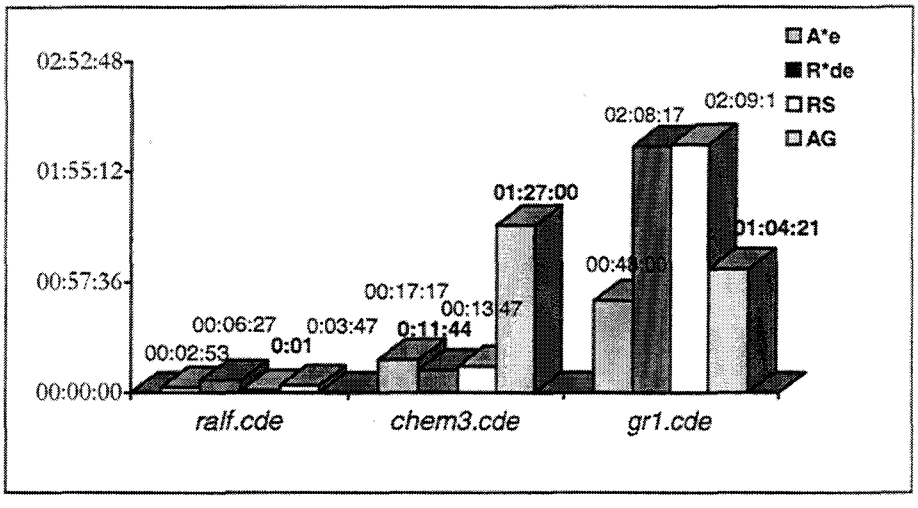


Figure V. 2: Exemple de durée de recherche pour les rendements de la Figure V. 1:. Le temps de calcul pour le carnet *chem3.cde* est élevé car le nombre de bandes est plus élevé (9 bandes).

Dans l'exemple de la Error! Reference source not found., l'algorithme évolutionniste est le plus coûteux en temps de calcul pour le carnet *chem3.cde.* Il faut noter que le temps de calcul d'un AG est proportionnel à la taille de la population (il faut évaluer chaque individu à chaque génération) et donc l'importance de la taille du carnet implique une taille de population assez importante pour avoir assez de diversité. Pour le carnet *gr1.cde*, le recuit simulé et l'algorithme de $R^*_{\delta\varepsilon}$ sont assez similaires en temps de calcul, alors que l'AG et A^*_{ε} sont concurrents. On pourrait dire que la stratégie AG est moins complexe que les recherches en arbres et plus

efficace que le recuit simulé. La conclusion de cette comparaison pourrait sembler un peu hâtive, mais nous sommes convaincus que les AGs offrent plus des possibilités d'étendre la méthode vers une stratégie adaptative. Il reste encore plusieurs aspects des AGs qui pourraient beaucoup améliorer les performances que nous avons obtenues dans ce document. Il s'agit notamment des notions de niches écologiques que nous avions mentionnées dans le chapitre 4. Ces propriétés permettraient à l'AG de mieux localiser plusieurs minima.

V.2. Conclusions

Dans ce document, nous avons étudié trois types de méthodes d'optimisation ainsi que leur application au problème de placement. Les méthodes qui ont été présentées sont:

1. Les algorithmes dérivés de A*. Ces algorithmes, A^*_ε et $R^*_{\delta\varepsilon}$, sont des stratégies de recherche semi-exhaustives utilisant des fonctions heuristiques pour explorer un grand espace de recherche. Cette propriété d'exploitation permet de confiner la recherche dans une zone limitée de l'espace de recherche. Ces méthodes sont souvent utilisées pour une recherche locale et exhaustive. Elles garantissent une solution optimale s'il en existe, et détecte l'absence de solution s'il n'en existe pas. Elles sont en général beaucoup plus performantes qu'une simple descente mais aussi beaucoup plus coûteuses en ressources informatiques, leur efficacité demande un effort non négligeable pour ajuster les paramètres qu'elles font intervenir dans le but de guider la recherche à travers de l'espace de recherche.

2. Le recuit simulé est une recherche aléatoire informée. Mais si le nombre de minima locaux est élevé (ou peu élevé mais inconnu), le nombre d'itérations devient prohibitif. Cependant, cet algorithme a la propriété d'exploration aléatoire locale, au voisinage d'un point donné. Il est utile pour une recherche locale rapide.

3. L'algorithme évolutionniste est une recherche aléatoire distribuée. Il a la propriété de scruter un large espace de solutions mais il a des difficultés de focaliser la recherche sur une région particulière. Il est utile pour détecter les régions potentielles de cet espace de solutions.

Après avoir appliqué ces méthodes à quelques exemples de placement, nous avons observé qu'elles fournissent des performances à peu près équivalentes. L'algorithme génétique apporte en plus la possibilité d'exploitation grâce à l'évolution par population de configurations et à l'opérateur de croisement. Son temps de calcul peut être amélioré en évitant les ré-évaluations des individus.

V.2.1. Expérience Acquise

En fait, ces méthodes, bien que la terminologie utilisée soit différente, partagent certaines caractéristiques communes. Par exemple, le concept de population correspond à la liste d'expansion dans les algorithmes dérivés de A*; les opérateurs croisement/mutation sont similaires aux opérateurs de transformation d'états des arbres; la sélection correspond à la stratégie d'expansion des noeuds alors qu'il correspond au critère de Metropolis dans le recuit simulé (voir Tableau 5-1).

Algorithmes Evolutionnistes	Recuit simulé	Recherche ne arbre
Individu	Configuration	Noeud
Population	-	Liste d'expansion
Fitness	Niveau d'énergie /coût	Coût d'une solution/état
Chromosome	Configuration	Représentation d'un état
Gène	-	Sous-solution
Croisement/mutation	Opérateurs de transformation	Opérateurs de transformation d'états
Sélection	Critère de Metropolis	Choix d'un noeud pour être développé

Tableau V. 1 : Correspondances entre les AE, le recuit simulé et la recherche en arbre.

Le problème de placement se résout alors par une transformation d'un état de placement à un autre (ou des états en d'autres états pour l'AG) en appliquant des opérateurs locaux ou globaux dans un ordre approprié. Un opérateur local (ex. la mutation en AG) provoque des petits changements dans les éléments de l'état courant et se traduit par un faible déplacement dans l'espace d'états. Un opérateur global, comme le croisement en AG, affecte plusieurs éléments et le changement à partir de l'état actuel peut être très important.

Dans certaines applications, il est montré que les méthodes hybrides fournissent de meilleurs résultats mais sont aussi très coûteux en temps de calcul. Le choix de la méthode à utiliser dépend donc en partie du temps de traitement disponible pour résoudre un problème particulier [**Cotta95**]. Pour résoudre un problème complexe, il serait intéressant de pouvoir varier les stratégies au cours de la recherche.

Actuellement, la communauté évolutionniste [**Bäck92, Hoffmeister92, Kanada94, Bilchev96, Parmee96, Streltsov96**] se concentre sur l'aspect adaptatif et auto-adaptatifde l'AG et particulièrement le développement de systèmes appelés la vie artificielle [**Collins91**].

L'auto-adaptation signifie aussi modification génétique et intelligence de l'individu pour s'adapter à son environnement. Cette perspective peut être considérée pour la conception d'une cellule flexible en combinant plusieurs stratégies de recherche au cours de la résolution. L'algorithme proposé dans **[Baluja97]** (COMIT)[2], est basé sur cette idée. Il est conçu pour être incorporé dans les stratégies de recherche multiple et il peut aussi servir à initialiser la population pour les AGs.

Dans **[Logan96, Poli96]**, l'idée est de combiner les différents caractéristiques des autres techniques de recherche classiques avec les algorithmes évolutionnistes. Ils aboutissent à l'algorithme qu'ils appellent GA*, incorporant l'estimation du coût futur de A* dans la recherche par l'AG, jonglant croisement avec le *bactracking*. L'algorithme est appliqué dans le problème de planification et le résultat montre une performance égale au GA classique mais avec un gain de temps dû au nombre réduit d'évaluations.

V.3. Perspectives

L'idée est d'utiliser de façon flexible toutes les méthodes de recherche que nous avons implantées. Une méthodologie basée sur les agents peut être envisageable **[Ferber94, Grecu96, Maher96]**. Cette approche peut être vue comme un mécanisme qui contrôle globalement les agents actifs qui sont représentés par les différents algorithmes de recherche.

Un algorithme de contrôle supervise l'optimisation globale faisant intervenir plusieurs types de recherche. Cet algorithme sera de type programmation génétique avec comme fonctions les algorithmes (A^*_ε, $R^*_{\delta\varepsilon}$, RS, AG). L'alphabet utilisé sera par exemple, l'ensemble: { AG, A^*_ε, $R^*_{\delta\varepsilon}$, RS} (Figure V. 3).

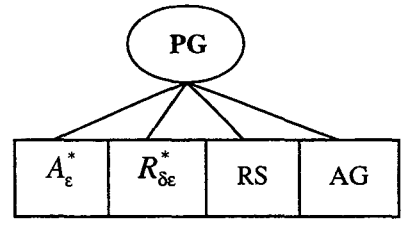


Figure V. 3 : Hiérarchie du macro-algorithme d'optimisation.

[2] COMIT: Combining Optimizers with Mutual Information Trees.

Ainsi, le programme "maître" peut faire basculer d'une stratégie à une autre parmi les méthodes de l'ensemble $\{A^*_\varepsilon, R^*_{\delta\varepsilon}, RS, AG\}$ au cours de la recherche. Les schéma seront par exemple de type : $\{AG, \#, \#, A^*_\varepsilon, R^*_{\delta\varepsilon}\}$, qui signifie que la recherche doit commencer par un AG puis au milieu, n'importe quelle stratégie peut être appliquée et à la fin elle doit se terminer par les algorithmes A^*_ε et $R^*_{\delta\varepsilon}$. Pour cela, il faut considérer aussi les partage de ressource pour chaque algorithme, comme pour des agents. L'attribution du temps d'actions est basée sur l'estimation du nombre d'opérations qui seront effectuées par chaque algorithme pour un certain résultat escompté, ainsi que les coûts intermédiaires et la capacité d'apprentissage de chaque agent.

V.3.1. Organisation entre les Agents de Recherche

Le système est hiérarchique : au niveau de l'algorithme maître, c'est la génération de la liste des algorithmes à appliquer. A chaque étape, un nombre d'agents de recherche est sélectionné. La structure de chaque agent peut être assez complexe, mais il est caractérisé par un nombre limité de paramètres. A court terme, les agents se comportent indépendamment, mais à long terme, un mécanisme d'apprentissage peut leur être incorporé pour qu'ils puissent s'interagir et coopérer entre eux.

Le modèle d'apprentissage est basé sur une population de méthodes heuristiques, l'ordre d'application des méthodes peut changer de façon dynamique selon les sorties des applications précédentes. Le but est de trouver la meilleure combinaison de des méthodes dans un environnement à ressource contrainte et par rapport au compromis qualité/coût. On pourrait ainsi développer un système ouvert et adaptatif composé de modules indépendants. De nouvelles composantes peuvent être ajoutées ou les anciennes supprimées sans détruire pour autant la structure de la recherche.

Le plan doit être organisé de façon à pouvoir présenter le travail sous forme d'un "macro-algorithme" dont les composantes sont les algorithmes mis en oeuvres (Figure V. 4).

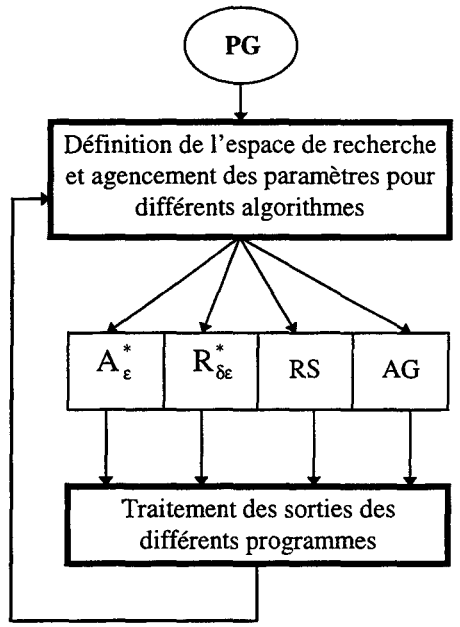


Figure V. 4 : Structure d'un macro-algorithme utilisant des agents de stratégies de recherche.

Le module "*espace de recherche de paramètres*" aura pour but de déterminer les paramètres optimaux que chaque programmes fait appel. L'implantation distribuée peut être envisageable. Les individus peuvent "migrer" d'une méthode à l'autre. Le traitement de sortie sera une sorte de filtre sélectif que le programme-moniteur pourra distribuer/agencer. Cependant, de telle méthode de recherche généralement nécessite une technologie distribuée ou parallèle.

V.4. Références Bibliographiques

[Baluja97] Shumeet Baluja, Scott Davies, "Combining Multiple Optimization Runs with Optimal Dependency Trees", rapport Technique, Carnegie-Mellon University, Juin 1997, CMU-CS-97-157.

[Bilchev96] G. Bilchev, "Evolutionary Metaphors for the Bin Packing Problem", *Proc. of the 5th Annual Conf. On Evolutionary Programming*, San Diego, California, USA, Feb. 29-Mar 2 1996.

[Bäck92] Th. Bäck, "Self-Adaptation in Genetic Algorithms", F.J. Varela and P. Bourgine, editors: *Proceedings of the 1st European Conference on Artificial Life*, pp. 263-271, The MIT Press, Cambridge MA, 1992.

[Collard94] Ph. Collard and J-Ph. Aurand and J. Biondi, "De l'intérêt des sosies dans un système autonome", *Actes des journées de Rochebrune*, Jan. 17-21, 1994.

[Collins91] R.J. Collins, D.R. Jefferson, "AntFarm:Towards Simulated Evolution", in J.D. Farmer, C. Langton, S. Rasmussen, and C. Taylor (Editors), *Artificial Life II*, Addison Wesley, 1991, pp. 579-601.

[Cotta95] C. Cotta, J.F. Aldana, A.J. Nebro, J.M. Troya, "Hybridizing Genetic Algorithms with Branch and Bound Techniques For the Resolution of The TSP", in *Proc. of the Int. Conf. on Artif. N.Net. and GAs*, April 1995, in Alès, France, pp.277-280.

[Ferber94] J. Ferber, *Les Systèmes Multi-Agents*, coll. Informatique, Intelligence Artificielle, InterEditions Paris, 1994

[Grecu96] D. L. Grecu, D. C. Brown, "Dimensions of Learning in Agent-based Design", *AID'96 workshop on Machine Learning in Design*, via Internet *http://cs.wpi.edu/~dgecu*

[Hoffmeister92] F. Hoffmeister, Th. Bäck, "Genetic Self-Learning", F.J. Varela and P. Bourgine, editors: *Proceedings of the 1st European Conference on Artificial Life*, pp. 227-235, The MIT Press, Cambridge MA, 1992.

[Kanada94] Y. Kanada, M. Hirokawa, "Stochastic Problem Solving by Local Computation Based on Self-Organization Paradigm", *Proc. HICSS-27, Emerging Paradigms*, 1994.

[Logan96] Brian Logan, Riccardo Poli, "Route Planning with GA*", Proc. !st On-line Workshop on Soft Computing, Aug 19-30, 1996, `http://www.bioele.nuee.nagoya-u.ac.jp/wscl/`

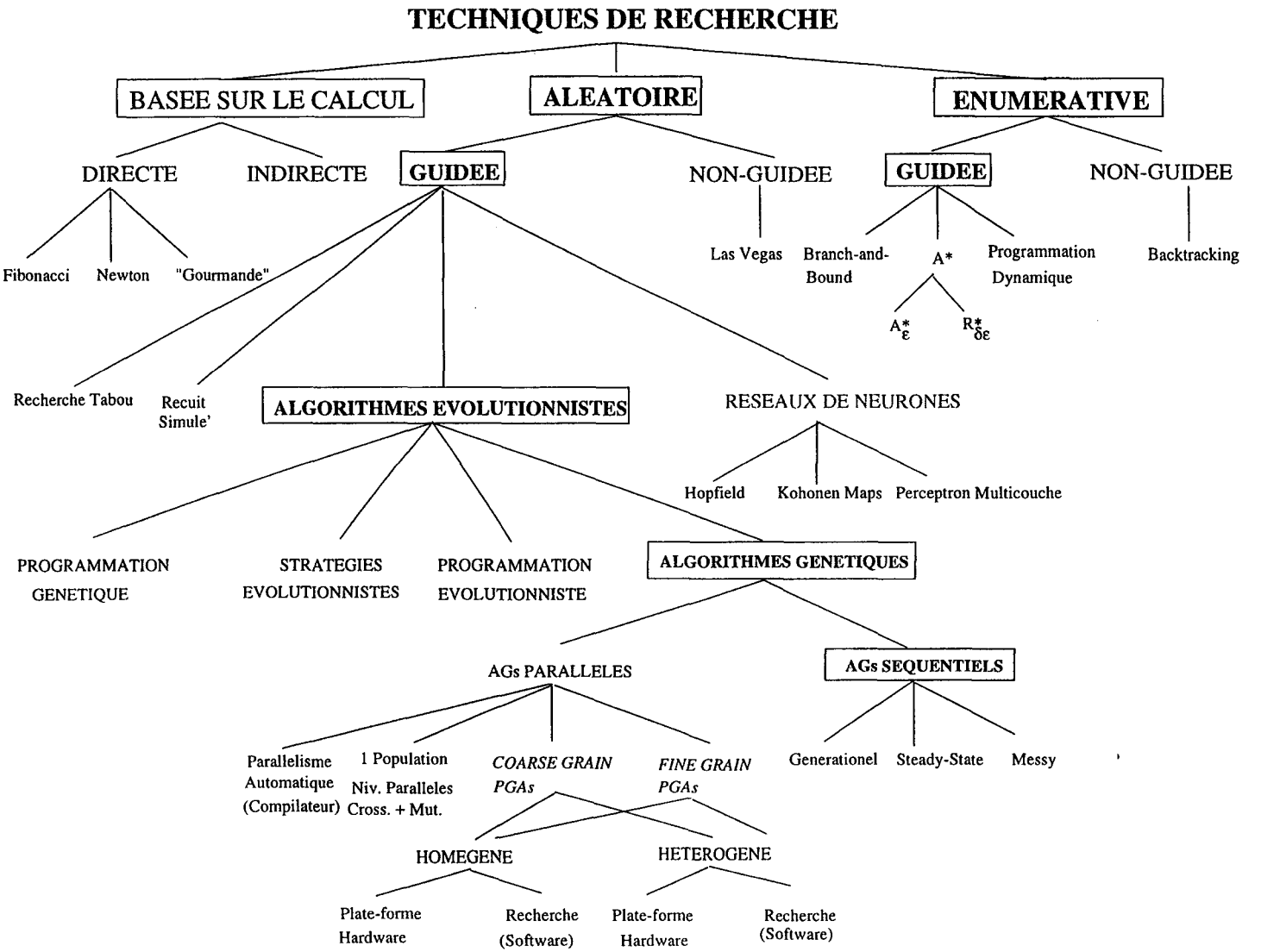
[Maher96] M. L. Maher, J. Poon and S. Boulanger, "Formalising Design Exploration as Co-evolution. A combined Gene Approach", Advances in Formal Design Methods for CAD, John S. Gero and Chapman&Hall ed., 1996.

[Parmee96] I.C. Parmee, "The Maintenance of the Search Diversity for Effective Design Space Decomposition Using Cluster-Oriented Genetic Algorithms

(COGAs) and Multi-Agent Strategies (GAANT)", in *Proceedings of ACEDC'96*, PEDC University of Plymouth, UK.

[Poli96] Riccardo Poli, Brian Logan, "Evolutionary Computation Cookbook: Recipes for Designing New Algorithms", *Proc. 2nd On-line Workshop on Evolutionary Computation (WEC2)*, Mar 11-22, 1996, pp. 33-36, `http://www.bioele.nuee.nagoya-u.ac.jp/wec2/`

[Streltsov96] S. Streltsov, P. Vakili, I. Muchnik, "Competing Intelligent Search Agents in Global Optimization", *Proc. Of NIST Conference "Intelligent Systems : A Semiotic Perspective"*, Aug. 1996.



calssification d'apres The Flying_Circus d'EvoNet, http://www.wi.leidenuniv.nl/~gusz/Flying_Circus/

Annexe 2

STRATEGIES EVOLUTIONNISTES

Développée en Allemagne dans les années 70 par Rechenberg et Schwefel, ces techniques diffèrent des algorithmes génétiques par plusieurs points. Dans cette section, nous allons définir les caractéristiques principales de la stratégie à évolution.

Principe

Les algorithmes SE sont basés sur des variables réelles et normalement distribuées avec l'espérance mathématique nulle. A l'origine, l'algorithme conçu par Rechenberg était un simple algorithme de sélection-mutation sur un seul individu. Cet individu un descendant par mutation et le meilleur des deux parent et enfant est sélectionné de façon déterministe pour devenir parent de la génération suivante. C'est l'algorithme (1+1)-SE, un individu est généré par un seul individu. Mais l'idée d'utiliser une population d'individus a introduit les algorithmes $(\mu+\lambda)$-SE et (μ,λ)-SE.

La population est un ensemble de μ individus qui, par la recombinaison r et la mutation m, génèrent λ nouveaux individus. Ces derniers forment une population intermédiaire P'(t). La sélection se fait parmi les $(\mu+\lambda)$ individus pour la stratégie $(\mu+\lambda)$-SE et parmi les λ nouveaux individus pour l'algorithme (μ,λ)-SE pour obtenir la population P(t+1).

L'algorithme est le suivant :

1°) t = 0
2°) Initialiser P(0) := { $a_1(0)$, ... , a_μ (0)};
3°) a_k = (x_i, c_{ij}) ;avec $c_{ij} = c_{ji}$, $\forall i, j \in \{1, \ldots, n\}$, $j \geq i$.
4°) Evaluer P(0);
5°) Tant que le critère d'arrêt n'est pas satisfait faire :
  5-1°) recombiner $a'_k(t) := r(P(t)) \ \forall k \in \{1, \ldots, \lambda\}$;
  5-2°) muter $a''_k(t) := m(a'_k(t))$;
  5-3°) évaluer $P'(t) := \{a_1''(t), \ldots, a_\lambda''(t)\}$;
    $(\{ f(x_1''(t)), \ldots, f(x_\lambda''(t))\})$;
  5-4°) sélectionner
    si (μ,λ)-SE alors P(t+1) := s(P'(t))
    si $(\mu+\lambda)$-SE alors P(t+1) := $s(P'(t) \cup P(t))$;
6°) t := t+1 ;
Fin

Les points sur lesquels il faut accorder de l'importance sont les caractéristiques de la fonction objectif (espérance, écart-type etc.) et la mutation (liée aux écarts-types).

Les opérateurs

Le passage d'une population P(t)à P(t+1) s'effectue par la sélection. Les individus sélectionnés subissent ensuite la mutation puis la recombinaison. L'écart-type σ_i est soumis lui-même à la mutation avant d'être appliqué au contrôle de la mutation des variables objet x_i.

$\mathbf{m}(a) = a' = (x', \sigma') ; x' \in \mathbf{R}^n, \sigma' \in \mathbf{R}^n$

$\sigma' = \sigma . \exp N_0(\Delta\sigma)$

Le vecteur $N_0(\Delta\sigma)$ est un vecteur de nombres aléatoires indépendants et normalement distribués, avec l'espérance mathématique nulle.

Comme la combinaison de mutation et de sélection est une sorte de *Hill-climbing* , σ_i est similaire à la largeur des pas.

La recombinaison.

La recombinaison créant un individu $a' = (x', \sigma')$ à partir de deux parents $a = (x_a, \sigma_a)$ et $b= (x_b, \sigma_b)$ est représentée par les relations suivantes :

avec $v' = x', \sigma'$

$v_{a,i}$	pas de recombinaison (1)
$v_{a,i}$ ou $v_{b,i}$	recombinaison discrète (2)
$v'_i = \quad 1/2 (v_{a,i} + v_{b,i})$	recombinaison intermédiaire (3)
$v_{ai,i}$ ou $v_{bi,i}$	recombinaison globale discrète (4)
$1/2 (v_{ai,i} + v_{bi,})$	recombinaison globale intermédiaire (5)

De façon empirique, T. Bäck affirme que la recombinaison discrète sur les variables objet et la recombinaison intermédiaire sur les paramètres de stratégie donnent de meilleurs résultats.

Comparaison

La méthode à stratégie d'évolution est applicable pour les problèmes qui peuvent se formuler par une fonction mathématique bien définie statistiquement (i.e. espérance mathématique, écart-type, etc. calculables).

	Algorithmes génétiques	**Algorithmes SE**
code / représentation	Génotype → chaîne binaire (ou non)	Phénotype → valeur réelle
fonction objectif	pas de connaissance préalable sur ses propriétés	connaissance de sa dimension (Nb de variables)
Restrictions	restriction sur l'espace de décodage	pas de restrictions
sélection	proportionnelle au fitness	(μ,λ) ou $(\mu+\lambda)$
opérateur principal	recombinaison ; peu de mutation	mutation
adaptation	pas d'auto-adaptation *collective* des affectations des paramètres	Auto-adaptation *collective* des paramètres de stratégies

Tableau de comparaison [Bäck93]

Références

[Bäck93] Th. Bäck, G. Rudolph, H.-P. Schwefel: Evolutionary Programming and Evolution Strategies: Similarities and Differences , D.B. Fogel and W. Atmar, editors: *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 11-22, Evolutionary Programming Society, San Diego CA 1993.

Annexe 3

PROGRAMMATION GENETIQUE

1. Introduction

La programmation génétique est une extension de l'algorithme génétique dans lequel les structures qui compose la population sont des programmes (ou sous-programmes) qui, une fois exécutés, fournissent des solutions candidats au problème. Chaque programme est exprimé sous forme d'arborescence constituée de codes pris dans un alphabet de codes de programme tels que {FOR, IF, AND OR, WHILE }...) [**Koza92**].

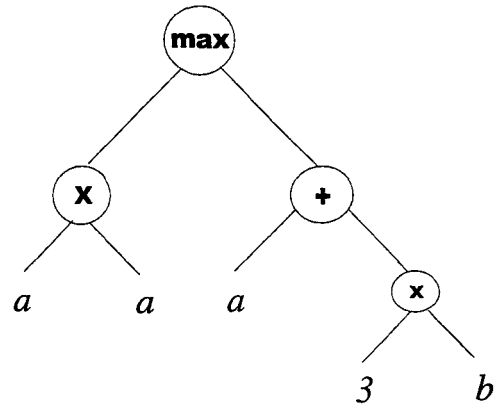


Figure 0-1 : Exemple d'arbre avec **max**($a \times a$, $a + 3 \times b$)

2. Principe

2.1. DÉFINITIONS

- L'ensemble des noeuds internes possibles est appelé l'ensemble de fonctions F = {f_1,..,f_{NF}}, ex : F = {max, ×, +}.
  Les fonctions incluses dans F sont : arithmétique (+, -, ×, /), mathématique (**sin, cos, exp**), booléen (**ET, OU, NON**), conditionnelle (**IF-THEN-ELSE**), récursive (**FOR, REPEAT**) ....
- L'ensemble des noeuds terminaux (feuilles) est appelé l'ensemble des terminaux T = {t_1,.., t_{NT}}, ex : T = {a, b, 3}.

Les terminaux sont des variables, des constantes ou des fonctions 0-arité (**random, aller_à**).

- Toutes les fonctions de F ont une arité (nombre d'arguments $\chi(f_i) \geq 1$
- F et T peuvent se réunir en un groupe uniforme C = F $\cup$ T si les terminaux sont considérés comme des fonction 0-arité. Ex. : C = {**max,** $\times$, +}$\cup${*a*, *b*, 3} = { **max,** $\times$, +, *a*, *b*, 3}
- L'espace de recherche de la PG est un ensemble de combinaisons possibles (et récursives) des fonctions de C.

Pour que la PG fonctionne proprement, F et T doivent posséder deux propriétés : la *fermeture* et la *suffisance* .

2.1.1. PROPRIÉTÉ DE FERMETURE

La fermeture exige que chaque fonction soit capable d'accepter pour arguments toute valeur ou type de données que toute fonction dans C est susceptible de retourner. Ceci évite des erreurs d'exécution.

Exemples d'ensembles fermés :

- C = {**AND, OR, NOT**, *a*, *b*, true} où *a* et *b* sont des variables booléens et true est une constante booléenne.
- C = {+, -,$\times$, *a*, *b*, 1, 0} où *a* et *b* sont des entiers : il n'y a pas de risque de générer des expression non valides.
- C = {+, -,sin, cos, exp , *a*, *b*} où *a* et *b* sont des variables réelles flottantes et les fonctions acceptent des arguments dans (-∞, +∞°).

Exemples d'ensembles non fermés :

- C = {+, -,$\times$, /, *a*, *b*, 1, 0} où *a* et *b* sont des variables réelles flottantes et une division par 0 risque de se produire.
- C = {+, -,sin, cos, log , *a*, *b*} où *a* et *b* sont des variables réelles flottantes et le logarithme d'un argument négatif peut se produire.
- C = {+, $\times$, $\sqrt{}$, *a*} peut être fermé ou non fermé selon le domaine de la variable *a*.

La fermeture peut être forcée en introduisant une contrainte (ex. : $\sqrt{x}$ forcé par $\sqrt{|x|}$).

2.1.2. PROPRIÉTÉ DE SUFFISANCE

- La suffisance est garantie seulement pour certains problèmes quand la théorie (ou une expérience par d'autres méthodes) indique qu'il existe une solution en combinant les éléments de C.
- Si C n'est pas suffisant, alors la PG ne peut que développer des programmes qui fournissent un résultat qui soit proche (le plus possible) de celui souhaité.

Exemple :

Trouver la fonction **exp** (*a*) avec l'ensemble C = {+, -, ×, /, *a*, 0, 1,2}.

La PG serait typiquement :

$\exp(a) \approx 1$

$\exp(a) \approx 1+a$

$\exp(a) \approx 1+a+\frac{1}{2}a^2$

$\exp(a) \approx 1+a+\frac{1}{2}a^2+\frac{1}{1+2+2+1}a^3$

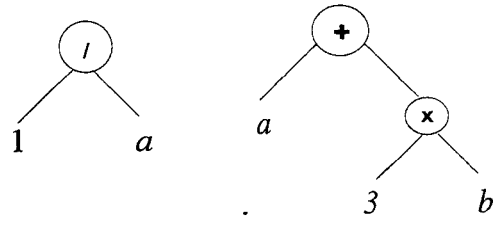
Notons que la qualité de l'approximation dépend de la définition de la fonction fitness ainsi que l'intervalle dans lequel *a* varie.

2.2. POPULATION INITIALE

La population initiale est créée en sélectionnant aléatoirement des fonctions dans C.

La taille et la forme des programmes initiales dépend du choix des noeuds dans F et des feuilles dans T selon la profondeur de l'arbre.

- Les arbres peuvent être représentés par des listes de listes, par conséquent, l'initialisation est basée sur des procédures récursives sur des listes.
- Par exemple, les listes [/ 1 *a*] et [+ *a* [× 3 *b*]] mèneront aux arbres suivants :



1) 2)

- La méthode « heuristique » sélectionne les noeuds dans F tant que leur profondeur est en dessous d'une valeur maximale, ensuite il choisit dans les noeuds feuilles dans T. Ainsi, on s'assure que tous les arbres initiaux ont la même profondeur.

Exemple de population initiale pour C = {+, -, *, /, *y*, *z*, 0, 2, 3} avec la profondeur d=2.

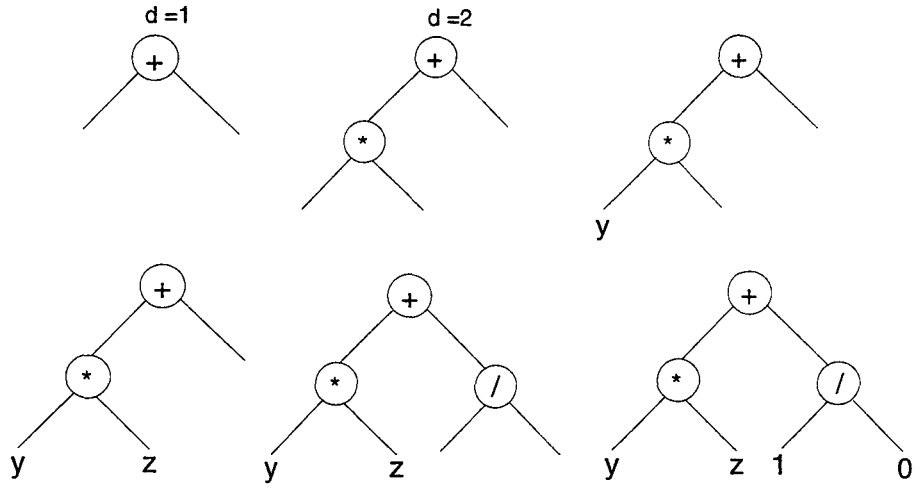


Figure 0-2 : Cette méthode, appelée "Full initialization", choisit les noeuds dans F tant que la profondeur est inférieure à d.

[/ *y z*], [+, [× 3 *y*] 0], [- *z* [/ [+ *y z*] [- 2 *y*]]]

Une autre méthode consiste à sélectionner soit dans C soit dans F tant que la profond :

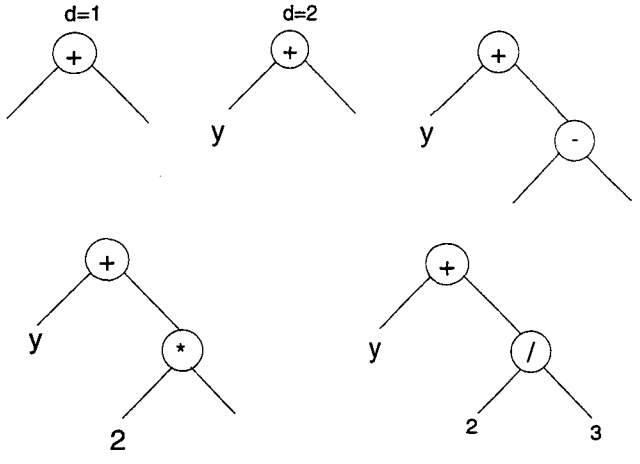


Figure 0-3 : Méthode d'initialisation "grow".

2.3. PRINCIPE DE TRANSFORMATION

Le croisement consiste à sélectionner aléatoirement un point (un lien entre les noeuds) dans chaque arbre parent et échanger les sous-arbres sous ce point.

La mutation consiste à supprimer un sous-arbre et le remplacer par un autre sous-arbre généré aléatoirement.

Les résultats de ces opérateurs peuvent fournir un programme non-valides. Des précautions nécessaires doivent être prises pour préserver le contexte [**D'haeseleer95**].

2.4. INTERPRÉTER LES ARBRES

Certain langages (Lisp) possèdent déjà un interpréteur dans l'environnement du développement. Pour tous les autres langages, il est nécessaire de créer un interpréteur.

Interpréter un arbre implique de le lire vers le bas dans le sens profondeur d'abord. Ensuite évaluer tous les noeuds commençant par les feuilles. L'évaluation en profondeur d'abord évite que les noeuds soient évalués avant de connaître les valeurs de ses arguments. Ainsi, la valeur de la racine est la sortie du programme. Les valeurs des feuilles étant les entrées, il est important de les définir de façon adéquate avant d'invoquer l'interpréteur.

Exemple:

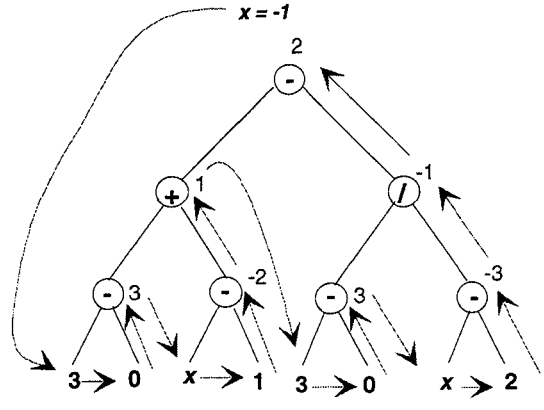


Figure 0-4: Interprétation d'un programme génétique. L'évaluation s'effectue toujours à partir de noeuds feuilles vers la racine.

Remarque:

Pour les instructions qui ne sont pas des fonctions (si <condition>, alors...), l'implantation pose quelque problèmes et pour des boucles, c'est impossible. La solution serait d'utiliser des macro-fonctions. Donc au lieu de recevoir en entrée les valeurs de leurs arguments, ils reçoivent les arguments eux-mêmes.

2.5. FONCTION DE FITNESS EN PG

comme en AG, les techniques d'échelle et de pénalité s'appliquent, mais définir une fonction fitness appropriée est délicat. La différence majeure est de savoir si un programme est bon ou non, ou si on doit l'exécuter une ou plusieurs fois avec différentes données en entrées ou dans différents contextes etc.

Parfois, les sorties du programme sont servies comme fitness et par ailleurs seuls les effets sont tenus compte. Une classe de problème pour laquelle la PG s'avère particulièrement adaptée est la "régression symbolique".

2.6. EXEMPLE D'APPLICATION : "REGRESSION SYMBOLIQUE" [POLI96]

La régression symbolique est une technique souvent utilisée pour tester et interpréter les données expérimentales. Cela consiste à trouver les coefficients d'une fonction préfixée, (souvent simple) de sorte que la fonction résultante corresponde le mieux aux points des données.

Ex. : Trouver l'expression symbolique qui correspond le mieux l'ensemble des points suivants :

$\{ (x_i, y_i) \}$ = { (-1.0, 0.0) (-0.9, 0.1629) (-0.8, -0.2624) (-0.7, -0.3129) ...(1.0, 4.0)},

les points étant générés avec la fonction

$$y = f(x) = x + x^2 + x^3 + x^4 \quad \text{avec } x \in [-1, 1]$$

2.6.1. LES PARAMÈTRES DU PG

Taille de la population :	1000
Ensemble des fonctions :	{+ - * plog pexp sin cos pdiv }
Ensembles des terminaux :	{ x }
Profondeur initiale maximale :	4
Méthode d'initialisation:	"full" (total)
Nombre de générations :	50
Probabilité de croisement :	0.7
Probabilité de mutation :	0
Fitness :	$\sum_i \lvert y_i - \text{eval}(prog, x_i) \rvert$

2.6.2. RÉSULTATS

Génération 1 (fitness = -8.20908)

[+ [- [plog [pexp x]] [+ [sin x] [- x x]]] [+ [pexp [plog x]] [sin [plog x]]]]

...

Génération 6 (fitness = -2.6334)

[* [+ [+ [+ x [pexp [plog x]]] [pdiv x x]] x] [plog x [pexp x]]]

...

Génération 26 (fitness = 0.841868)

[* [+ [+ [+ [* [+ [pexp [plog x]] [sin [plog [+ x x]]]] x]
[pexp [plog x]]] [pdiv x x]] x] [plog [pexp [+ [plog [- [sin
[+ [plog [pexp [plog [- [sin [+ [+ x x] [pdiv [* [+ [+ [+ x
[pexp [plog x]]] [pdiv x x]] x] x] x]] [pexp x]]]]] x]]
[pexp [plog [- [sin [+ [+ [+ x x] [pdiv [* [+ [+ [+ x [pexp [plog x]]]
[pdiv x x]] x] x] x]] x]] [pexp x]]]]]] [plog [- x x]]]]]

Remarque : il est possible d'appliquer la PG aux fonction non-continues ou non dérivables et dans un domaine discret. On peut aussi trouver un circuit logique avec des portes logiques (AND NAND XOR etc.).

2.7.FONCTION DÉFINIE AUTOMATIQUEMENT (FDA)

Pour un problème complexe, il est bon d'utiliser la stratégie "diviser puis conquérir" pour permettre au programme de découvrir et utiliser de bonnes sous routines. Ceci peut être obtenu par :

- l'usage d'un noeud racine spécial avec un nombre fixe de branches
- interpréter certains sous arbres connectés à ces branches en tant que "fonction définies automatiquement", c'est-à-dire sous routines.
- interpréter une des sous routines connectées à la racine en tant que programme principal. La valeur retournée par la branche "programme principal" est prise pour la valeur retournée par la racine elle-même.

Exemple :

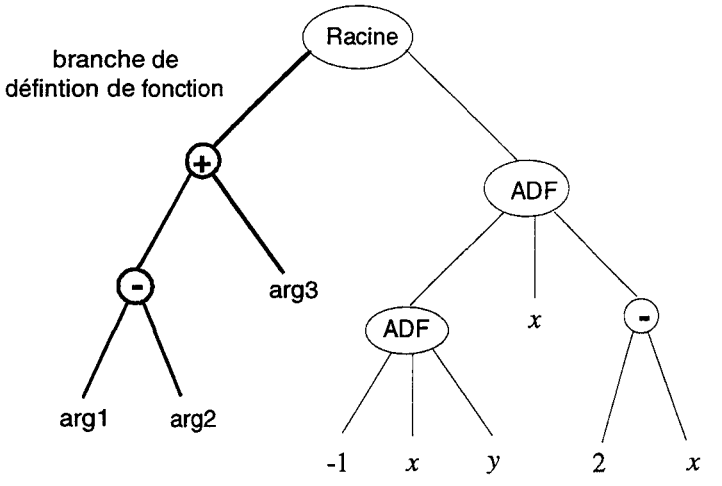


Figure 0-5 : Programme FDA : arg1, arg2 et arg3 sont des variables muettes.

Ce programme a pour expression:

ADF (arg1, arg2, arg3) = arg1 - arg2 - arg3

Racine (x, y) = **ADF** (**ADF** (-1, x, y), x, 2 - x)

2.7.1. QUELQUES RÈGLES POUR UTILISER UNE FDA

L'utilisation de FDA permet de réduire la complexité du programme principal (et donc le temps de calcul). Toutefois, les modifications par rapport à un PG classique doivent être effectuées en observant quelques points suivants:

1. le programme principal ne peut pas avoir d'arguments muets mais peut utiliser des FDA;
2. des FDA ne devraient pas utiliser des variables globales i.e. des variables qui ne sont pas passées en arguments (pour une meilleure "réusabilité" sans d'effets secondaires) à moins que...;
3. éviter les boucles récursives infinies, des FDA ne devraient pas appeler d'autres FDA qui les rappellent à leur tour;
4. des FDA peuvent utiliser avec sûreté des FDA qui sont définies à leur gauche;

5. L'initialisation aléatoire doit générer des arbres avec un noeud spécial racine e(t doit utiliser des terminaux et des fonctions différents pour les noeuds descendants de la racine.
6. L'opérateur de croisement doit être légèrement modifié pour éviter de permuter des sous arbres pris à partir des différentes branches de la racine.
7. La mutation n'a pas besoin d'être modifiée.
8. Le noeud racine doit être implanté en tant que macro qui définit d'abord toutes les fonctions automatiques puis déclenche les évaluation du programme principal.

L'utilisation de FDA permet de réduire la complexité du programme principal (et donc le temps de calcul).

3. REFERENCES BIBLIOGRAPHIQUES

[Koza92] J.R. Koza, *Genetic Programming : On the Programming of Computers by means of Natural Selection*, Cambridge, MA, MIT Press, 1992.

[D'haeseleer95] D'haeseleer Patrick, " Context Preserving Crossover in Genetic Programming", Tech. Report, by pdhaes@cs.stanford.edu, 1995.

[Poli96] Ricardo POLI, Electronic document on Evolutionary Algorithms, Oct. 1996, R.Poli@cs.bham.ac.uk

Annexe 4

OPÉRATEURS GÉNÉTIQUES

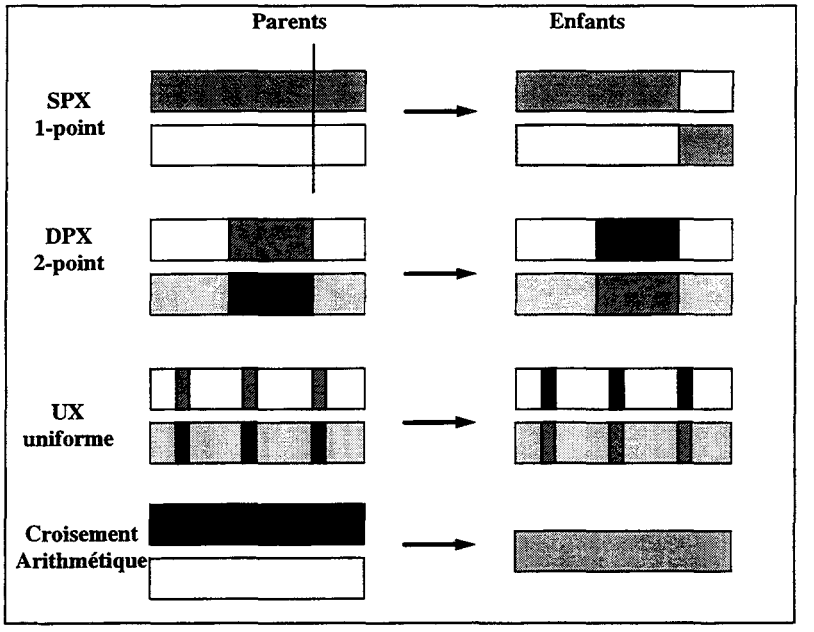


Figure 1 : quatre types de croisement: SPX, le croisement à un point, DPX, le croisement à deux-points de coupure, UPX, le croisement à plusieurs points. Enfin, le croisement arithmétique qui ne produit qu'un seul enfant par recombinaison arithmétique des éléments des deux parents[1].

Des tableaux récapitulatifs sur les propriétés des différents opérateurs existants sont donnés dans [**Goodman94**].

Les modes de sélection pour les algorithmes génétiques

"Sharing scheme" [Golberg89]

On utilise cette technique lorsque l'utilisateur est intéressé à trouver l'ensemble des optima (et non l'optimum global). Le mécanisme de partage est basé sur la notion de "niche écologique", l'ensemble d'individus partageant une même ressource. Ceci se traduit par la modification de la fonction de fitness qui s'écrit: $fi' = \frac{fi}{\sum_{i=1..N} s(d(i,j))}$ avec d(i,j) une mesure de distance entre les individus *i* et *j* et *s(d)* la fonction de partage, décroissante par rapport à *d*.

[1] http://www.wi.leidenuniv.nl/~gusz/Flying_Circus/1.Reading/Tutorial/

Avec cette nouvelle fonction f', on favorise la formation de sous population au niveau des optima locaux, avec d'autant moins d'individus que l'optimum local est peut élevé.

Crowding scheme **[Davis91]**

On ne renouvelle qu'une partie de la population suivant un taux t : le taux de renouvellement. Ainsi, on choisit tN individus pour la recombinaison, les tN enfants produits peuvent remplacer les parents ou remplacer les individus qui leur sont plus proches génétiquement.

Sélection par rang **[Davis91]**

Une autre solution est de ranger les individus par ordre croissant (ou décroissant pour un problème de maximisation) de leur fitness et d'attribuer une probabilité de sélection p_i selon leur rang ("Ranking selection"):

$N.p_i = \Phi - (r_i - 1)*(2\Phi - 2)/(N-1)$ est le nombre moyen d'enfants de l'individu i.

Où r_i est le rang de i, N est le nombre d'individus et Φ est la "pression de sélection" ($\Phi \in [1,2]$, représente le nombre moyen d'enfants du meilleur individu et nécessairement le moins bon en aura $2-\Phi$). Ce type de sélection représente un changement d'échelle dynamique non-linéaire.

Les opérateurs de croisement pour le code en liste

Le code en liste d'éléments se rencontrent surtout dans les problèmes de permutation tels que le problème du voyageur de commerce (TSP), le bin-packing , où chaque élément de l'alphabet n'est utilisé qu'une seule fois dans une même chaîne. Il est délicat d'effectuer le croisement par la méthode classique, alors des opérateurs spéciaux ont été définis pour ce cas spécifique.

Croisement ordonné

Il est fréquent dans les codages non binaires de respecter l'ordre des éléments, il fallait créer des opérateurs qui respectent cette contrainte.

On définit deux sites entre lesquels l'ordre doit être préservé (fenêtre sélectionnée). En dehors de cette zone, on enlève les occurrences des éléments du *tronçon* de l'autre chaîne. A la place de la fenêtre sélectionnée, on remplace par la fenêtre de l'autre chaîne.

Parent1 : 8 7 1 | **2 3 9** | 5 4 6 Enfant1 : 2 3 9 | **5 6 7** | 4 8 1

Parent2 : 9 8 4 | **5 6 7** | 1 3 2 Enfant2 : 5 6 7 | **2 3 9** | 1 8 4

Comme la solution au problème de TSP est un circuit hamiltonien, il est possible de créer un opérateur respectant ce cycle.

Croisement circulaire.

On commence par créer la chaîne enfant1 à partir de parent1. Le premier élément de enfant1 est le premier élément de parent1, les suivants proviennent de la permutation entre les éléments des parent1 et parent2, sans répéter les occurrences. On procède de même pour enfant2.

Exemple :

parent1 : **1** 2 **3** **4** 5 6 7 **8** enfant1 : **1** 6 **3** 4 2 7 5 **8**

↓ ↓ ↓ ↓ ↓

parent2 : **8** 6 **4** 1 2 7 5 **3** enfant2 : **8** 2 **4** **1** 5 6 7 **3**

A noter que si les deux chaînes parentes commencent par le même élément, ce croisement devient un croisement à 1-point. On peut voir aussi dans ce croisement une succession de mutations.

L'enfant1 résulte de parent1 par mutations successives :

1°) permutation entre 6 et 2 : → 1 **6** 3 4 5 **2** 7 8

2°) Mutation RAR de 5 derrière 7 : → 1 6 3 4 2 **7** **5** 8 .

Dans le cas où les éléments de deux chaînes parents sont issus du même alphabet ou du même ensemble d'éléments dans lequel la mutation peut prendre sa valeur (cas du codage binaire, du TSP ...), on pourra définir le croisement comme étant une composition de mutations : C = MoMoMo...oM (du point de vu AG, uniquement !).

Si l'on accepte l'idée que le croisement, en AG, est une composition de mutations, cela signifie que l'évolution ne s'effectue que par sélection et par mutation. Cela relève l'idée de la programmation évolutionniste et de la stratégie à évolution. Avis aux connaisseurs ! ...

Croisement par adjacence ("Edge-map") [Whitley91]

Le croisement par recombinaison d'arêtes ou "edge recombination" consiste à prendre deux chemins et créer une liste de voisinage entre les villes (à partir d'une ville trouver les autres villes accessibles) pour générer un autre chemin (c'est la recombinaison par *edge*).

Par exemple, soient deux chemins ***C1*** *(BACDEF)* et ***C2*** *(FCDBEA)*. Les villes adjacentes à *A* sont : *B* et *C* dans ***C1***, *E* et *F* dans ***C2***, donc son "*edge recombination*" est *BCEF*. De même pour les autres villes.

Les adjacences pour les deux cycles sont :

$A \rightarrow BCEF$ $\quad\quad$ $D \rightarrow CEB$

$B \rightarrow AFDE$ $\quad\quad$ $E \rightarrow DFBA$

$C \rightarrow ADF$ $\quad\quad$ $F \rightarrow EBCA$

On choisit ensuite une des deux villes de départ possibles : *B* ou *F*. Soit *B*, au hasard. Alors *B* est la ville courante, et toutes les occurrences de *B* sont supprimées dans les autres *adjaceneces*. La prochaine ville courante sera choisie parmi les villes présentes dans le *edge map* de *B*. La ville choisie sera celle ayant le plus petit *edge map*. Ici ce sera *D* (car son edge map est *CE*). En cas d'égalité, on choisit au hasard. Ainsi de suite, jusqu'à l'obtention du chemin complet.

Un chemin fils obtenu est ***Cf*** *(BDCFEA)*.

On peut aussi coder le TSP sous forme binaire par l'utilisation de l'adjacence. Les mêmes auteurs proposent le principe suivant :

A partir d'un cycle, on écrit toutes les adjacences possibles. Ensuite, on crée une chaîne binaire de même longueur que le nombre d'adjacences. L'élément à la position i vaut 1 si l'adjacence correspondante est vraie dans le cycle sélectionné.

Par exemple : Villes à parcourir : A B C D E F de l'exemple précédent.

Adjacences possibles : *ab ac ad ae af bc bd be bf cd ce cf de df ef*

C1 (BACDEF) et C2 (FCDBEA) autorisent les adjacences respectives suivantes :

C1 $\rightarrow$ *ab ac cd de ef fb* $\quad$ C2 $\rightarrow$ *fc cd db be ea* $\quad$ Cf $\rightarrow$ bd dc cf fe ea ab

ab ac ad ae af bc bd be bf cd ce cf de df ef

(ABCDEF) : 1 1 0 0 0 0 0 0 1 1 0 0 1 0 1

(BCDEAF) : 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0

Le code binaire permet l'utilisation de la théorie des schémas, mais dans le cas du TSP, la faisabilité d'un cycle n'est pas explicite.

Mutation pour le codage non-binaire

Le code non-binaire type est le code utilisé pour le problème du voyageur de commerce. Comme les éléments de l'alphabet doivent être utilisés une et une seule fois, l'opération de mutation est faite essentiellement de "permutations".

Mutation par Permutation

Choisir deux sites pour cette mutation et permuter les deux éléments qui s'y trouvent:

individu géniteur : (1 2 3 4 5 6 7 8 9) → individu généré : (1 2 3 7 5 6 4 8 9)

Mutation RAR (Remove-And- Reinsert)

Choisir deux éléments, puis enlever le premier et l'insérer derrière le deuxième :

Individu géniteur : (1 2 3 4 5 6 7 8 9) → individu généré : (1 2 3 5 6 7 4 8 9)

Mutation par Inversion

Choisir deux sites, inverser l'ordre des villes entre ces points et permuter les deux villes désignées.

individu géniteur : (1 2 3 4 5 6 7 8 9) → individu généré : (1 2 3 7 6 5 4 8 9)

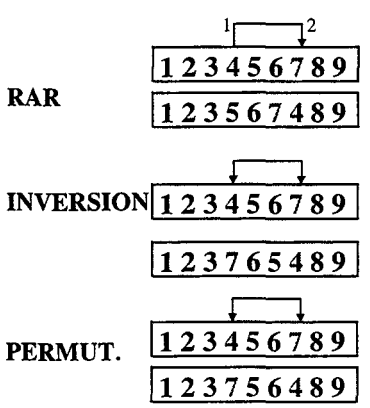


Figure 2 : Mutation d'un code basé sur l'ordre des éléments.

Pour les références bibliographiques : Voir références du **chapitre IV**.

BIBLIOGRAPHIE PERSONNELLE

1. S. Maouche, C. Bounsaythip, **"An Algorithm for a Marking System. e-Admissible Resolution"**, *Proc. of the 12th Int. Conf. On Syst. Science*, Wroclaw, Poland, Sept. 12-15 1995, pp. 345-354.

2. C. Bounsaythip, S. Maouche, **"ε-Admissible Algorithm and Limited Risk Algorithm for Marking System"**, présenté à the. *17th IFIP TC7 Conf. on System Modeling and Optimization*, Prague, Czech Republic, July 10-14, 1995.

3. C. Bounsaythip, S. Maouche, M. Neus **"Evolutionary Search Techniques Application to Automated Lay-Planning Optimization Problem"**, in *Proc. IEEE SMC'95*, Vancouver, Canada, Oct. 22-25, 1995, vol. 5, pp. 4497-4503.

4. S. Maouche, C. Bounsaythip, **"Optimizing Textile Shape Placement by Tree Genetic Annealing"**, in *Proc. of the Society for Computer Simulation Conference (SCSC'96)*, July 21-25, 1996.

5. C. Bounsaythip, S. Maouche, **"A Genetic Approach to the Marker Making Problem in Textile Manufacturing Industry"**, in *Proc. of the 2nd Nordic Workshop on Genetic Algorithms (2NWGA)*, Vaasa, Finland, August 19-23, 1996, Chap. 8, pp. 89-104. Disponible via ftp://ftp.uwasa.fi/cs/2NWGA/Bounsaythip.ps.Z

6. C. Bounsaythip, S. Maouche, **"Irregular Shape Nesting and Placing with Evolutionary Approach"**, article invité pour *IEEE International Conference on Systems, Man, and Cybernetics (SMC'97)*, Hyatt Orlando, Florida (USA), October 12-15, 1997.

7. C. Bounsaythip, J.T. Alander, **"Genetic Algorithms Applied to Image Processing - A Review"**, article invité, *Proc. of the 3rd Nordic Workshop on Genetic Algorithms (3NWGA)*, Helsinki, Finland, August 18-23, 1997, Chap. 14, pp. 172-192. Disponible via ftp://ftp.uwasa.fi/cs/3NWGA/Bounsaythip.ps.Z

8. I. Karanta, T. Mikkola, C. Bounsaythip, M.C. Riff, **"Modeling Timber Collection for Wood Processing Industry. The case of ENSO"**, Rapport de recherche, TTE1-2-98, VTT Information Technology, Information Systems, Octobre 1998.

9. A paraître (article invité): C. Bounsaythip, T. Honkela, **"Combination of Neural and Evolutionary Methods for Data Organization"**, , The 5th International Conference on Foundations of Data Organization (FODO'98), Kobe, Japon, novembre 12-14, 1998.

Errata

UNIVERSITÉ DES SC. ET TECHN.
BIBLIOTHÈQUE UNIVERSITAIRE
LILLE

ch0_intro:

p.5: remplacer $A^*_{\delta\varepsilon}$ par A^*_{ε}

ch1_etat_art:

> **p.20**: remplacer **Erreur! source de renvoi introuvable.** par 6.

ch2_arbre:

> **p.50:** En dessous de l'Equation II.2, lire:

Dans un arbre orienté, si on denote par $Ch(u, v)$ l'ensemble des chemins menant de u à v ; l un chemin quelconque de u à v; et $c(u, v)$ le coût du chemin menant du noeud u au noeud v., alors $g^*(u)$ est le coût minimal d'un chemin de l'état initial à l'état courant :

> **p.56:** dernier paragraphe, 6eme ligne, lire:

Cette information est basée sur l'incertitude sur l'estimation de h^*; elle est représentée par la fonction de densité de probabilité conditionnelle $\rho_{h^*}(x|h)$, qui permet de mesurer la vraisemblance relative que h^* se trouve au voisinage de x sachant l'estimation h **[Pearl82]**, c-à-d, le risque de manquer une opportunité de réduction de coût si on abandonne ce noeud u.

> **p.76:** Dans le graphique Figure II.8.7: deux légendes en trop

> **p. 77:** 4eme paragraphe, 1ere ligne, remplacer "n'est" par "ne sont"

> **p. 78:** enlever "**Erreur!source de renvoi introuvable.**"

> **p. 79:** remplacer "**Erreur!source de renvoi introuvable.** " par II.9.1

ch3_sa:

> **p.101**: remplacer section 3.2 par section III.3.2

> **p.103**: remplacer Equation 3.1.1 par équation III.3

> **p.105**: remplacer M par MT dans l'équation

> remplacer Equation 3.1.2 par Equation III.4

> **p.113** en bas de la page: remplacer g=1, par γ=1

> **p. 115**: Légende de Figure III.7: Evolution du coût par palier de température.