# Algorithmes évolutionnaires et résolution de problèmes de satisfaction de contraintes en domaines finis

Blaise Madeline

UNIVERSITÉ de NICE-SOPHIA ANTIPOLIS – UFR SCIENCES

École Doctorale STIC
Département d'Informatique

# THÈSE

Présentée pour obtenir le titre de

**Docteur en SCIENCES**
**de l'Université de Nice-Sophia Antipolis**

Spécialité : **INFORMATIQUE**

par

**Blaise Madeline**

Sous la direction de **Bertrand Neveu**

# Algorithmes évolutionnaires et résolution de problèmes de satisfaction de contraintes en domaines finis

Soutenue publiquement le 18 décembre 2002 devant le jury composé de :

| | | | | |
|---|---|---|---|---|
| M. | **Bertrand** | **NEVEU** | Ingénieur en Chef ENPC | (Directeur) |
| M. | **Jin Kao** | **HAO** | Professeur | (Rapporteur) |
| M. | **Gérard** | **VERFAILLIE** | Ingénieur de Recherche | (Rapporteur) |
| M. | **Philippe** | **COLLARD** | Professeur | (Président) |
| M. | **Philippe** | **JÉGOU** | Professeur | (Examinateur) |
| M. | **Marc** | **SCHOENAUER** | Directeur de Recherche | (Examinateur) |

# Remerciements

Je tiens particulièrement à remercier les personnes suivantes :

- Bertrand Neveu, qui m'a accueilli au sein du projet contraintes du CERMICS et que j'ai ensuite suivi lors de la formation du projet commun COPRIN. Pour la liberté qu'il m'a laissée, pour nos nombreuses discussions des plus enrichissantes, pour sa grande disponibilité, pour son honnêteté scientifique, je ne saurais trop le remercier sans que sa modestie n'en souffre.
- Jin Kao Hao et Gérard Verfaillie pour l'honneur qu'ils m'ont fait en rapportant cette thèse et pour avoir participé au jury. Leur remarques constructives sur le manuscrit m'ont permis de clarifier certains points.
- Philippe Jegou et Marc Schoenauer pour l'honneur qu'ils m'ont fait en acceptant de faire partie du jury.
- Philippe Collard pour l'honneur qu'il m'a fait en acceptant de faire partie du jury, pour m'avoir initié aux algorithmes génétiques et pour m'avoir présenté à Bertrand Neveu.
- Fabrice Didierjean pour sa bibliothèque AgCSP, ses conseils et nos nombreuses discussions.
- Les membres de l'équipe COPRIN, Jean-Pierre Merlet, Michel Rueher, Gilles Trombettoni, Yves Papegay, Claude Michel, Christophe Jermann, Heikel Batnini, ...
- Les personnels du CERMICS, de l'INRIA et de l'UNSA, particulièrement Corinne, Patricia, Sabine et Christian.
- On se souvient tous de certains enseignants du primaire ou du secondaire pour l'envie d'apprendre qu'il nous ont inculquée : qu'il me soit permis de remercier ici Mme Irlinger, Mme Durand-Delga et M. Brosse.
- Emmanuel Kounalis, qui dès la Licence m'a poussé vers la recherche, m'a donné de judicieux conseils, et a encadré mon monitorat.
- Mon père, ma mère, Yann pour tellement de choses... Ma soeur et mes frères : Chloé, Julien, Florian (et sa sempiternelle question «alors ? tu as trouvé ?») et Thibault. Ma famille en général.
- Mes amis niçois : David, Hervé, Pascal, Céline, Gilles, Caroline, Jérôme, Caroline, Guillaume, Mélinda, Jérémie, Christelle, Bruno, Sandrine, Nath, Raphaelle, Stfb, Pédro...
- Mes amis Bellifontains : Bruno, Marc, Ingrid, Romain, Catherine, Nicolas, Stéphanie et Lionel, qui malgré l'éloignement ont toujours été là.
- Isa, qui m'a soutenu, supporté, aimé, cajolé, secoué, ... et qui est toujours à mes côtés.
- Ainsi que tous ceux que je n'ai pas nommément cités, mais que je n'oublie pas.

# Table des matières

# Introduction

Les problèmes de satisfaction de contraintes en domaines finis sont très largement étudiés dans de nombreux domaines d'application de l'informatique, comme par exemple, les problèmes d'emplois du temps, d'ordonnancement de tâches, d'allocation de fréquences, de coloriage de graphe. Comme beaucoup de problèmes intéressants, les CSP sont généralement NP-complets. Il n'existe donc pas de méthode générale pour les résoudre efficacement. Cependant, beaucoup de méthodes sont disponibles pour tenter de les résoudre. Les méthodes de recherche arborescente complète, les méthodes de recherche locale (par exemple la recherche avec liste taboue) et les algorithmes génétiques (AG) en font partie. Le but de nos travaux est de montrer que pour certains problèmes de coloriage de graphes, notamment sur-contraints, les algorithmes génétiques peuvent être d'un intérêt majeur sans spécialisation particulière.

Les algorithmes génétiques sont sujets à un certain nombre de critiques récurrentes, lesquelles ne sont pas forcement toutes justifiées :

1. « Je ne suis pas sûr d'obtenir la solution optimale»
2. «J'ai essayé en presse-bouton, et c'est nettement plus mauvais que mon algorithme»
3. «On est obligé d'intégrer toute la connaissance du problème pour avoir des résultats»
4. «Les paramètres nécessitent une période de réglage longue et fastidieuse»

La première affirmation est fondamentalement vraie. En effet, comme tous les algorithmes non systématiques, rien n'assure que la solution optimale puisse être trouvée. La seule chose que l'on puisse faire est d'expérimenter, et de donner une réponse statistique sur un ensemble de problèmes types.

La deuxième affirmation est nettement plus sujette à caution. Lorsque un algorithme efficace existe pour un problème donné, il est évident qu'un algorithme génétique a peu de chance d'être concurrentiel, surtout utilisé tel quel, sans spécialisation particulière. Mais si on se base sur un protocole de comparaison équitable, nous verrons qu'il peut en être autrement (chapitre 5). Nous verrons aussi que les troisième et quatrième affirmations peuvent être remises en question. L'idée est de comparer différents algorithmes, dont les

algorithmes génétiques, sur un problème particulier, de telle sorte qu'aucun des algorithmes ne subit de réglages ou spécialisations importants sinon ceux d'usages.

En ce qui concerne les deux dernières affirmations, il est vrai qu'un algorithme génétique doit intégrer une certaine connaissance du problème pour être efficace, et qu'il faut en général, passer un certain temps pour déterminer le meilleur réglage. Cependant, nous verrons dans le chapitre 6 que des opérateurs n'intégrant pas de connaissance particulière peuvent se révéler plus efficaces que les opérateurs classiques, tout en nécessitant des réglages moins fins.

La motivation principale de cette thèse était de montrer que ces idées reçues, bien que non dénuées de fondements, pouvait se révéler fausses dans certains cas. Nous voulions particulièrement trouver des solutions pour limiter le réglage de paramètres, soit en proposant un cadre où ce réglage n'est pas nécessaire, soit en proposant des opérateurs dont le réglage est moins fastidieux. L'idée est aussi de rester au plus proche de l'algorithme génétique standard, c'est à dire celui qui permettrait à un novice une implantation simple et rapide.

Cette thèse se divise en trois parties. La première partie présentera d'une part le cadre formel des CSP, ainsi que quelques exemples (chapitre 1), et d'autre part un tour d'horizon des différentes méthodes disponibles pour les résoudre (chapitre 2).

La deuxième partie traite des algorithmes évolutionnaires, en présentant tout d'abord la version canonique des algorithmes génétiques, puis différentes variantes que l'on peut trouver dans la littérature (chapitre 3) et enfin l'application de ces algorithmes aux problèmes de satisfaction de contraintes en domaines finis (chapitre 4).

La troisième partie présente nos contributions. Nous allons tout d'abord proposer un cadre de comparaison des algorithmes évolutionnaires, de la recherche arborescente et de la recherche locale qui soit équitable, et dans une optique de réglage minimal des paramètres quelle que soit la méthode (chapitre 5). Puis nous allons proposer des opérateurs génétiques nouveaux qui, bien que développés et testés dans le cadre des problèmes de satisfaction de contraintes en domaines finis, peuvent être efficaces dans d'autres domaines d'applications des algorithmes évolutionnaires (chapitre 6).

# Première partie

# Les problèmes de satisfaction de contraintes

# Chapitre 1

# Problèmes de satisfaction de contraintes en domaines finis

Ce chapitre s'attache à présenter brièvement le modèle utilisé pour les problèmes de satisfaction de contraintes(CSP[1]). Après une brève présentation, nous introduirons un certain nombre de concepts, notations et définitions utilisés dans la suite de la thèse, puis quelques notions de complexité. Enfin, nous présenterons quelques exemples types, ainsi que leurs formulation en CSP. Le lecteur pourra trouver une information plus complète concernant les CSP dans [AS94] par exemple.

## 1.1 Présentation

Les problèmes de satisfaction de contraintes permettent de modéliser un grand nombre de problèmes de la vie courante tels que :

- les problèmes d'emploi du temps
- les problèmes d'ordonnancement de tâches
- les problèmes d'affectation de fréquences
- les problèmes de rangement d'objets
- ...

Ce modèle a été très largement utilisé ces dernières décennies, et son efficacité n'est plus à démontrer. La formulation CSP n'a pas changé la complexité des problèmes, et savoir si un CSP admet une solution reste cependant un problème NP-complet, mais permet généralement une formulation plus concise.

[1] Constraint Satisfaction Problem

## 1.2 Concepts et Notations

Un problème de satisfaction de contraintes (CSP) est constitué d'un ensemble de variables dont les valeurs sont issues d'un ensemble de valeurs appelé domaine, et dont l'affectation est soumise à certaines conditions appelées contraintes. Plus formellement nous avons :

**Définition 1.2.1** *(CSP) CSP en domaines finis*
*un CSP P est défini par un triplet* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *avec :*

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ *un ensemble fini de n variables*
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ *un ensemble de domaines où* $D_i$ *est l'ensemble fini des valeurs possibles pour la variable* $x_i$
- $\mathcal{C} = \{C_1, \ldots, C_m\}$ *un ensemble fini de contraintes. Chaque contrainte* $C_i$ *est définie par :*
  - *L'ensemble* $\mathcal{X}_i = \{x_{i1}, \ldots, x_{ik}\} \subset \mathcal{X}$ *des k variables impliquées dans la contrainte* $C_i$
  - *L'ensemble des k-uplets, sous ensemble du produit cartésien* $D_{i1} \times \ldots \times D_{ik}$*, des valeurs qui peuvent être affectées aux variables impliquées dans la contraintes* $C_i$*.*

On appelle arité d'une contrainte le nombre de variables impliquées dans cette contrainte. Un CSP ne contenant que des contraintes d'arité deux est appelé *CSP binaire*. A partir de maintenant, sauf indication contraire, on entendra par CSP un CSP binaire. On notera

- $n = |\mathcal{X}|$ le nombre de variables.
- $d = max_{i \in 1..n}(|Di|)$ la taille du plus grand domaine.
- $m = |\mathcal{C}|$ le nombre de contraintes.

**Définition 1.2.2** *(Test de cohérence)*
*On appelle test de cohérence le fait de tester si un ensemble de valeurs est autorisé par une contrainte.*

**Définition 1.2.3** *(sous-domaine)*
$\mathcal{D}' = \{D'_1, \ldots, D'_n\}$ *est un sous-domaine de* $\mathcal{D} = \{D_1, \ldots, D_n\}$ *si et seulement si* $\forall i \in [1..n], D'_i \subseteq D_i$*.*

**Définition 1.2.4** *(voisinage)*
*Dans un CSP, le voisinage d'une variable* $x_i$*, noté* $\Gamma(x_i)$*, est l'ensemble des variables* $x_j$ *telles qu'il existe une contrainte entre* $x_i$ *et* $x_j$*.*

**Définition 1.2.5** *(degré)*
*Le degré d'une variable $x_i$ est la taille $|\Gamma(x_i)|$ de son voisinage.*

**Définition 1.2.6** *(Graphe de contraintes)*
*A chaque CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est associé un graphe des contraintes (ou graphe associé) obtenu en représentant chaque variable du réseau par un sommet et chaque contrainte $C_{ij} \in \mathcal{C}$ par une arête entre $x_i$ et $x_j$.*

**Définition 1.2.7** *(densité)*
*Un CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est complet s'il comporte le nombre maximum (e) de contraintes ($e = \frac{n(n-1)}{2}$). La densité de $P$ est le rapport $\frac{2e}{n(n-1)}$ entre le nombre de contraintes dans $\mathcal{C}$ et e le nombre maximum de contraintes.*

**Définition 1.2.8** *(Instanciation)*
*Soit un CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, on appelle instanciation $\mathcal{I}$ une application qui associe à chaque variable $x_i \in \mathcal{X}$ une valeur $\mathcal{I}(x_i) \in D_i$.*

On utilisera indifféremment le terme *instanciation* et le terme *solution potentielle*. Cependant le terme *instanciation* sera préféré lorsqu'on utilisera des méthodes systématiques, et le terme *solution potentielle* lorsqu'on utilisera des méthodes locales.

**Définition 1.2.9** *(Instanciation partielle)*
*Une instanciation partielle $\mathcal{I}_S$ d'un ensemble de variables $S = \{x_{i1}, \ldots, x_{ik}\}$ affecte à chaque variable de S une valeur de son domaine. En d'autres termes, une instanciation de S correspond à un k-uplet de $D_{i1} \times \ldots \times D_{ik}$. On notera $\mathcal{I}_S[x_i]$ la valeur que $\mathcal{I}_S$ affecte à $x_i$.*

**Définition 1.2.10** *(Instanciation partielle localement cohérente)*
*Une instanciation partielle d'un ensemble de variables S est localement cohérente si et seulement si elle satisfait toutes les contraintes portant uniquement sur les variables de S.*

**Définition 1.2.11** *(Compatibilité)*
*Une valeur $(x_i, v_i)$ est dite compatible avec une instanciation localement cohérente $\mathcal{I}$ de S si l'instanciation $\mathcal{I}_0$ de $S \cup \{x_i\}$ obtenue en étendant $\mathcal{I}$ par affectation de $v_i$ à $x_i$ est localement cohérente.*

**Définition 1.2.12** *(Solution)*
*Une solution d'un CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est une instanciation de $\mathcal{X}$ localement cohérente.*

**Définition 1.2.13** *(instanciation partielle globalement cohérente)*
*Une instanciation partielle est globalement cohérente [Dec92] si et seulement si elle peut-être étendue à une solution.*

**Définition 1.2.14** *(Satisfiabilité)*
*Un CSP est satisfiable si et seulement si il admet au moins une solution. Il est dit insatisfiable dans le cas contraire.*

On peut être amené à traiter des CSP insatisfiables, on parle alors de CSP sur-contraints. On a alors définie le cadre MaxCSP. Dans ce cas, le but est de trouver une solution qui satisfasse le maximum de contraintes. Ces problèmes sont réputés difficiles car les méthodes de filtrage sont peu efficaces.

**Définition 1.2.15** *(Équivalence de CSP)*
*Deux CSP ayant le même ensemble de variables sont dits équivalents si et seulement si ils possèdent le même ensemble de solutions.*

**Définition 1.2.16** *(Espace de recherche)*
*L'espace de recherche d'un CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ est l'ensemble des instanciations possibles de $\mathcal{X}$ , c'est-à-dire $D_1 \times D_2 \times \ldots \times D_n$.*

## 1.3 Complexité

Nous ne nous étendrons pas sur l'étude des classes de complexité car elle n'entre pas dans le cadre de cette thèse. Dans leur livre «Intelligence Artificielle & Informatique Théorique» [AS94] J.-M. Alliot et T. Schiex y consacrent un chapitre entier et le lecteur intéressé pourra s'y reporter. Nous nous devons cependant d'en évoquer les principes de base. La théorie de la complexité permet de classer les problèmes en problèmes faciles, problèmes difficiles et problèmes infaisables. On appelle problèmes faciles ceux pour lesquels il existe un algorithme efficace en temps polynomial ($O(n^k)$, $k$ étant constant). On dit alors que ces problèmes font partie de la classe $P$. On appelle problèmes difficiles ceux pour lesquels il existe un algorithme efficace qpour vérifier de manière déterministe qu'une instance du problème est valide (ou invalide) en temps polynomial. On dit alors que ces problèmes font partie de la classe $NP$.

Nous nous intéresserons, bien sûr, qu'aux problèmes traitables informatiquement[2], et plus particulièrement aux problèmes de la classe $NP$. Ces problèmes ont en général un

[2]Par exemple un problème dont une instance nécessite de mémoriser un nombre de bits supérieur au nombre d'atomes dans l'univers est intraitable.

nombre de solutions exponentiel en fonction du nombre de variables, que l'on pourra vérifier chacune en utilisant un algorithme polynomial. Ceci n'est pas satisfaisant pour autant : prenons l'exemple d'un problème de taille 100 (variables booléennes) et dont le nombre de solutions potentielles est exponentiel ($2^{100}$ par exemple). Si générer une solution potentielle prend une nanoseconde, les générer toutes prendra $3 \times 10^{11}$ siècles. D'où l'intérêt de trouver des techniques permettant de réduire le nombre de solutions à tester. Ces techniques sont appelées heuristiques, car elles intègrent une connaissance du problème afin d'éliminer les solutions manifestement mauvaises. Nous verrons dans les chapitres suivants un certain nombre de ces heuristiques.

## 1.4 Exemples de CSP

Comme nous l'avons dit au début de ce chapitre, de très nombreux problèmes peuvent se modéliser sous forme de CSP. Nous allons présenter ici quelques problèmes classiques, soit issus de problèmes réels, soit de jeux (*puzzles* en anglais).

### Placement de reines

Il s'agit d'un jeu sous forme d'énigme : comment placer huit reines sur un échiquier de telle sorte qu'elles ne soient pas en prise les unes par rapport aux autres. On ne peut donc placer qu'une reine par ligne, colonne et diagonale. Une version plus générale du problème propose de placer $n$ reines sur un échiquier de taille $n \times n$ La formulation en CSP est assez triviale :

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ avec $x_i$ le numéro de la colonne où se place la reine de la ligne[3] $i$.
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ avec $\forall i, D_i = \{1, \ldots, n\}$.
- $\mathcal{C}$, l'ensemble des contraintes tel que $\forall i, \forall j : x_i \neq x_j, x_i + i \neq x_j + j, x_i + j \neq x_j + i$.

Ce problème dont une solution pour $n = 8$ est présenté figure 1.1, s'est révélé être un problème facile. Minton et al. ont obtenu un résultat en un temps raisonnable pour $n = 1000000$ en utilisant leur heuristique min-conflit[MJPL92]. Une variante de ce problème consiste à colorier un échiquier de telle sorte que deux cases ne peuvent être de la même couleur si elles se trouvent sur le trajet d'une reine. Ce problème revient à placer $n$ ensemble de $n$ reines sur un échiquier de telle sorte qu'aucun ne se chevauche. Nous retrouverons ce problème dans le chapitre 5.

[3] On élimine d'avance toutes les solutions ayant plusieurs reines sur une même ligne

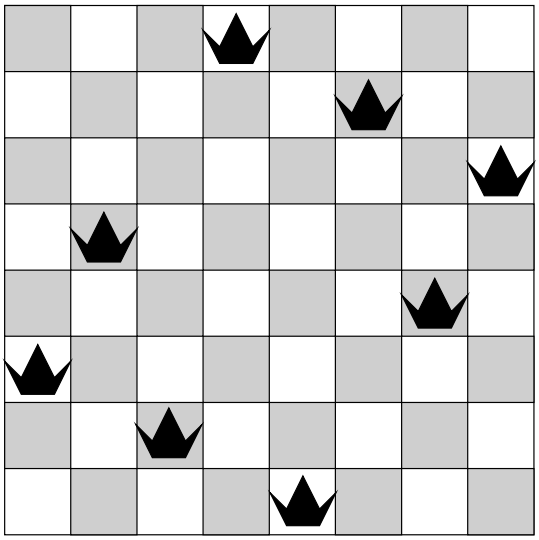

FIG. 1.1 – Une solution pour le problème des 8-reines.

### La rencontre d'équipages

Le problème de rencontre d'équipages (*Progressive Party Problem*) est un problème très difficile, qui consiste à planifier la rencontre de différents équipages en respectant un certain nombre de contraintes. Ce problème a été proposé par un *yacht club*, il s'agissait d'organiser des rencontres entre les équipages sur un certain nombre de périodes de temps. Une partie des bateaux est composé de bateaux hôtes, et ceux-ci vont recevoir tour à tour les équipages invités. Pour chaque période de temps, chaque équipage invité doit rendre visite à l'un des bateaux hôtes, en respectant les contraintes suivantes.

- Un équipage ne peut rencontrer plus d'une fois un autre équipage.
- Un équipage ne peut aller deux fois sur le même bateau hôte.
- Un équipage doit être invité à chaque période de temps.
- Un bateau hôte ne peut recevoir plus de personnes que sa capacité.

Les approches par programmation linéaire en nombre entier se sont révélées inefficaces par rapport à l'approche CSP [SBHW95]. L'avantage de la formulation CSP est qu'elle est assez compacte comparée à la formulation en programme linéaire.

Appelons $I$ le nombre d'équipages invités, $H$ le nombre de bateaux hôtes et $T$ le nombre de périodes de temps. Soit $h_{it}$ les variables qui représentent le bateau hôte qui a reçu l'équipage $i$ pendant la période de temps $t$, en respectant la contrainte : $\forall i$, les variables $h_{i1}, \ldots, h_{iT}$ sont toutes différentes. Les contraintes de capacité sont données par des variables binaires $v_{ijt}$, avec $v_{ijt}$=1 si et seulement si l'équipage invité $i$ a été reçu par l'hôte $j$ pendant la période de temps $t$. Les relations entre ces variables et les variables $h_{it}$ sont

définies par la contrainte : $\forall i, \forall j, t$ on a $v_{ijt} = 1$ si et seulement si $h_{it} = j$. La contrainte de capacité est définie comme suit :

$$\forall j, \forall t \sum_i c_i v_{ijt} \leq C_j \tag{1.1}$$

où $c_i$ est la taille de l'équipage $i$ et $C_j$ est la capacité du bateau hôte $j$.

Il nous reste à modéliser la contrainte qui dit que deux équipages ne peuvent se rencontrer deux fois. On fera cela en introduisant de nouvelles variables binaires : $r_{klt} = 1$ si et seulement si les équipages $k$ et $l$ se sont rencontrés au temps $t$. Nous avons la relation suivante : $\forall k, \forall l, \forall t; k < l$ ; si $h_{kt} = h_{lt}$ alors $r_{klt} = 1$. On peut exprimer la contrainte de rencontre par :

$$\forall k, \forall l; k < l; \sum_t r_{klt} \leq 1 \tag{1.2}$$

### Coloriage de graphes

Un problème de coloriage de graphe peut se formuler, si on se place du point de vue théorie des graphes, de la façon suivante : étant donné un graphe $G = (V, E)$ et un entier $k$, trouver une partition de $V$ en $k$ classes $C_c$ tel que $\forall i \in C_c, \forall j \in C_c$, $(i, j) \notin E$. On dit alors qu'un tel graphe est k-coloriable. Un problème de coloriage de graphe peut aussi se formuler sous forme de CSP. Il est défini comme suit :

- $\mathcal{X} = \{x_1, \ldots, x_n\}$ un ensemble de variables représentant les nœuds à colorier
- $\mathcal{C}$ un ensemble de $m$ contraintes sur ces variables, qui sont uniquement des contraintes de différence.
- $\mathcal{D} = \{D_1, \ldots, D_n\}$ Le domaine des variables avec $D_1 = \ldots = D_n = \{1, \ldots, k\}$, $k$ étant le nombre de couleurs utilisées.

Chaque contrainte $C_j$ relie deux variables qui ne peuvent donc être de la même couleur. Une solution au coloriage consiste en une instanciation de toutes les variables qui respecte simultanément toutes les contraintes.

Dans un problème de coloriage de graphe, on peut chercher à résoudre soit le problème de décision : trouver un coloriage en $k$ couleurs, soit le problème d'optimisation : quel est le plus petit nombre de couleurs $k$, qui satisfasse toutes les contraintes.

Un problème très connu de coloriage de graphe consiste à colorier une carte de telle sorte que deux pays partageant une frontière ne soient pas de la même couleur. Nous présentons dans la figure 1.2 un exemple de coloriage de carte avec le coloriage de graphe associé.

La plupart des CSP binaires peuvent être facilement réduits à un problème de coloriage de graphe, d'où l'intérêt de ce type de problème.

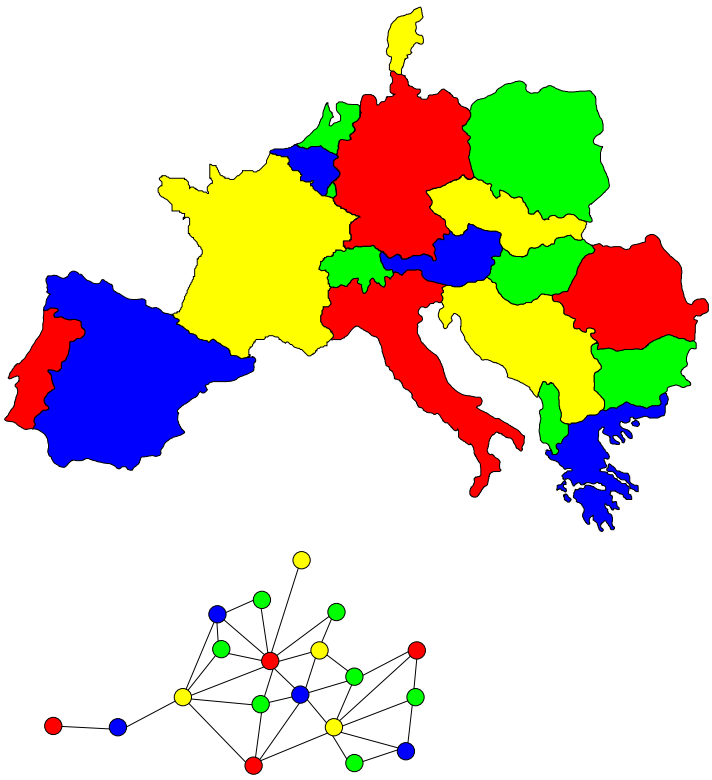

FIG. 1.2 – Carte de l'Europe et le graphe de coloriage associé.

## 1.5 Conclusion

Nous avons vu dans ce chapitre que la formulation CSP permettait de modéliser un grand nombre de problèmes réels. De plus, cette formulation est souvent assez intuitive et concise. Bien que les problèmes traités restent des problèmes difficiles, cette modélisation a permis l'émergence de nouvelles méthodes qui se sont révélées efficaces sur nombre de problèmes. Le chapitre suivant se propose de présenter certaines de ces méthodes.

# Chapitre 2

# Méthodes de résolution

Il existe de très nombreuses méthodes de résolution pour les problèmes d'optimisation combinatoire et pour les CSP en particulier. De nombreuses classifications de ces méthodes ont été proposées dans la littérature : méthodes complètes et incomplètes, méthodes issues de la recherche opérationnelle et méthodes issues de l'intelligence artificielle ... Nous ne verrons pas ici toutes les méthodes existantes ni toutes les classifications proposées. Nous pouvons cependant distinguer trois grandes familles d'algorithmes pour résoudre des problèmes d'optimisation combinatoire qui vont utiliser des approches fondamentalement différentes. Dans la première section, nous présenterons l'approche constructive : on construit une solution par instanciation successive des variables. Cette approche regroupe la plupart des méthodes systématiques. Ces méthodes garantissent, en un temps fini, soit d'obtenir une solution, soit qu'aucune solution n'existe. Cependant, ces méthodes se révèlent relativement peu efficaces sur des problèmes de grande taille. On trouve aussi quelques méthodes constructives non déterministes. Dans la deuxième section, nous présenterons des algorithmes qui utilisent l'approche locale : on opère par réparation locale d'une instanciation complète globalement incohérente. Dans la troisième section, nous présenterons l'approche par algorithme à population, qui fait évoluer une population de solutions potentielles. Ces deux dernières approches font partie des algorithmes dits non-systématiques (aussi appelés méta-heuristiques), donc incomplets, c'est à dire que rien ne garantit de trouver la solution optimale en un temps fini. Ces algorithmes sont souvent associés à un processus stochastique, pour éviter qu'un mauvais choix de départ ou des biais systématique ne les pénalise. Ce chapitre présente différentes méthodes issues de ces trois approches, puis conclut en présentant une quatrième approche qui consiste à combiner ensemble différentes approches. On parle alors d'approche hybride.

## 2.1 Méthodes constructives

Dans cette section, nous présentons les différentes méthodes qui procèdent par construction itérative de la solution. La première partie présente l'algorithme *Backtrack*, la deuxième partie présente différentes heuristiques de choix de variables et de choix de valeurs. Dans les troisième et quatrième parties, nous nous intéresserons aux méthodes rétrospective et prospective. La cinquième partie présente différents parcours d'arbre, et enfin, la sixième partie évoque rapidement les CSP sur-contraints.

### 2.1.1 L'algorithme *Backtrack*

La méthode de base de la plupart des méthodes constructives est le retour-arrière [GV89](*Backtrack* en anglais). L'arbre de recherche est généralement exploré en utilisant une recherche arborescente en profondeur d'abord (DFS[1]) pour des raisons de complexité en espace. Cette méthode explore successivement tout l'espace de recherche. Cela consiste à construire la solution en instanciant successivement les variables du problème en utilisant un ordre défini au départ[2]. A chaque nouvelle affectation on teste si l'instanciation partielle est localement cohérente. Si c'est le cas, on continue (en instanciant une nouvelle variable), sinon on essaye avec une autre valeur du domaine de la variable courante. Si toutes les valeurs du domaine ont été essayées, on procède à un retour-arrière vers la dernière variable instanciée.

Une manière formelle de représenter le parcours de l'espace de recherche est d'utiliser un arbre de recherche dont chacun des nœuds représente la valeur d'une variable. L' affectation de la $i^{ième}$ variable de l'instanciation partielle a donc lieu à la profondeur $i$ de l'arbre. Le problème est que *Backtrack* ne filtrant pas les domaines des variables non instanciées à partir de l'instanciation partielle courante, on peut affecter l'intégralité des variables avant de s'apercevoir qu'il ne s'agit pas d'une solution.

Dans le pire des cas, l'algorithme *Backtrack* est contraint de tester toutes les instanciations possibles pour trouver une solution. Pour tester la cohérence de chacune de ces instanciations, toutes les contraintes doivent être considérées. D'où une complexité temporelle dans le pire des cas en $O(ed^n)$ si on considère qu'un test de cohérence peut être effectué en temps constant. En pratique, cet algorithme n'est pas vraiment utilisable car trop lent. Cependant de nombreuses améliorations ont été proposées dans la littérature dont nous allons voir quelques exemples.

Les méthodes que nous allons présenter dans cette section sont toutes inspirées de l'algorithme *Backtrack*. Ces méthodes cherchent toutes à construire une solution en instanciant

[1] pour *Depth First Search*

[2] Pour le *Backtrack* classique, nous verrons dans la suite du chapitre que ce n'est pas toujours le cas

une par une les variables et en effectuant des retour-arrière à chaque fois que l'instanciation partielle n'est plus cohérente.

### 2.1.2 Heuristiques

Nous allons voir dans cette section des méthodes, appelées heuristiques, qui permettent de faciliter la recherche de solution, en introduisant une certaine connaissance de la structure du problème à traiter. Ce sont les heuristiques de choix de variables, et les heuristiques de choix de valeurs.

#### Choix de variables

La façon dont on va générer l'arbre de recherche, c'est-à-dire l'ordre dans lequel les variables sont choisies pour étendre l'instanciation, joue de manière déterminante sur la rapidité à trouver la solution. Par exemple, si l'arbre de recherche contient moins de nœuds intermédiaires, il engendre moins de retour-arrière, et peut donc être parcouru plus efficacement. On peut générer de tels arbres en choisissant d'abord les variables ayant le plus petit domaine. Dans l'exemple de la figure 2.1, les variable $x, y$ et $z$ on respectivement des domaines de taille 4,3 et 2. En choisissant d'abord la variable $z$, puis la variable $y$ et enfin la variable $x$, on génère un arbre de recherche contenant deux fois moins de nœuds intermédiaires que dans l'ordre inverse.

Le *Fail First Principle* [HE80], préconise de choisir d'abord la variable qui a le plus de chance de mener à un échec, le but étant d'élaguer rapidement de grands sous-arbres sans solution. Hormis le choix de la variable ayant le plus petit domaine évoqué plus haut, il existe d'autres heuristiques de choix de variables, comme choisir d'abord la variable ayant le plus grand degré dans le graphe des contraintes ou bien choisir d'abord la variable contrainte par le plus grand nombre de variables déjà instanciées. Ces heuristiques peuvent être combinées entre elles [GMP+96, BR96] afin d'obtenir un critère plus fin. Ces combinaisons peuvent cependant devenir coûteuses, il faut donc chercher le meilleur compromis entre temps de calcul et efficacité à trouver la solution.

#### Choix de valeurs

L'ordre d'instanciation des valeurs pour chaque variable influence lui aussi les performances. Si on ne recherche qu'une solution, il est généralement plus efficace de choisir la valeur qui est compatible avec un maximum de valeurs des variables non instanciées. On aimerait pouvoir choisir pour chaque variable la valeur qui nous conduit à la bonne solution, on va donc chercher des heuristiques qui vont nous permettre de se rapprocher

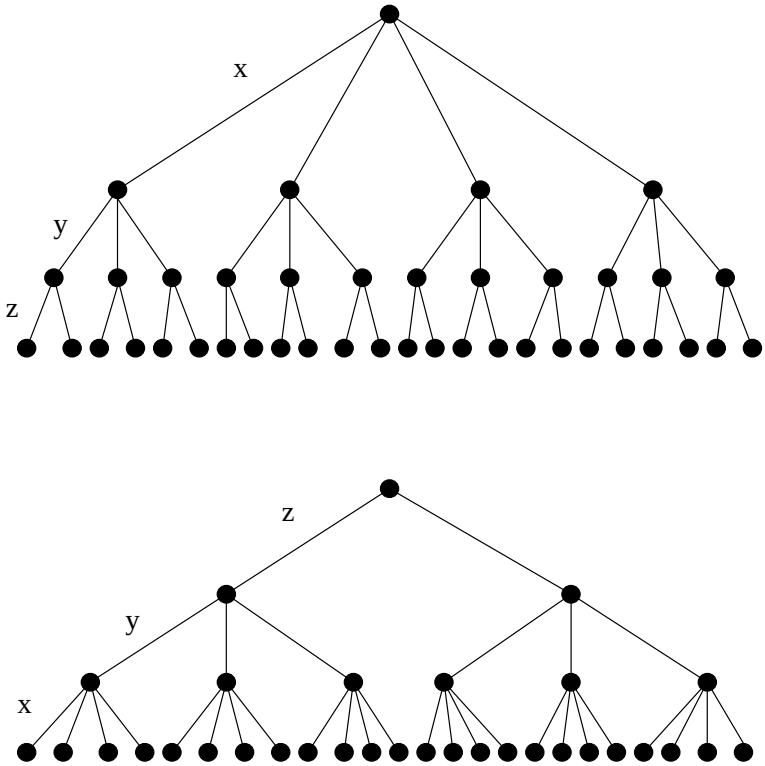


FIG. 2.1 – Deux arbres de recherche selon l'ordre d'instanciation.

de cet état parfait. La technique *Lookahead Value Ordering* [FD95] ordonne les valeurs en comptant le nombre de conflits que la valeur de la variable courante a avec les valeurs de variables non encore instanciées. Une autre technique consiste à compter le nombre de solutions potentielles [Gee92]. Lorsqu'on traite des CSP sur-contraints, on a plutôt intérêt à choisir les valeurs qui minimisent les conflits avec les variables déjà instanciées [MJPL92].

Ces heuristiques permettent d'améliorer sensiblement la recherche de solution mais ne suffisent généralement pas. Nous allons donc maintenant voir différentes méthodes d'amélioration du *Backtrack*. Elles sont de deux types :

- Les premières sont rétrospectives : elles cherchent à identifier les causes de l'échec afin de ne pas retomber dans des configurations similaires.
- Les secondes sont prospectives, on élimine certaines valeurs du domaine des variables non encore instanciées en fonction de la connaissance apportée par les variables déjà instanciées.

### 2.1.3 Méthodes rétrospectives (*Lookback*)

Contrairement au *Backtrack* qui revient sur l'instanciation de la variable précédant immédiatement la variable courante, les méthodes rétrospectives s'intéressent aux causes de l'échec. Elles cherchent la variable plus en amont qui est la cause de l'échec (par exemple en utilisant le *Backjumping* [Dec89] ou le *Conflict-directed Backjumping* [Pro93]). Elles peuvent aussi chercher à tirer partie de l'information donnée par cet échec (par exemple en utilisant le *Nogood Recording* [SV94] ou encore le *Dynamic Backtracking* [Gin93, Bli98]). Bien qu'efficaces, ces méthodes se révèlent généralement moins performantes que les méthodes prospectives étudiées dans la section suivante. Pour une description complète de la plupart des méthodes rétrospectives, nous conseillons [DF98].

### 2.1.4 Méthodes prospectives (*Look-ahead*)

Le principe, utilisé par les méthodes prospectives, est de mettre en œuvre les techniques de filtrage tout au long de la recherche. On supprime dynamiquement les sous arbres de l'arbre de recherche qui ne contiennent pas de solution. Pour cela, on utilise une propriété de cohérence locale et on retire des valeurs qui, compte tenu de l'instanciation courante, ne vérifient pas cette propriété. On arrive ainsi à réduire l'espace de recherche aux seules zones prometteuses, et donc à rendre la recherche plus efficace. Les propriétés de filtrage que l'on va utiliser dépendent de l'intensité de filtrage désirée. On pourra pour cela tester la cohérence de nœuds, d'arcs ou de chemins.

La méthode prospective la plus connue est le *forward checking* (FC) [HE80] qui élimine les valeurs des variables non instanciées qui sont directement incompatibles avec l'instanciation courante. Cette technique s'est révélée comme l'une des plus efficaces, notamment sur des problèmes de taille raisonnable. La conception de méthodes utilisant des cohérences plus fortes comme le *real-full lookahead* [HE80] ou des techniques plus récentes comme le maintien de cohérence d'arcs durant la recherche (MAC) [SF94] se sont révélées plus efficaces sur des problèmes difficiles de grande taille. D'autres techniques de cohérence sont basées sur les cohérences locales inverses, un cadre général pour ces techniques est proposé dans [VMB99].

Utiliser des méthodes de filtrage pour améliorer la recherche est indispensable. Cela permet de réduire de manière significative l'espace de recherche et permet de ne pas retomber sur certaines incohérences. Il faudra cependant là aussi savoir trouver le meilleur compromis entre le temps de calcul imposé par le filtrage et la proportion de l'espace de recherche qui a été enlevée. Pour une étude approfondie des cohérences locales, le lecteur pourra consulter [Deb98].

Bien que ne faisant pas partie des méthodes prospectives, certaines méthodes s'y ap-

parentent car elles cherchent à réduire la complexité du CSP avant de lancer la recherche. Il s'agit des méthodes structurelles. Ces méthodes commencent par une phase de prétraitement qui analyse la structure du CSP, et essayent de le rendre soit plus simple, soit plus petit. Sans entrer dans les détails, nous pouvons citer les techniques de coupe-cycle [Jeg90] ou encore de *tree-clustring* [DP89].

Il existe d'autres méthodes qui améliorent le *Backtrack* et qui peuvent de plus être avantageusement combinées avec les méthodes vues précédemment. Ces méthodes proposent de parcourir l'arbre de recherche généré de manière différente, en prenant plus en compte notamment l'efficacité des heuristiques de choix de valeurs. Nous allons présenter ici les plus connues.

### 2.1.5 Différents parcours d'arbres

Après avoir choisi les heuristiques qui paraissaient pertinentes, la façon dont on va parcourir cet arbre va influer sur la rapidité à trouver une solution. Pour cette raison, différents parcours d'arbre ont été proposés dans la littérature. Nous allons présenter les plus connus. Ces trois méthodes partent du principe que DFS ne parcourt pas les feuilles de manière cohérente par rapport à l'ordre de parcours que préconise l'heuristique de choix de valeurs. En effet, celle-ci remet systématiquement en cause les derniers choix effectués et ne remet en cause les premiers choix que très tardivement. De ce fait un mauvais choix fait au début ralentit considérablement la recherche. Il faudra cependant être attentif à la qualité de l'heuristique, car ces différents parcours obligent à générer nettement plus de nœuds internes. Pour une description complète et une étude comparée de ces méthodes, nous invitons le lecteur à consulter [Prc98].

#### IDFS

La recherche en profondeur d'abord entrelacée (IDSF[3]) [Mes97] a été proposée par Meseguer sous deux versions. Cette méthode propose de remettre en cause les choix les plus anciens contrairement au *Backtrack* qui remet en cause les choix les plus récents. La première version proposée par l'auteur, appelée IDFS pure, consiste à remettre systématiquement en cause les premiers choix avant de passer aux suivants. De fait, on ne repasse pas par un nœud avant d'avoir visité tous ses frères. C'est une manière d'imiter la parallélisation de la recherche arborescente. Cependant cela impose de mémoriser tous les nœuds intermédiaires, dont la quantité est exponentielle en fonction du nombre de variables du problème. Cette première version est donc inapplicable. L'auteur propose une deuxième

[3]Interleaved Depth-First Search

version, IDFS Limité, qui restreint l'entrelacement à un nombre fixé de sous-arbres situés à une profondeur donnée.

### LDS

La recherche à divergence limitée (LDS[4]) proposé par Harvey et Ginsberg [HG95], et améliorée par Korf [Kor96], propose de ne visiter que les feuilles dont le nombre de violations des choix préconisés par l'heuristique est borné. De cette façon, on ne visite que les zones de l'arbre de recherche les plus prometteuses. Cette méthode présuppose que l'heuristique utilisée est très bonne, et qu'elle n'est pas fondamentalement meilleure en haut de l'arbre qu'en bas. LDS procède par itérations successives, en augmentant à chaque itération le nombre de divergences autorisé (figure 2.2). Il faut noter que cette méthode génère jusqu'à deux fois plus de nœuds internes qu'une recherche en profondeur d'abord classique. Cette méthode a été définie sur les arbres binaires mais peut, dans une certaine mesure, être étendue aux arbres n-aires.

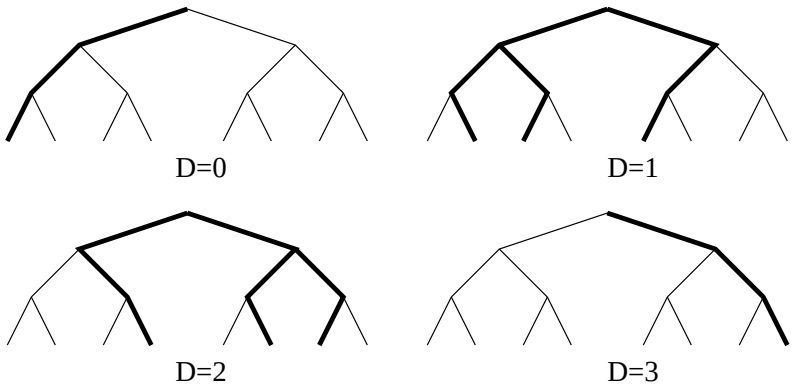


FIG. 2.2 – Les différentes itérations de LDS en fonction du nombre de divergences $D$ autorisé.

### DDS

La recherche à divergence limitée en profondeur (DDS[5]) proposée par Walsh [Wal97] s'inspire des deux dernières méthodes. Elle va procéder par itérations successives comme le fait LDS, mais va présupposer que l'heuristique de choix de valeur est plus efficace en bas de l'arbre qu'en haut, présupposition faite aussi par IDFS. DDS va donc d'abord remettre en cause les premiers choix effectués. Contrairement à LDS qui fixe le nombre de

[4]Limited Discrepancy Search
[5]Depth-bounded Discrepancy Search

divergences maximal autorisé, DDS fixe la profondeur maximale à laquelle on autorise les divergences. On va donc parcourir de manière systématique tous les chemins jusqu'à une certaine profondeur, puis ensuite suivre uniquement les choix préconisés par l'heuristique (figure 2.3), en évitant cependant les chemins déjà explorés.

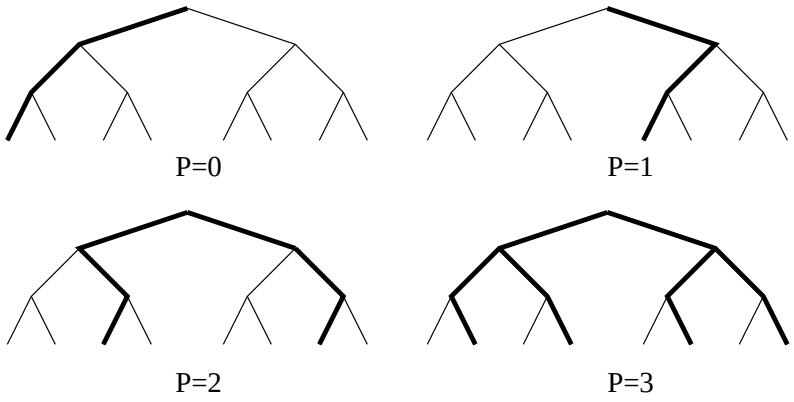


FIG. 2.3 – Les différentes itérations de DDS en fonction de la profondeur $P$ jusqu'à laquelle les divergences sont permises.

### 2.1.6 CSP sur-contraints

Lorsqu'on traite des CSP sur-contraints, le but n'est plus de trouver une solution, puisque par définition il n'y en a pas, mais de minimiser un critère comme la somme des coûts des contraintes non satisfaites. Les méthodes utilisées pour résoudre de tels problèmes sont légèrement différentes de celles utilisées pour résoudre les CSP classiques.

#### *Branch and Bound*

La méthode de séparation/évaluation (*Branch and Bound* en anglais) en profondeur d'abord, permet d'élaguer l'arbre de recherche en éliminant les branches qui ne mènent pas à des solutions ayant un coût inférieur au coût de la meilleure solution trouvée jusqu'à ce moment. Pour ce faire elle utilise une borne $B$ qui correspond au coût de la meilleure solution trouvée (cette borne étant fixée à l'infini en début de résolution). Après chaque nœud généré dans l'arbre de recherche, on va se demander s'il va mener à une solution moins bonne, en calculant une minoration des coûts des solutions constructibles à partir de ce nœud. Si la somme de cette minoration et du coût de la solution partielle est supérieure à $B$, on rejette le nœud. Le *Branch and Bound* peut facilement être combiné aux méthodes vues précédemment, avec les modifications qui s'imposent. Par exemple, on utilisera le

*forward checking* avec les compteurs d'incohérence de Freuder et Wallace [FW92], ou bien les compteurs d'incohérence de type PFC-MRDAC [LMS99].

Nous n'avons pas évoqué ici la recherche gloutonne. Cette méthode constructive élabore elle aussi une solution en suivant diverses heuristiques, mais s'arrête dès la première instanciation complète, qu'elle soit ou non cohérente. En fait, on utilise surtout cette méthode comme initialisateur pour les méthodes de recherche locale que nous allons présenter maintenant.

## 2.2 Méthodes de recherche locale

Les méthodes de recherche locale ne sont pas systématiques, elles ne peuvent garantir de trouver une solution s'il y en a ni de démontrer qu'il n'y en a pas le cas échéant. Elles vont généralement utiliser des processus stochastiques, c'est-à-dire qu'elles vont utiliser des processus aléatoires au cours de la recherche afin d'introduire de la diversification dans le parcours de l'espace de recherche. Ces méthodes se révèlent généralement efficaces, et permettent de traiter des problèmes trop gros pour être traités efficacement par les méthodes constructives. Le lecteur intéressé pourra se reporter à l'article de synthèse [HGM99].

Les méthodes de recherche locale ne procèdent pas par construction d'une solution mais par améliorations successives d'une solution initiale (i.e. d'une instanciation globalement incohérente). Pour cela, plusieurs outils sont nécessaires, principalement, une fonction de voisinage qui permettra de se déplacer dans l'espace de recherche à partir de la solution initiale et une fonction d'évaluation qui permettra, entres autres, de choisir parmi les voisins. On peut schématiser la procédure générale de la plupart des méthodes locales comme suit :

1. choisir une solution initiale $s$.
2. choisir parmi les voisins de $s$ une solution $\bar{s}$
3. vérifier si la condition de terminaison est vraie.
4. décider si $\bar{s}$ doit remplacer $s$, revenir à l'étape 2

Selon la méthode utilisée, ce sont les trois derniers points qui vont suivre des stratégies différentes. Pour les différentes méthodes de recherche locale que nous allons présenter, nous appellerons $s$ la solution courante, $\bar{s}$ la nouvelle solution prise dans le voisinage $V(s)$, et enfin $E(s)$ sera la fonction qui calcule le coût de la solution $s$. En règle générale, $E(S)$ calculera le nombre de contraintes violées (ou satisfaites). De nombreuses fonctions de voisinage peuvent être définies, cependant on choisira généralement, dans le cadre des

CSP, la fonction qui associe à $s$ l'ensemble des solutions qui ne diffèrent que par la valeur d'une variable. On peut restreindre ce voisinage en ne choisissant que les variables qui sont en conflit [MJPL92].

### 2.2.1 *Hill-climbing* et GSAT

La méthode du *Hill-climbing* ou méthode de descente stricte procède par améliorations successives strictes. On choisit un voisin strictement meilleur, qui remplace la solution courante et on s'arrête lorsque l'on est sur un optimum. Plus formellement :

- $s \leftarrow \bar{s}, \bar{s} \in V(s) si\ E(\bar{s}) < E(s)$.
- si $\forall \bar{s} \in V(s), E(\bar{s}) \geq E(s)$ on s'arrête.

Cette méthode est très simple, mais pèche sur plusieurs points. Tout d'abord, différents choix s'offrent à nous, et vont déterminer l'efficacité de l'algorithme : doit-on choisir le premier voisin qui améliore la solution courante ou bien le meilleur ? Si on cherche le meilleur voisin, est-il possible d'évaluer rapidement tous ses voisins ? Dans le cadre des CSP, il est facile de ne compter que les incohérences locales générées par le nouveau choix de valeur. Ensuite, l'algorithme s'arrête au premier optimum trouvé, ce qui fait que l'initialisation a un rôle très important. De plus si le problème contient beaucoup d'optima locaux, et peu de globaux, il y a de grandes chances pour que ce soit un optimum local qui soit trouvé. Il existe de nombreuses améliorations possibles, et parmi elles, la relance aléatoire (GSAT [SLM92]). Cela consiste à recommencer avec un nouvelle solution initiale à chaque fois qu'un optimum local a été atteint (en mémorisant le meilleur trouvé) un certain nombre de fois décidé à l'avance. Cette méthode permet en fait de donner à la méthode de descente stricte plus de chance d'obtenir une bonne solution. Une autre variante consiste à autoriser le choix d'un voisin équivalent (en terme d'évaluation) s'il n'y a pas de voisin meilleur. Cette méthode permet d'explorer des plateaux de l'espace de recherche.

### 2.2.2 Recuit simulé

Le recuit simulé est une méthode locale inspirée du recuit thermique utilisé dans l'industrie métallurgique pour obtenir un cristal parfait. La technique industrielle utilisée consiste à abaisser progressivement la température du métal en fusion, de manière à ce qu'en refroidissant lentement les molécules prennent un «axe parfait». L'algorithme qui s'inspire de cette méthode a été introduit par Kirkpatrick et al. [KGV83].

Le principe du recuit simulé est le suivant : on choisit aléatoirement une solution potentielle $s$, et on fixe la température $T$. On va ensuite entrer dans la boucle de l'algorithme :

- on choisit aléatoirement un voisin $\bar{s}$ de la solution courante

– on calcule la différence entre l'évaluation de $s$ et de $\bar{s}$ : $\Delta = E(s) - E(\bar{s})$[6]
– $s \leftarrow \bar{s}$ avec une probabilité $e^{\frac{\Delta}{T}}$ si $\Delta$ est négatif, 1 sinon.
– on fait décroître $T$.

Différents critères d'arrêt peuvent être envisagés : lorsque les diminutions de température s'avèrent inefficaces, lorsqu'un certain nombre de mouvements a été effectué, .... On va généralement démarrer en utilisant un température très élevée qu'il va falloir diminuer par palliers successifs au cours de l'exécution. La difficulté étant de trouver une fonction de décroissance de la température qui soit efficace. Cet algorithme garde systématiquement la nouvelle solution si elle améliore la solution courante, sinon la nouvelle solution qui dégrade la solution courante est choisie avec une probabilité qui diminue au cours du temps. Donc, au début de l'exécution on va avoir tendance à choisir n'importe quel voisin, pour ensuite se focaliser progressivement sur un sous-espace de l'espace de recherche en acceptant de moins en moins de solution dégradant la solution courante. L'avantage de cette méthode par rapport à la méthode de descente est qu'elle peut sortir des optima locaux (tant que la température est suffisamment élevée). Cependant le choix de la température initiale et la fonction de décroissance associée à cette température restent des paramètres assez difficiles à régler. Une variante appelé algorithme de Métropolis, et qui consiste à utiliser une température constante (généralement faible) pour toute l'exécution a été proposée dans [Con90].

### 2.2.3 Liste Tabou

La recherche Tabou introduite par Glover [Glo86], est une méthode qui cherche à guider la recherche de manière intelligente dans l'espace des solutions. Pour ce faire, elle dispose d'une mémoire (la liste Tabou) des solutions par lesquelles on est déjà passé qui lui permet d'éviter les bouclages. La méthode de recherche Tabou peut être résumée comme suit : on commence par initialiser $s$, en utilisant par exemple un algorithme glouton, et on initialise la mémoire à vide. Ensuite, on entre dans la boucle de l'algorithme :

– on cherche $\bar{s} \in V(s)$ tel que $\bar{s}$ minimise $E(\bar{s})$[7] et que $\bar{s} \notin M$.
– on met à jour la mémoire $M$ en y ajoutant $s$ et en retirant l'élément le plus ancien.

Le critère d'arrêt peut être le nombre maximum de mouvements autorisé, d'avoir atteint un coût acceptable, de ne pas avoir obtenu d'amélioration depuis un certain nombre de mouvements, ...

La recherche Tabou va donc chercher le meilleur voisin non tabou, même si celui-ci dégrade la solution courante. Une solution qui a été choisie va être mise dans la mémoire,

[6] En convenant que l'on cherche à minimiser $E(s)$.
[7] On note que $E(\bar{s})$ peut être moins bonne que $E(s)$

et interdite (taboue) pendant un certain nombre d'itérations[8]. De ce fait, on va éviter les bouclages et de plus cela va permettre à l'algorithme de sortir des optima locaux.

De nombreux paramètres vont influer sur l'efficacité de la méthode Tabou. La taille de la liste Tabou, la manière dont on va parcourir le voisinage, le fait de ne parcourir qu'une portion du voisinage lorsque celui-ci est trop grand pour le parcourir entièrement, la méthode utilisée pour l'initialisation sont des paramètres déterminants.

Il existe de nombreuses variations de la recherche Tabou. On peut par exemple ne pas mémoriser les solutions passées mais les couples valeur/variable qui ont changé, on interdit ainsi un ensemble de solutions (toutes celles qui contiennent cette instanciation), on peut aussi intégrer de l'aléatoire dans le choix des voisins, ou encore mettre en place des critères d'aspiration, lorsque les critères d'interdiction de la liste sont trop astreignants. Le lecteur pourra consulter [GL97] pour une description complète de la recherche Tabou.

## 2.3 Méthodes à population

L'originalité de ces méthodes réside dans le fait qu'elles utilisent un ensemble de solutions potentielles (appelé population) qui vont être confrontées à des processus stochastiques via des opérateurs dédiés. Cela leur permet d'exploiter plus largement l'espace de recherche, mais induit un surcoût dû au traitement de cette population. Ces méthodes utilisent d'une certaine manière un parallélisme implicite et vont chercher à tirer profit de l'information globale qui en ressort. Nous n'allons que très brièvement présenter les méthodes évolutionnaires puisqu'elles sont amplement présentées dans le chapitre suivant. Nous présenterons ensuite deux autres méthodes utilisant des populations de solutions dont le mode opératoire est différent.

### 2.3.1 Algorithmes génétiques

Les algorithmes génétiques ont été introduits par Holland en 1975[Hol75]. Ils s'inspirent des mécanismes génétiques de l'évolution des espèces afin de permettre à une population de solutions de converger vers les solutions optimales. Pour ce faire, ils vont utiliser un mécanisme de sélection des individus de la population (les solutions potentielles). Les individus sélectionnés vont être croisés entre eux (exploitation), et certains vont être mutés (exploration). Ces mécanismes d'exploitation et d'exploration vont permettre de converger vers les bonnes solutions en évitant, autant que faire se peut, les optima locaux.

[8]En général, on mémorise les mouvements qui ont été effectué pour passer d'une solution à une autre

### 2.3.2 Go With the Winners

Cette méthode («Va avec les gagnants» en francais) [AV94, DI96] ne fait pas partie des algorithmes évolutionnaires. Elle utilise une population de particules (les solutions potentielles) choisies aléatoirement. Ces particules sont évaluées, et ne doivent pas dépasser un certain seuil sous peine d'être éliminées et remplacées par une copie d'autres particules plus compétitives de la population courante. Ensuite on effectue une marche aléatoire à partir des particules sélectionnées pour diversifier les particules regroupées, et enfin le seuil est décrémenté. Différentes améliorations ont été testées, notamment en essayant de guider la recherche de nouvelles particules plutôt que d'effectuer une marche aléatoire.

### 2.3.3 Colonie de fourmis

Une autre méthode utilisant des populations a été introduite par Dorigo, Maniezzo et Colorni [DMC96, DCG99]. Bien que cette méthode s'apparente plus à des techniques d'apprentissage réparti, elle permet de résoudre efficacement des problèmes tels que le voyageur de commerce. Cette méthode a été inspirée par le travail réalisé par les fourmis vivant en colonie. Les fourmis sont capables de trouver le plus court chemin de la colonie vers les stocks de nourriture. Lorsqu'elles se déplacent, elles déposent sur le sol des phéromones et elles suivent les phéromones déposées par d'autres fourmis avec une certaine probabilité. Les phéromones subissant l'évaporation, seuls les chemins suivis par le plus grand nombre de fourmis sont conservés. De fait, les chemins les plus empruntés seront les plus courts. Pour comprendre ce phénomène, il suffit de se représenter une fourmilière d'où partent des fourmis aléatoirement à la recherche de nourriture. Chaque fourmi qui quitte la fourmilière dépose des phéromones sur le sol. Lorsqu'une fourmi a trouvé une réserve de nourriture, elle revient à la fourmilière en suivant les phéromones qu'elle a déposées, et en en déposant d'autres. De fait le chemin qu'a suivi cette fourmi contient plus de phéromones que celui des autres fourmis qui n'ont encore rien trouvé. Les autres fourmis partant de la fourmilière vont donc avoir tendance à suivre prioritairement ce chemin, et vont elles aussi déposer des phéromones. Au fur et à mesure, et jusqu'à ce que la réserve de nourriture soit épuisée, ce chemin sera de plus en plus suivi et aura donc de plus en plus de chance d'être suivi.

Le système de colonie de fourmis propose d'imiter ce comportement en utilisant des agents (les fourmis) qui s'échangeront des informations via les phéromones déposées sur les arêtes d'un graphe. En utilisant ce principe on pourra chercher à optimiser des problèmes tels que le voyageur de commerce, ou encore le problème de routage dans les réseaux. Pour une description complète des colonies de fourmis, et des problèmes qu'elles permettent de résoudre, nous conseillons la lecture de la première partie de [CDG99].

## 2.4 Hybridations

La dernière méthode utilisée, dans la classification que nous avons proposée, consiste à hybrider les méthodes précédemment présentées. On entend par hybridation le fait de mêler intimement les propriétés d'algorithmes fondamentalement différents. Par exemple, on peut combiner la capacité d'exploitation de l'espace de recherche que permettent les algorithmes génétiques avec les capacité d'exploration que proposent les méthodes de recherche locale. Ces méthodes hybrides se sont révélées très efficaces sur de nombreux problèmes d'optimisation combinatoire, comme le voyageur de commerce, la coloration de graphe, l'affectation quadratique, ...et ont parfois obtenu des résultats meilleurs que les autres méthodes utilisées seules. Cependant pour que leur efficacité soit réelle, ces méthodes doivent généralement intégrer une bonne connaissance du problème pour éviter que les bonnes solutions obtenues ne soient perdues. Par exemple, avec un algorithme hybride AG/recherche locale, un croisement non spécialisé détruira les bonne solutions obtenues par la recherche locale. Le problème sous-jacent est qu'une méthode hybride très efficace sur un problème donné pourra se révéler désastreuse sur un autre problème, voire sur une autre instance du premier problème. Certaines hybridations utilisant les AG seront présentées dans le chapitre suivant, le lecteur pourra cependant consulter [DH98] pour une hybridation entre algorithme génétique et recherche Tabou, [Sha98] pour une hybridation entre recherche locale et programmation par contraintes (LNS pour *Large NeighBourhood Search*), [RFR99] pour une hybridation entre colonie de fourmis et recherche Tabou ou encore [LKH93, Lan98] pour une hybridation entre recuit simulé et algorithme génétique par exemple. Nous renvoyons de nouveau à la lecture de [CDG99], particulièrement pour la description des algorithmes mémétiques et la proposition de cadre général pour l'hybridation qui est proposé.

## 2.5 Conclusion

Nous avons présenté dans ce chapitre de nombreuses méthodes qui peuvent s'appliquer aux CSP. Nous avons vu des méthodes systématiques qui permettent de certifier qu'une solution est la meilleure, ou de prouver qu'aucune solution n'existe. Nous avons vu d'autres méthodes qui cherchent à améliorer une ou plusieurs solutions potentielles par itérations successives. Toutes ces méthodes ont leurs avantages et leurs inconvénients. Les premières réclament un temps d'exécution bien supérieur, mais garantissent le résultat. Les secondes peuvent trouver rapidement un résultat satisfaisant et permettent de traiter des problèmes de grande taille, mais réclament souvent une période de réglage des paramètres longue et fastidieuse. Nous comparerons dans le chapitre 5, les capacités de certaines des méthodes présentées à trouver de bonnes solutions sans réglage particulier.

# Deuxième partie

# Méthodes évolutionnaires

# Chapitre 3

# Algorithmes Évolutionnaires

## 3.1 Introduction

Ce chapitre s'attache à présenter de manière générale les méthodes évolutionnaires, et en particulier les algorithmes génétiques. Dans la classification que nous avons utilisée au chapitre 2, ces méthodes font partie des méthodes stochastiques basées sur une population. Elles se sont essentiellement et grossièrement inspirées du modèle biologique sélectionniste introduit par Charles Darwin : les individus les plus adaptés survivent et se reproduisent. Ces méthodes se sont vite imposées comme méthodes d'optimisation globale, permettant d'optimiser des problèmes très variés et non triviaux. Particulièrement, elles permettent de traiter des problèmes de grande taille, ou encore des problèmes non décrits de manière explicite mais sous forme de programme. Leur vaste champ d'action, leur implantation généralement rapide et aisée, et l'utilisation récurrente de métaphores ayant trait à la biologie sont certainement à l'origine de leur succès. Nous commencerons par une introduction présentant les fondements biologiques qui ont inspiré ces méthodes, le but étant d'amener le lecteur à pressentir les intuitions sous-jacentes qui ont précédé le développement de méthodes s'inspirant du vivant. Nous introduirons ensuite quelques définitions nécessaires à une bonne compréhension du modèle évolutionnaire. La section suivante présentera l'algorithme génétique standard ainsi que quelques bases théoriques qui expliquent les raisons de l'efficacité des algorithmes génétiques et les cas limites (théorie des schémas, problèmes déceptifs et *Royal Road*). Dans les trois dernières sections, nous présenterons brièvement différentes approches : la programmation génétique, les stratégies d'évolution ainsi que différentes approches hybrides.

## 3.2 Fondements Biologiques

L'existence d'espèces diverses et variées n'a pas toujours été expliquée par la théorie de l'évolution. La théorie du fixisme, et la vision empreinte de catastrophisme largement soutenue par l'église était la plus couramment admise avant le $XIX^{ème}$ siècle[1] .

Les travaux de Lamarck, Mendel et surtout Darwin, sont venus chambouler cette vision statique du monde et ont posé les bases de toute la génétique moderne. L'origine de la vie, et par là l'origine des espèces a toujours été un sujet controversé. D'un côté, les religions ont une vision statique d'un monde créé soudainement par une ou plusieurs entités surnaturelles, de l'autre les sciences ont plutôt une vision d'une évolution lente où l'hérédité et la sélection naturelle jouent un grand rôle. Nous n'allons ici nous intéresser qu'aux fondements scientifiques et biologiques qui ont donné naissance ensuite aux algorithmes évolutionnaires. Nous avons divisé cette section en deux parties, la première concerne ce que nous avons appelé le niveau macroscopique car c'est par observation directe du vivant que les découvertes ont été faites. La deuxième partie concerne le niveau microscopique, c'est-à-dire les bases biochimiques.

### 3.2.1 Mendel, Darwin et les autres : le niveau macroscopique

Très tôt, les hommes ont eu conscience de l'hérédité, certaines populations refusant les unions avec des étrangers [2](« le sang bleu»), d'autre évaluant le mérite en fonction de la naissance, pour ne citer que ces exemples. Cependant les principes fondamentaux n'ont été découverts qu'aux $XIX^{ème}$ et $XX^{ème}$ siècles.

#### Hérédité

On appelle hérédité la capacité qu'a un être vivant à transmettre ses caractères à ses descendants. De tout temps, l'homme a su utiliser les principes de l'hérédité sans vraiment en comprendre les mécanismes fondamentaux. Tous les éleveurs savent bien qu'il vaut mieux choisir les meilleurs animaux[3] comme reproducteurs. Bien que la transmission des

[1] A noter cependant la citation d'Aristote présentée par Darwin dans son introduction à l'origine des espèces : «Qu'est-ce qui empêche les différentes parties (du corps) d'avoir dans la nature ces rapports purement accidentels ? [...]Partout donc, toutes choses réunies (c'est-à-dire l'ensemble de parties d'un tout) se sont constituées comme si elles avaient été faites pour quelque chose ; celles façonnées d'une manière appropriée par une spontanéité interne se seront conservées, tandis que dans le cas contraire elles auront péri et périssent encore. »

[2] Alors que l'ovule a tendance à fusionner avec le spermatozoïde le plus différent d'elle au niveau génétique

[3] Mâle généralement : on a longtemps pensé que le spermatozoïde n'était qu'un embryon (nommé homonculus) , et l'ovule qu'un substrat nutritif nécessaire.

caractères d'un individu à ses descendants semblait évidente, la disparition de certains caractères et la réapparition d'autres [4] sont longtemps restées inexpliquées. Les travaux de Mendel sur les pois ont permis de comprendre enfin les principes qui régissaient cette transmission des caractères. Il exhiba notamment le fait que le matériel héréditaire est donné pour moitié par chacun des parents, mais que cette moitié contient quand même toute l'information nécessaire et que ce sont des mécanismes de dominance et de récessivité qui détermineront lesquels seront exprimés. Le mécanisme de l'hérédité avait son modèle, cependant on ne savait toujours rien des principes biochimiques qui rendaient tout cela possible. A ce stade, on considère le cycle de vie comme un modèle de développement programmé qui se transmet d'une génération à l'autre grâce à des facteurs héréditaires, le matériel héréditaire devant être capable de se répliquer, de contrôler les activités cellulaires et d'évoluer.

### Différentes hypothèses concernant la sélection naturelle

Les espèces vivantes présentent d'innombrables variations individuelles ou raciales, à tel point qu'il existe toujours entre deux populations d'une même espèce, isolées l'une de l'autre, d'infimes différences. La multiplication des individus se faisant à un rythme supérieur à celle de leurs ressources, elle conduit à une impitoyable concurrence vitale, entraînant la mort sélective des plus faibles et la survie des plus aptes. Mais cette sélection des individus les plus adaptés ne semble pas être le seul critère. Comment expliquer sinon l'évolution qu'a suivi le paon par exemple[5]. Certaines espèces ont évolué en fonction des préférences sexuelles du partenaire, quitte à ce que ce soit au détriment de la logique de survie. D'autres points posent problème : une variation étant toujours un infime changement, généralement insuffisant pour donner un avantage réel lors de la sélection naturelle, le caractère ne devient généralement avantageux qu'après une longue orthogenèse. En fait, la nature semble éliminer les pires individus, mais laisse le soin au hasard en ce qui concerne tous les autres. Ce mécanisme permet deux choses. Premièrement, il y a conservation de la diversité génétique qui permet entre autre de générer de nouvelles variations et d'éviter l'expression de mutation létale. Deuxièmement, il permet à la population d'avoir une meilleure capacité d'adaptation en cas de variations du milieu. L'hypothèse neutraliste prétend que certaines mutations ne confèrent aucun avantage notable, mais aucun désavantage non plus. Ceci permet d'expliquer entre autre la disparition d'organes devenus inutiles, mais aussi l'impressionnant polymorphisme régnant dans la faune et la flore terrestre.

[4] «Il a les yeux de son grand père!»

[5] Dont la queue en panache est un véritable appel pour les prédateurs

### 3.2.2 Watson, Crick et les autres : le niveau microscopique

La découverte des chromosomes[6], véritable support matériel des caractères héréditaires, puis celle de la structure moléculaire de l'acide désoxyribonucléique[7] (ADN) ont permis de comprendre le fonctionnement exact des principes héréditaires. Les chromosomes sont constitués d'ADN, c'est lui qui contient toute l'information nécessaire à la « fabrication » d'un individu[8]. La question a longtemps été de savoir comment une molécule pouvait contenir une telle masse d'information.

#### Le code génétique

L'ADN est un polymère de nucléotides complexe en double hélice. Il est composé entre autre d'une succession de bases azotées, qui sont au nombre de quatre : adénine, thymine, cytosine et guanine, la plupart du temps représentées par leurs initiales. L'adénine s'apparie toujours avec la thymine et la guanine avec la cytosine. Ainsi, lorsque la molécule se réplique, il se forme un brin complémentaire de chaque brin de la double hélice. Il y a quatre bases azotées et vingt-et-un[9] acides aminés (éléments constitutifs de toutes les protéines), il faut donc trois bases pour coder un acide aminé. Ce sont ces suites de triplets qui vont déterminer l'intégralité du code génétique.

**La biosynthèse des protéines** La biosynthèse des protéines est un processus chimique très complexe. En schématisant, L'ADN n'est que le vecteur d'information, celui-ci est lu et copié dans un ARN messager[10]. Mais cet ARN messager ne contient que le code d'un gène, il va donc y avoir un ARN messager par gène. Cette molécule sera ensuite lue par un ribosome qui effectuera la synthèse de la protéine codée par le gène.

Les processus de réplication de l'ADN ne sont par pour autant totalement fiables, ce qui explique en partie les mutations, d'autres pouvant être dues à des facteurs externes. De plus, lors de la division cellulaire il arrive que des brins d'ADN se trouvent échangés, soit à l'intérieur d'un même chromosome, soit entre deux chromosomes qui s'apparient. Les trois principes vus au niveau macroscopique se sont donc révélés exacts au niveau microscopique : l'ADN est capable de se répliquer, de contrôler l'activité cellulaire via la

[6] Découvert par Thomas Hunt Morgan au début des années 30

[7] Découverte par Watson et Crick en 1953

[8] L'expérience de Hershey et de Chase a démontré que cet acide nucléique est le composant des facteurs héréditaires

[9] Un $22^{ème}$ acide aminé vient d'être découvert dont la séquence d'activation est un triplet que l'on pensait être un signal d'arrêt de lecture

[10] L'ARN messager est une sorte d'ADN à un seul brin, contenant d'autres bases azotées correspondant à celles de l'ADN

synthèse des protéines et de changer via les mutations et les échanges de brins.

### 3.2.3 Des Modèles Biologiques aux Implantations Informatiques

L'idée d'appliquer les principes du Darwinisme pour des implantations informatiques à été introduit par John Holland [Hol75]. Puisque sur des millions d'années les êtres vivants, par le principe de sélection naturelle, se sont spécialisés et complexifiés afin d'être de plus en plus adapté à leur environnement, on pouvait légitimement penser que l'application de ces principes pour l'optimisation de systèmes complexes s'avèrerait fructueuse. Mais ce passage d'un principe biologique des plus complexes vers un modèle nécessairement réducteur ne s'est pas fait sans pertes. Nous ne nous attarderons pas sur toutes les parties qui ont été laissées de côté par les algorithmes évolutionnaires pour nous attacher aux trois principes fondamentaux.

#### Sélection

Le mécanisme de sélection ne se résume pas à une fonction d'adaptation au milieu. Une multitude de paramètres rentre en ligne de compte, de plus certaines modifications du patrimoine génétique ne se révèlent qu'après une longue période, les gènes n'étant pas exprimés. La pression sélective utilisée dans les AG est généralement unidirectionnelle et sans aucune interaction ni avec le milieu, ni avec les autres individus : elle privilégie les meilleurs individus au détriment de tous les autres. Certaines théories issues du néo-darwinisme envisagent plutôt une hypothèse neutraliste : seuls les individus manifestement inadaptés sont éliminés, de telle sorte qu'un maximum de variations est conservé dans la population.

#### Croisement

Dans la nature, il existe plusieurs types de croisement. Tout d'abord, lors de la reproduction sexuée, chacun des parents ne donne à l'enfant qu'une partie (la moitié) de son patrimoine génétique. Ce premier type de croisement ne change en rien la structure des gènes. Le deuxième type de croisement a lieu lors de la méiose, deux chromosomes peuvent s'échanger une partie de leur matériel génétique. Enfin, mais c'est plus rare, on peut observer des modifications des brins d'ADN au sein même d'un chromosome : inversion, coupure, boucle... La version informatique n'a pour ainsi dire retenu que le deuxième type (*crossover* en anglais), phénomène rarissime dans la nature et quasi systématique avec les AG. Il faut noter que l'on trouve aussi dans la littérature le terme recombinaison qui semble plus approprié.

**Mutation**

La mutation est aussi un phénomène rare dans la nature, de plus il n'y a pas automatiquement une modification phénotypique liée à une mutation. La structure de l'ADN et le codage des acides aminés font que beaucoup de mutations restent dormantes. Par exemple, quatre triplets codent pour l'arginine, ne différant que par la troisième base. Une mutation affectant cette dernière base n'aura donc aucun effet. De plus, une mutation peut avoir lieu sur la partie, très importante, d'ADN non codant.

Les algorithmes génétiques sont généralement utilisés comme des algorithmes d'optimisation. Il n'était donc évidemment pas question d'imiter les lents processus évolutifs naturels, mais bien de s'inspirer des mécanismes évolutionnaires, tout en les rendant rapides et efficaces. Nous avons cependant précisé ces points, car il est courant que des personnes ayant des notions de biologie se retrouvent perdues dans cette utilisation «mal adaptée» du vocabulaire des généticiens.

## 3.3 Les algorithmes génétiques

Nous allons dans cette partie expliquer le fonctionnement d'un algorithme génétique standard. Nous allons aussi donner un certain nombre de définitions de vocabulaire nécessaires à une bonne compréhension. En effet, la communauté AG utilise nombre d'analogies issues du monde biologique pour illustrer ses idées. Le lecteur pourra aussi consulter [SS96] pour une description de l'algorithme génétique standard et du contrôle des paramètres.

### 3.3.1 Vocabulaire

**Définition 3.3.1** *(Chromosome) L'ensemble des chromosomes d'un individu regroupe l'intégralité de son patrimoine génétique. C'est grâce à l'information qu'ils contiennent qu'il va se construire. Généralement les AG n'utilisent qu'un chromosome par individu, on appellera donc chromosome, le codage d'une solution potentielle.*

Par abus de langage, on utilisera souvent indifféremment les termes chromosome, individus et solution potentielle.

**Définition 3.3.2** *(Gène) Un gène est une suite de bases azotées qui contient le code d'une protéine donnée. On appellera gène la suite de symboles qui codent la valeur d'une variable.*

Dans le cas général, un gène correspond à un seul symbole (0 ou 1 dans le cas binaire). Une mutation changera donc systématiquement l'expression du gène muté.

**Définition 3.3.3** *(Allèle) On appellera allèle une valeur admise pour un gène donné. Dans le cadre des CSP, un gène représente généralement une variable (voir chapitre 4), l'ensemble allélique correspond alors au domaine de la variable en question.*

**Définition 3.3.4** *(Population) On appellera population l'ensemble des solutions potentielles qu'utilise l'AG.*

**Définition 3.3.5** *(Individu) On appellera individu une des solutions potentielles. Dans la plupart des cas un individu sera représenté par un seul chromosome, dans ce cas, par abus de langage, on utilisera indifféremment individu et chromosome.*

**Définition 3.3.6** *(Génotype) Le génotype correspond à l'ensemble des valeurs des gènes.*

**Définition 3.3.7** *(Phénotype) le phénotype correspond à l'expression du génotype, c'est-à-dire à l'évaluation d'un génotype particulier.*

Il ne faut surtout pas confondre génotype et phénotype. Deux individus dont les génotypes sont différents peuvent avoir exactement le même phénotype.

**Définition 3.3.8** *(Locus) Le locus correspond à la position d'un gène sur le chromosome.*

**Définition 3.3.9** *(Génération) On appellera génération l'ensemble des opérations qui permettent de passer d'une population $\Pi_i$ à une population $\Pi_{i+1}$. Ces opérations seront généralement : sélection des individus de la population courante, application des opérateurs génétiques, évaluation des individus de la nouvelle population.*

Par abus de langage, on pourra aussi appeler *$i^{ème}$ génération* l'ensemble des individus après $i$ itérations de l'algorithme.

### 3.3.2 Les algorithmes génétiques

Les algorithmes génétiques font partie de la classe des algorithmes dits stochastiques. En effet, une grande partie de leur fonctionnement est basée sur le hasard. Cependant, ce hasard est dirigé grâce à la fonction d'évaluation qui permet d'introduire dans les opérateurs génétiques une quantité de déterminisme utile afin d'obtenir une solution. Le schéma global d'un algorithme génétique est constitué de 6 éléments principaux.

1. Une population de configurations initiales ;
2. Une fonction de codage/décodage des chromosomes ;
3. Des opérateurs génétiques (Mutation, Croisement,...) ;

4. Une fonction d'évaluation ;
5. Un algorithme de sélection (Tournoi, Roulette, ...) ;
6. Des paramètres.

Ensuite, il suffit d'appliquer l'algorithme suivant :

**debut**
$t = 0$
Initialisation de la population $\Pi_t$
Evaluation des individus de la population $\Pi_t$
**tant que** (Condition de terminaison non satisfaite) **faire**
  $t \leftarrow t + 1$
  Sélection et copie des parents de la population $\Pi_{t-1}$ dans $\Pi_{sel}$
  Application des opérateurs génétiques sur $\Pi_{sel}$
  $\Pi_t = \Pi_{sel}$
  Evaluation des individus de la population $\Pi_t$
**fin tant que**
Décoder la meilleure solution et l'afficher
**fin**

FIG. 3.1 – Algorithme génétique standard

Un algorithme génétique standard (SGA) utilise des chromosomes codés sur un alphabet binaire $\Omega$ et cherche à optimiser une fonction $F$ définie de $\Omega$ vers $\mathbb{R}^+$.

$$F : \Omega = \{0, 1\}^n \rightarrow \mathbb{R}^+ \tag{3.1}$$

Chaque chromosome est donc constitué d'une chaîne de $n$ bits. Les opérateurs génétiques vont donc effectuer des remaniements de ces chaînes binaires en vue d'améliorer l'évaluation des chromosomes. Nous allons maintenant suivre chaque étape de l'algorithme présenté sur la figure 3.1.

### Initialisation de la population

Généralement, la population initiale $\Pi_0$ est choisie aléatoirement, chacun des bits de chacun des chromosomes étant choisi au hasard. Cependant, rien n'interdit d'utiliser des

initialisations gloutonnes, de fournir des solutions déjà connues que l'on cherche à améliorer ou d'utiliser d'autres algorithmes de recherche qui fourniront à l'AG des solutions ayant subi une première phase d'optimisation. Le choix de l'initialisation se fera en fonction des connaissances que l'utilisateur a sur le problème. Si il n'a pas d'informations particulières, on préférera une initialisation aléatoire la plus uniforme possible afin de favoriser une exploration de l'espace de recherche maximum. Mais dans d'autres cas, on peut vouloir simplement améliorer des solutions connues. Dans ce cas, on pourra donner à l'AG l'ensemble des solutions connues.

### Évaluation

L'opérateur d'évaluation n'est pas anodin. Il est utilisé par l'opérateur de sélection pour faire son choix des individus à conserver. Il va donc influencer la convergence de l'algorithme. Dans certains cas, on pourra différencier la fonction d'évaluation et la fonction de *fitness* : la fonction d'évaluation calcule le coût d'un individu, alors que la fonction de *fitness* calcule le coût de cet individu par rapport à la population courante. Lorsqu'une sélection de type roulette est utilisée, il est important de faire la différence entre fonction d'évaluation et fonction de *fitness*. En effet, il va falloir définir le fonction de *fitness* de telle sorte qu'elle ne favorise pas trop les meilleurs individus, mais aussi qu'elle permette de repérer les individus manifestement mauvais et inutiles pour la recherche. Par contre, lorsque qu'une sélection par tournoi est utilisée, il n'est pas nécessaire de faire de mise à l'échelle, la fonction d'évaluation suffit donc amplement. Dans cette thèse, seule la sélection par tournoi est utilisé, nous utiliserons donc indifféremment les termes évaluation et *fitness*. L'évaluation est aussi l'opérateur le plus dépendant du problème à traiter. On doit généralement définir un opérateur d'évaluation par type de problème, voire par problème.

### Condition de terminaison

La condition de terminaison d'un AG diffère selon le type de problème que l'on traite. Si on traite un problème de décision, c'est-à-dire que l'on connaît la valeur de l'optimum que l'on cherche à atteindre, c'est assez simple. En revanche pour les problèmes d'optimisation, on ne connaît pas l'optimum et on ne sait jamais si celui-ci est atteint. Dans le cas général on se bornera par le nombre de générations maximum à effectuer depuis le début ou depuis la dernière amélioration trouvée. On peut aussi décider d'arrêter lorsqu'il y a convergence prématurée. On parle de convergence prématurée lorsque la diversité entre les individus est trop faible pour espérer sortir d'un bassin d'attraction.

**Sélection**

L'opérateur de sélection est l'opérateur qui va devoir choisir dans la population courante les individus qui auront le «droit» de survivre et de se reproduire. Plus formellement cet opérateur va générer à partir de la population courante $\Pi_{t-1}$, une population $\Pi_{sel}$ par copie (une ou plusieurs) des individus choisis dans $\Pi_{t-1}$. Les opérateurs génétiques seront ensuite appliqués sur cette population $\Pi_{sel}$. Le nombre d'occurrences d'un individu $x_i$ de la population $\Pi_{t-1}$ dans la population $\Pi_{sel}$ tend généralement à être proportionnel au rapport entre sa *fitness* et la *fitness* moyenne. De fait, les individus les mieux adaptés ont le plus de chance de figurer (parfois en plusieurs exemplaires) dans la population $\Pi_{sel}$, et les individus les moins adaptés ont peu de chance d'y figurer. Généralement, la sélection s'effectue sur toute la population $\Pi_{t-1}$, il existe cependant des cas où on peut préférer ne prendre en compte qu'une partie de la population, et générer ainsi un fossé de génération [Gol89]. On aurait ainsi moins de chance de perdre les bons individus, et on permet ainsi le croisement entre individus issus de générations différentes. L'idée est alors de conserver plus de stabilité. L'algorithme génétique stationnaire (*Steady State* en anglais) [Sys89], par exemple, ne choisit que deux parents dans la population courante. Aucune de ces variantes ne s'est réellement détachée des autres à ce jour, cependant la variante *Steady State* est souvent utilisée comme base lors de l'élaboration d'algorithme hybride utilisant la recherche locale (voir section 3.7). Plusieurs méthodes sont possibles pour choisir les individus. Nous allons évoquer ici les plus courantes.

La sélection par roulette place sur une roue des cases de tailles différentes proportionnelles au rapport *meilleure fitness* / *fitness moyenne* de chaque individu. La figure 3.2 montre une roulette de sélection pour une population de cinq individus dont les évaluations figurent à droite de la roulette (l'objectif est ici de maximiser la fonction d'évaluation). On lance ensuite $n$ fois la roue, et pour chaque case désignée, on fait une copie de l'individu correspondant dans la population $\Pi_{sel}$. Des variantes existent, comme par exemple, utiliser le rang des individus plutôt que le rapport *meilleure fitness* / fitness *moyenne*, ce qui permet de moins favoriser les individus dont la *fitness* est très au dessus de la *fitness* moyenne.

La sélection par tournoi (*k-tournament* en anglais), qui a été exclusivement utilisée dans cette thèse, prend au hasard $k$ individus parmi les $n$ de la population $\Pi_{t-1}$, et leur fait faire un tournoi. Celui qui a la meilleure *fitness* est copié dans la population $\Pi_{sel}$. Généralement, le nombre d'individus $k$ est assez faible (2 à 5). Plus ce nombre est élevé, plus les individus ayant une *fitness* élevée seront représentés dans la population $\Pi_{sel}$. Cette méthode présente différents avantages. Elle permet de régler la force de la pression sélective en modifiant la

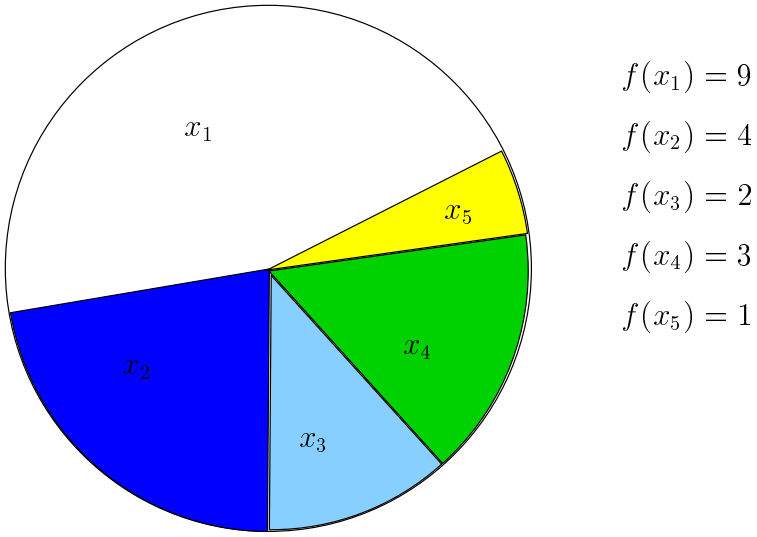


FIG. 3.2 – Roulette de sélection

valeur $k$ et ne nécessite pas de mise à l'échelle.[11]

### Croisement

L'opérateur de croisement est généralement considéré comme l'opérateur d'exploitation. On entend par là qu'il va permettre de découvrir de meilleures solutions en combinant les avantages de solutions déjà découvertes. Dans sa version classique, on va choisir deux individus de la population $\Pi_{sel}$, et leur appliquer l'opérateur de croisement ou bien les recopier dans la population $\Pi_t$ tels quels, en fonction de la probabilité de croisement. Le croisement à un point va créer deux enfants à partir des deux parents de telle sorte que chacun des enfants ait une partie du chromosome de chaque parent. On va pour cela, choisir un entier $i$ tel que $1 \leq i < n$, l'enfant 1 va copier les gènes $1..i$ du parent 1 et les gènes $i + 1..n$ du parent 2, et réciproquement pour l'enfant 2 (figure 3.3). Une variante très couramment utilisée consiste à choisir deux points de croisement, le croisement va alors s'effectuer comme sur la figure 3.4, la position du ou des points de croisement étant généralement choisie au hasard.

Il existe de très nombreuses variations de l'opérateur de croisement, plus ou moins dédiées à des problèmes particuliers. Nous verrons le croisement à n points et le croisement uniforme dans le chapitre 6. Pour une comparaison des opérateurs de croisement les plus connus, nous conseillons [ECS89], et pour une étude théorique [Cer94] et [Spe98].

[11] La mise à l'échelle est une technique de réajustement de l'espérance du nombre de copies d'un individu dans la population $\Pi_{sel}$

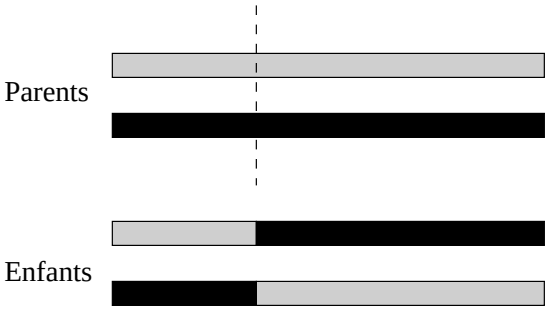


FIG. 3.3 – Croisement à un point

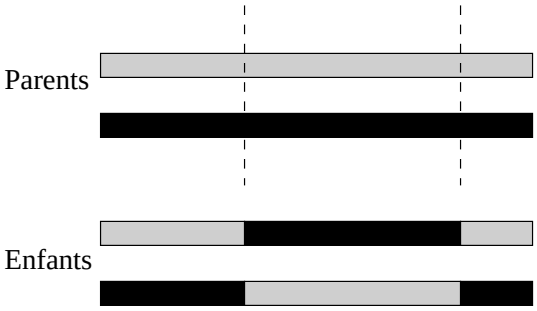


FIG. 3.4 – Croisement à 2 points

#### Mutation

L'opérateur de mutation va permettre l'exploration de l'espace de recherche. En effet, les autres opérateurs ne permettent que des déplacements dans certaines zones de l'espace de recherche (le croisement), voir aucun (la sélection). En utilisant la mutation, on peut théoriquement atteindre n'importe quelle zone de l'espace de recherche. Dans le cas général, cela consiste à effectuer une altération aléatoire d'un (ou plusieurs) gène(s) du chromosome (figure 3.5), en fonction du taux de mutation. Ce taux étant généralement assez faible, entre 0.05 et 0.001. De très nombreuses variations de l'opérateur de mutation ont été proposées dans la littérature, que ce soit en faisant varier le taux en fonction du temps comme le préconise Holland [Hol75], ou encore en particularisant l'opérateur en fonction du problème. Nous verrons quelques unes de ces variations dans le chapitre 6.

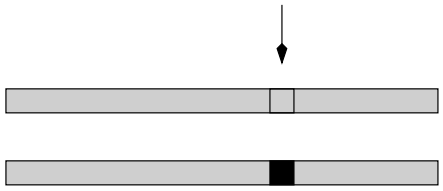

FIG. 3.5 – Mutation

### 3.3.3 Variations

En restant sur le modèle des AG classiques, de très nombreuses variations ont été proposées dans la littérature. Premièrement, chaque type de problème nécessite généralement un codage particulier et une fonction d'évaluation propre, mais cela n'explique pas toute l'«algo-diversité» que l'on peut trouver.

Certaines versions ont exploré le principe de niche écologique, et pour ce faire, diverses méthodes ont été proposées :

- en utilisant une fonction d'évaluation qui va noter plus favorablement les individus peu représentés dans la population [DG89],
- en utilisant un opérateur de croisement restreint qui va interdire les croisement entre les individus trop différents [BBM93].

Ces deux méthodes peuvent généralement être avantageusement combinées. Ce principe de niche écologique est généralement utilisé dans le cadre de problèmes ayant plusieurs optima, et permet à l'AG de converger vers les différents optima simultanément.

Des comportements de société ont parfois été modélisés dans un AG. Par exemple, les algorithmes mimétiques [SSR97] proposés par Sébag et al. modélisent des comportements en fonction de deux individus particuliers (les médiateurs) qui sont le meilleur individu et le pire. Ensuite chaque individu peut choisir d'imiter ou de fuir chacun des médiateurs. On aura des comportements sociaux très différents tel que «mouton», «asocial», «suicidaire» ou encore «brave». Thomsen et al. proposent un algorithme génétique religieux [TRK00], où chaque individu a une religion, et ne pourra en principe que croiser les individus de même religion. Mais des possibilités de convertir des individus, ou même de croiser des individus de religions différentes existent en fonction de différents critères (en fonction de la *fitness*, du nombre d'individus ayant la même religion,...). D'autres comportements ont été modélisés comme la prévention de l'inceste [ES91], ou encore des croisements polygames et polyandres, voir orgiaques [EvKK95], proposant des recombinaisons multi-parentales.

En se rapprochant du modèle biologique, des codages utilisant deux chromosomes par individu (diploïdie), avec des règles de dominance et de récessivité ont été proposés dans [RG93]. Ce modèle est dédié aux environnements changeants, et permet d'obtenir un effet de mémoire. On pourra trouver une généralisation multiploïde dans [CCR96]. Des modèles utilisant plusieurs chromosomes par individu ont aussi été proposés, et à l'inverse certains ont proposé de supprimer la génétique (i.e. le croisement et la représentation chromosomique) de l'AG [BC95]. D'autres encore ont proposé des algorithmes qui utilisent les principes de symbiose et de parasitisme [WRP00, WP00].

## 3.4 Bases théoriques

Cette section est consacrée d'une part aux théories qui tentent d'expliquer pourquoi les algorithmes génétiques convergent, et d'autres part aux cas où ceux-ci rencontrent des problèmes (problèmes dits AG-difficile). La partie concernant la théorie des schémas à été librement empruntée à la thèse de Maria-Cristina Riff-Rojas [Rif97] car elle nous a paru claire, concise et s'intégrant très bien à notre propos.

### 3.4.1 Théorie des *Schémas*

Un des concepts centraux développés dans l'analyse théorique des algorithme génétiques est le concept de "schéma" [Hol75]. Pour comprendre cette notion, nous devons adopter le point de vue suivant : pour guider sa recherche, un algorithme génétique qui agit sur une population traite en fait des entités appelées "schémas" et tente de découvrir les meilleurs schémas et de les combiner.

**Définition 3.4.1** *(Schéma)*
*Un schéma $H$ décrit un sous-ensemble de chromosomes ayant des caractéristiques communes à certaines positions de la chaîne de caractères.*

C'est donc une hyper-surface particulière de l'espace de recherche. L'ensemble de tous les schémas couvre la totalité de l'espace de recherche avec une redondance élevée. Un schéma $H$ peut être défini sur l'alphabet $(0, 1, \#)$, où le métasymbole "#" est le symbole de l'indifférence. Par exemple, le schéma $1\#\#\#0$ désigne tous les chromosomes qui commencent par 1 et qui finissent par 0. En conséquence, tout chromosome de longueur $l$ est un représentant de $2^l$ schémas. La fonction d'évaluation ou performance d'un schéma $H$, $f(H)$, correspond à la valeur moyenne de la fonction d'évaluation du sous-ensemble de chromosomes qu'il décrit. Certains schémas sont plus spécifiques que d'autres. Par exemple, le schéma $01\#\#1$ est plus spécifique que le schéma $1\#\#\#0$. De plus, certains schémas couvrent une plus grande partie de la longueur totale de la chaîne de caractères que d'autres. Par exemple, le schéma $1\#\#\#0$ couvre une plus grande portion de la chaîne que $1\#0\#\#$. Pour quantifier ces notions, il est nécessaire de définir deux caractéristiques propres aux schémas : *ordre* et *longueur utile*

**Définition 3.4.2** *(Ordre d'un schéma)*
*L'ordre $o(H)$ d'un schéma $H$ est le nombre de positions fixes (le nombre de uns et de zéros) dans $H$. Par exemple $o(\#1\#00) = 3$.*

**Définition 3.4.3** *(Longueur utile d'un schéma)*
*La longueur utile $\delta(H)$ d'un schéma $H$ est la distance entre la première et la dernière position fixée dans $H$, donc correspond à l'écart maximal entre deux bits fixés. Par exemple, $\delta(\#1\#00) = 5 - 2 = 3$.*

Le concept de schéma et les propriétés d'ordre et de longueur permettent d'analyser l'effet de la reproduction sur le nombre attendu de schémas d'un certain type dans la population. Supposons qu'à l'instant $t$, il y ait $m$ exemplaires d'un schéma $H$ contenus dans la population $P(t)$ dénoté par $m(H, t)$ et soit $\hat{f}(H)$ la moyenne des adéquations des

$m$ individus. Lors de la reproduction, un chromosome $C_{bin}^{p}$ est copié en fonction de son adéquation (sa valeur de la fonction d'évaluation) avec la probabilité $Prob_p = \frac{f_p}{f_{moyen}}$, où :

$$f_{moyen} = \frac{(\sum_{p=1}^{n} f_p)}{n} \tag{3.2}$$

Soit $n$ la taille de la population depuis la population $P(t)$, soit $m(H, t+1)$ l'espérance de nombre de représentants du schéma H dans la population à l'instant $t+1$. L'équation de développement par reproduction des schémas (avant de faire une transformation) est la suivante :

$$m(H, t+1) \geq m(H, t)\frac{\hat{f}(H)}{f_{moyen}} \tag{3.3}$$

Le développement d'un schéma dépend du rapport de la fonction d'évaluation du schéma dans la population sur la valeur moyenne de la fonction d'évaluation de la population. Les schémas au-dessus de la moyenne auront un nombre de représentants croissant exponentiellement, tandis que les schémas dont les valeurs sont en-dessous de la moyenne seront moins copiés. La survie d'un schéma dépend aussi du croisement et de la mutation. Considérons l'opérateur de croisement à un point qui choisit aléatoirement un point où couper les chaînes de caractères des parents. Il y a $l-1$ points possibles de croisement pour choisir. La probabilité qu'un schéma $H$ dépérisse dépendra de son ordre $o(H)$ et de sa longueur utile $\delta(H)$. Plus les positions d'un schéma sont fixées et plus elles se trouvent éloignées, plus sa probabilité de disparition augmente. La probabilité de disparition d'un schéma est $Prob_d = \frac{\delta(H)}{(l-1)}$, donc sa probabilité de survie est $Prob_s = (1 - Prob_d)$. Une borne inférieure de survie après croisement pour un schéma $H$ est :

$$Prob_s \geq 1 - Prob_c\frac{\delta(H)}{l-1} \tag{3.4}$$

où $Prob_c$ est la probabilité d'effectuer un croisement. Pour qu'un schéma $H$ survive, toutes ses positions fixées doivent survivre. Si on considère la mutation, chaque allèle survit avec la probabilité $(1-Prob_m)$, où $Prob_m$ est la probabilité de mutation. Un schéma quelconque survit quand chacune des $o(H)$ positions fixées dans le schéma n'a pas subi un changement, puisque chaque mutation est indépendante. En multipliant la probabilité de survie par elle-même $o(H)$ fois, on obtient la probabilité de la survie à la mutation $(1-Prob_m)^{o(H)}$. Quand $Prob_m \ll 1$ la probabilité peut être approximée par $1 - o(H)Prob_m$, [Gol89]. En général, la probabilité de mutation dans un algorithme génétique est assez petite, cette condition est donc valable. En considérant les effets du croisement et de la mutation ensemble, nous obtenons l'expression suivante pour la survie d'un schéma $H$ :

$$Prob_s \geq 1 - Prob_c\frac{\delta(H)}{l-1} - o(H)Prob_m \tag{3.5}$$

Finalement pour être plus précis, nous incorporons ces résultats dus aux opérateurs, dans l'équation 3.3, et nous obtenons le Théorème Fondamental des Algorithmes Génétiques :

**Théorème 3.4.4** *(Théorème des Schémas)*

$$m(H,t+1) \geq m(H,t)\frac{\hat{f}(H)}{f_{moyen}}\left(\frac{\delta(H)}{1-Prob_c l} - (1-o(H)Prob_m)\right) \tag{3.6}$$

**Hypothèse 3.4.5** *(Blocs de Construction)*
*Un algorithme génétique cherche à accroître sa performance près de l'optimum en utilisant la juxtaposition des schémas ayant une fonction d'évaluation au-dessus de la moyenne, courts, et peu spécifiques ($o(H)$ petit). Ces schémas sont dénommés blocs de construction.*

En utilisant le Théorème des Schémas nous pouvons prévoir que le nombre d'occurrences d'un schéma court et performant augmentera en moyenne d'une manière exponentielle d'une génération à la suivante.

**Définition 3.4.6** *(Parallélisme Implicite) L'évaluation de la performance d'un chromosome est également l'évaluation de tous les schémas auxquels il appartient.*

L'augmentation exponentielle des blocs de construction est dénommée "parallélisme implicite". Holland a estimé qu'avec une population de taille "n" pour chaque génération, un algorithme génétique utilise implicitement $O(n^3)$ schémas.

Une généralisation du théorème des schémas à été proposée par Radcliffe [Rad91]. Cette généralisation s'appelle théorie des *formae* et à l'avantage de ne s'attacher à aucune représentation particulière. Par définition, les *formae* représentent des ensembles de solutions qui partagent certaines propriétés supposées être avantageuses par rapport à l'évaluation.

Le Théorème des Schémas aide à comprendre comment les algorithmes génétiques travaillent, mais il ne constitue pas une preuve du succès et il n'existe pas encore d'analyse stochastique rigoureuse sous des circonstances réelles (telles qu'une population finie, une fonction d'évaluation non triviale, des opérateurs spécialisés). Le lecteur intéressé par les théories concernant les preuves de convergence de AG pourra consulter [RW91, NV92, GM00]

### 3.4.2 Problèmes trompeurs et *Royal Roads*

Le principe de problèmes trompeurs (AG-difficile) a été défini à partir d'observations faites sur des relations particulières entre la représentation des problèmes et leurs fonctions d'évaluation [Gol89, Rad91]. Formellement, une fonction est dite trompeuse si des schémas d'ordre $n$ ne contenant pas d'optima globaux ont une meilleure performance moyenne que les schémas de même ordre contenant des optima globaux. Plus intuitivement, s'il existe une

zone de l'espace de recherche ne contenant pas d'optima globaux, mais contenant beaucoup de bonnes solutions, et que les zones de l'espace de recherche contenant des optima globaux sont situées dans des zones contenant beaucoup de mauvaises solutions, l'AG aura tendance à faire converger l'ensemble de la population vers la zone la plus prometteuse à savoir celle qui contient le plus de bonnes solutions en moyenne. Une fois dans ce sous-espace, il ne pourra pas reconverger vers les zones contenant des optima globaux.

Les *Royal Roads* [MFH92] sont des fonctions qui ont été conçues au départ pour être facilement optimisables par un AG. Elles se sont finalement révélées comme particulièrement difficiles. Le principe de ces fonctions est de valider l'hypothèse des blocs de construction en récompensant explicitement la présence de certains *building blocks* dans les chromosomes. Le principe de ces fonctions est de récompenser la présence de certains schémas dans le chromosome, la fonction d'évaluation se contentant de cumuler les récompenses des différents schémas présents, le but étant de maximiser le nombre de 1 dans le chromosome. La figure 3.6 présente les schémas récompensés pour des chromosomes de soixante-quatre bits. Par exemple, un chromosome dont les seize premiers bits sont à 1 et contenant en tout trente bits à 1 sera évalué de la façon suivante :

- 30 pour les trente bits à 1
- 8*2 pour les schémas de huit 1 consécutifs (les huit premiers et les huit suivants)
- 16 pour les seize bits consécutifs

Soit une évaluation totale de 62. Ce système de récompense des blocs de construction favo-

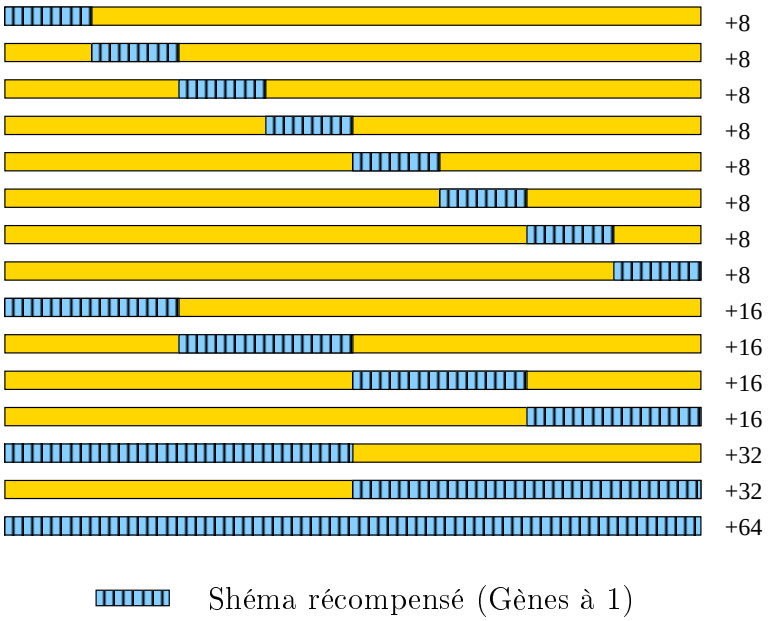


FIG. 3.6 – Principe de récompense des *Royal Roads*

rables devait permettre de favoriser la convergence de l'AG. Cependant, l'étude empirique de ces fonctions a très largement contrarié les attentes de leurs concepteurs car elles se sont révélées difficiles. La raison de la divergence des AG sur de telles fonctions à été dénommée *Hitch Hiking* par Forrest, Mitchell et Holland. Le principe est que les schémas fortement récompensés, mais dont les solutions contiennent des parasites, «emmènent» la population vers des zones de l'espace de recherche non prometteuses. Les solutions plus proches de l'optimal mais ne contenant pas (ou peu) de schémas récompensés sont alors perdues. Par exemple un chromosome contenant sept 1 consécutifs puis un 0, puis à nouveau sept 1 consécutifs et ainsi de suite n'obtiendra qu'une évaluation de 56, ce qui est moins bon que l'exemple précédent (seize 1 consécutifs et trente 1 en tout) mais qui est nettement plus proche de la solution au niveau génotypique. On notera que la formalisation des *Royal Roads* engendre des optima locaux dont il est difficile de sortir, alors que la fonction sans récompense des blocs de constructions n'en contenait aucun. On appellera effet *Royal Road* la capacité d'une fonction à récompenser des blocs de construction aux dépens d'autres blocs.

## 3.5 Programmation Génétique

La programmation génétique introduite par [Koz92], dont le modèle qui s'inspire des algorithmes génétiques, a pour but de créer automatiquement un programme pour résoudre un problème. Pour cela on va utiliser non pas des chromosomes codant pour une solution, mais de petits programmes qui vont évoluer et s'assembler pour fournir un programme plus complet. Ces programmes sont en général des S-expressions Lisp, qui ont l'avantage d'être facilement représentés par des arbres syntaxiques, où les noeuds sont des opérateurs, et les feuilles des variables ou des valeurs. L'algorithme utilisé pour la programmation génétique, suit le même schéma qu'un algorithme génétique standard : après une initialisation aléatoire (chaque individu est un programme) on effectue la boucle suivante un certain nombre de fois :

- Evaluation des individus (en mesurant l'adéquation entre le programme, et la solution voulue) et sélection
- Application des opérateurs génétiques

Nous allons illustrer chacune des étapes utilisées en programmation génétique à partir d'un exemple simple. Le problème qu'on se pose est de retrouver l'équation d'une courbe, par exemple la représentation d'un polynôme. On dispose des coordonnées de différents points de cette courbe. Le but est alors de trouver une fonction qui approxime ce polynôme. L'initialisation va consister à générer un ensemble de petits programmes (dans notre exemple des fonctions simples) sous forme d'arbres syntaxiques de S-expressions. Pour évaluer les

programmes, on va comparer les valeurs des ordonnées des points avec les valeurs retournées par les programmes pour un certain nombre d'abscisses. La mutation va consister à remplacer un opérateur par un autre si on est sur un noeud, ou bien remplacer une valeur ou variable par une autre si on est sur une feuille. Le croisement va permettre à deux programmes de s'échanger des sous-arbres syntaxiques. L'idée principale de la programmation génétique est que des S-expressions de petite taille dont la *fitness* est élevée vont se recombiner en formant des S-expressions plus grandes et de *fitness* plus élevées.

La programmation génétique permet en général de manipuler des structures complexes, permettant d'optimiser des réseaux de neurones ou encore des automates cellulaires. Ce modèle va permettre, en utilisant des structures simples au départ, de créer une structure plus complexe dont le comportement devra répondre aux attentes de l'utilisateur. Le lecteur intéressé par la programmation génétique pourra consulter [Mic94].

## 3.6 Évolution Stratégique

Les algorithmes d'évolutions stratégiques (SE) ont été développés à peu près à la même époque que les AG et indépendamment par Rechenberg [Rec73] et améliorés plus tard par Schwefel. Contrairement aux AG qui utilisent des chaînes binaires, les SE ont été développées pour travailler sur les réels. La version initiale des SE n'utilise qu'un seul individu qui peut subir des mutations, la sélection choisissant le meilleur du couple parent/enfant : c'est le (1+1)-SE. Ensuite des variantes ont utilisé une population d'individus. Ce sont les algorithmes $(\mu + \lambda)$-SE et $(\mu, \lambda)$-SE. La population $\Pi_t$ est un ensemble de $\mu$ individus qui, par recombinaison et mutation, génèrent $\lambda$ nouveaux individus. La sélection se fait parmi les $(\mu + \lambda)$ individus pour la stratégie $(\mu + \lambda)$-SE et parmi les $\lambda$ nouveaux individus pour l'algorithme $(\mu, \lambda)$-SE pour obtenir la population $\Pi_{t+1}$. On pourra trouver une comparaison entre SE et AG dans [HB91].

## 3.7 Approches hybrides

De nombreuses approches hybrides ont été proposées dans la littérature, et se sont souvent révélées très efficaces sur des problèmes donnés. Dans la plupart des cas, ces approches ne sont efficaces que sur le type de problème pour lequel elles ont été développées, mais sont aussi généralement d'une grande efficacité par rapport aux autres méthodes. La raison principale est généralement que l'opérateur de croisement doit être spécialisé pour ne pas avoir un effet destructeur sur les solutions potentielles découvertes par la recherche locale. Nous avons aussi volontairement classé le principe d'opérateurs spécialisés parmi les approches hybrides.

### Avec la liste Tabou

Une approche hybride d'algorithme génétique avec la méthode Tabou est proposé par Galinier dans [Gal99]. Après une initialisation où chaque individu subit une recherche Tabou sur un certain nombre d'itérations, on entre dans la boucle de l'AG. A chaque génération deux individus sont choisis aléatoirement dans la population[12], un croisement spécifique qui génère un enfant est effectué. Ensuite, cet individu subit là aussi une recherche Tabou pendant un certain nombre d'itérations.

Cet algorithme hybride a obtenu de très bons résultats sur les coloriages de graphe issus du challenge DIMACS [JT93], améliorant parfois les meilleures bornes connues.

### Avec le Recuit Simulé

Plusieurs algorithmes hybrides utilisant les AG et le recuit simulé ont été proposés, dont le fonctionnement est fondamentalement différent. Brown et al. [BHS89] proposent une version où chaque individu subit une amélioration par recuit simulé à chaque génération. Koakutsu et al. [SK90, KKD96] proposent deux versions différentes. La première (PGSA) est un algorithme de recuit simulé parallèle qui échange et croise certains individus, la deuxième (GSA) est basé sur un algorithme *Steady State*. Deux individus choisis au hasard vont être croisés pour ne donner qu'un seul enfant, celui-ci remplace le pire individu de la population. Cet individu est ensuite passé à un processus de recuit simulé pendant une certain nombre d'itérations.

### Algorithmes Mémétiques

Les algorithmes Mémétiques ont été proposés par Moscato et Norman [MN89]. Le principe de base est d'utiliser un AG dont la population est constituée exclusivement de minima locaux. Pour ce faire, à chaque génération les individus subissent un processus de descente stricte, qui s'arrête comme nous l'avons vu au premier optimum local trouvé. Aujourd'hui, les auteurs proposent le concept d'algorithme mémétique comme cadre général pour les méthodes hybrides basées sur les algorithmes évolutionnaires et la recherche locale ( quatrième partie de [CDG99]) et affirment que nombres de travaux récents s'intègrent à ce modèle.

Durant la thèse, nous avons expérimenté un algorithme génétique qui intègre une recherche de type Metropolis[13]. Cet algorithme fait subir à une partie de la population (de

[12] Il s'agit donc plus d'un algorithme de type *steady state*

[13] Sorte de Recuit simulé à température contante (voir 2.2.2 page 24)

10% à 50%) un processus de Metropolis pendant un certain nombre d'itérations à intervalle régulier (généralement une période de 1 à 30 générations). Mais ces expérimentations ne se sont pas révélées concluantes et ne seront pas traitées dans ce manuscrit.

#### Avec des Opérateurs Spécialisés

Beaucoup d'approches que l'on peut classer dans les algorithmes hybrides consistent à spécialiser les opérateurs génétiques en fonction du problème. Par exemple on peut faire une recherche exhaustive du meilleur voisin lors d'une mutation, ou bien effectuer un croisement qui prend en compte la structure du problème en interdisant les croisements sur des loci inappropriés. Dès que l'on sort du domaine de l'étude des algorithmes génétiques sur des fonctions dédiées (comme les fonctions trappes, les *Royal Road*,...), et qu'on s'attaque à des problèmes particuliers, on va chercher à intégrer une partie de la connaissance particulière que l'on a des problèmes. La littérature foisonne de contributions aussi diverses que variées intégrant des opérateurs génétiques qui utilisent des heuristiques connues pour leur efficacité. Sans intégration de ces connaissances, les AG se révèlent généralement inefficaces par rapport aux autres méthodes, mais peuvent devenir d'une efficacité redoutable si cette intégration est bien faite.

## 3.8 Conclusion

Après avoir entrevu les fondements biologiques qui ont conduit à la conception des méthodes évolutionnaires, nous avons présenté les algorithmes génétiques dans leur version canonique. Nous avons aussi présenté quelques variations et hybridations connues. Le domaine d'application des AG est assez large, que ce soit pour optimiser des fonctions complexes, faire de l'optimisation multi-critère mélangeant le continu et le discret, faire de la reconnaissance d'images, optimiser des structures comme les ailes d'avions,... L'autre avantage des AG est que par nature, ils n'ont pas besoin d'avoir une connaissance du problème, mais seulement d'une fonction qui mesure l'adéquation entre les données en entrée et les donnée en sortie. On appelle ce type de traitement optimisation de «boîtes noires».

Les algorithmes génétiques ont connu un grand succès en partie grâce à la rapidité et la facilité d'implantation qu'ils permettent. Cependant, pour être efficaces, ils nécessitent généralement une forte spécialisation, et une intégration des connaissances du problème à traiter. Ceci va à l'encontre de la facilité de développement. Plus on va spécialiser l'algorithme, plus l'implantation va être longue, plus le réglage des paramètres sera difficile, et plus le domaine d'application sera restreint. C'est pour ces raisons que dans cette thèse nous sommes plus intéressé aux AG classiques, en essayant de rester au plus près d'une

certaine généricité.

Nous allons, dans le chapitre suivant, présenter les spécificités des algorithmes génétiques lorsqu'ils sont développés afin de résoudre des CSP.

# Chapitre 4

# Algorithmes évolutionnaires et CSP

## 4.1 Introduction

Les deux premiers chapitres de cette thèse étaient consacrés aux problèmes de satisfaction de contraintes en domaines finis. Nous avons vu les méthodes les plus courantes utilisées pour les traiter. Le chapitre précédent présentait les algorithmes évolutionnaires de manière générale. Ce chapitre est consacré à l'application des méthodes évolutionnaires aux CSP. Nous évoquerons la problématique de cette application ainsi que différentes approches qui ont été proposées. Enfin, nous terminerons par une brève présentation du moteur évolutionnaire que nous avons utilisé pour nos expérimentations.

## 4.2 Préliminaires

Les problèmes de satisfaction de contraintes ne sont pas à proprement parler des problèmes d'optimisation. Le but, lorsqu'on traite un CSP est de trouver une solution qui satisfasse toutes les contraintes, quelles qu'elles soient. Si on veut en plus trouver la meilleure solution parmi celles qui satisfont toutes les contraintes, et à condition d'avoir une fonction de coût qui permette d'évaluer les solutions, on entre dans le cadre des problèmes d'optimisation sous contraintes (COP[1]) [MJ91, SX93] qui ne sont pas traités ici. Lorsqu'on veut appliquer les algorithmes évolutionnaires aux CSP [ERR94], plusieurs problèmes se posent :

- Les algorithmes évolutionnaires sont des algorithmes d'optimisation, on se doit donc de transformer le problème de satisfaction en problème d'optimisation.
- Quel codage utiliser pour les chromosomes, le codage binaire étant généralement inadapté ?

[1] Contrained Optimisation Problem

- Les opérateurs génétiques classiques peuvent ils être utilisés, ou bien faut il en définir de nouveaux ?
- ...

Nous allons présenter dans la suite de ce chapitre, les approches les plus courantes.

## 4.3 Approches

Cette section présente différentes approches, généralement utilisées conjointement. Nous allons tout d'abord présenter différentes manières de coder un CSP dans un AE, puis nous présenterons quelques fonctions d'évaluations parmi les plus courantes, enfin évoquerons l'approche de réparation.

### 4.3.1 Codage

Le premier problème auquel on est confronté lorsqu'on veut appliquer les algorithmes évolutionnaires aux CSP est le problème du codage. Les AE standards utilisent un codage binaire qui est souvent mal adapté aux CSP, cependant, certains problèmes peuvent l'utiliser. C'est le cas par exemple, du problème des $n$-reines (voir section 1.4 page 11), où on peut représenter la présence ou l'absence d'une reine sur une case par des variables binaires. Le problème du sac à dos peut aussi se représenter facilement par des variables binaires, la présence ou l'absence d'un objet dans le sac étant là aussi représenté par les variables 0 ou 1. Cependant la majeur partie des CSP ne peuvent se représenter aussi facilement par des variables binaires sans que la transformation ne soit coûteuse.

Les problèmes de coloriage de graphes se prêtent assez mal à une représentation binaire car certaines solutions n'auraient alors aucun sens. Pour coder sous forme de chromosome un coloriage de graphe, on va généralement représenter la couleur affectée à un nœud du graphe par un gène dans le chromosome. C'est le codage utilisé pour les gènes qui peut poser problème si on utilise un codage binaire. Prenons l'exemple d'un 3-coloriage, chaque gène devra représenter une couleur possible pour le nœud correspondant. Si on utilise un codage binaire, on va utiliser deux bits pour chaque gène, mais ceux-ci permettent de coder quatre couleurs, donc les croisements et les mutations vont pouvoir proposer des solutions qui vont utiliser des valeurs qui ne font pas partie du domaine des variables. Une manière simple de remédier à ce problème est d'utiliser un codage en nombre entier [Evv98]. On limitera les entiers utilisables au domaine des variables. L'avantage de cette solution est qu'elle permet d'utiliser les opérateurs génétiques standards sans modification. On pourra bien sûr utiliser aussi des opérateurs spécialisés.

Une deuxième solution pour les coloriages de graphe, est d'essayer de découvrir les en-

sembles indépendants du graphe. Pour se faire, on va subdiviser le chromosome en zones qui correspondront aux couleurs utilisées. Chaque zone va recevoir un ensemble de nœuds qui devront minimiser le nombre de contraintes les reliant. Le but étant à terme d'avoir dans chaque zone des nœuds qui ne sont reliés par aucune contrainte. Une contrainte globale s'ajoute à cela : tous les nœuds du graphe doivent être présents une et une seule fois dans le chromosome. Le problème de cette représentation est qu'elle impose des opérateurs de croisement et de mutation spécifiques. L'opérateur de croisement doit respecter la contrainte globale sous peine d'obtenir des solutions incohérentes, de même que l'opérateur de mutation. Il existe plusieurs croisements spécifiques pour cette représentation dont on trouvera une description dans [Gal99]. L'opérateur de mutation va généralement consister à intervertir deux nœuds dans le chromosome.

Pour les problèmes du type «Voyageur de commerce», on pourra utiliser un codage ordonné des variables du problème. Le problème du voyageur de commerce consiste à trouver le chemin le plus court reliant un ensemble de villes. Pour coder un tel problème sous forme d'un chromosome, on va en fait coder l'ordre dans lequel les villes sont traversées. La fonction d'évaluation va simplement cumuler le nombre de kilomètres parcourus, fonction qu'il s'agira de minimiser. Dans ce cas aussi, les opérateurs génétiques standards ne peuvent être utilisés, on devra donc utiliser des opérateurs spécialisés qui prennent en compte ce codage spécifique.

### 4.3.2 Pénalisation

Comme toutes les méthodes qui manipulent des solutions potentielles globalement incohérentes, les AE doivent se munir d'une fonction qui puisse évaluer une solution en fonction du nombre de contraintes satisfaites. Plusieurs fonctions ont été proposées dans la littérature afin d'évaluer les solutions courantes. Ce sont des fonctions qui pénalisent les solutions les moins adaptées.

La plus simple se contente de compter le nombre de contraintes non satisfaites [Tsa90] :

**Définition 4.3.1** *(Fonction d'évaluation standard)*
*Soit $Tc$ la fonction qui teste la cohérence d'une contrainte $\mathcal{C}_i$ pour une instanciation $\mathcal{I}$*

$$\forall i, Tc(\mathcal{C}_i, \mathcal{I}) = \begin{cases} 0 & si \quad \mathcal{C}_i \quad est \quad satifaite \\ 1 & sinon \end{cases}$$

*la fonction d'évaluation standard $E_{std}$ se calcule comme suit :*

$$E_{std}(\mathcal{I}) = \sum_{i=0}^{m} Tc(\mathcal{C}_i, \mathcal{I}) \tag{4.1}$$

Le but est alors de minimiser cette fonction. D'autres fonctions pénalisantes ont été proposées [ER96], en affectant des poids différents aux contraintes violées :

$$E_{pond}(\mathcal{I}) = \sum_{i=0}^{m} w_i \times Tc(\mathcal{C}_i, \mathcal{I}) \tag{4.2}$$

où $E_{pond}$ est la fonction d'évaluation pondérée et $w_i$ est le poids affecté à la contrainte $\mathcal{C}_i$.

Ces fonctions pénalisantes sont basées sur les contraintes, c'est-à-dire qu'elles prennent en compte les contraintes non satisfaites pour calculer la pénalisation infligée à la solution potentielle. On peut aussi calculer cette pénalisation en fonction des variables impliquées dans des contraintes non satisfaites :

**Définition 4.3.2** *(Fonction d'évaluation par rapport aux variables)*
*Soit $Vc$ la fonction qui teste la cohérence de l'ensemble des contraintes $\mathcal{C}_j^i$ où la variable $\mathcal{X}_i$ est impliquée, pour une instanciation $\mathcal{I}$*

$$\forall i, Vc(\mathcal{X}_i, \mathcal{I}) = \begin{cases} 0 & si \quad \forall j, \mathcal{C}_j^i \quad est \quad satifaite \\ v_i & sinon \end{cases}$$

*La fonction d'évaluation prenant en compte les variables se calcule alors comme suit :*

$$E_{var}(\mathcal{I}) = \sum_{i=0}^{n} Vc(\mathcal{X}_i, \mathcal{I}) \tag{4.3}$$

La difficulté inhérente aux deux dernières fonctions d'évaluation est de fixer correctement $w_i$ et $v_i$. Pour ce faire, il faut savoir évaluer les contraintes les plus difficiles à satisfaire dans le premier cas, et les variables qui vont poser le plus de problèmes dans le deuxième cas. Pour palier cette difficulté, des schémas adaptatifs comme SAW[2] [CEM00] ont été proposés, qui font augmenter progressivement et périodiquement les poids. Cependant, ces fonctions d'évaluation adaptatives se révèlent très difficiles à régler. Eiben lui-même reconnaît que son algorithme SAW, bien qu'efficace, a des paramètres dont les réglages sont très sensibles et nécessite une longue période de mise au point. Ce type de fonction adaptative est généralement plus efficace sur des problèmes dont la structure est déséquilibrée. On entend par là, des problèmes dont certaines contraintes sont beaucoup plus dures à satisfaire que d'autres. Elles permettent de repérer les contraintes les plus dures et d'augmenter leur poids, ces contraintes seront donc les premières à être satisfaites. Par contre, cette approche est manifestement inadaptée pour des problèmes très structurés, et contenant beaucoup de contraintes toutes aussi difficiles à satisfaire. En effet, l'algorithme va commencer par augmenter le poids des contraintes non satisfaites au départ, les satisfaire, ensuite d'autres

[2]Stepwise Adaptive Weight

contraintes vont être violées, il va donc augmenter leur poids jusqu'à dépasser le poids affecté aux premières contraintes, et on risque alors d'obtenir un effet de bouclage. Un autre cas où ces méthodes sont inadaptées est le traitement de CSP sur-contraints pour lesquels on peut préférer satisfaire un maximum de contraintes, quitte à laisser les plus dures non satisfaites.

### 4.3.3 Réparation

L'approche de réparation concerne en général les COP, pour lesquels on veut d'abord satisfaire toutes les contraintes, puis trouver la meilleure solution parmi celles qui ne violent aucune contrainte. On parlera alors de solutions infaisables (i.e. qui ne satisfont pas toutes les contraintes) et de solutions faisables (i.e. dont toutes les contraintes sont satisfaites). On peut alors décider que l'algorithme évolutionnaire ne travaille que dans l'espace des solutions faisables. On va donc à chaque génération *réparer* les solutions potentielles infaisables, pour les transformer en solutions faisables. On peut aussi utiliser cette approche de réparation pour certains problèmes sur-contraints qui contiennent des ensembles de contraintes qu'il est impératif de satisfaire (contraintes dures), et d'autres qui peuvent ne pas être satisfaites (contraintes molles). On va donc réparer les solutions potentielles afin qu'elles satisfassent toutes les contraintes dures.

Dans cette thèse, nous avons traité des coloriages sur-contraints dans lesquels toutes les contraintes sont des contraintes molles, nous ne détaillerons donc pas les méthodes de réparations.

## 4.4 AgCSP

De nombreux moteurs d'algorithmes évolutionnaires sont disponibles et libres de droits.[3] Ces moteurs proposent différentes approches évolutionnaires, que ce soient les algorithmes génétiques, la programmation génétique ou encore les stratégies d'évolutions. Ce sont généralement des outils d'étude et de recherche, et n'ont aucune prétention d'outils professionnels. Nous nous sommes tout d'abord intéressé à des bibliothèques telles que SuGal[4] et GAlib [Wal96]. Cependant, un doctorant du projet, Fabrice Didierjean, finissait de développer sa propre bibliothèque d'algorithmes évolutionnaires [Did02][5] et nous avons préféré utiliser cette dernière. Ce choix a été motivé par différentes raisons :

[3] Dont un certain nombre est disponible à http ://www.aic.nrl.navy.mil :80/galist/src/
[4] Par Andrew Hunter : http ://www.dur.ac.uk/andrew1.hunter/Sugal/
[5] Disponible à http ://AgCSP.sourceforge.net/

- Cette bibliothèque a été développée expressément pour traiter des CSP en domaines finis
- Elle propose une interface graphique de définitions des problèmes, et de tests interactifs
- son auteur était disponible en cas de problèmes d'implantation !

Nous allons d'abord décrire le modèle évolutionnaire utilisé par cette bibliothèque appelée AgCSP, puis nous présenterons ses caractéristiques principales.

AgCSP utilise un codage des chromosomes en nombre entier[6], et propose la fonction d'évaluation pénalisante standard, ainsi que l'évaluation adaptative SAW vues dans la section précédente. La plupart des opérateurs génétiques standards sont disponibles (croisement à un point, croisement à p-points, croisement uniforme, mutation allélique classique, mutation swap, ...). Deux modes de sélection (Roulette et Tournoi) sont proposés, ainsi que deux initialisations (aléatoire et gloutonne). Trois critères de terminaison (nombre de générations, nombre d'évaluations, solution trouvée) sont disponibles et peuvent être combinés. De plus, AgCSP propose une version classique et une version *Steady State* de l'algorithme évolutionnnaire. Toutes ces fonctionnalités peuvent être aisément paramétrés grâce à l'interface graphique.

Bien qu'AgCSP a été développée pour traiter des problèmes de satisfaction de contraintes (pour l'instant seuls les coloriages de graphe et les CSP aléatoires peuvent être traités), elle peut également être utilisée pour traiter d'autres types de problèmes. Si le problème peut être représenté sous forme d'un ensemble de contraintes, dans ce cas il n'y aura aucun travail supplémentaire. Si par contre il s'agit d'un problème d'optimisation d'une fonction par exemple, il faudra au moins écrire un opérateur d'évaluation spécifique à cette fonction. Le fait de ne pas utiliser de contraintes ne ralentit pas les traitements, cela ne fait que rendre inutile une partie du programme

Cependant, dixit son auteur, «cette bibliothèque est un outil de recherche en version Alpha, il n'est donc en aucun cas jugé comme fiable et stable. Beaucoup de travail reste à faire».

Nous allons maintenant décrire les caractéristiques de cet outil. La bibliothèque AgCSP est écrite en C++ et est composée de 2 programmes.

- AgCSP : Il est utilisé pour résoudre un problème à partir de la ligne de commande.
- wxAgCSP. C'est une version graphique d'AgCSP. Elle peut être utilisée pour examiner et modifier dynamiquement les paramètres de l'AE.

Ces deux programmes utilisent un fichier de configuration en ASCII. AgCSP et wxAgCSP utilisent les mêmes commandes pour être configurés. De plus, wxAgCSP peut être utilisé pour créer ces fichiers de configurations, ne nécessitant ainsi aucune connaissance de leurs

[6] L'ordre des variables dans le chromosome étant définie par le fichier d'entré (Dimacs par exemple)

formats internes. Ces commandes servent à décrire le problème que l'on cherche à résoudre.

Voici les caractéristiques principales de ces programmes :

- Configuration grâce à un fichier texte
- Génération de ces fichiers grâce à une interface graphique
- Chargement des graphes (ASCII) définis par le DIMACS Challenge
- Chargement des CSP aléatoires générés par urbcsp[7]
- Conception objet des algorithmes.
- Configuration grâce aux outils GNU (tester sous Linux et SunOS)
- Dynamicité des paramètres possibles
- Insertion et suppression automatique d'opérateurs dans la librairie.
- Génération et intégration automatique de squelettes de nouveaux opérateurs
- Opérateurs sous forme de *plugins* : c'est-à-dire qu'on n'est pas obligé de tout recompiler à chaque modification.
- Intégration automatique de ces nouveaux opérateurs dans l'interface graphique

En plus de ces caractéristiques techniques, AgCSP permet, toujours avec le système de *plugins*, d'ajouter de nouvelles fonctionnalités telles que le chargement de nouveaux types de problèmes, la définition de nouveaux algorithmes évolutionnaires, ou encore d'opérateurs globaux qui permettront d'implanter par exemple, des algorithmes hybrides. Nous avons participé à l'enrichissement de cette bibliothèque durant la thèse et présentons dans le chapitre 6 de nouveaux opérateurs qui participent à cet enrichissement.

## 4.5 Conclusion

Nous avons présenté dans ce chapitre les approches classiques d'application des algorithmes évolutionnnaires aux CSP, pour un tour d'horizon de ces approches, nous conseillons au lecteur [Coe99, Eib01]. Nous avons aussi présenté la bibliothèque et les programmes associés qui nous ont permis de faire toutes nos expérimentations. Cette bibliothèque n'est pas parfaite, et se révèle de par sa généricité plus lente que d'autres bibliothèques telle que GALLOPS [Goo95]. Mais ses nombreux avantages, particulièrement en terme d'adaptabilité et d'accessibilité, nous ont conduit à la préférer aux autres.

[7] http ://www.lirmm.fr/ bessiere/generator.html

# Troisième partie

# Contributions

# Chapitre 5

# Coloriage de graphes sur-contraints

## 5.1 Introduction

Nous nous intéressons dans cette partie aux coloriages de graphes sur-contraints. Dans ce cas, on cherche à colorier un graphe en utilisant un nombre insuffisant de couleurs. Le but est alors de minimiser le nombre de conflits. Nous comparons les résultats obtenus par différentes méthodes sur quelques graphes particuliers. L'idée ici n'est pas de chercher le meilleur algorithme, ni le meilleur réglage, mais de s'intéresser aux comportements de ces algorithmes lorsqu'ils sont utilisés dans leur version canonique. Dans la première partie, nous présentons quelques définitions sur les ensembles stables, les différents outils que nous avons utilisés et nous proposons un protocole de comparaison des différentes méthodes. Dans la deuxième partie, nous nous intéressons au problème de coloriage des $n$-reines. Dans la troisième partie, nous présentons deux graphes issus des problèmes de multiplexage en longueurs d'ondes (WDM pour *Wavelength Division Multiplexing*) dans les réseaux tout optique [BBG+97]. Nous montrerons que les algorithmes évolutionnaires peuvent se révéler très efficaces lorsqu'ils sont utilisés en «presse-bouton» mais que les autres méthodes réclament souvent peu d'améliorations pour devenir réellement compétitives.

## 5.2 Préliminaires

Avant de présenter les différents résultats que nous avons obtenus sur les coloriages de graphes sur-contraints, nous nous devons d'introduire les définitions concernant les ensembles stables. Nous allons aussi préciser les différentes méthodes utilisées dans ce chapitre, ainsi que le protocole que nous avons choisi pour les comparer.

### 5.2.1 Définition des ensembles stables

Nous allons utiliser dans ce chapitre les ensembles stables, ou ensembles indépendants, afin de trouver un minorant du nombre de conflits d'un coloriage de graphes sur-contraints.

**Définition 5.2.1** *(Stable)*
*On appelle ensemble stable d'un graphe, un ensemble de nœuds tel qu'il n'existe aucune arête les reliant deux à deux. Plus formellement : Soit un graphe $G = (V, E)$, un ensemble $\mathcal{S}_G = \{s_1, \ldots, s_n\}, s_i \in V$ du graphe $G$ est un ensemble stable si et seulement si :*
$\forall s_i, \forall s_j \in \mathcal{S}_G, (s_i, s_j) \notin E$

**Définition 5.2.2** *(Stable maximal) On appelle stable maximal, un ensemble stable qui contient un nombre maximal de nœuds indépendants, c'est à dire que tout nœud ne faisant pas partie du stable a au moins une arête commune avec un des nœuds du stable. Plus formellement : Soit le graphe $G = (V, E)$, $\mathcal{S}_{max}$ est un ensemble stable maximal de $G$ si et seulement si : $\forall s_i \in V \backslash \mathcal{S}_{max}, \exists s_j \in \mathcal{S}_{max}$ tel que $(s_i, s_j) \in E$*

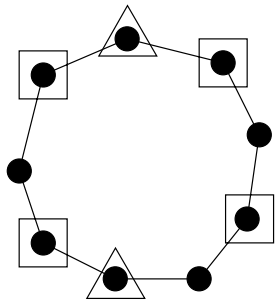

FIG. 5.1 – Un graphe et deux de ses stables.

Sur l'exemple de la figure 5.1, on a représenté un stable par les nœuds entourés d'un triangle, et un stable maximal par ceux entourés d'un carré. A partir de ces définitions, on se rend compte que trouver une partition de $G$ en $n$ ensembles stables $\mathcal{S}_{G_i}$ équivaut à colorier le graphe $G$ avec $n$ couleurs. Trouver le nombre minimal de couleurs pour colorier un graphe revient donc à trouver le nombre minimal de stables vérifiant cette définition. Utiliser des stables maximaux pour trouver ce nombre est généralement une bonne idée, cependant il existe des graphes dont on peut trouver le nombre chromatique sans utiliser un seul stable maximal. Sur l'exemple 5.1 les stables maximaux contiennent tous quatre nœuds, on peut cependant colorier ce graphe en utilisant trois stables contenant trois nœuds. Pour une information plus complète sur les ensembles indépendants, se reporter à [Ber83].

Le nombre de stables maximaux d'un graphe est exponentiel en fonction du nombre de nœuds dans le pire des cas, et donc a fortiori le nombre de stables quelconques. Trouver le nombre minimal de stables recouvrant un graphe est un problème NP-difficile[1]. Nous ne pourrons nous servir de cette définition que dans le cas de graphe ayant une structure particulière permettant de connaître rapidement les stables maximaux.

### 5.2.2 Outils utilisés

Nous allons présenter dans cette section les différentes méthodes et implantations utilisées dans ce chapitre. Ces méthodes sont issues des trois familles définies dans le chapitre 2 : évolutionnaire, constructive et locale.

**Méthode évolutionnaire :** Pour nos expérimentations, nous avons utilisé un moteur d'AG appelé AgCSP [Did02], qui est dédié à la résolution de CSP en domaines finis. Son fonctionnement est le même que celui d'un AG classique sauf que ses chromosomes sont codés en nombre entier plutôt qu'en binaire. De ce fait, les problèmes de coloriage de graphes ou de CSP binaires classiques (lorsque les domaines des variables sont finis) sont beaucoup plus faciles à traiter (voir chapitre 4). Le croisement est un croisement à 1 point, la mutation est une mutation allélique et on utilise une sélection par tournoi. La fonction d'évaluation compte le nombre de contraintes violées. Pour toutes les expérimentations, nous avons fixé le taux de croisement à 1, et une initialisation aléatoire. Nous aurions pu utiliser une initialisation gloutonne, mais nous n'avons pas voulu donner cet avantage à l'AG pour rester cohérent avec ce que nous voulions illustrer, à savoir l'utilisation d'un AG sans réglage ni spécialisation particulier et dont l'implantation doit être rapide.

**Méthode arborescente *Branch and Bound* :** Pour la méthode de recherche arborescente, nous avons utilisé la bibliothèque ILOG solver [ILO98]. Les contraintes sont représentées par des variables booléennes qui indiquent si elles sont satisfaites ou non. On cherche à minimiser le nombre de contraintes violées (c'est à dire la somme de ces variables) par un Branch and Bound (voir chapitre 2). Nous avons aussi utilisé quelques heuristiques.

- L'heuristique de choix de variables est dom/deg (on choisit le rapport minimum entre la taille du domaine et le degré) introduit par Bessière et Régin [BR96].
- L'heuristique de choix de valeur est min-conflit ( on minimise les conflits avec les variables déjà instanciées, on choisit aléatoirement parmi les ex-aequos) introduit par Minton et al. [MJPL92]

[1] Problèmes de la coloration

- Pour éviter de rester bloqué dans les sous-arbres issus des premier choix où l'heuristique de choix de valeur est peu performante, nous avons utilisé comme méthode de recherche arborescente la recherche à divergence limitée (LDS) introduite par Harvey et Ginsberg [HG95].
- Pour le filtrage, nous avons préféré celui réalisé avec le *forward checking* et les compteurs d'incohérences introduits par Freuder et Wallace [FW92]. Les nouveaux compteurs d'incohérences (PFC-MRDAC) [LMS99] basés sur la cohérence d'arcs orientée ne sont pas utiles pour le problème de coloriage de graphes qui ne comprend que des contraintes de différence. En effet, on ne peut rien propager tant qu'une variable de la contrainte n'est pas instanciée.

Nous avons aussi utilisé une autre heuristique pour réduire les symétries qui sera décrite un peu plus tard (section 5.4.2). Dans le reste de l'article, le programme de recherche arborescente sera appelé RASCG.

**Méthodes locales** Pour les algorithmes de recherche locale (Tabou, Recuit Simulé, Descente)[2] nous avons aussi utilisé la bibliothèque ILOG, avec parfois quelques raffinements. Nous avons notamment utilisé deux versions pour l'exploration du voisinage avec Tabou. On entend par voisinage, l'ensemble des solutions potentielles ne différant que d'une valeur d'une variable par rapport à la solution courante. La taille du voisinage est donc égale au nombre de couleurs multiplié par le nombre de variables. Le voisinage étant souvent trop grand pour être exploré intégralement à chaque mouvement, nous avons généralement choisi d'en n'explorer qu'une proportion de ce voisinage choisie aléatoirement.

Pour aucune des méthodes, nous n'avons cherché le réglage optimal, ou le raffinement le plus efficace. Ce n'était pas le but. Nous avons plutôt cherché à les mettre sur un pied d'égalité, en intégrant cependant les raffinements les plus courants de chacune des méthodes, et en choisissant un panel de réglages correspondant aux paramétrages classiques, en fonction du problème. Par exemple, pour AgCSP, le taux de mutation est choisi en fonction du nombre de variables du problème (et donc de la taille du chromosome), c'est à dire une valeur proche de $\frac{1}{n}$.

### 5.2.3 Protocole de comparaison

Il est très difficile de comparer des méthodes stochastiques et des méthodes complètes. En effet les méthodes complètes, bien que généralement plus lentes que les méthodes stochastiques, peuvent certifier qu'une solution est optimale. Pour RASCG, l'espace de re-

[2] Voir section 2.2 page 23

cherche étant trop grand pour laisser terminer la méthode complète, lancer une seule exécution pendant un temps très long ne nous a pas paru intéressant. En effet, le choix de la première variable à instancier et celui de la graine du générateur pseudo aléatoire utilisé par les heuristiques, sont des paramètres qui jouent de manière significative sur l'exécution de RASCG. Par contre, une méthode stochastique nécessite des tests statistiques pour être validée, afin d'obtenir un pourcentage de convergence vers la bonne solution. Nous avons donc préféré laisser une enveloppe de temps comparable à chacune des méthodes utilisées, et d'utiliser différemment cette enveloppe selon la méthode.

**Comparaison de AgCSP et RASCG :** Pour comparer AgCSP et RASCG, il nous a semblé plus judicieux de laisser 5 à 10 fois plus de temps par exécution à la méthode complète, et des temps d'exécution équivalents pour l'ensemble des tests avant de comparer les résultats. Par exemple, avec une enveloppe de temps de 6000 secondes, 100 tests d'une minute sont effectués avec AgCSP, et 10 tests de dix minutes avec RASCG, en utilisant différents paramétrages (nombre de divergences autorisées pour LDS par exemple) et après avoir cherché la variable qui, instanciée la première, permet d'obtenir la meilleure solution de départ. De ce fait, nous donnons le droit à AgCSP d'avoir plusieurs mauvaises exécutions avant de trouver les solutions attendues. Si, en un même temps total d'exécution, RASCG n'a pas trouvé mieux que les meilleures solutions trouvées par AgCSP, alors nous estimons qu'AgCSP est meilleur, sinon c'est RASCG.

**Comparaison de AgCSP et Tabou :** La recherche Tabou nécessite moins de validation statistique qu'un algorithme génétique. En effet, le mode opératoire de la recherche Tabou lui permet de mieux sortir des optima locaux qu'un AG. Lorsqu'un AG se retrouve dans un optimum local, sa population a tendance à s'homogénéiser et la pression sélective devient souvent trop forte pour permettre aux opérateurs génétiques de l'en sortir. A l'inverse la recherche avec liste Tabou, de part la présence de cette liste peut généralement sortir d'un optimum local en quelques mouvements. Nous avons donc décidé de laisser plus de temps au Tabou qu'à AgCSP, mais par contre de n'essayer qu'une poignée de graines aléatoires avec Tabou. Là encore l'enveloppe de temps globale reste sensiblement la même. Si le taux de convergence de AgCSP est supérieur, alors nous estimons qu'il est meilleur. Si le taux de convergence est équivalent alors nous comparons les temps d'exécution et considérons AgCSP comme vainqueur seulement si ses temps d'exécution sont nettement meilleurs.

Lorsque l'efficacité de la recherche arborescente ou celle de Tabou n'était pas satisfaisante et qu'un raffinement simple pouvait être ajouté, nous recommencions les tests avec une enveloppe de temps remise à zéro.

Établir un protocole de comparaison avec les autres méthodes n'a pas été nécessaire, mais nous verrons pourquoi dans la troisième partie. Tous les tests ont été effectués sur un P-III 450Mhz.

## 5.3 Le problème de coloriage des $n$-reines

C'est un problème très connu donné par Michael Trick, issu de la base de graphes de Standford de Donald Knuth[3], et repris pour le challenge de coloriage de graphes de CP'2002. Le but est de colorier les cases d'un échiquier de taille $n \times n$, de telle sorte que deux cases ne soient pas de la même couleur si elles sont sur la même ligne, la même colonne ou la même diagonale. Nous étudions ici le problème avec un échiquier de taille $8 \times 8$. Nos expérimentations sur le problème de coloriage des échiquiers $8 \times 8$ (NQPC) montrent qu'AgCSP obtient de bons résultats sans réglages particuliers, et que RASCG doit être guidé par de bonnes heuristiques pour obtenir des résultats au moins équivalents.

NQPC se colorie en 9 couleurs, mais puisque nous voulions étudier les problèmes sur-contraints, nous avons essayé de le colorier avec 5, 6, 7 et 8 couleurs. Pour ces coloriages, le nombre minimal de conflits est respectivement 66, 40, 18 et 2. Pour trouver ces valeurs, nous avons cherché un minorant. Pour les trois premières valeurs, on peut compter le nombre minimum de conflits par ligne, colonne et diagonale. Ce minorant ayant été atteint par l'une des méthodes, nous avons donc le nombre minimal de conflits. La dernière valeur a été trouvée différemment, puisqu'on trouve zéro comme minorant si on utilise la première méthode. Si on énumère les ensembles stables de 8 nœuds dans ce graphe[4], et que l'on cherche à superposer ces ensembles stables de sorte qu'ils recouvrent de manière optimale l'échiquier, on se rend compte que l'on ne peut pas poser plus de six stables complets. Le septième et le huitième ne peuvent comporter que 7 nœuds au maximum, il reste donc au minimum 2 cases sans couleur qui devront être coloriées chacune avec une couleur d'un des stables utilisés, ce qui génère au moins deux conflits. Nous avons donc un minorant, et ce minorant a lui aussi été atteint.

Les résultats obtenus sont exposés dans la table 5.1, où CV est le nombre minimum de contraintes violées trouvé par AgCSP et % le pourcentage d'essais d'AgCSP ayant trouvé une solution avec au plus CV conflits. Par exemple en utilisant 5 couleurs, 77% des exécutions ont trouvé 68,67 ou 66 conflits. Le taux de mutation est de 0.01, la taille de la population est 40 avec un élitisme de 10%. Nous avons arrêté chaque exécution à 100 000 évaluations (soit 60 secondes sur un P-III 450). AgCSP donne de bonnes solutions assez

[3] `http ://mat.gsia.cmu.edu/COLOR/instances.html`

[4] ce qui revient à énumérer les configurations de 8 reines n'étant pas en prise les unes par rapport aux autres

| Nb Couleurs | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | | 6 | | 7 | | 8 | |
| CV | % | CV | % | CV | % | CV | % |
| 66 | 11% | 40 | 3% | ≤21 | 3% | ≤10 | 8% |
| ≤67 | 47% | ≤41 | 15% | ≤22 | 7% | ≤11 | 28% |
| ≤68 | 77% | ≤42 | 43% | ≤23 | 24% | ≤12 | 56% |
| ≤69 | 91% | ≤43 | 76% | ≤24 | 58% | ≤13 | 74% |
| ≤70 | 100% | ≤44 | 91% | ≤25 | 81% | ≤14 | 91% |

TAB. 5.1 – Pourcentage d'essais d'AgCSP ayant trouvé une solution avec CV conflits ou moins pour le problème NQPC.

rapidement (généralement en moins de 30 secondes) mais a beaucoup de mal à trouver la meilleure solution. RASCG obtient de meilleurs résultats que AgCSP lorsque la méthode LDS est utilisée, par contre, lorsque qu'on utilise une méthode de recherche arborescente standard *branch and bound* en profondeur d'abord, RASCG ne peut obtenir (en dix minutes) de solutions équivalentes à celles trouvées par AgCSP, quels que soient les réglages. Les résultats obtenus ne sont pas aussi bons pour AgCSP que nous l'avions espéré, mais nous ont encouragé à trouver des problèmes plus gros et plus difficiles, pour valider notre approche.

Au vu des premiers résultats nous n'avons pas effectué de tests avec les autres méthodes. La méthode complète étant efficace, il n'était pas intéressant d'essayer d'autres méthodes locales. De fait, les quelques tests préliminaires n'ont pas donné de résultats satisfaisants au regard de ce que donne RASCG. Nous avons donc préferé nous intéresser à des graphes plus difficiles, dont les cliques sont plus importantes et dont le nombre chromatique est plus élevé.

## 5.4 Deux problèmes de coloriage issus de WDM

Trouver des problèmes de coloriage de graphes difficiles, tout en restant de taille raisonnable, pour pouvoir faire des comparaisons efficaces, n'est pas chose aisée, surtout si on veut en plus connaître le nombre minimum de couleurs nécessaires pour le colorier. Nous avons cherché des graphes contenant des cycles de longueur impaire et dont la clique maximum est de taille inférieure au nombre de couleurs nécessaires au coloriage. Nous voulions aussi que ce nombre de couleurs soit élevé (supérieur à 30 couleurs). Nous avons trouvé de tels graphes dans les problèmes de multiplexage de longueurs d'ondes (WDM).

### 5.4.1 Définitions concernant le WDM

Dans les réseaux de communications optiques utilisant la technologie WDM (*Wavelength Division Multiplexing*), une même fibre optique peut transporter plusieurs signaux logiques, chacun étant transmis sur une longueur d'onde différente (couleur différente). Cette technique permet d'accroître la bande passante disponible.

Un réseau de communication optique est modélisé par un graphe orienté symétrique, $G = (V, A)$ dans lequel chaque sommet $u \in V$ correspond à un nœud du réseau et chaque arc $(u, v) \in A$ correspond à un lien de communication optique (fibre optique). Nous avons les définitions suivantes :

- $P(x, y)$ décrit un chemin de $G$ du nœud $x$ au nœud $y$, c'est un chemin orienté défini par l'ensemble des arcs consécutifs reliant $x$ à $y$.
- Une requête est une paire de nœuds ordonnés $(x, y)$ de $G$, qui correspond à la réservation d'un chemin de communication optique de $x$ à $y$. Autrement dit, on réserve une longueur d'onde le long du chemin $P(x, y)$.
- Une instance de communication $I$ est un ensemble de requêtes (une requête $(x, y)$ pouvant apparaître plusieurs fois) pour lequel on cherche un routage et une coloration.
- Un routage $R$ pour une instance $I$ dans $G$ est un ensemble de chemins $R = \{P(x, y) \| (x, y) \in I\}$
- Le graphe des conflits associé à un routage $R$ est un graphe non orienté $G_c(V, E)$, où le sommet $s_\alpha \in V$ correspond au chemin $P_\alpha(i, j) \in G$, et tel que pour deux sommets $s_\alpha$ et $s_\beta$ sont reliés par une arête si et seulement si leurs chemins correspondants $P_\alpha(i, j)$ et $P_\beta(k, l)$ ont au moins un arc en commun dans $G$.

Autrement dit, soit un ensemble de requêtes et leurs routages dans un réseau WDM, chaque requête est associée à un chemin entre deux nœuds dans le réseau. Le but est de trouver un coloriage des chemins tel que deux chemins traversant le même arc, aient des couleurs différentes. Par conséquent, nous construisons le graphe des conflits entre les chemins, où chaque nœud du graphe des conflits correspond à un chemin dans le réseau, et deux nœuds du graphe des conflits sont connectés si leurs chemins correspondant dans le réseau partagent le même arc. De ce fait, colorier les sommets du graphe des conflits donne un coloriage des chemins du réseau. Dans la figure 5.2 (graphe que nous appellerons *Htree*), chaque chemin représente une seule requête, nous pouvons cependant généraliser à $n$ requêtes par chemin. Ainsi, un nœud du graphe des conflits représente une clique de $n$ nœuds dans le problème de coloriage de graphe, et une arête représente un graphe biparti complet entre les nœuds. La figure 5.3 montre le résultat obtenu avec deux requêtes par chemin.

Nous aurions pu utiliser des méthodes de coloration par liste pour colorier de tels

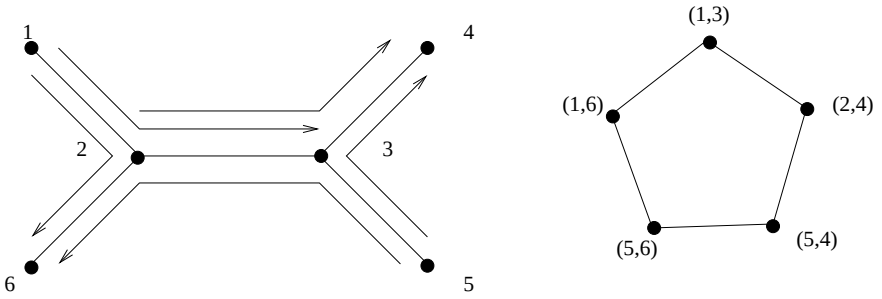


FIG. 5.2 – un routage pour cinq requêtes dans le graphe Htree et son graphe des conflits associé.

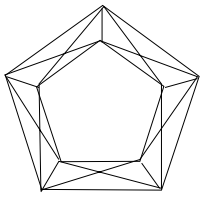

FIG. 5.3 – le graphe de coloriage associé avec 2 requêtes par chemin.

graphes, mais ce n'était pas le but ici. On pourrait nous objecter que les solutions trouvées en coloriage de graphe classique peuvent contenir des conflits dans les cliques correspondant aux listes. Mais tant que le nombre de couleurs utilisées est supérieur à deux fois la taille de la clique, il existe une méthode simple pour déplacer les conflits, sans en changer le nombre. Si on exhibe une solution sous forme d'une coloration par liste, et qu'une liste contient des conflits, il existe une méthode simple qui permet de déplacer les conflits à l'exterieur des listes[5]

L'intérêt de tels graphes est que le fait de les colorier avec un nombre de couleurs inférieur au nombre chromatique a une signification en terme de coût pour le réseau physique. On voudrait pouvoir dimensionner les fibres optiques en fonction du nombre de requêtes qui les traversent. Hors ce n'est pas toujours possible. Par exemple, dans le graphe 5.3, chaque fibre est traversée par quatre requêtes, mais le graphe est au mieux 5-coloriable, et devra donc comporter des fibres laissant passer 5 requêtes. Plus les fibres laissent passer de requêtes plus elles sont chères, et de plus les coûts ne sont pas linéaires. L'idée est alors de chercher un compromis. On va chercher à colorier le graphe avec un nombre insuffisant

[5]Il suffit de changer la couleur d'un des éléments en conflit dans la liste en choisissant parmi l'ensemble constitué de l'union des couleurs des listes des nœuds adjacents moins leur intersection et moins les couleurs de la liste du nœud courant.

de couleurs (correspondant à la capacité des fibres). On va ensuite chercher à minimiser les conflits. Une fois les conflits identifiés, on pourra placer des convertisseurs de longueur d'ondes en certains nœuds du réseau. Ces convertisseurs vont permettre de résoudre les conflits, et peuvent revenir moins cher qu'un sur-dimensionnement du réseau.

Nous avons implémenté un générateur de graphes, qui prend en entrée un réseau, une instance de communication et son routage, et rend un graphe des conflits au format standard Dimacs [JT93]. Nous allons présenter successivement les résultats des différentes méthodes de recherche utilisées dans ce chapitre sur deux instances particulières des graphes de coloriage WDM. Pour chacun d'eux, nous avons généré deux versions, structurellement identiques mais dont la numérotation des nœuds est différente. Nous nous sommes cantonnés à ces deux instances pour deux raisons. La première est que leurs structures particulières nous permettaient de calculer facilement un minorant pour le nombre de contraintes violées. La deuxième est que nous pouvions étudier leurs structures afin de comprendre ce qui mettait certaines méthodes en échec.

### 5.4.2 Graphe Htree

Pour le graphe Htree, si l'ensemble des requêtes est petit (moins de 10 requêtes par chemin) le problème est facile pour RASCG. En revanche, s'il est grand (supérieur à 20 requêtes par chemin) le problème devient difficile. Nous avons donc choisi d'utiliser une instance de communication comportant 30 requêtes par chemin. Le graphe de coloriage ainsi généré comporte 150 nœuds et 6675 arêtes, la clique maximale a 60 nœuds. Les deux versions dont la numérotation des nœuds est différente seront appelée «Htree 1» et «Htree 2». Ce graphe est coloriable en 75 couleurs. Nous avons essayé de le colorier en 60, 65 et 70 couleurs avec respectivement un nombre minimal de conflits de 30, 20 et 10 conflits (en nombre de contraintes violées). Nous trouvons ces valeurs en utilisant la définition des ensembles stables d'un graphe. Pour comprendre comment nous obtenons ces valeurs, utilisons le graphe de coloriage avec 2 requêtes par chemin (Fig 5.3). Dans ce graphe, les stables maximaux contiennent tous 2 nœuds, ils sont au nombre de 20. Nous pouvons trouver 5 stables qui ne partagent aucun nœud, et qui donc couvrent la totalité du graphe. Si on affecte à chaque stable une couleur différente, nous avons un coloriage de ce graphe en cinq couleurs. Maintenant si nous n'avons que quatre couleurs à notre disposition, nous ne pouvons utiliser que quatre stables maximaux. Pour colorier ce graphe en minimisant les conflits, nous devons alors utiliser une couleur déjà prise par un des stables pour chacun des deux nœuds non encore colorié. Cela va générer au moins un conflit par nœud. On peut faire une première généralisation en disant que lorsque le nombre de couleurs reste important(supérieur ou égal à deux fois le nombre de requêtes), il suffira de multiplier

par deux la différence entre le nombre chromatique du graphe et le nombre de couleurs utilisées. Cette technique permet de trouver des minorants, les minorants étant atteints par au moins l'une des méthodes, le nombre minimal de conflits est donc connu.

La table 5.2 nous montre qu'AgCSP obtient de très bons résultats. Le nombre minimal de conflits est trouvé en moins d'une minute pour chaque exécution. Ces tests ont été effectués avec un taux de mutation de 0.005, une population de 50 individus et 8% d'élitisme. RASCG n'arrive pas à trouver les meilleures solutions en trente minutes, avec seulement les heuristiques *dom/deg* et *min-conflit*, et la méthode LDS. Nous avons dû ajouter une nouvelle heuristique qui essaye de casser les symétries en cherchant une clique, et en réduisant les domaines des variables dans cette clique. Nous avons appelé cette heuristique *nosym*. Avec cette nouvelle heuristique, l'efficacité de RASCG pour trouver une bonne solution dépend de l'ordre des nœuds dans le graphe généré. Si nous choisissons les meilleurs réglages (comme la variable de départ, la graine ...) et le graphe généré ayant un ordre dans la numérotation des nœuds très favorable (c'est-à-dire celui qui donne les meilleurs résultats) la meilleure solution est trouvée dès l'initialisation (dans 50% des cas en 65 couleurs et dans 20% des cas en 70 couleurs), mais avec les autres graphes générés, aucune bonne solution n'est trouvée en 10 minutes.

| Temps moyen | | | |
|---|---|---|---|
| Nb couleurs | 60 | 65 | 70 |
| AgCSP | 19s | 26s | 43s |

TAB. 5.2 – Temps moyen en seconde pour trouver le nombre minimal de conflits pour le graphe Htree avec AgCSP.

En ce qui concerne la recherche avec liste Tabou (table 5.3) en utilisant une initialisation gloutonne basique[6], avec quatre tailles de liste (5, 10, 20 et 50), quatre proportions de voisinage (1%, 2%, 5% et 10%), et six graines aléatoires, les résultats sont assez décevants. En effet lorsqu'on cherche à colorier en 60 couleurs, la solution est trouvée dès l'initialisation par la méthode gloutonne, mais dès que l'on cherche à colorier avec 65 ou 70 couleurs, le Tabou se révèle nettement moins bon qu'AgCSP. On remarque de plus que la deuxième version du graphe se révèle plus difficile pour la recherche Tabou. En fait, les différences de réussite de Tabou et de RASCG en fonction de l'ordre dans lequel les nœuds sont générés sont principalement dues à l'algorithme d'initialisation. En effet, celui-ci obtient des résultats très différents selon l'ordre de numérotation des nœuds, et donc les méthodes qui l'utilisent

[6]En fait, il s'agit d'un glouton qui choisit systématiquement la première valeur parmi celles qui minimisent le nombre de conflits

démarrent la recherche à partir des solutions de différentes qualités. AgCSP, utilisant une initialisation aléatoire, ne rencontre pas ces problèmes.

Le graphe Htree est certes difficile de part sa taille, mais a une structure simple qui peut parfois favoriser la recherche arborescente. Tous ces nœuds ont le même degré ce qui favorise les heuristiques utilisées par le *Branch and Bound* lorsque l'ordre des nœuds le permet. Nous avons donc cherché à générer un graphe plus complexe, qui est étudié dans la section suivante.

| | % Voisinage | | | | |
|---|---|---|---|---|---|
| Graphe | 0,01 | 0,02 | 0,05 | 0,1 | Total |
| 65 couleurs | | | | | |
| Htree 1 | 70s | 66s | 68s | 107s | 78s |
| Htree 2 | 90s | 90s | 139s | 226s | 136s |
| 70 couleurs | | | | | |
| Htree 1 | 112s | 125s | 176s | 306s | 180s |
| Htree 2 | 113s | 131s | 206s | 299s | 187s |

TAB. 5.3 – Temps moyen en seconde pour trouver le nombre minimal de conflits pour le graphe Htree avec la méthode Tabou

### 5.4.3 Graphe 3-Penta

Nous avons effectué des tests sur un autre graphe (issu des problèmes WDM) plus difficile que nous avons appelé 3-penta. La figure 5.4 représente le graphe des conflits de 3-penta avec une requête par chemin. Nous avons généré le graphe de coloriage avec 21 requêtes par chemin. On obtient un graphe ayant 231 sommets et 8043 arêtes, la clique maximale est de 42, et le graphe se colorie en 53 couleurs. Nous avons essayé de le colorier en 45 couleurs (le nombre minimal de conflits est 30), et en 50 couleurs (le nombre minimal de conflits est 10). Nous trouvons un minorant pour le nombre de conflits en utilisant la même méthode que pour le graphe Htree. Les résultats obtenus avec AgCSP sont dans la table 5.4, le taux de mutation utilisé est 0.004, la taille de la population est de 50 individus, l'élitisme est de 8% et l'exécution est arrêté après 200 000 évaluations (800 secondes sur un P-III 450).

AgCSP a trouvé le nombre minimal de conflits (en nombre de contraintes violées) dans 70% des cas pour 45 couleurs et trouvé de bonnes solutions (entre le meilleur et une divergence de 2) pour 50 couleurs dans 66% des cas, indépendamment de l'ordre des nœuds dans le graphe généré. RASCG a le même comportement que pour le problème du

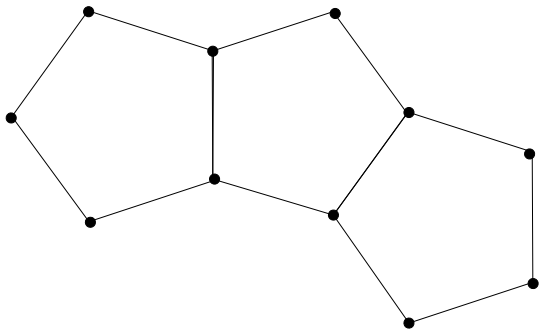

FIG. 5.4 – Le graphe des conflits de 3-penta.

| Nb couleurs | | | |
|---|---|---|---|
| 45 | | 50 | |
| CV | % | CV | % |
| 30 | 70% | 10 | 10% |
| ≤31 | 100% | ≤11 | 48% |
| | | ≤12 | 74% |
| | | ≤13 | 94% |
| | | ≤14 | 100% |

TAB. 5.4 – Pourcentage d'essais d'AgCSP ayant trouvé pour solution CV conflits ou moins pour le problème 3-penta avec AgCSP.

graphe Htree. Le fait de trouver de bonnes solutions dépend de l'ordre des nœuds dans le graphe généré. De plus, toutes les heuristiques décrites ci-dessus doivent être utilisées, en particulier la méthode LDS. Avec ces réglages, et un graphe dont les nœuds sont générés dans le «bon ordre», les solutions trouvées sont presque aussi bonnes que celles trouvées par AgCSP mais l'optimal n'est jamais atteint. Dans les autres cas, RASCG ne trouve pas de solutions intéressantes en près d'une heure. Les résultats de RASCG sont dans les tables 5.5 et 5.6. La colonne «Meilleure solution initiale» donne le pourcentage de variables qui, lorsqu'elles sont choisies comme variables de départ, donnent tel nombre de conflits minimum comme solution initiale ( première solution trouvée). Nous avons procédé en effet en deux étapes. Une première étape qui est arrêtée à la première solution trouvée est essayée pour toutes les variables de départ ( première variable instanciée). Ensuite, nous avons choisi les variables qui donnaient les meilleures solutions initiales pour effectuer les tests en 3000 secondes. « 3-penta 1 » et « 3-penta 2 » représentent deux graphes dont les nœuds sont générés dans des ordres différents. Nous avons effectué une dizaine de tests en

3000 secondes pour nous donner le même temps de calcul global qu'AgCSP.

| 3-penta 1 | | | 3-penta 2 | | |
|---|---|---|---|---|---|
| Meilleure solution initiale | | Meilleure amélioration en 3000 s | Meilleure solution initiale | | Meilleure amélioration en 3000 s |
| 33 | 0.8% | 33 | 33 | 9,5 % | 31 |
| 34 | 3,4% | 31 | 34 | 9% | 31 |
| 35 | 3,4% | 31 | 35 | 18,5% | 32 |
| 37 | 3,4% | - | 36 | 9,5% | - |
| $\geq$38 et $\leq$ 45 | 89% | - | $\geq$46 et $\leq$ 49 | 53,5% | - |

TAB. 5.5 – Résultats obtenu avec RASCG sur le problème 3-penta en utilisant 45 couleurs

| 3-penta 1 | | | 3-penta 2 | | |
|---|---|---|---|---|---|
| Meilleure solution initiale | | Meilleure amélioration en 3000 s | Meilleure solution initiale | | Meilleure amélioration en 3000 s |
| 22 | 9% | 20 | 15 | 18% | 12 |
| 23 | 36,2% | 21 | 16 | 9% | 14 |
| 25 | 9% | 21 | 17 | 9% | 11 |
| 28 | 27% | - | 18 | 5,2% | - |
| $\geq$30 et $\leq$ 32 | 19% | - | $\geq$19 et $\leq$ 27 | 58,8% | - |

TAB. 5.6 – Résultats obtenus avec RASCG sur le problème 3-penta en utilisant 50 couleurs

Nous avons effectué les tests de la méthode Tabou avec quatre tailles de liste (5, 10, 20, 50), quatre proportions de voisinages (1% ,5%, 10% et 20%), et six graines aléatoires. La méthode Tabou est elle aussi sensible à l'ordre de génération des nœuds. Lorsque nous utilisons le graphe dont l'ordre des nœuds est favorable (c'est-à-dire celui qui donne les meilleurs résultats), les résultats obtenus sont généralement meilleurs. Ce comportement est flagrant lorsqu'on essaye de colorier le graphe 3-penta en 45 couleurs : dans le premier cas (3-penta 1), la méthode Tabou trouve le nombre minimal de conflits en moins de cinq minutes dans le pire des cas et en moins de deux minutes en moyenne (table 5.7),alors que dans le deuxième cas (3-penta 2) le meilleur résultat trouvé est à une distance 4 de l'optimal (table 5.8) pour un temps d'exécution variant entre 1500 secondes et 3000 secondes selon la proportion de voisinage visité. Cet effet est un peu moins visible lorsque l'on colorie en 50

couleurs ( Table 5.9 et 5.10), mais un écart subsiste (Rapport entre le nombre d'optimaux trouvés et le nombre total de tests : 8% pour le graphe 1 contre 23% pour le graphe 2). Cependant, lorsque l'on essaye de colorier en 50 couleurs l'optimal est trouvé en moyenne en 900 secondes pour le graphe 1 et en 1200 secondes pour le graphe 2. Le temps d'exécution alloué pour trouver l'optimal était là aussi de 1500 à 3000 secondes selon la proportion de voisinage visité.

| | %Voisinage | | | | |
|---|---|---|---|---|---|
| Liste | 0,01 | 0,05 | 0,1 | 0,2 | Total |
| 5 | 209s | 119s | 64s | 33s | 106s |
| 10 | 298s | 103s | 78s | 33s | 128s |
| 20 | 163s | 107s | 84s | 33s | 96s |
| 50 | 173s | 109s | 76s | 33s | 98s |
| Total | 211s | 109s | 75s | 33s | 107s |

TAB. 5.7 – Temps moyen en secondes pour trouver le nombre minimal de conflits pour la version 1 du graphe 3-penta en 45 couleurs

| | Liste | | | | |
|---|---|---|---|---|---|
| CV | 5 | 10 | 20 | 50 | Total |
| 34 | - | 13% | 4% | 4% | 5% |
| ≤35 | 8% | 17% | 8% | 17% | 12% |
| ≤36 | 29% | 30% | 29% | 24% | 28% |
| ≤37 | 58% | 55% | 58% | 53% | 56% |
| ≤38 | 100% | 100% | 100% | 100% | 100% |

TAB. 5.8 – Pourcentage d'exécution de Tabou ayant trouvé pour solution CV conflits avec la version 2 du graphe 3-penta en 45 couleurs pour la méthode Tabou

Ces mauvais résultats de la méthode Tabou nous ont conduit à étudier d'un peu plus près la méthode de parcours de voisinage utilisé par la bibliothèque ILOG. Nous nous sommes rendu compte qu'après avoir choisi aléatoirement une portion de voisinage, le parcours se faisait de manière ordonnée. En quelque sorte les voisins reçoivent une numérotation et sont ensuite toujours parcourus de manière croissante. Si le voisinage comporte beaucoup de voisins équivalents, ce sont souvent les mêmes qui sont choisis, à moins d'avoir une liste Tabou très longue. Nous avons donc choisi un parcours aléatoire de la portion de voisinage sélectionnée. Ces tests ont été réalisés avec six tailles de liste(5, 10, 20, 50),

| | Liste | | | | |
|---|---|---|---|---|---|
| CV | 5 | 10 | 20 | 50 | Total |
| 10 | 21% | 25% | 21% | 25% | 23% |
| ≤11 | 25% | 25% | 25% | 25% | 25% |
| ≤13 | 38% | 38% | 38% | 38% | 38% |
| ≤14 | 96% | 100% | 100% | 100% | 99% |
| ≤15 | 100% | - | - | - | 100% |

TAB. 5.9 – Pourcentage d'exécution de Tabou ayant trouvé pour solution CV conflits avec la version 1 du graphe 3-penta en 50 couleurs

| | Liste | | | | |
|---|---|---|---|---|---|
| CV | 5 | 10 | 20 | 50 | Total |
| 10 | 13% | 8% | 4% | 8% | 8% |
| ≤11 | 17% | 16% | 17% | 16% | 16% |
| ≤12 | 21% | 20% | 25% | 33% | 24% |
| ≤13 | 42% | 37% | 38% | 54% | 42% |
| ≤14 | 50% | 50% | 59% | 62% | 55% |
| ≤15 | 67% | 67% | 67% | 79% | 70% |
| ≤16 | 88% | 88% | 88% | 92% | 89% |
| ≤17 | 96% | 96% | 100% | 100% | 98% |
| ≤18 | 100% | 100% | - | - | 100% |

TAB. 5.10 – Pourcentage d'exécution de Tabou ayant trouvé pour solution CV conflits avec la version 2 du graphe 3-penta en 50 couleurs

quatre proportions de voisinage (1%, 5%, 10%, et 20%), et six graines aléatoires. Ceci a permis d'obtenir des résultats nettement meilleurs pour la méthode Tabou. En effet l'optimal est trouvé dans la plupart des cas. Cependant, les temps d'exécution sont parfois très longs (table 5.11). On remarque tout de même que selon le graphe utilisé les résultats sont différents. La première version du graphe est plus difficile à colorier en 50 couleurs alors que la deuxième version est plus difficile à colorier en 45 couleurs. Nous pouvons tout de même conclure qu'au vu des résultats, et en utilisant ce raffinement dans le parcours du voisinage, la méthode Tabou est gagnante face à AgCSP et RASCG.

Le comportement du recuit simulé sur ces problèmes, que ce soit le graphe Htree ou le graphe 3-penta, est intéressant. Quels que soient les réglages utilisés, l'algorithme diverge et ne fait que détériorer la solution initiale. Lorsque la température devient nulle, l'algo-

| | % Voisinage | 45 Couleurs | | 50 Couleurs | |
|---|---|---|---|---|---|
| 3-penta 1 | 0,01 | 100% | 118 | 83% | 953 |
| | 0,05 | 100% | 92 | 88% | 1244 |
| | 0,1 | 100% | 105 | 88% | 1748 |
| | 0,2 | 100% | 113 | 42% | 2283 |
| 3-penta 2 | 0,01 | 100% | 1107 | 100% | 801 |
| | 0,05 | 92% | 1897 | 100% | 915 |
| | 0,1 | 25% | 2011 | 92% | 1188 |
| | 0,2 | 4% | 2840 | 79% | 1549 |

TAB. 5.11 – Pourcentage d'exécution ayant trouvé le nombre minimal de conflits et temps moyen pour y parvenir pour les deux versions du graphe 3-penta en 45 et 50 couleurs en utilisant la méthode Tabou avec parcours aléatoire de la portion de voisinage

rithme se met alors à converger, et dans la plupart des cas trouve la solution optimale. Finalement, la méthode la plus inattendue et qui se révèle être globalement aussi efficace que les autres, est un algorithme de Descente qui choisit dans une portion du voisinage le premier mouvement qui a un delta inférieur ou égal à 0 (on accepte de rester sur les plateaux). On appelle delta la différence entre la solution courante et un de ses voisins.

### Structure du voisinage

Ce fait assez surprenant nous a conduit à étudier de plus près la structure du voisinage des solutions potentielles. On peut se figurer le paysage de ce voisinage comme un grand plateau, où une grande partie des voisins ne détériore ni n'améliore la solution courante, une autre grande partie la détériore légèrement, et une infime partie l'améliore lorsqu'on n'est pas sur un minimum local. Généralement, l'écart entre la meilleure et la pire solution du voisinage ne dépasse pas 4 pour le Htree et 6 pour le 3-penta. De plus, les minima locaux sont représentés par de grands bassins où les voisins sont tous équivalents avec un delta très faible. Sur la figure 5.5, nous avons affiché l'intégralité du voisinage de chaque variable à distance 1 d'une solution potentielle du Htree, avec en abscisse les variables, en ordonnée les valeurs du domaine et la hauteur représente la différence entre le nombre de conflits de la solution courante et le celui du voisin ayant une autre valeur pour une variable donnée. On peut remarquer l'aspect structuré de ce paysage, ainsi que l'absence de réels pics. La majeure partie du paysage est constituée de points dont la hauteur est comprise entre 0 et 2 ( 95% du paysage). Pour plus de lisibilité, nous avons tracé l'évolution du nombre de voisins en fonction de leur distance à la solution potentielle courante pendant

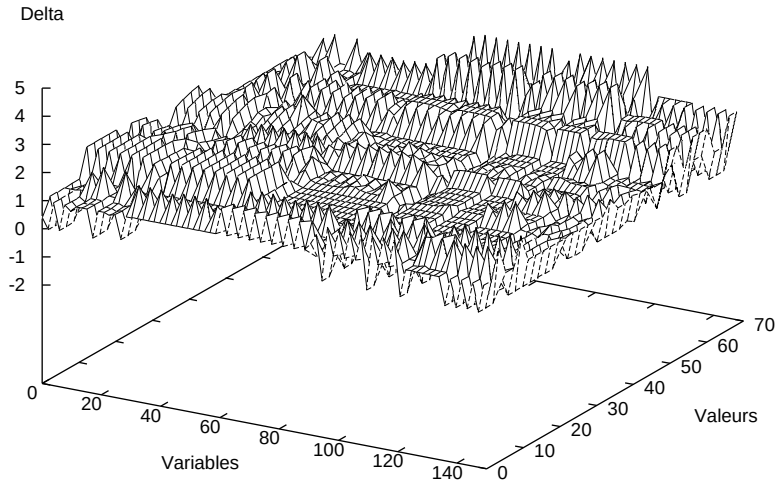


FIG. 5.5 – Voisinage d'une l'instanciation complète du Htree ayant 12 conflits avec 70 couleurs

l'exécution d'un algorithme de descente sur la figure 5.6. Les voisins améliorant la solution courante (Delta négatif) ne sont pas visibles sur le graphique car leur nombre est rarement supérieur à deux hormis en début d'exécution lorsqu'on est loin de l'optimal. On peut remarquer que le nombre de voisins dont le delta est égal à deux et trois est relativement stable durant l'exécution. Le nombre de voisins dont le delta est égal à zéro et un évolue de manière symétrique. Les premiers diminuent et les seconds augmentent. Ceci s'explique assez facilement : plus on se rapproche de l'optimal plus les cliques du graphe sont coloriées de manière optimale, et donc il est plus rare d'avoir un valeur ne changeant pas l'évaluation de la solution courante. Par exemple, si on essaye de colorier le graphe Htree en soixante-dix couleurs, sachant que les cliques sont de tailles soixante, seulement dix valeurs du domaine permettent aux voisins de ne pas détériorer une clique de la solution courante, alors que les soixante autres génèrent au moins un conflit. Ces particularités du voisinage sont similaires pour le graphe 3-penta. Nous avons choisi d'utiliser le graphe Htree plutôt que le 3-penta pour illustrer nos propos uniquement par souci de lisibilité. En effet, bien que la structure du voisinage et son évolution au cours de l'exécution soient apparentées, les différences entre les voisins sont plus grandes (de 0 à 5) et le paysage plus chaotique. L'évolution du nombre de voisins ayant le même delta est moins flagrante, mais similaire.

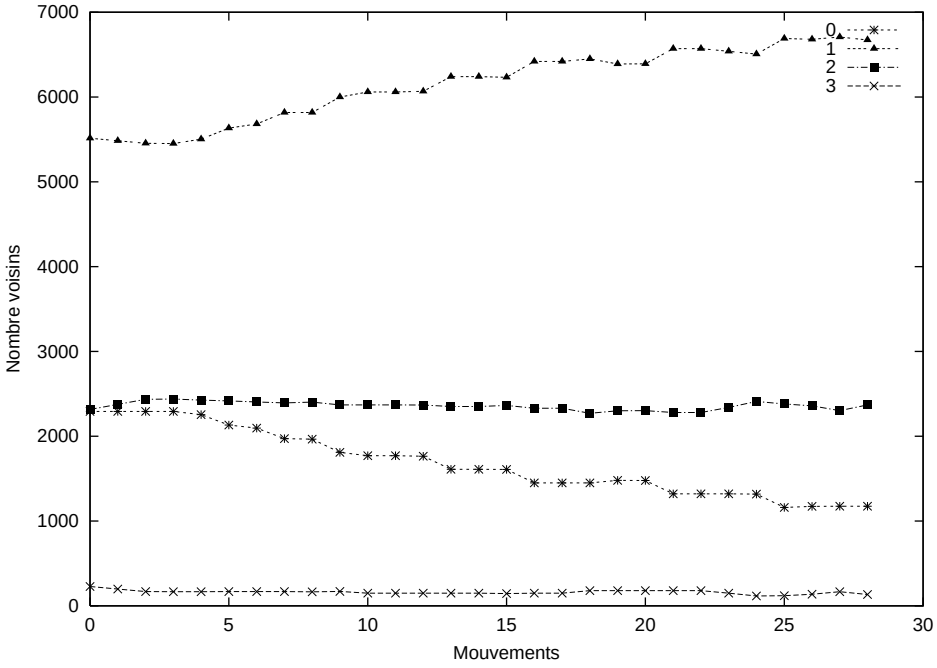


FIG. 5.6 – Évolution de la différence entre une solution et ses voisins au cours d'une exécution

Cette structure particulière du voisinage et le fait que les voisins améliorant la solution courante (s'il y en a), sont très peu nombreux, moins de 1 pour 10000, lorsque l'on est relativement proche de l'optimal. Ceci explique le comportement du recuit simulé. Celui-ci va, la plupart du temps, sélectionner des solutions équivalentes ou moins bonnes, et n'aura pas le temps de converger lorsqu'il trouve une solution qui améliore la solution courante puisqu'il la détériorera aussitôt après. Par contre, dès que la température sera nulle, il va pouvoir converger sans trop de problème. Cela explique aussi le comportement de la méthode Tabou lorsque le parcours du voisinage est trop systématique ou que la portion de voisinage choisie est trop grande. Le Tabou parcourt les grands plateaux de solutions équivalentes sans pouvoir trouver de meilleures solutions, et la liste ne lui sert presque à rien. Pour que la liste serve réellement, il faudrait qu'elle soit plus longue que le nombre de mouvements maximum permis. En fait le Tabou se sert très peu de sa liste et a donc un comportement similaire à un algorithme de Descente non stricte.

### 5.4.4 Bilan sur les coloriages WDM

Avant de conclure, nous allons dresser un bilan des résultats obtenus par les différentes méthodes sur les coloriages WDM. Ce bilan va nous permettre d'avoir une vision plus

globale des résultats en fonction des améliorations.

| | | | | Tabou | | RASCG | |
|---|---|---|---|---|---|---|---|
| Graphe | # | Critères | AgCSP | V1 | V2 | LDS | LDS + nosym |
| Htree 1 | 60 | Temps | 19 | | GL | GL | GL |
| Htree 2 | | | 19 | | GL | GL | GL |
| Htree 1 | 65 | | 26 | | 78 | NA(1800s) | **GL(50%)** |
| Htree 2 | | | **26** | | 136 | NA(1800s) | NA(600s) |
| Htree 1 | 70 | | 43 | | 180 | NA(1800s) | **GL(20%)** |
| Htree 2 | | | **43** | | 187 | NA(1800s) | NA(600s) |
| 3-penta 1 | 45 | Temps | 800 | **100** | **107** | NA | 3000 |
| | | Meilleur | 30 | 30 | 30 | | 31 |
| | | % | 70% | 100% | 100% | | NA |
| 3-penta 2 | 45 | Temps | **800** | NA | 1964 | NA | 3000 |
| | | Meilleur | 30 | 34 | 30 | | 31 |
| | | % | 70% | 5% | 52% | | NA |
| 3-penta 1 | 50 | Temps | 800 | 900 | **1114** | NA | 3000 |
| | | Meilleur | 10 | 10 | 10 | | 11 |
| | | % | 10% | 23% | 76% | | NA |
| 3-penta 2 | 50 | Temps | 800 | 1200 | **1500** | NA | 3000 |
| | | Meilleur | 10 | 10 | 10 | | 21 |
| | | % | 10% | 8% | 92% | | NA |

TAB. 5.12 – Bilan sur les graphe WDM

Dans ce tableau, la première colonne représente le graphe utilisé, la deuxième (#) représente le nombre de couleurs utilisé. La troisième colonne représente le critère que l'on fera figurer dans les colonnes concernant les méthodes :

- Dans la partie concernant le graphe Htree, le temps représente le temps moyen pour trouver l'optimal (les deux méthodes stochastiques ayant toujours trouvé l'optimal)
- Dans la partie concernant le 3-penta, le critère temps représente le temps moyen mis pour trouver l'optimal (sachant que l'optimal n'est pas systématiquement trouvé).
- Le critère meilleur représente la meilleure solution trouvée, et le critère %, le pourcentage moyen d'exécution ayant trouvé cette meilleure solution.

Par exemple, pour la version 1 de la méthode Tabou, sur le graphe 3-penta 1, avec 50 couleurs, on peut lire qu'en moyenne 23% des exécutions ont trouvé comme meilleure solution 10 et que 900 secondes sont nécessaires.

Pour la méthode Tabou, V1 et V2 représente les deux versions de Tabou utilisé (selon leur façon de parcourir le voisinage). Pour RASCG, la première colonne (LDS) représente le fait qu'on utilise LDS, et la deuxième colonne (LDS + nosym) représente qu'on a utilisé LDS et l'heuristique qui supprime une partie des symétries. Les cases contenant GL signifie que l'optimal est trouvé dès l'initialisation (par la méthode gloutonne), et NA indique que la méthode n'a pas trouvé l'optimal. Nous avons fait figurer en gras les résultats que nous estimons comme les meilleurs. On peut déduire plusieurs choses de ce tableau. L'AG a le comportement attendu : il est relativement efficace pour se rapprocher de la solution optimale, mais trouver l'optimal est souvent difficile. Il n'est pas sensible à l'ordre des nœuds dans les graphes générés. Les grands plateaux de solutions équivalentes lui permettent de conserver une hétérogénéité suffisante pour se rapprocher des solutions optimales. Cependant, l'évaluation des solutions potentielles coûte cher lorsque le problème est gros, et les temps d'exécution s'en ressentent. Pour la méthode Tabou, il est nécessaire de s'intéresser particulièrement à la structure du problème afin d'intégrer les raffinements qui la rendent efficace. Cependant, ces raffinements sont assez simples à mettre en œuvre. Il faut, entre autre, intégrer un processus stochastique dans le parcours de la portion de voisinage (cette possibilité étant déjà incluse dans la bibliothèque Ilog solver). La méthode arborescente devient moins efficace si le problème est plus gros, mais ce type de résultat était attendu, de plus les heuristiques ont une efficacité très variable selon le graphe utilisé. Cela démontre que les heuristiques standards, bien qu'améliorant sensiblement la recherche, ne permettent pas de traiter seules des problèmes de grande taille efficacement.

## 5.5 Conclusion

Nous avons étudié l'efficacité des algorithmes génétiques, sans réglages particuliers ou spécialisations (hybridations), pour trouver de bonnes solutions aux problèmes de coloriages de graphes sur-contraints. Nous montrons aussi que les AG peuvent être plus efficaces que les méthodes de recherche arborescente, surtout lorsque le nombre de couleurs est élevé. Les graphes que nous avons étudiés sont denses, et ont plusieurs cliques maximales elles-mêmes très inter-connectées. De plus, le nombre minimal de couleurs nécessaire est supérieur à la taille de ces cliques maximales. De fait, ce sont des problèmes difficiles pour les méthodes de recherche arborescente, même en utilisant de bonnes heuristiques. Ces résultats sont intéressants lorsqu'on n'a pas le temps de chercher une bonne heuristique ni d'essayer beaucoup de réglages pour les AG. Les AG permettent d'avoir, sur ce type de problème, des solutions intéressantes rapidement. Des travaux très récents de van Hemert [vH02] comparent les méthodes évolutionnaires aux méthodes arborescentes sur des CSP aléatoires, et concluent que ces dernières sont plus efficaces, cependant les expérimentations ont été

faites sur des instances de relativement petites tailles (10 à 45 variables, domaine de taille 15). De fait, les instances de petites tailles favorisent la recherche arborescente, ces résultats ne remettent donc pas en question nos propres travaux.

Nous pensions au départ que la méthode Tabou pourrait faire mieux que les AG dès la première implantation, mais il nous a fallu plusieurs implantations, et une prise en compte de la structure du voisinage pour qu'elle devienne réellement efficace. Bien qu'une résolution «en aveugle» ne soit pas efficace, peu d'améliorations sont cependant nécessaires pour obtenir des résultats acceptables. Il serait donc intéressant de poursuivre la comparaison entre la recherche Tabou et les AG, en suivant toujours le protocole que nous avons fixé, sur un panel assez large de problèmes de coloriage sur-contraints, en restant toujours dans l'optique de faire peu ou pas de réglage ni spécialisation. Le but final serait de caractériser les types de problèmes où les AG sont performants.

Une autre caractérisation qu'il serait intéressant de faire est celle des problèmes déceptifs pour le recuit simulé. Nous avons vu que les problèmes que nous présentons font diverger le recuit simulé classique, et nous pensons qu'une caractérisation générale des problèmes engendrant ce type de comportement serait intéressant dans l'optique d'une classification entre problèmes et méthodes les mieux adaptées. Bien que peut être trop ambitieux, nous pensons que le fait de pouvoir déterminer rapidement les caractéristiques d'un problème afin de choisir ensuite la méthode la mieux adaptée serait des plus profitable.

Nous nous devons cependant de faire une parenthèse sur l'implantation des méthodes utilisées. Les longs temps d'exécution observés pour la résolution des problèmes ne peuvent seulement s'expliquer en fonction des algorithmes utilisés. Le moteur d'AG que nous utilisons n'est en aucune façon une implantation professionnelle, de ce fait un certain temps est perdu par des processus de gestion mémoire ou encore d'évaluation non incrémentale. Ce problème existe aussi pour la méthode Tabou. Bien que la bibliothèque utilisée soit professionnelle (ILOG solver), il existe une perte de temps non négligeable pendant l'évaluation des solutions à cause là aussi d'un manque d'incrémentalité dans le test des contraintes. Ce biais n'est pas présent pour la méthode de recherche arborescente, mais il est vrai qu'ILOG solver a plus une vocation tournée vers les méthodes complètes. L'idée principale de ce chapitre était non seulement de ne pas faire de réglage particulier, mais de ne pas passer beaucoup de temps pour l'implantation des méthodes. La relative efficacité des AG sur les problèmes que nous présentons est déjà en soit intéressante, et d'autant plus que le développement d'un AG tel que celui que nous utilisons est assez rapide du fait de sa très faible spécialisation. Peut-être faudrait-il élargir le protocole en prenant en compte le temps de développement des différentes méthodes.

Ceci nous conduit à penser que dans une optique de recherche de solutions approchées, sur des problèmes de coloriage sur-contraint de grande taille, et lorsque le temps manque

pour chercher la meilleure implantation, les meilleurs réglages et les meilleures heuristiques, l'utilisation d'un AG standard semble être la méthode la mieux adaptée.

# Chapitre 6

# Nouveaux Opérateurs

## 6.1 Introduction

Dans la littérature, de nombreuses variations des opérateurs génétiques ont été proposées (voir chapitre 3 et 4). Leurs coûts en terme de temps d'exécution sont très variables, et ils sont plus ou moins dédiés à des problèmes particuliers. L'idée principale dans ce chapitre est certes de trouver des opérateurs qui soient plus efficaces que les opérateurs classiques, mais surtout dont les réglages nécessitent moins d'expertise et de temps. Nous cherchons aussi à définir des opérateurs dont le comportement soit peu lié au problème traité, l'idée étant toujours de rester au plus proche d'un AG standard dans l'optique d'un développement rapide. Les nouveaux opérateurs que nous présenterons sont conçus pour des problèmes de satisfaction de contraintes mais nous pensons qu'ils peuvent, pour la plupart, être employés pour beaucoup d'autres problèmes d'optimisation. De plus, ils sont peu coûteux en terme de temps d'exécution. Dans la section 6.2 nous nous intéressons au croisement, nous montrons les limites et les avantages des opérateurs connus les plus classiques (à 1 point, à points multiples, uniforme) avant de proposer un nouvel opérateur de croisement. Dans la section 6.3, nous nous intéressons aux opérateurs de mutation. Après avoir présenté quelques opérateurs de mutation déterministes connus, nous présentons deux nouveaux opérateurs de mutation adaptatifs et déterministes. Enfin, dans la section 6.4, nous proposons un opérateur de diversification pour le coloriage de graphe avant de conclure.

## 6.2 Opérateurs de croisement et CSP

Dans cette section, nous discutons de l'efficacité du croisement classique à 1 point et à p-points pour la résolution CSP. Nous montrons qu'à cause de la structure du graphe de contraintes, ces croisements ne sont pas les plus appropriés. Nous présentons le croise-

ment uniforme bien connu [Sys89] et proposons un nouvel opérateur multi-points dont la dynamique est différente de celle du croisement uniforme.

### 6.2.1 Croisement à 1 point et à p-points

Les croisements à 1 point et à p-points ont été directement inspirés de la nature où le code génétique a un sens de lecture et où deux gènes sont d'autant plus liés qu'ils sont proches l'un de l'autre. Donc l'utilisation du croisement à un point dans un AG a semblé être la meilleure idée. Mais pour un CSP, la représentation linéaire des variables du CSP dans un chromosome ne reflète pas la structure du graphe de contraintes[1]. La codépendance entre deux variables ne peut pas être représentée par deux gènes proches. Avec le croisement à un point, beaucoup de partitions du graphe de contraintes sont omises, donc nous pouvons manquer quelques bonnes solutions du fait que la convergence suit une seule direction. En effet, dans un graphe donné avec $n+1$ nœuds, il y a $2^n$ partitions possibles en deux sous-graphes et le croisement à 1 point propose seulement $n$ partitions parmi l'ensemble des possibilités, alors que le but du croisement est d'exploiter le maximum de reconfigurations possibles. Pour éviter ce problème, le croisement à p-points a été proposé , mais avec un nombre fixé de points $p$ nous avons seulement : $C_n^p$ partitions. Ce croisement peut être très intéressant pour quelques problèmes d'optimisation, mais n'est pas satisfaisant pour les CSP et les problèmes de coloriage de graphe. Pour une étude comparée du croisement à p-points et du croisement uniforme, on peut se reporter à [SD91, ECS89].

### 6.2.2 Le croisement uniforme

Le croisement uniforme [Sys89] a été conçu pour tenir compte de la spécificité du partitionnement de graphe et avoir un meilleur croisement pour des problèmes où la co-dépendance entre deux variables n'est pas représentée par des gènes proches. Il permet de plus de générer des enfants beaucoup plus différents les uns des autres du fait du nombre de croisements possibles. Avec cet opérateur, chaque locus a la même chance d'être choisi. Chaque gène de chaque parent a une probabilité de $\frac{1}{2}$ d'être représenté dans le chromosome fils. Avec ce croisement, nous avons $2^n$ croisements possibles et une probabilité de $\frac{1}{2^n}$ pour chacun, d'où son nom : croisement uniforme. Mais le nombre de points choisis pour le croisement n'est pas uniforme : un croisement à un point a une probabilité de $\frac{n}{2^n}$ d'être représenté alors qu'un croisement à $p$ points a une probabilité de $\frac{C_n^p}{2^n}$ d'être représenté. La probabilité pour avoir un croisement à $p$ points suit une distribution binomiale tracée sur

[1] Des représentations non linéaires des chromosomes (sous forme matricielle par exemple) ont déjà été proposées, mais nous ne nous intéressons ici qu'aux représentations classiques, les autres représentations impliquant des opérateurs particuliers dédiés.

la figure 6.1. Les croisements avec peu de points et ceux avec beaucoup de points sont mal représentés et ceux avec en moyenne $\frac{n}{2}$ points sont fortement représentés. Pour tenir compte de la spécificité de quelques problèmes, nous avons voulu avoir une probabilité uniforme pour le nombre de points de croisement, nous avons donc défini un nouvel opérateur appelé croisement à nombre de points aléatoire (NPA). De nombreux autres opérateurs

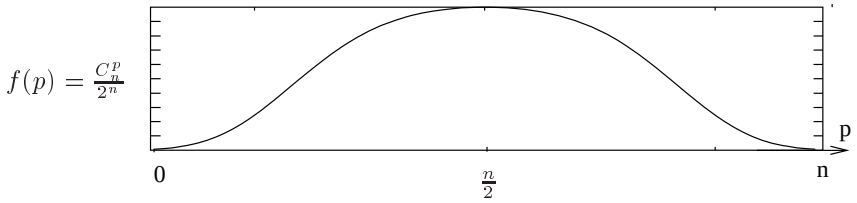


FIG. 6.1 – la Probabilité pour avoir $p$ des points de croisement

multi-points ont été proposés dans la littérature, ils ne seront pas étudiés ici en raison de leur particularité. En effet, nous nous sommes intéressés ici qu'aux opérateurs qui ne présupposent rien du problème à traiter (i.e. aucune information sur le problème) sinon le degré de corrélation entre les gènes induite par le codage. Le lecteur peut se reporter à [SMM+91] pour une comparaison et une présentation des autres opérateurs les plus courants.

### 6.2.3 Le croisement NPA

Nous désirons obtenir un croisement qui permet une répartition uniforme du nombre de points de croisement. Mais nous ne voulons pas calculer une probabilité complexe qui augmenterait la complexité de l'algorithme. Nous avons choisi un algorithme simple en trois étapes qui a le comportement suivant :

- Choisir aléatoirement le nombre $p$ de points de croisement entre 0 et $n$.
- Effectuer le croisement comme un croisement classique à p-points, en choisissant aléatoirement les positions des points de croisement, entre les $C_n^p$ positions possibles.

Calculons le nombre de croisements différents possibles :

- Soit $n$ le nombre de points de croisement possible.
- Pour $p$ points choisis parmi $n$, nous avons $C_n^p$ positions possibles.
- Nous avons donc $\sum_{p=0}^{p=n} C_n^p = 2^n$ croisements différents possibles

Comme pour le croisement uniforme, nous obtenons toutes les partitions de graphe possibles. Nous avons une probabilité uniforme pour avoir un certain nombre $p$ de points de croisement. Cependant, pour un croisement particulier avec $p$ points à une position parti-

culière, la probabilité que ce croisement apparaisse est : $\frac{1}{n} \times \frac{1}{C_n^p}$. La probabilité d'apparition d'un croisement particulier suit donc une distribution inverse d'une binomiale tracée sur la figure 6.2.

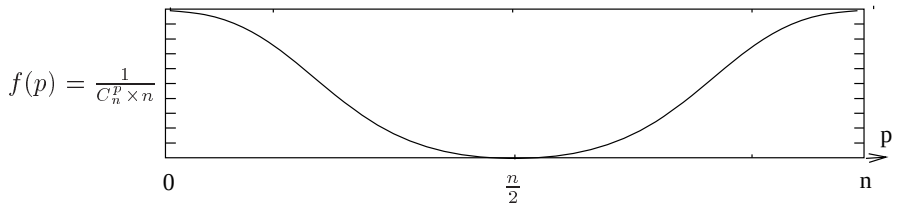


FIG. 6.2 – la Probabilité pour avoir une configuration donnée de croisement avec $p$ points

Cet opérateur a quelques particularités : nous ne supposons pas que le nombre de points de croisement est meilleur s'il est proche de $\frac{n}{2}$. Nous supposons qu'il est intéressant de ne pas casser un grand nombre de dépendances induites par le codage à chaque croisement, en particulier pour des problèmes comme le problème de coloriage n-reines (NQCP) introduit dans le chapitre précédent(voir section 5.3 page 68). Dans le NQCP le but est de colorier toutes les cases d'un échiquier tel que deux cases ne peuvent être de la même couleur si elles appartiennent à la même colonne, la même ligne ou la même diagonale. Pour coder ce problème dans un chromosome, les lignes de l'échiquier sont généralement représentées côte à côte. Préserver quelques codépendances sur une même ligne peut être très utile pour la recherche.

C'est notamment un croisement intéressant quand une initialisation gloutonne, qui suit l'ordre des nœuds, est employée[2]. Dans ce cas, casser les contraintes déjà satisfaites par la méthode gloutonne n'est pas forcément intéressant. Nous avons cependant choisi de conserver le principe purement stochastique de croisement plutôt que de chercher à identifier les contraintes satisfaites à conserver en se tournant vers des opérateurs dédiés.

Toujours en suivant l'idée qu'une partie des codépendances sont représentées par des gènes proches, il peut être intéressant d'éviter les croisements ayant un trop grand nombre de points (supérieur à $\frac{n}{2}$). Pour ce faire, on peut restreindre le nombre de points maximum que notre croisement peut choisir. Nous appellerons NPA restreint, noté NPA-$i$, ce croisement où $i$ est le nombre de points de croisement maximum (avec $i \leq n$). Bien entendu, le NPA restreint perd la possibilité d'avoir tous les partitionnements de graphe possibles[3]. La raison qui nous a motivé à restreindre le nombre de points de croisement autorisé, est qu'un grand nombre de points de croisement est beaucoup trop destructeur lorsque des

[2]Une manière simple pour faire une initialisation gloutonne est de choisir un premier gène aléatoirement et d'initialiser les gènes suivants en minimisant les conflits

[3]à noter qu'avec $i = \frac{n}{2}$ le nombre de croisements différents est supérieur ou égal à $2^{n-1}$ (suivant la parité de $n$)

dépendances entre gènes proches existent, et donc, l'efficacité du croisement s'en trouve amoindri. Le croisement NPA, dans sa forme originelle, utilise plus de $\frac{3n}{4}$ points de croisement en moyenne une fois sur quatre, ce qui nous a paru excessif. Nous avons donc préféré introduire ce réglage supplémentaire, qui nous permettra de mieux respecter la structure du chromosome. En fait le croisement NPA restreint cherche à cumuler les avantages du croisement uniforme et du croisement à p-points. Il permet de ne pas préjuger du nombre de points le mieux adapté au problème, mais plutôt de proposer un grand nombre de croisements possibles utilisant différents nombres de points.

Nous allons présenter les résultats que nous avons obtenus avec cet opérateur, sur le NQPC, le 3-penta et sur des CSP aléatoires. Pour chacun des deux premiers graphes, nous avons choisi d'utiliser, là encore, un nombre de couleurs inférieur au nombre chromatique.

**Coloriage des 8 reines (NQPC)** Pour ces expérimentations, le nombre de couleurs utilisées est 8, car c'est avec ce nombre de couleurs qu'AgCSP a le plus de mal à converger (voir chapitre 5). Cela va nous permettre de mesurer plus facilement les différences de convergence entre les opérateurs. Nous avons utilisé le croisement uniforme, et avons comparé ses performances au croisement NPA en utilisant quatre réglages différents pour ce dernier :

- NPA-64 : c'est la forme originelle, toutes les possibilités de croisement sont prises en compte.
- NPA-48, NPA-32, NPA-16 : c'est la forme restreinte respectivement à 48, 32 et 16 points de croisement maximum (choisi arbitrairement)

Les taux de croisement utilisés sont 0.8, 0.9 et 1, le taux de mutation est $\frac{1}{64}$, la sélection est une sélection par tournoi à 2 individus, l'élitisme est de 5% sur un total de 80 individus. Pour chaque paramétrage, nous avons effectué cent exécutions, en limitant à 100 000 évaluations. Toutes les courbes représentent en ordonnée le nombre total d'exécutions ayant atteint au pire la valeur en abscisse.

Les deux premières courbes (Figure 6.3 et 6.4) présentent les résultats globaux quel que soit le taux de croisement, en utilisant une initialisation gloutonne de la population, et une initialisation aléatoire. On peut déjà remarquer que le croisement uniforme est légèrement moins efficace avec l'initialisation gloutonne. Cependant, ces graphiques montrent surtout que les différences de convergence selon l'opérateur de croisement utilisé sont très faibles en moyenne. Nous allons maintenant détailler les résultats des différents opérateurs en fonction du taux de croisement, avec une initialisation gloutonne.

En détaillant les résultats (Figure 6.5, 6.6 et 6.7) en fonction du taux de croisement, on remarque que le croisement uniforme a généralement de moins bons taux de convergence que les différentes versions du croisement NPA. On ne peut cependant pas prétendre à

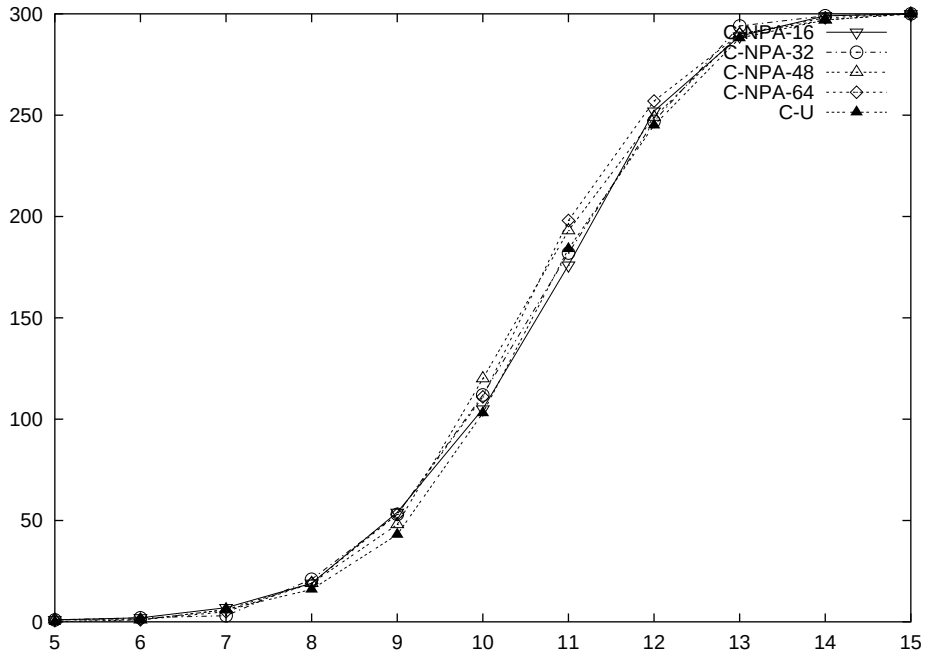


FIG. 6.3 – NQPC : Total, initialisation gloutonne

une suprématie du croisement NPA face au croisement uniforme, les taux de convergence restant relativement proches. Nous nous sommes donc tournés vers un graphe plus difficile.

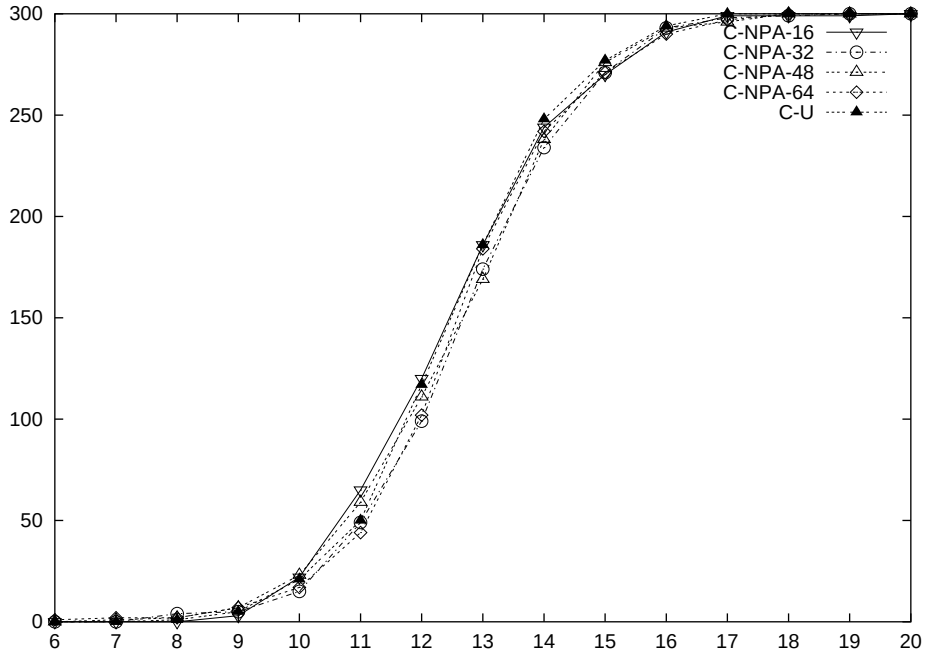


FIG. 6.4 – NQPC : Total, initialisation aléatoire

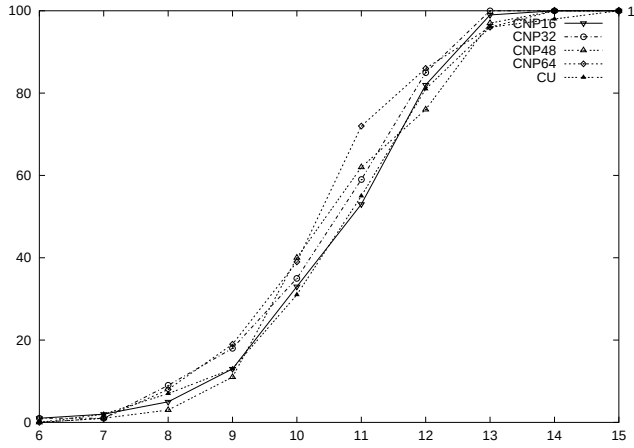


FIG. 6.5 – NQPC : Taux de croisement à 0,8, initialisation gloutonne

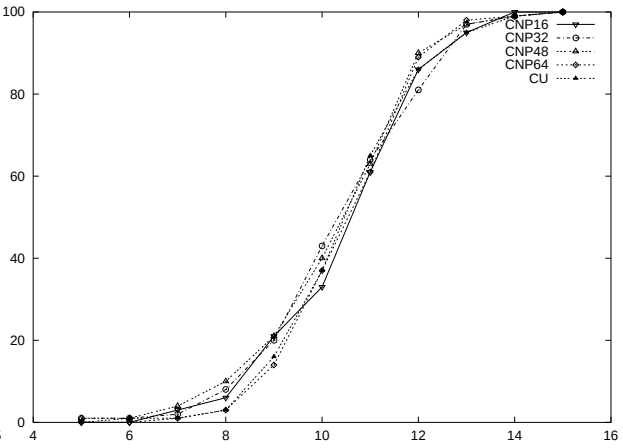


FIG. 6.6 – NQPC : Taux de croisement à 0,9, initialisation gloutonne

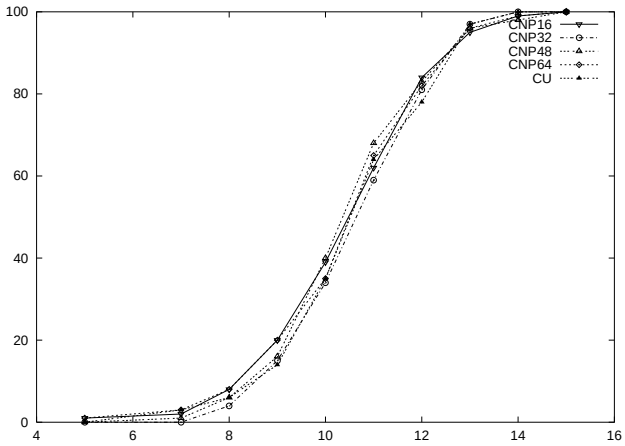


FIG. 6.7 – NQPC : Taux de croisement à 1,0, initialisation gloutonne

**3-Penta** Nous avons choisi d'utiliser à nouveau le graphe 3-penta[4] pour nos expérimentations pour plusieurs raisons. La première est que ce graphe répond tout à fait aux critères d'utilisation du croisement NPA. C'est un graphe dense, dont les variables sont fortement interconnectées, et deux gènes proches ont de bonnes chances de faire partie de la même clique, il peut donc être intéressant d'avoir moins de $\frac{n}{2}$ points de croisement. La seconde est que c'est un graphe difficile, et donc la méthode gloutonne ne pourra pas faire « tout le travail». Comme précédemment, nous avons utilisé deux versions de ce graphe, dont l'ordre des nœuds est différent. De ce fait, les chromosomes n'auront pas la même structure. La première version est partiellement ordonnée en fonction des cliques du graphe, la deuxième n'a pas d'ordre particulier. Nous les nommerons, par abus de langage, 3-penta-ord et 3-penta-mél.

Pour cet ensemble d'expérimentations, le taux de mutation est 0.004, la taille de la population est 50, l'élitisme est fixé à 8%, la sélection s'effectue par tournoi (avec 2 individus), deux taux de croisement ont été essayés (0.7 et 1.0), on limite chaque exécution à 200 000 évaluations. Nous avons décidé de comparer l'efficacité du croisement uniforme au croisement NPA avec trois réglages différents :

- NPA-231 : prend en compte tous les croisements possibles.
- NPA-100 et NPA-50 : deux formes restreintes du croisement NPA (choisis arbitrairement, sans préréglage de ce paramètre, l'idée étant de prendre en compte environ $\frac{n}{2}$ et $\frac{n}{4}$ points de croisement)

Pour chaque réglage, 50 exécutions ont été effectuées.

Les deux premières courbes représentent les résultats globaux quels que soient le taux de croisement et le graphe utilisé. La première présente les résultats avec une initialisation gloutonne (figure 6.8), la deuxième avec une initialisation aléatoire (figure 6.9). On remarque tout de suite que notre opérateur de croisement n'est efficace par rapport au croisement uniforme que lorsque une initialisation gloutonne est utilisée. En effet, en cas d'initialisation aléatoire aucun opérateur de croisement n'est réellement meilleur qu'un autre. On remarque aussi que le croisement NPA complet (i.e. qui permet de choisir tout les points) est assez mauvais. Cela confirme les prémisses énoncées plus haut, à savoir que notre opérateur de croisement NPA-restreint respecte mieux les contraintes déjà satisfaites par l'algorithme glouton, et aussi qu'un trop grand nombre de points de croisement est alors néfaste. Nous allons donc maintenant détailler ces résultats, en fonction du graphe utilisé et en fonction du taux de croisement utilisé.

Lorsqu'on détaille les résultats en fonction des graphes, on remarque des différences de convergence, quel que soit l'opérateur utilisé. Ceci est dû, comme nous l'avions vu au chapitre précédent, à l'initialisation gloutonne qui est plus ou moins efficace selon l'ordre

[4] voir 5.4.3 page 74

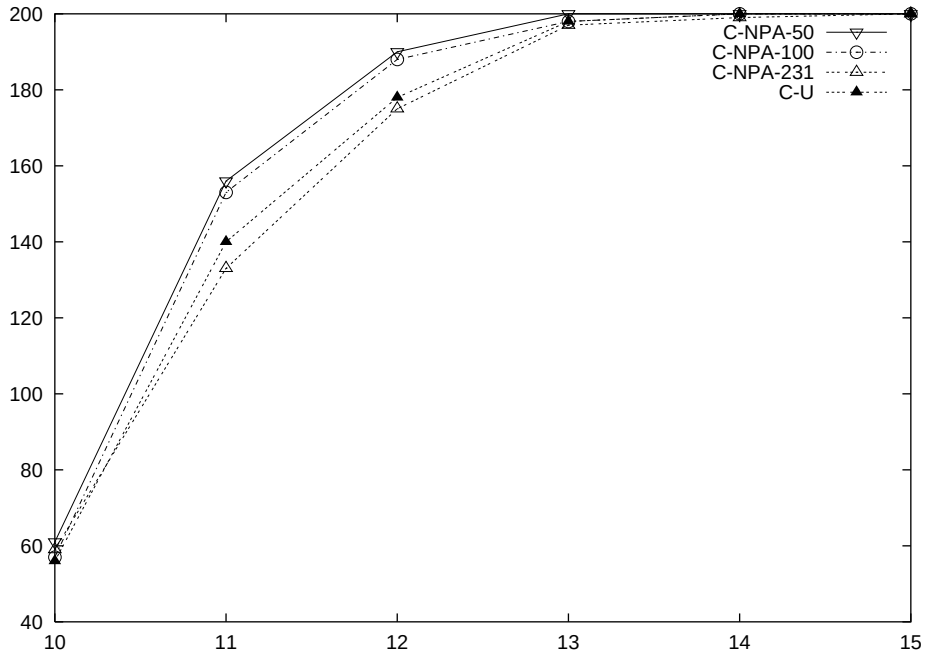


FIG. 6.8 – 3-penta : Total : initialisation gloutonne

des nœuds dans le graphe généré. Cependant, l'opérateur NPA restreint reste globalement meilleur que le croisement uniforme, quel que soit le type de graphe utilisé (Figure 6.10 et 6.11).

Les figures suivantes (6.12, 6.13, 6.14 et 6.15) présentent les résultats obtenus pour chaque graphe et chaque taux de croisement. Nous remarquons que globalement les deux réglages du croisement NPA restreint sont les meilleurs. Cependant, on peut remarquer que le croisement uniforme obtient le plus de valeurs optimales pour le graphe 3-penta-mél, avec un taux de croisement de 1.0 (19/50, figure 6.13), mais ne conserve pas son avance pour les valeurs proches de l'optimal. Le plus grand nombre de valeurs optimales et le meilleur taux de convergence global, est obtenu avec l'opérateur NPA-50, le graphe 3-penta-ord et un taux de croisement à 1.0 (25/50, figure 6.15)[5].

Nous avons pu voir que sur le graphe 3-penta, notre opérateur dans sa forme restreinte se révélait globalement plus efficace que le croisement uniforme. Si on se reporte à la figure 6.8, les deux formes du croisement restreint sont globalement équivalentes en terme de convergence, et sont meilleures que le croisement uniforme et que le croisement NPA dans sa forme complète, ces deux derniers étant équivalents en terme de convergence. On

[5]Nous rappelons que le croisement à 1 point, utilisant une initialisation aléatoire obtient un taux de convergence à l'optimal de 10% (voir table 5.4 page 75)

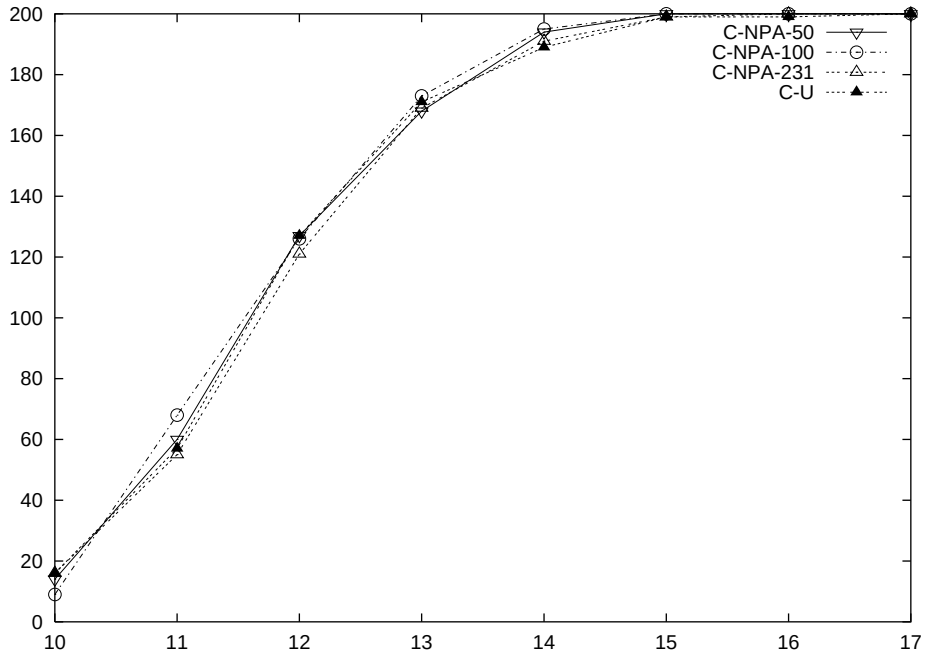


FIG. 6.9 – 3-penta : Total : initialisation aléatoire

peut cependant remarquer que le NPA-50 est plus efficace sur le 3-penta-ord, alors que le NPA-100 l'est sur 3-penta-mél. Ceci confirme le fait que sur de tels problèmes très structurés, des croisements qui utilisent trop de points de croisement sont néfastes. En effet, le graphe 3-penta-mél étant moins structuré que le graphe 3-penta-ord, le croisement NPA-100 qui utilise plus de points est favorisé, et réciproquement pour le croisement NPA-50. Le croisement à 1 point n'a pas été représenté car il est toujours inférieur en terme de convergence aux autres opérateurs de croisement.

Nous avons dit au début de cette section que notre croisement NPA avait une dynamique différente de celle du croisement uniforme, et que son utilisation était plutôt destinée à des graphes très structurés. Afin de vérifier cette assertion, nous avons fait une série de tests sur dix CSP binaires aléatoires, que nous allons maintenant présenter.

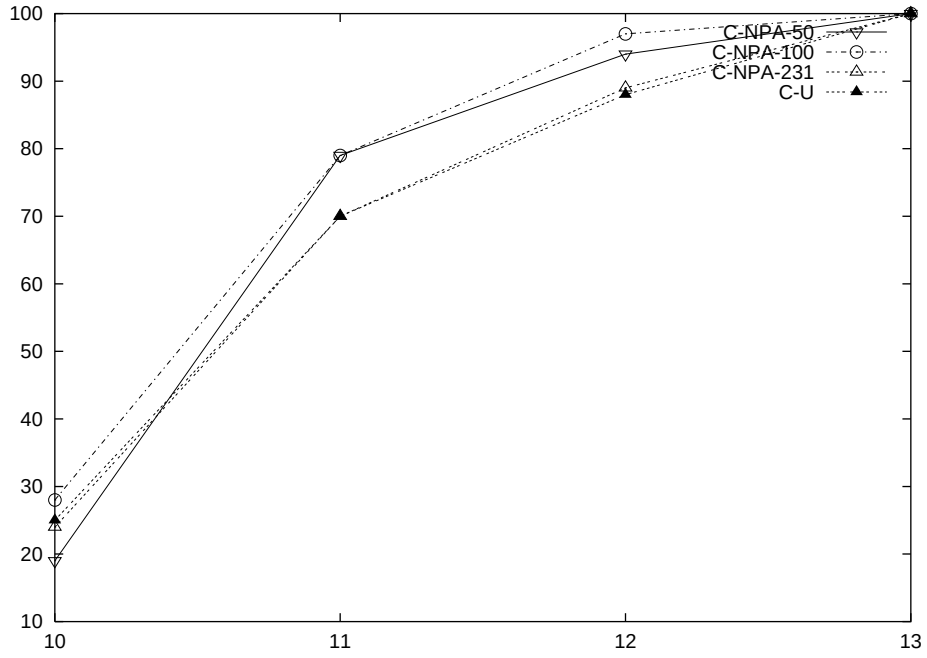


FIG. 6.10 – Total 3-penta mélangé, initialisation gloutonne

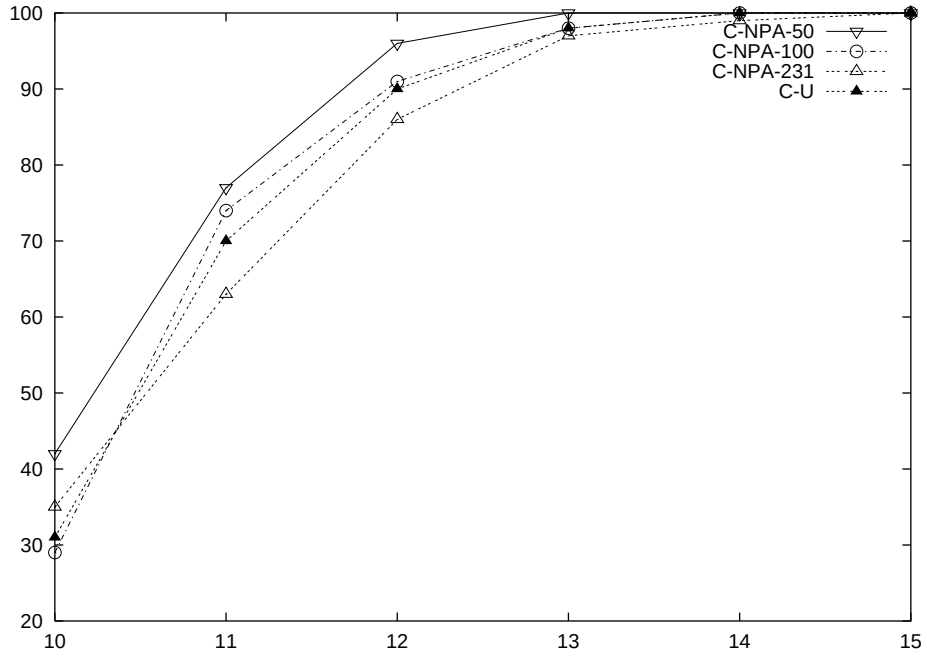


FIG. 6.11 – Total 3-penta ordonné, initialisation gloutonne

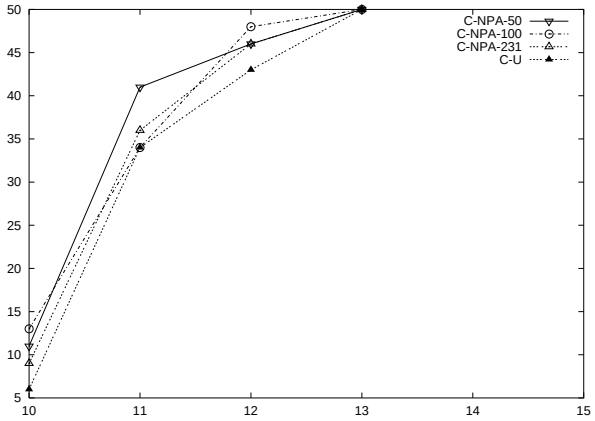


FIG. 6.12 – 3-penta mélangé, initialisation gloutonne, taux de croisement 0.7

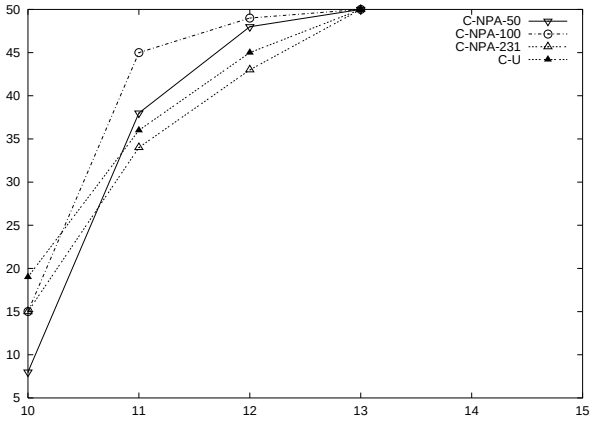


FIG. 6.13 – 3-penta mélangé, initialisation gloutonne, taux de croisement 1.0

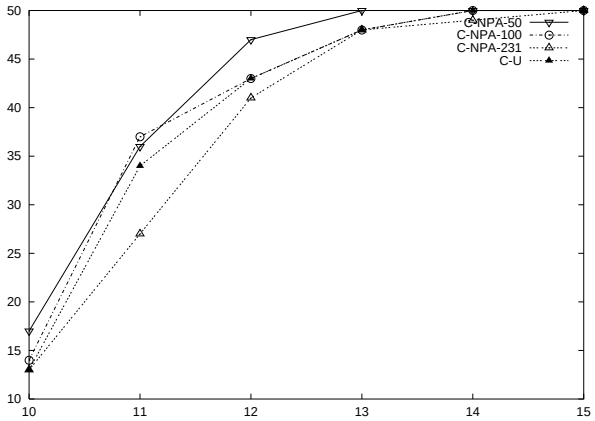


FIG. 6.14 – 3-penta ordonné, initialisation gloutonne, taux de croisement 0.7

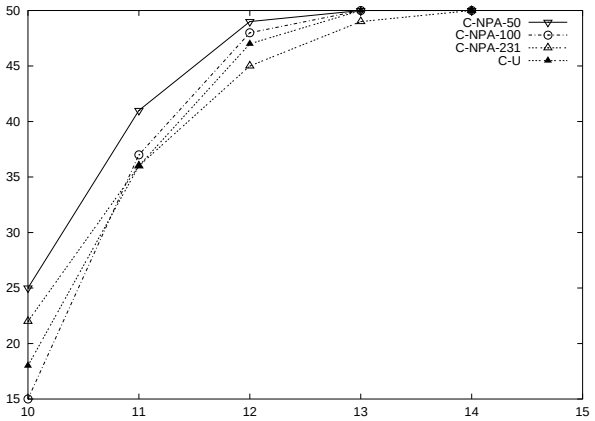


FIG. 6.15 – 3-penta ordonné, initialisation gloutonne, taux de croisement 1.0

**CSP binaires Aléatoires** Nous avons généré dix CSP binaires aléatoires, défini selon le modèle B de [MPSW98], ayant les mêmes caractéristiques globales, à savoir :

- 50 variables.
- toutes les variables ont le même domaine : $D = \{1, \ldots, 10\}$.
- 600 contraintes.
- chaque contrainte interdit 20 couples de valeurs.

Pour chaque exécution, la taille de la population est de 50 individus, le taux de mutation est 0.02, l'élitisme est fixé à 1, la sélection est un tournoi à deux individus, le taux de croisement est 1.0. Pour chacun des dix problèmes, nous avons utilisé huit croisements différents :

- le croisement à 1 point.
- le croisement à n-points classique, avec n=10, n=25, n=50.
- le croisement NPA, avec NPA-10, NPA-25, NPA-50[6].
- le croisement uniforme.

Et enfin, deux initialisations de la population ont été utilisées : aléatoire et gloutonne. Chaque exécution est stoppée après 30000 évaluations.

Les deux premières figures (6.16 et 6.17) représentent respectivement l'ensemble des résultats obtenus avec l'initialisation gloutonne et avec l'initialisation aléatoire, en combinant les résultats de tous les CSP aléatoires.

Les résultats sont conformes à nos attentes : sur des problèmes denses mais n'ayant aucune structure particulière, le croisement uniforme obtient les meilleurs taux de convergence. On remarque de plus que tous les croisements multi-points (à part le croisement à 50 points) se situent entre le croisement uniforme et le croisement à 1-point, ceci étant nettement plus visible sur la figure 6.17. Ces résultats confirment le fait que dans le cadre des CSP, le croisement à 1 point n'est pas le mieux adapté. En effet, les gènes fortement liés par des contraintes n'étant pas forcément côte à côte, le croisement uniforme qui prend en compte toutes les partitions possibles du graphe de manière uniforme obtient de meilleurs résultats. De plus, comme l'ont démontré Spears et De Jong [SD91], avec le croisement uniforme la probabilité d'altération d'un schéma est toujours la même, quel que soit sa longueur, ce qui là aussi est un facteur favorisant.

Le cas du croisement à 50 points (C-NP-50) n'est pas étonnant. Le fait qu'il soit bien moins efficace que les autres opérateurs s'explique par le fait qu'il effectue toujours le même croisement, en prenant pour créer un enfant alternativement le gène 1 du parent 1 puis le gène 2 du parent 2, et ainsi de suite (et inversement pour le deuxième enfant). De ce fait, l'exploitation ne suit qu'une seule direction.

[6] les valeurs 10, 25 et 50 ont été choisies arbitrairement à $\frac{n}{5}$, $\frac{n}{2}$ et $n$, $n$ étant la longueur du chromosome

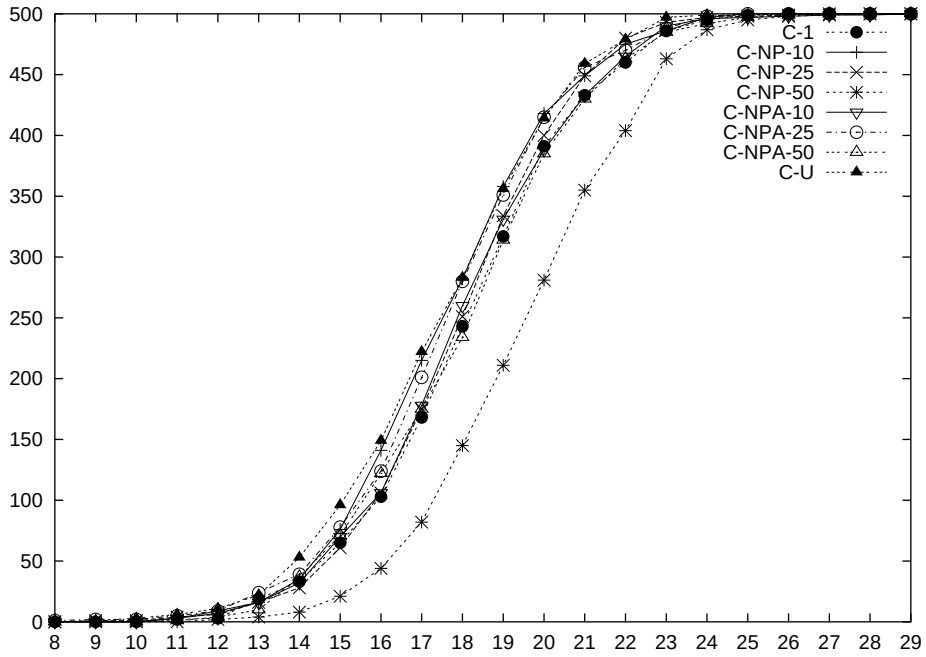


FIG. 6.16 – CSP aléatoire : Total initialisation gloutonne

A titre d'exemple, les deux figures suivantes (6.18, 6.19) représentent respectivement les résultats sur le CSP aléatoire le moins favorable au croisement uniforme, ainsi que le plus favorable. Ces deux figures illustrent que le choix de l'opérateur va évidemment influer sur la qualité de la recherche, mais aussi que les différence de convergence entre les différents opérateurs ne sont pas assez significatives pour décider d'une véritable suprématie de l'un d'eux.

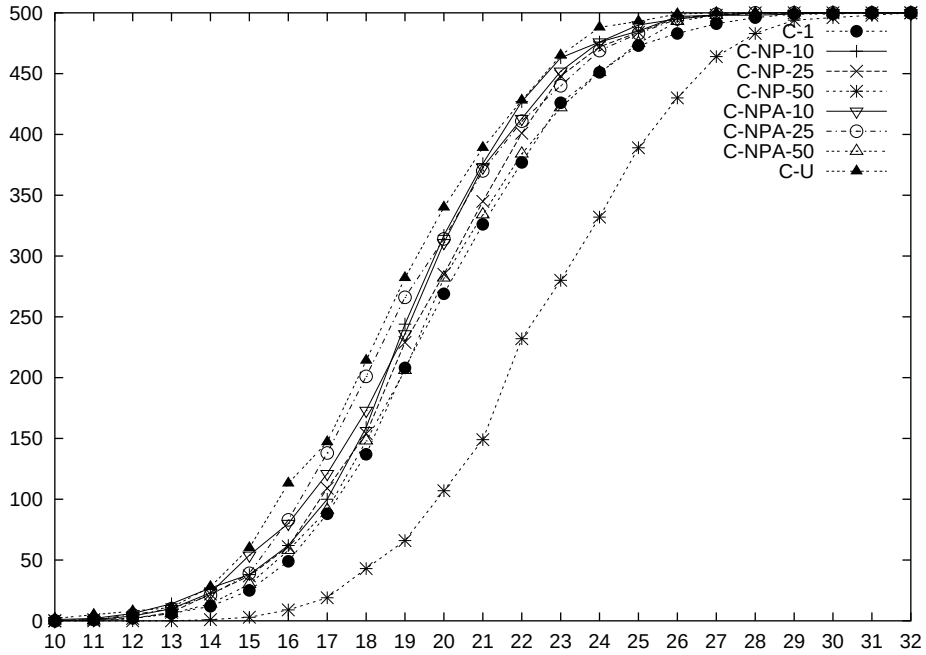


FIG. 6.17 – CSP aléatoire : Total initialisation aléatoire

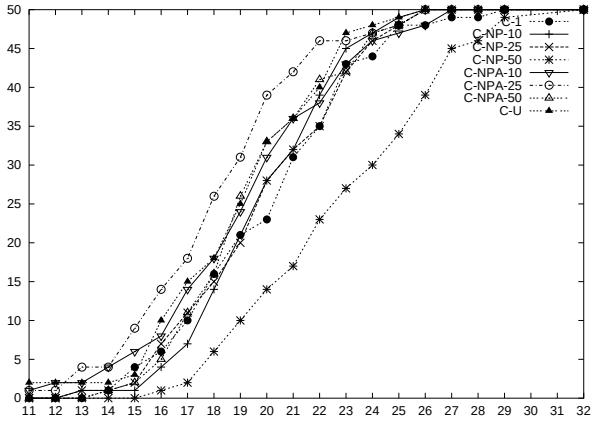


FIG. 6.18 – CSP aléatoire : Résultat sur le problème le moins favorable au croisement uniforme

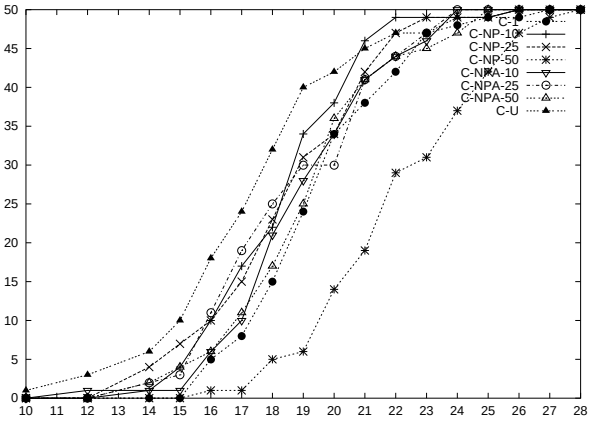


FIG. 6.19 – CSP aléatoire : Résultat sur le problème le plus favorable au croisement uniforme

### 6.2.4 Bilan sur le croisement NPA

Le croisement NPA que nous proposons se révèle globalement efficace, dans sa forme restreinte, sur des problèmes de coloriage de graphes denses et très structurés, et lorsque une initialisation gloutonne est utilisée. Il obtient de bons résultats du fait qu'il respecte mieux les contraintes déjà satisfaites par l'initialisation gloutonne. La version complète du croisement NPA est nettement moins bonne qu'attendu. Ceci est dû au fait qu'un grand nombre de croisements utilise un nombre de points de croisement trop élevé. Le croisement uniforme n'a pas ce problème car les croisements utilisant un grand nombre de points ont une probabilité très faible d'apparaître (cf. figure 6.1 page 89), alors que le croisement NPA complet a une chance sur quatre de choisir un nombre de points supérieur à $\frac{3n}{4}$. D'où la nécessité du croisement NPA restreint. Il serait intéressant de tester ce croisement sur d'autres types de problèmes, dont la structure et les liens entre les gènes satisfont les conditions d'application que nous avons évoquées plus haut. Il serait aussi intéressant d'effectuer une étude complète concernant les biais qu'engendre ce croisement, notamment au niveau des schémas.

On peut cependant se poser la question du choix de l'opérateur de croisement quand ils ne sont pas particulièrement dédiés à un problème. Sans remettre en question l'utilité des opérateurs de croisement, les différences de convergence entre ces différents opérateurs restent assez faibles. Dans l'optique de trouver rapidement une solution acceptable à un problème nouveau (et sur lequel on n'a pas ou peu d'information particulière), on doit pouvoir se satisfaire du croisement à 1 point et du croisement uniforme qui seront choisis en fonction du type de problème et du codage chromosomique utilisé.

Notre propos n'est pas de dire que les autres opérateurs de croisements non spécifiques (n-point, NPA, ...), ne sont pas intéressants : ils sont efficaces sur des problèmes particuliers. Cependant, nous pensons que dans le cadre d'une utilisation «presse-bouton», le croisement uniforme et le croisement à 1 point suffisent amplement pour se faire une idée de l'efficacité de l'AG sur le problème en question et de la qualité des solutions que l'on va obtenir.

## 6.3 Opérateurs Adaptatifs

Le contrôle des paramètres est une des difficultés inhérentes aux AGs [EHM99, CST96] et des méthodes adaptatives sont une des solutions pour faciliter le réglage des paramètres. Chaque opérateur peut inclure un comportement adaptatif. Dans [Ev97], Eiben et van der Hauw ont proposé une fonction d'évaluation adaptative avec des poids adaptatifs pour chaque variable (SAW pour *Stepwise Adaptive Weight*) pour résoudre le problème de

coloriage de graphe : la pénalité associée à une variable est augmentée quand certaines contraintes associées à cette variable sont violées. Un croisement adaptatif avec contraintes dynamiques est proposé dans [Rif98] par Riff-Rojas. Avec ce croisement, l'enfant hérite des gènes en employant une procédure gloutonne, qui analyse chaque contrainte selon une priorité dynamique. Cette priorité dynamique tient compte de la structure de réseau et de la valeur des parents. La première contrainte à être analysée est la plus dure à être satisfaite. Smith et Fogarty proposent une recombinaison multi-parentale [SF96], qui tient compte des gènes liés, définis dynamiquement par un drapeau comme un bloc de gènes sur certains loci. Ils proposent aussi une mutation auto-adaptative où le taux de mutation est incorporé pour chaque bloc, le taux de mutation de chacun des blocs étant défini comme la somme des taux de mutation des gènes le constituant.

Il y a beaucoup d'autres façons de concevoir des opérateurs adaptatifs. Néanmoins, les méthodes cherchant à faire varier les paramètres pendant l'exécution peuvent être classées en trois catégories [EHM99, Ang95] : déterministe, exogène (c'est-à-dire adaptatif) et endogène (c'est-à-dire auto-adaptatif). Les méthodes déterministes contrôlent les paramètres selon une stratégie choisie avant l'exécution et n'emploient pas de réajustement pendant l'exécution. Les méthodes exogènes utilisent les informations données par l'exécution pour changer les valeurs des paramètres et les méthodes endogènes codent le contrôle de paramètres dans les gènes du chromosome lui-même. Nous considérons que, généralement, les méthodes déterministes sont plus intéressantes pour les problèmes statiques, que les méthodes endogènes sont bien conçues pour les problèmes dynamiques (dont la fonction d'évaluation change en fonction du temps) et que les méthodes exogènes peuvent être employées sur les deux types de problèmes. En effet, les méthodes déterministes décident à l'avance du comportement à adopter et ne peuvent donc prendre en compte une variation imprévue dans la topologie du problème traité. Les méthodes exogènes vont adapter les paramètres en fonction des informations recueillies pendant l'exécution et vont pouvoir aussi bien traiter un problème statique en fonction de la convergence de l'AG sur celui-ci qu'une variation du problème lui-même. Nous considérons que les méthodes endogènes sont mal adaptée aux problèmes statiques car elles impliquent de converger à la fois vers le paramétrage idéal et la solution au problème, ce qui implique un ralentissement de la convergence que subiront moins les deux autres méthodes. Par contre dans le cas de problèmes dynamiques, les méthode endogènes permettent de s'adapter à presque tout changement du problème et nous semblent donc les plus valides. Nous nous intéressons ici aux méthodes exogènes et déterministes pour des problèmes statiques.

Les opérateurs de mutation sont les opérateurs d'exploration. Sans exploration, un AG converge vers un optimum local et ne peut pas s'en échapper. Mais avec un AG classique, il est très difficile de trouver le meilleur taux de mutation, de fait $\frac{1}{n}$ semble être une bonne

valeur dans le cas général [Müh92]. L'idée est ici de trouver de nouveaux opérateurs de mutation, avec un faible coût en temps d'exécution et non spécialisés pour un problème particulier.

### 6.3.1 Mutation allélique classique

Avec la mutation allélique chaque gène d'un chromosome a une probabilité d'être muté. Si on utilise un taux de mutation proche de $\frac{1}{n}$ , chaque chromosome a en moyenne un gène muté[7]. Bien que le taux de $\frac{1}{n}$ semble généralement être un bon taux de mutation, cela devient moins intéressant quand l'algorithme a convergé vers un bassin d'attraction d'un optimum local ou d'un site trompeur. L'idée de faire varier le taux de mutation s'est donc dès le départ imposé, et différents schémas ont été proposés, que ce soit par des méthodes déterministes, endogènes ou exogènes.

### 6.3.2 Mutation adaptative

Nous nous intéressons ici aux méthodes adaptatives exogènes, c'est-à-dire celles qui utilisent des informations globales données par l'exécution pour modifier le taux de mutation. Il existe de nombreux schémas d'adaptation du taux de mutation, comme par exemple utiliser une mesure de pression sélective. Dans ce cas, si la pression sélective devient trop forte, on va augmenter le taux de mutation afin d'augmenter l'exploration, et le diminuer sinon. On pourra aussi analyser la vitesse de convergence, dans l'espoir de détecter une convergence prématurée, et dans ce cas augmenter la taux de mutation. A chaque génération, un certain nombre d'informations sont à notre disposition, celles-ci nous permettant d'obtenir une sorte de bilan sur l'état de la population. Parmi elles nous trouvons la meilleure évaluation, l'évaluation moyenne, la pire évaluation, dans certain cas la distance à l'optimal recherché ou encore la distance entre chacun des individus. En sauvegardant ces information à chaque génération, on peut obtenir des informations supplémentaires comme la stagnation de la convergence, l'évolution de l'écart entre la meilleure évaluation et l'évaluation moyenne, les processus d'homogénéisation de la population. Toutes ces informations peuvent donc servir pour adapter les différents paramètres de l'AG.

Nous proposons un opérateur de mutation adaptatif dont le comportement est très classique. Le principe est d'augmenter le taux de mutation quand la population est trop homogène et de le diminuer quand la population est trop hétérogène, cette opération n'étant effectuée qu'une fois par génération. Nous avons besoin d'une fonction de mesure de diversité, mais nous en avons voulu une très simple à calculer. Dans notre moteur d'AG,

[7]Dans le cadre des CSP, cela revient à choisir aléatoirement un voisin dans le voisinage complet à distance 1 de la solution courante

la fonction d'évaluation compte le nombre de contraintes violées. Cela nous donne deux informations importantes :

- la meilleure *fitness* espérée est 0 et nous avons directement une mesure de distance à l'optimum en nombre de contraintes violées.
- Nous avons une mesure entre le meilleur trouvé et la *fitness* moyenne en terme de contraintes violées.

Nous supposons que la différence entre la meilleure *fitness* et la *fitness* moyenne peut être une bonne fonction mesure de diversité simple à calculer et peu coûteuse car elle utilise des valeurs déjà calculées. Notre mesure de diversité (DM) est définie comme suit :

$$DM = MeilleureFitness - FitnessMoyenne \tag{6.1}$$

Cet opérateur adaptatif a des paramètres : nous devons définir les limites pour le taux de mutation, c'est-à-dire le taux minimal ($p_{m_{min}}$) et le taux maximal ($p_{m_{max}}$) accepté pour l'exécution. Par exemple nous ne pouvons pas accepter un taux de mutation à 0.6. Nous avons aussi deux limites pour la mesure de diversité. Les taux sont adaptés seulement quand la mesure de diversité DM est plus grande (resp. plus petite) que $DM_{max}$ (resp. $DM_{min}$). La mesure de diversité que nous avons choisie a l'avantage de permettre un réglage relativement intuitif. Il s'agit de définir les bornes $DM_{min}$ et $DM_{max}$ en fixant le nombre de contraintes violées moyen entre la meilleure solution trouvée et les autres solutions. Finalement nous définissons un facteur de changement $f$. L'algorithme d'adaptation du taux de mutation est :

calculer $DM$
**si** ($DM < DM_{min}$ et $p_m < p_{m_{max}}$)
  **alors**
    $p_m \leftarrow p_m \times f$
**fin si**
**si** ($DM > DM_{max}$ et $p_m > p_{m_{min}}$)
  **alors**
    $p_m \leftarrow \frac{p_m}{f}$
**fin si**

FIG. 6.20 – Algorithme de la mutation adaptative

Les paramètres sont définis comme suit :

- $p_{m_{min}}$ (le taux minimal) est le taux de mutation le plus bas permis (généralement $\frac{1}{10\times n}$). Sous ce taux, aucune exploration réelle n'est possible.
- $p_{m_{max}}$ (le taux maximal) est le taux de mutation le plus haut permis (généralement $\frac{10}{n}$) . Mais cela peut être plus haut pour des problèmes difficiles.
- $DM_{min}$ (la diversité minimale) est l'écart minimal permis entre la *fitness* moyenne et la meilleure *fitness*.
- $DM_{max}$ (la diversité maximale) est l'écart maximal permis entre la *fitness* moyenne et la meilleure *fitness*.
- $f$ le facteur de diminution et d'augmentation (la meilleure valeur semble être entre 1.1 et 1.5)

Remarque : Le facteur de changement est un facteur de réactivité : plus haut il est, plus rapide est la réaction sur le taux. Les figures que nous présentons représentent le nombre d'exécutions ayant trouvé l'optimal en fonction du nombre de générations. Les meilleurs résultats sont donc les courbes ayant la croissance la plus rapide. Cela nous permet de visualiser en même temps le taux de convergence et la vitesse de convergence.

Les figures 6.21 à 6.24, présentent les résultats obtenus sur différents coloriages de graphes (jean, myciel7, et queen5_5)[8]. Nous avons effectué ces tests en utilisant différents réglages pour le taux de mutation classique et pour les réglages des paramètres de la mutation adaptative. Pour les autres paramètres nous avons utilisé un croisement à 1 point avec un taux de 1.0. La sélection se fait par tournoi (à 2 individus), l'initialisation est aléatoire. Nous pouvons remarquer que notre opérateur, bien que n'ayant pas de mauvais résultats, ne peut généralement surpasser un bon réglage de la mutation classique. Nous avions choisi pour les réglages de cet opérateur certains paramétrages qui nous paraissaient intuitivement bons, et d'autres intuitivement mauvais afin d'évaluer son comportement. Cependant les résultats n'ont pas été ceux attendus : certains «mauvais» réglages n'étaient pas si mauvais que ça et réciproquement. Par exemple, sur les résultats concernant le graphe myciel7, les réglages 10-10 (on augmente $p_m$ si $DM < 10$, on le diminue si $DM > 10$) et 10-8 (on augmente $p_m$ si $DM < 8$, on le diminue si $DM > 10$), semblaient assez proches et peu prometteurs. En effet, ce réglage impose d'avoir une différence d'environ 10 contraintes non satisfaites entre la meilleure *fitness* et la *fitness* moyenne. Cependant, le réglage 10-10 s'est révélé être assez mauvais, comme nous nous y attendions, mais pas le réglage 10-8 qui figure parmi les meilleurs.

Il en ressort que le réglage des paramètres s'avère difficile. En effet, bien que rarement vraiment mauvais, l'écart entre les différents réglages est important. De plus, il est difficile de trouver une conjecture de réglage des paramètres comme on peut l'avoir pour la mutation

[8]On peut trouver ces instances sur la page du challenge de coloriage de graphe de CP'2002 : `http ://mat.gsia.cmu.edu/COLORING02/`

classique.

L'idée d'un taux de mutation qui alterne entre valeur forte et valeur faible nous a cependant paru intéressant à conserver, mais nous nous sommes tourné vers un opérateur adaptatif déterministe. L'idée étant toujours d'avoir des opérateurs dont les réglages sont moins longs que pour les opérateurs classiques.

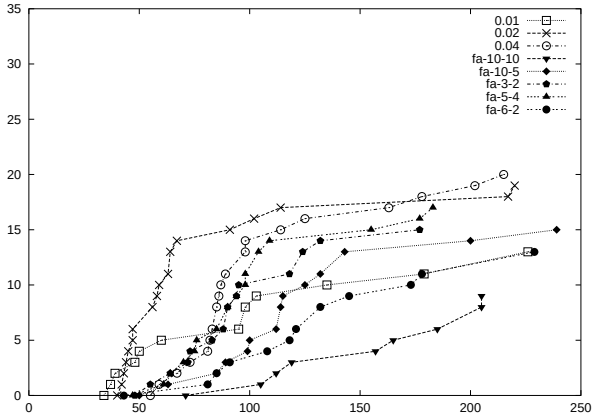


FIG. 6.21 – Résultats avec le graphe jean : mutation adaptative et mutation classique

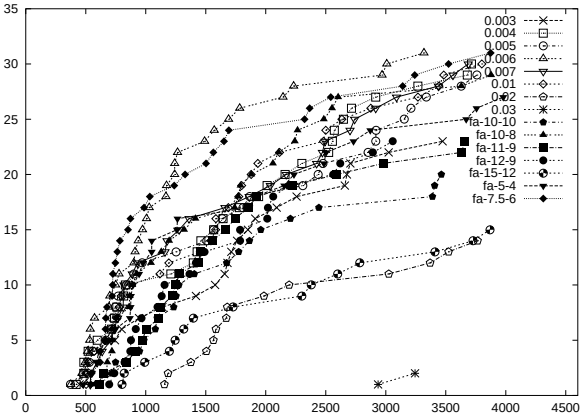


FIG. 6.22 – Résultats avec le graphe myciel7 : mutation adaptative et mutation classique

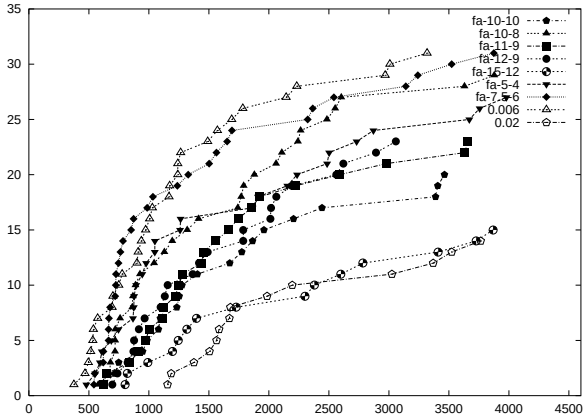


FIG. 6.23 – Résultats avec le graphe myciel7 : mutation adaptative et mutation classique (seulement les extrema)

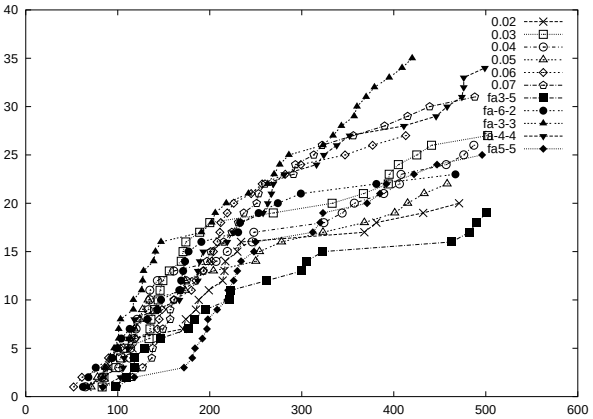


FIG. 6.24 – Résultats avec le graphe queen5_5 : mutation adaptative et mutation classique

### 6.3.3 Mutation déterministe

On sait qu'un fort taux de mutation augmente la diversité, qu'un taux faible aide la convergence et qu'il est difficile de trouver le bon compromis. Holland [Hol75] a dès le départ préconisé un taux de mutation dépendant du temps. Une méthode consiste à diminuer le taux de mutation pendant l'exécution. Fogarty a proposé la première de ces méthodes dans [Fog89]. Beaucoup de schémas de décroissance déterministe ont été proposés pour le taux de mutation dans la littérature. L'idée est venue du recuit simulé, qui accepte moins de mauvaises valeurs dans le voisinage à la fin de l'exécution qu'au commencement. Hesser et Manner ont proposé un taux de mutation décroissant pour le problème de comptage des 1 dans [HM90], qui tient compte du nombre de générations et de la taille de la population :

$$p_m(t) = \sqrt{\frac{\alpha}{\beta}} \times \frac{exp(\frac{-\gamma t}{2})}{\lambda\sqrt{n}} \tag{6.2}$$

où $\alpha$, $\beta$, $\gamma$ sont des constantes, $\lambda$ la taille de population, $n$ le nombre de gènes dans un chromosome et $t$ le nombre de générations effectuées. Une autre idée qui est de diminuer le taux de mutation en fonction de la distance à l'optimum a été proposée par Bäck dans [Bäc92] :

$$p_m(f(\vec{x})) \approx \frac{1}{2(f(\vec{x})+1) - n} \tag{6.3}$$

Bäck à aussi proposé avec Schültz [BS96] un taux de mutation diminuant de 0.5 à $\frac{1}{n}$ à condition de connaître le nombre d'évaluations avant l'exécution :

$$p_m(t) = \left(2 + \frac{n-2}{T} \times t\right)^{-1} \tag{6.4}$$

$n$ est la longueur du chromosome, $T$ le nombre maximal d'évaluations et $t$ le nombre actuel d'évaluations. Ces opérateurs sont très intéressants mais posent problème lorsque l'AG est bloqué dans un optimum local vers la fin de l'exécution. Il est alors impossible de sortir de cet optimum local. Quand l'AG est bloqué dans un optimum local, il serait intéressant d'augmenter le taux de mutation. Mais il est difficile de se rendre compte d'un tel blocage pendant l'exécution. En outre, le taux de mutation doit être diminué quand l'algorithme s'est échappé de cet optimum local.

Une solution serait d'augmenter lentement le taux de mutation pendant l'exécution. L'idée vient du fait que l'initialisation aléatoire permet une bonne diversité dans la population initiale et que la diversité diminue au cours de l'exécution en raison de la pression sélective. Augmenter le taux de mutation devrait pouvoir préserver une bonne diversité, mais va à l'encontre de l'exploitation. En effet, si le taux de mutation est trop élevé, la convergence de l'algorithme est altérée voire stoppée. Donc le schéma d'augmentation linéaire du taux de mutation est plutôt une mauvaise idée. Nous pensons cependant qu'en

alternant l'augmentation et la diminution on peut obtenir un opérateur efficace, donc nous proposons un opérateur de mutation dont le taux suit une sinusoïde. Cela permet à l'algorithme de sortir des optima locaux et conserver une bonne convergence. Le taux de mutation est défini comme suit :

$$p_m(t) = P_M + \left( sin\left(\frac{t}{\gamma.\pi}\right) \times \alpha \right) \tag{6.5}$$

où $\alpha$, $\gamma$ sont des constantes, $P_M$ le taux de mutation par défaut (généralement $\frac{1}{n}$), $n$ la longueur du chromosome et $t$ le nombre de générations.

- la période du sinus est définie par la constante $\gamma$ (en nombre de générations).
- l'amplitude du sinus est définie par la constante $\alpha$ et $\alpha < P_M$.

Pendant l'exécution, l'AG va alterner entre exploration et exploitation, de fait la convergence peut prendre un peu plus de temps. Cependant, l'algorithme devrait rester bloqué moins longtemps dans des optima locaux. Deux cas se présentent :

- Après une période de forte exploration, la population est assez hétérogène. Le taux de mutation devenant plus faible, le croisement va pouvoir faire son travail d'exploitation à partir de cette population et ne sera pas gêné par une mutation trop élevée.
- Après un période de faible exploration, la population s'est homogénéisée autour des meilleures solutions, le taux de mutation redevenant fort, l'exploration va pouvoir se faire beaucoup plus largement autour des meilleures valeurs trouvées.

Nous donnons un exemple sur la figure 6.25 de l'évolution de la meilleure évaluation, de l'évaluation moyenne de la population et du taux de mutation sur une portion d'une exécution ( de la génération 250 à la génération 400) pour illustrer ces phénomènes.

Nous présentons les résultats que nous avons obtenus avec cet opérateur sur le graphe myciel7, en comparaison avec l'opérateur de mutation classique dans le tableau 6.1. Pour cela, nous avons choisi les quatre meilleurs taux de mutation classique (0.004 à 0.007), et pour chacun d'eux nous avons essayé quatre périodes pour le sinus (10, 20, 50 et 100 générations), ainsi que quatre amplitudes (0.0005, 0.001, 0.002, 0.003). Pour chaque paramétrage, nous avons effectué 50 exécutions. Les autres paramètres de l'AG sont le croisement à un point avec un taux à 1.0, une sélection par tournoi (à 2 individus), une population de 40 et une initialisation aléatoire. Chaque exécution est arrêtée après 150000 évaluations. Chaque case du tableau représente le pourcentage d'exécutions qui ont atteint l'optimal. Nous avons fait figurer en gras, les cas où notre opérateur fait au moins aussi bien que la mutation classique quel que soit son réglage.

On peut remarquer que lorsque l'amplitude est faible (environ 10% du taux de mutation), notre opérateur de mutation a globalement de meilleurs résultats que la mutation classique (dans près de 90% des cas). Pour les autres réglages, et bien que certains résultats soient intéressants, il est difficile d'exhiber un comportement. Il y a cependant un

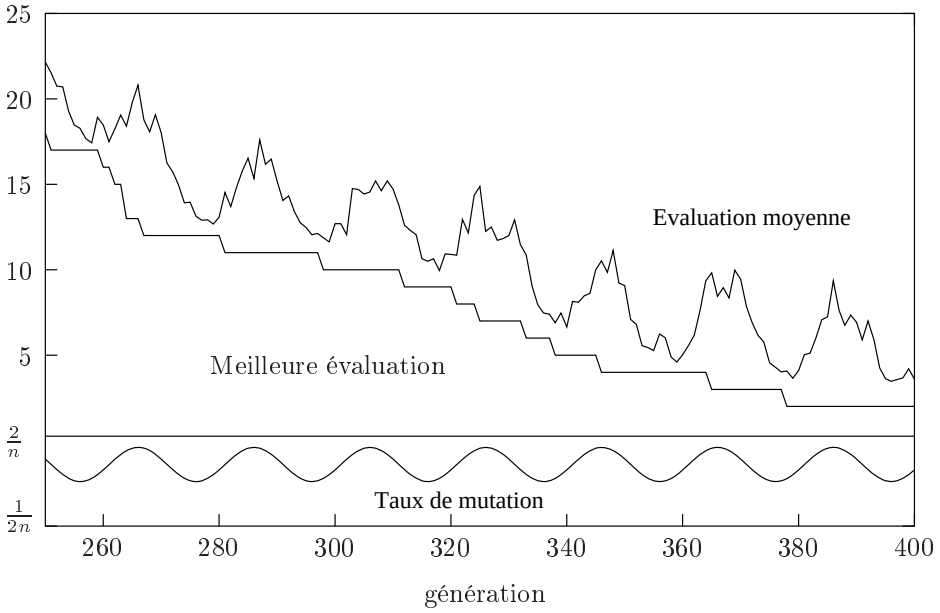


FIG. 6.25 – Mutation sinusoïdale : évolution de la meilleure *fitness*, de la *fitness* moyenne et du taux de mutation

intérêt certain à utiliser cet opérateur avec une faible amplitude. Cela permet de ne plus se focaliser sur le réglage du taux de mutation en cherchant le meilleur, mais de prendre un taux de mutation arbitraire (proche de $\frac{1}{n}$), et d'appliquer la mutation sinusoïdale avec une faible amplitude. En faisant cela, on a de très bonnes chances d'obtenir de meilleurs résultats qu'avec la mutation classique.

Beaucoup d'améliorations sont envisageables pour cet opérateur. Par exemple, définir l'amplitude du sinus comme un pourcentage du taux de mutation plutôt qu'en utilisant une valeur fixée. On peut aussi facilement envisager d'introduire un sinus amorti en fonction du nombre de générations, de manière à se focaliser progressivement vers la solution lorsqu'on s'en rapproche, particulièrement en combinant la mutation sinusoïdale amortie avec un opérateur de mutation décroissante.

| | | Taux de Mutation | | | |
|---|---|---|---|---|---|
| Amplitude | Fréquence | 0,004 | 0,005 | 0,006 | 0,007 |
| 0,0005 | 10 | **68%** | 60% | 54% | **72%** |
| | 20 | **62%** | **72%** | **72%** | **70%** |
| | 50 | **64%** | **70%** | **66%** | **80%** |
| | 100 | **62%** | **62%** | **74%** | **62%** |
| 0,001 | 10 | 44% | **64%** | **66%** | 60% |
| | 20 | 54% | **62%** | **78%** | **64%** |
| | 50 | 56% | 52% | 60% | 56% |
| | 100 | 50% | **64%** | 56% | **68%** |
| 0,002 | 10 | 56% | **70%** | **68%** | **64%** |
| | 20 | 56% | **64%** | 60% | 56% |
| | 50 | 60% | **68%** | **70%** | **70%** |
| | 100 | 54% | 56% | **66%** | 56% |
| 0,003 | 10 | 44% | **62%** | **68%** | 54% |
| | 20 | 58% | **74%** | **62%** | **66%** |
| | 50 | **62%** | **74%** | 56% | 60% |
| | 100 | 58% | 54% | **68%** | **64%** |
| Mutation classique | | 60% | 60% | 62% | 60% |

TAB. 6.1 – Taux de réussite de l'opérateur sinus mutation sur le graphe myciel7

## 6.4 Opérateur de Diversification

Empêcher la convergence prématurée, par perte de diversité, d'un AG est un des principaux problèmes, et donc beaucoup de méthodes s'y sont attachées. Des analyses concernant les raisons de la convergence prématurée des AG ont été proposées [LGX97, Rud94, GS00], utilisant différentes méthodes pour essayer d'en caractériser les principes. D'autres auteurs ont proposé des méthodes cherchant à limiter la perte de diversité. On trouvera entre autres, les algorithmes religieux de Thomsen et al. [TRK00], la prévention de l'inceste d'Eshelmann et Shaffer [ES91] ou encore les individus multiploïdes de Collingwood, Corne et Ross [CCR96][9]. D'autres méthodes essayent de réintroduire de la diversité dans la population pour conserver une bonne diversité. Différentes solutions s'offrent à nous : introduction de nouveaux individus aléatoires, changement de taux de mutation, recherche hybride, utilisation d'un algorithme *steady-state*, d'un algorithme co-évolutionnaire ou encore de technique de *sharing* par exemple. Bien que les deux dernières méthodes aient été

[9] Voir aussi section 3.3.3 page 42

plus prévues pour trouver plusieurs solutions simultanément, elles permettent, en explorant parallèlement plusieurs sous espaces de l'espace de recherche, d'éviter une convergence prématurée. Une autre solution consiste à trouver un moyen de déplacer les individus d'un sous espace de l'espace de recherche vers un autre, lorsque le sous espace exploré ne semble plus prometteur. L'opérateur que nous proposons combine deux de ces principes : réintroduire régulièrement des individus génétiquement nouveaux en se basant sur des individus de la population courante pour ne pas faire chuter la *fitness* moyenne.

### 6.4.1 Principe

Dans le coloriage de graphe, un grand nombre de solutions potentielles ont la même *fitness* avec des génotypes très différents. Deux solutions peuvent être génotypiquement différentes, mais phénotypiquement identiques. C'est une idée émergente dans la communauté AG, que de chercher à se servir de cette particularité que partagent un grand nombre de problèmes. On parle en général de réseaux de neutralité. L'idée est alors de chercher des opérateurs neutres, c'est à dire des opérateurs qui n'influent pas sur la *fitness* de l'individu, mais uniquement sur sa représentation génotypique. Par exemple sur la figure 6.26, les deux graphes ont exactement le même nombre de contraintes violées, mais sont vraiment différents. Aucun nœud n'a la même couleur, et leurs chromosomes correspondants sont différents aussi.

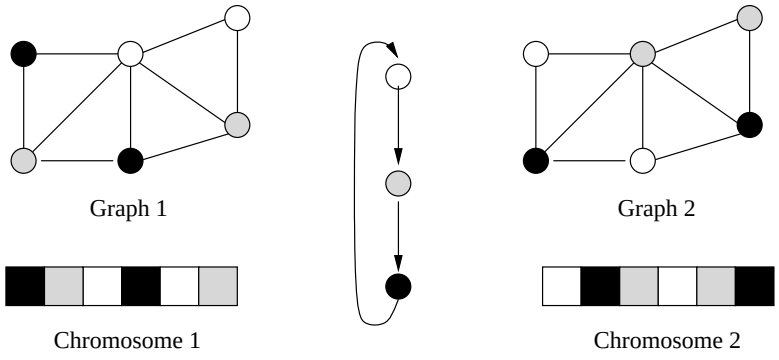


FIG. 6.26 – Deux solutions avec des génotypes différents et le même phénotype et l'opération d'échange du premier à la seconde

Dans le cas de cet exemple, trouver la seconde instanciation à partir de la première est simple : il suffit d'effectuer un décalage par rapport au domaine. Cette idée est facilement applicable à un opérateur génétique. Notre moteur d'AG emploie un codage en nombre entier pour les gènes. Cet opérateur choisit une valeur aléatoirement dans le domaine et ajoute cette valeur à la valeur de chaque gène du chromosome modulo la taille du domaine. Plusieurs paramètres ont été intégrés à cet opérateur :

- le pourcentage de la population sur lequel l'opérateur sera appliqué $p$
- la fréquence de l'application de l'opérateur $f$ en nombre de générations
- le nombre d'individus différents que l'on va pouvoir générer, c'est à dire limiter le choix de la valeur de décalage $d_{max}$

Il suit l'algorithme suivant :

**si** $ng$ le nombre de générations tel que $ng$ modulo $f = 0$
**alors**
  **pour** chaque individu **faire**
    choisir aléatoirement la valeur de décalage $d \in D$ tel que $d \leq d_{max}$
    **si** $r \in [0..1]$ un nombre aléatoire tel que $r \leq p$
    **alors**
      **pour** chaque gène $i \in C$ **faire**
        $i \leftarrow (i + d)$ modulo $|D|$
      **fin pour**
    **fin si**
  **fin pour**
**fin si**

FIG. 6.27 – Algorithme de l'opérateur de diversification

Nous allons motiver maintenant chacun des paramètres de cet opérateur. Nous pensons qu'appliquer notre opérateur sur l'intégralité de la population serait une erreur. En effet, l'action de cet opérateur consiste à déplacer les solutions vers d'autres régions de l'espace de recherche. Si toute la population est déplacée vers d'autres régions de l'espace de recherche, on perd alors la partie de l'espace de recherche que l'on était en train d'explorer.

Appliquer l'opérateur à chaque génération serait aussi une erreur. On n'aurait pas le temps d'exploiter les nouvelles parties de l'espace de recherche découvertes grâce à l'opérateur. Il est donc préférable de ne l'appliquer qu'à intervalle régulier.

Si on choisit la valeur de décalage parmi toutes les valeurs disponibles dans le domaine, on va générer trop d'individus différents, particulièrement quand le domaine est grand. On risque alors d'avoir une diversité génotypique trop importante, et donc d'avoir des croisements inefficaces, voire destructeurs.

### 6.4.2 Résultats

Les résultats que nous présentons dans la table 6.2 représentent le pourcentage de convergence à l'optimal sur le coloriage de graphe myciel7 (en huit couleurs). Nous avons

utilisé cinq taux de mutation (de 0.003 à 0.007), et pour l'opérateur de diversification nous avons utilisé deux fréquences d'applications (30 et 100 générations) trois proportions de la population (30%, 50% et 80 %) et trois décalages (3, 5 et 8 décalages maximum). Pour les autres paramètres de l'AG, nous avons utilisé un croisement à 1 point avec un taux de 1.0, une sélection par tournoi (à 2 individus) et une population de 40 individus. Pour chaque paramétrage, nous avons effectué 50 exécutions, chacune étant arrêtée après 150000 évaluations. Nous avons fait figurer en gras les résultats qui obtiennent mieux en utilisant l'opérateur de diversification que sans l'utiliser (à taux de mutation égal) sur le graphe myciel7 (figure 6.2). Dans ce tableau, Fréquence représente la fréquence en nombre de génération à laquelle l'opérateur est appliqué, # représente la valeur de décalage maximum permise, Taux représente la proportion de la population sur laquelle l'opérateur est appliqué. Référence donne les taux de convergence lorsque l'opérateur n'est pas utilisé, tous les autres réglages étant identiques.

| | | | Taux de Mutation | | | | |
|---|---|---|---|---|---|---|---|
| Fréquence | # | Taux | 0,003 | 0,004 | 0,005 | 0,006 | 0,007 |
| 30 | 3 | 0,3 | **54%** | 52% | 46% | 58% | **68%** |
| | | 0,5 | **52%** | 44% | **64%** | 52% | **64%** |
| | | 0,8 | 34% | 52% | 48% | 58% | 58% |
| | 5 | 0,3 | 40% | 56% | **66%** | **78%** | 54% |
| | | 0,5 | **50%** | **68%** | 54% | 48% | **66%** |
| | | 0,8 | **54%** | 56% | 48% | 54% | 42% |
| | 8 | 0,3 | **56%** | 56% | **62%** | **66%** | **68%** |
| | | 0,5 | 40% | 48% | **62%** | **64%** | **72%** |
| | | 0,8 | 38% | 54% | 48% | 48% | 48% |
| 100 | 3 | 0,3 | **50%** | **66%** | **72%** | **74%** | **62%** |
| | | 0,5 | **50%** | 56% | **72%** | **70%** | **68%** |
| | | 0,8 | **58%** | **62%** | **72%** | **66%** | **68%** |
| | 5 | 0,3 | 36% | **60%** | **68%** | 60% | **60%** |
| | | 0,5 | **58%** | 54% | 50% | 52% | **64%** |
| | | 0,8 | **46%** | **60%** | **62%** | 52% | **66%** |
| | 8 | 0,3 | **60%** | 56% | 58% | 60% | **72%** |
| | | 0,5 | **48%** | 58% | **68%** | **68%** | **64%** |
| | | 0,8 | 42% | 44% | 54% | 60% | **68%** |
| Référence | | | 46% | 60% | 60% | 62% | 60% |

TAB. 6.2 – Résultat de l'opérateur de diversification sur le graphe myciel7

Bien qu'on arrive à obtenir de meilleurs résultats en utilisant l'opérateur de diversification, qu'avec uniquement les opérateurs classiques (mutation, croisement), le réglage de ses paramètres s'avère tout aussi difficile et fastidieux. Ceci se confirme avec les tests effectués sur le graphe 3-penta (tableau 6.3 et 6.4). Pour ces tests, nous avons utilisé les réglages qui paraissaient les plus prometteurs, mais là encore trouver le bon réglage s'avère difficile, de plus le gain obtenu n'est pas très important.

| | Meilleur | | | | |
|---|---|---|---|---|---|
| Décalage | 10 | $\leq 11$ | $\leq$12 | $\leq$13 | $\leq$14 |
| 3 | 6 | 20 | 37 | 47 | 49 |
| 5 | 3 | 12 | 31 | 44 | 48 |
| 10 | 5 | 17 | 29 | 43 | 47 |
| 25 | 8 | 19 | 38 | 45 | 50 |
| réf. | 3 | 18 | 37 | 44 | 50 |

TAB. 6.3 – Meilleurs résultats sur le graphe 3-penta avec un taux de croisement de 0.9, un taux de mutation de 0.004, une fréquence de 200 générations et une proportion de la population concernée de 0.3

| | Meilleur | | | | |
|---|---|---|---|---|---|
| Décalage | 10 | $\leq 11$ | $\leq 12$ | $\leq 13$ | $\leq 14$ |
| 3 | 2 | 16 | 32 | 44 | 50 |
| 5 | 9 | 24 | 38 | 47 | 49 |
| 10 | 7 | 22 | 37 | 45 | 50 |
| 25 | 5 | 19 | 37 | 47 | 49 |
| Référence | 5 | 20 | 33 | 49 | 49 |

TAB. 6.4 – Meilleurs résultats sur le graphe 3-penta avec un taux de croisement de 1.0, un taux de mutation de 0.004, une fréquence de 50 générations et une proportion de la population concernée de 0.1

Nous pensons que les résultats mitigés que nous obtenons avec cet opérateur qui semblait prometteur, sont dus à l'effet destructeur des bonnes solutions que peut avoir le croisement entre deux individus issus de deux régions différentes de l'espace de recherche. Nous pensons que pour obtenir de meilleurs résultats, deux solutions sont envisageables. La première serait de favoriser les croisements entre les individus issus du même espace

de recherche, ou même d'interdire tout croisement d'individus trop différents. Ceci correspondrait en fait à utiliser un opérateur de croisement restreint comme il en existe dans les AG appliquant le principe de *sharing*. La deuxième, qui nous paraît plus intéressante, serait d'utiliser un opérateur de croisement qui prend en compte le graphe des contraintes et favoriserait les croisements les moins destructeurs. Un tel croisement a été proposé par Riff-Rojas dans [Rif97] (Arc-Crossover). Avec ce croisement, la probabilité de choisir un (ou plusieurs) point(s) de croisement n'est plus uniforme. Chaque point de croisement possible a un probabilité d'être choisi qui est inversement proportionnelle au nombre d'arcs reliant deux variables situées de part et d'autre de ce point. On favorise ainsi les croisements qui conservent les ensembles de gènes les plus interconnectés. De cette façon, l'opérateur de croisement effectue une exploitation plus fine des individus.

## 6.5 Conclusion

Dans ce chapitre, nous avons présenté un nouvel opérateur de croisement efficace sur les coloriages de graphe denses et très structurés. Il serait intéressant de l'évaluer sur d'autres problèmes ayant une structure similaire afin de dégager un type de problèmes sur lesquels cet opérateur peut s'appliquer avec succès. Il serait aussi très intéressant d'étudier la dynamique de cet opérateur par rapport à la théorie des schémas. On pourrait aussi envisager une forme adaptative, de manière à adapter le nombre de points de croisement maximum au cours de l'exécution. Une voie qui nous paraît particulièrement intéressante à explorer serait de diminuer progressivement le nombre maximum de points.

Nous proposons un opérateur de mutation sinusoïdal qui obtient de bons résultats en permettant d'effectuer moins de réglages du taux de mutation. Cet opérateur est particulièrement intéressant dans le cadre d'une utilisation sur des problèmes nouveaux sur lesquels on a très peu d'information. Il permet d'évaluer rapidement l'efficacité d'un AG sans faire de réglages fins du taux de mutation tout en obtenant de bons résultats en terme de convergence. Nous pensons qu'il serait intéressant d'évaluer l'efficacité de cet opérateur sur des problèmes autres que les CSP, par exemple sur des problèmes dédiés à l'étude de la dynamique de AG en codage binaire.

Enfin, nous présentons aussi un opérateur de diversification dédié au coloriage de graphe. Cet opérateur utilise les réseaux de neutralité intrinsèque des graphes de coloriage pour réintroduire de la diversité sans altérer les individus. Cependant, il paraît évident qu'un opérateur de croisement spécialisé est nécessaire pour exploiter correctement la diversité introduite. Nous pensons d'ailleurs, que de nombreuses améliorations sont possibles en exploitant ces réseaux de neutralité, dans tous les domaines où les AG s'appliquent.

# Conclusion

L'idée principale dans cette thèse était de lutter contre certaines idées reçues concernant les algorithmes génétiques. Bien que ces idées soient souvent fondées, nous voulions montrer que dans certains cas elles pouvaient se révéler fausses. Dans le chapitre 5, nous avons montré qu'un AG pouvait être efficace sur des problèmes de coloriage sur-contraints, par rapport aux méthodes arborescentes et aux méthodes locales, si toutes étaient utilisées sans réglage particulier. Ces coloriages ont cependant plusieurs particularités :

- leur nombre chromatique est grand,
- leurs nœuds sont très interconnectés,
- la clique maximum est plus petite que le nombre de couleurs nécessaires.

Ils serait intéressant d'étendre ces résultats à une famille de graphes plus vaste. L'idée du protocole de comparaison que nous avons proposé pourrait être étendu au temps nécessaire à l'implantation de chacune des méthodes, mais ceci risque d'être difficilement mesurable, de plus de nombreuses implantations libres sont disponibles et peuvent être avantageusement utilisées.

Il est évident que les algorithmes génétiques utilisés tels quels sans spécialisation ne peuvent être meilleurs que d'autres méthodes lorsqu'elles intègrent une connaissance suffisante du problème, et ce n'est pas notre propos ici. Notre démarche est simplement de dire que face à un problème nouveau dont on veut rapidement une solution acceptable, utiliser un AG peut être intéressant, particulièrement si le problème est de grande taille.

Toujours en suivant l'idée de rester au plus proche de l'AG standard, nous proposons de nouveaux opérateurs de croisement et de mutation peu coûteux en terme de temps de calcul. Ces opérateurs, bien que développés dans le cadre des CSP en domaines finis, devraient être applicables à d'autres champs d'actions des algorithmes génétiques. L'opérateur de croisement à nombre de points aléatoire que nous proposons se révèle efficace sur les coloriages de graphe dont les nœuds sont fortement interconnectés et obtient de meilleurs résultats que les autres opérateurs de croisement classiques. Il n'était pas question de se comparer à des opérateurs spécifiques, l'idée étant de proposer des opérateurs efficaces dans le cadre d'une utilisation sans réglage ni spécialisation. L'opérateur de mutation sinusoïdale que nous proposons permet de s'abstraire partiellement du réglage du taux

de mutation. En effet, nous avons vu qu'il obtient, lorsque l'amplitude du sinus n'est pas trop élevée, de meilleurs résultats que la mutation classique. De plus cet opérateur, bien qu'expérimenté sur des coloriages de graphe, peut s'utiliser sur tous types de problèmes. Il serait donc profitable de le tester sur d'autres problèmes, et notamment sur des problèmes dédiés à l'étude de la dynamique des algorithmes génétiques.

Le dernier opérateur que nous proposons tente d'exploiter les réseaux de neutralité des solutions potentielles d'un graphe de coloriage. Nous entendons par réseaux de neutralité, dans ce cas précis, un ensemble de solutions potentielles qui ont le même nombre de contraintes satisfaites mais dont les valeurs affectées aux variables sont différentes. Cependant cet opérateur, bien qu'obtenant des résultats satisfaisants, devrait être utilisé conjointement avec un opérateur de croisement spécifique. Nous pensons que l'exploitation de ces réseaux de neutralité font partie des axes prometteurs pour les algorithmes génétiques, que ce soit par exploitation directe, comme pour le coloriage de graphe, ou bien par création artificielle de tels réseaux par utilisation d'un codage particulier. Par exemple, les travaux de Collard et Clergue [CC99] proposent de dupliquer l'espace de recherche afin de permettre à l'algorithme d'éviter les sites trompeurs. On peut imaginer de nombreuses façons de créer artificiellement des réseaux de neutralité : par exemple on peut, dans le cadre de codage binaire des entiers, utiliser la représentation de suites géométriques dont la raison est inférieur à $2^n$. Ceci permet d'obtenir une redondance dans le codage qui peut ensuite être exploitée par des opérateurs neutres. Conjointement à l'exploitation des réseaux de neutralité, ils nous paraît intéressant d'utiliser de l'élitisme, et particulièrement de conserver les meilleurs individus issus de différents réseaux, afin de ne pas perdre les avantages que peuvent représenter certaines parties de l'espace de recherche.

Un autre point qui nous paraît important est que la plupart des bibliothèques libres qui sont proposées manquent généralement de généricité, et demandent à l'utilisateur un investissement non négligeable pour pouvoir les adapter à son problème. Bien qu'un effort ait été fait ces dernières années (notamment avec la plate-forme EASEA [CSLL01]), il manque un véritable outil générique qui pourrait traiter n'importe quel type de problème d'optimisation combinatoire sans effort particulier de la part de l'utilisateur. Nous pensons qu'une telle plate-forme mêlant les différentes métaheuristiques, qui proposerait de tester de nombreux types de problèmes en utilisant un paramétrage semi-automatique, pourrait permettre une meilleure reconnaissance des métaheuristiques dans le monde industriel. Les résolveur de contraintes commerciaux intègrent parfois des métaheuristiques mais, comme nous l'avons entrevu au chapitre 5, leur implantation n'est généralement pas très efficace et a donc plutôt un effet de contre publicité.

Nous voudrions évoquer un pressentiment concernant la communauté des chercheurs en algorithmique évolutionnaire. Nous voyons se dessiner deux courants qui prennent des

directions opposées. Le premier concerne la recherche de méthodes efficaces sur des problèmes d'optimisation combinatoire, utilisant des hybridations de plus en plus complexes. Ces méthodes s'éloignent de plus en plus du modèle évolutionnaire classique, pour ne garder que le principe d'exploitation de l'espace de recherche grâce à l'utilisation d'une population de solutions et de croisements spécifiques. Des applications industrielles semblent d'ailleurs voir le jour, tel que l'allocation de fréquences, la reconnaissance de caractères (La Poste) ou l'optimisation d'aile d'avion (Dassault). Le deuxième s'intéresse à la dynamique des AE en tant que modèle d'adaptation en se désintéressant plus ou moins de la partie optimisation combinatoire. L'idée est en général de s'inspirer encore plus du modèle naturel, et d'étudier la dynamique d'adaptation du modèle évolutionnaire sur des problèmes fluctuants. Ces deux voies sont toutes deux très intéressantes, mais une séparation, si elle se confirme, risque de conduire à un désintéressement des deux sous-communautés l'une par rapport à l'autre, et donc à un ralentissement des avancées dans le domaine évolutionnaire.

Nous ne voulons cependant pas terminer sur cette note pessimiste, mais plutôt sur les perspectives qui nous paraissent les plus prometteuses. En ce qui concerne les AE en général, nous pensons que l'étude des réseaux de neutralité que nous avons évoquée plus haut, mérite une attention particulière quel que soit le domaine d'application. En ce qui concerne les CSP en particulier, et les problèmes d'optimisation combinatoire, nous pensons que l'élaboration d'une plate-forme générique proposant différentes métaheuristiques, ainsi que la plupart des méthodes hybrides devrait être très profitable à l'ensemble de la communauté. De telles entreprises ont d'ailleurs commencé à voir le jour avec *SCOOP*[10] au SINTEF et *iOpt* [DVL+02] à British Telecom.

[10] http ://www.oslo.sintef.no/SCOOP/

# Table des figures

# Bibliographie


[Ang95] Peter J. Angeline. Adaptive and self-adaptive evolutionary computations. In M. Palaniswami, Y. Attikiouzel, R. Marks, D. Fogel, and T. Fukuda, editors, *Computational Intelligence: A Dynamic Systems Perspective*, pages 152–163. IEEE Press, Piscataway, NJ, 1995.

[AS94] J.M. Alliot and T. Schiex. *Intelligence Artificielle & Informatique Théorique*. Cépadués Édition, 1994.

[AV94] David Aldous and Umesh V. Vazirani. "go with the winners" algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 492–501, 1994.

[Bäc92] Thomas Bäck. The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, pages 85–94, Amsterdam, 1992. Elsevier.

[BBG+97] B. Beauquier, J.-C. Bermond, L. Gargano, S. Perennes, P. Hell, and U. Vaccaro. Graph problems arising from wavelength-routing in all- optical networks. In *Proc. of the 2nd Workshop on Optics and Computer Science*, 1997.

[BBM93] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

[BC95] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In A. Prieditis and S. Russel, editors, *The Int. Conf. on Machine Learning 1995*, pages 38–46, San Mateo, CA, 1995. Morgan Kaufmann Publishers.

[Ber83] C. Berge. *Graphes*. Bordas, Gauthier-Villars edition, 1983.

[BHS89] D.E. Brown, C.L. Huntley, and A.R. Spillane. A parallel genetic heuristic for the quadratic assignment problem. In *Proceedings of the third international conference on Genetic algorithms*, pages 406–415. Morgan Kaufmann Publishers Inc., 1989.

[Bli98] C. Bliek. Generalizing partial order and dynamic backtracking. In *Proceedings of AAAI*, 1998.

[BR96] C. Bessière and J.C. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proc. of CP'96*, volume 1113 of *LNCS*, pages 61–75, 1996.

[BS96] Thomas Bäck and Martin Schütz. Intelligent mutation rate control in canonical genetic algorithms. In *International Syposium on Methodologies for Intelligent Systems*, pages 158–167, 1996.

[CC99] Manuel Clergue and Philippe Collard. Genetic heuristic for search space exploration. In *IJCAI*, pages 1218–1226, 1999.

[CCR96] Emma Collingwood, David Corne, and Peter Ross. Useful diversity via multiploidy. In *Proc. of the IEEE International Conf. on Evolutionary Computing*, page ?, Piscataway, NY, 1996. IEEE Press.

[CDG99] D. Corne, M. Dorigo, and F. Glover, editors. *New Ideas in Optimization*. McGraw-Hill, 1999.

[CEM00] B. G. W. Craenen, A. E. Eiben, and E. Marchiori. Solving constraint satisfaction problems with heursitic-based evolutionary algorithms. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1571–1577, Piscataway, NJ, 2000. IEEE Service Center.

[Cer94] R. Cerf. *Une théorie assymptotique des algorithmes génétiques*. Thèse de doctorat, Université de Montpellier II, 1994.

[Coe99] Carlos A. Coello Coello. A survey of constraint handling techniques used with evolutionary algorithms. Technical Report Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada, 1999.

[Con90] D. T. Connolly. An improved annealing scheme for the qap. *European Journal of Operational Research*, (46):93–100, 1990.

[CSLL01] Pierre Collet, Marc Schoenauer, Evelyne Lutton, and Jean Louchet. EASEA: un langage de spécification pour les algorithmes évolutionnaires. Rapport de Recherche RR-4218, INRIA, Juin 2001.

[CST96] D. Coit, A. Smith, and D. Tate. Adaptive penalty methods for genetic optimization of constrained combinatorial problems, 1996.

[DCG99] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[Deb98] R. Debruyne. *Consistances locales pour les problèmes de satisfaction de contraintes de grande taille.* Thèse de doctorat, Université de Montpellier II, 1998.

[Dec89] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning and cutset decomposition. *Artificial Intelligence*, 1989.

[Dec92] R. Dechter. From local to global consistency. *Artificial Intelligence*, 55:87–107, 1992.

[DF98] R. Dechter and D. Frost. Backtracking algorithms for constraint satisfaction problems; a survey, 1998.

[DG89] K. Deb and D.E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 42–50. Morgan Kaufmann Publishers Inc., 1989.

[DH98] Raphaël Dorne and Jim-Kao Hao. A new genetic local search algorithm for graph coloring. In Agoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature – PPSN V*, pages 745–754. Springer, 1998.

[DI96] T. Dimitriou and R. Impagliazzo. Towards an analysis of local optimization algorithms. In *ACM Symposium on Theory of Computing*, pages 304–313, 1996.

[Did02] F. Didierjean. AgCSP, une bibliothèque de classes C++ pour le développement d'opérateurs génétiques pour CSP, 2002. Thèse en preparation.

[DMC96] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996.

[DP89] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artificial Intelligence*, 38:353–366, 1989.

[DVL+02] R. Dorne, C. Voudouris, D. Lesaint, A. Liret, and C. Ladde. La plate-forme *iOpt* dédiée au développement des systèmes d'optimisation basés sur les méthodes heuristiques. In *Journées Nationales sur la Résolution Pratique des Problèmes NP-Complets*, pages 99–114, Nice, 2002.

[ECS89] Larry J. Eshelman, Richard A. Caruana, and J. David Schaffer. Biases in the crossover landscape. In *Proceedings of the third international conference on Genetic algorithms*, pages 10–19. Morgan Kaufmann Publishers Inc., 1989.

[EHM99] A. E. Eiben, R. Hinterding, and Z. Michalewicz. Parameter control in Evolutionary Algorithms. *IEEE Transaction on Evolutionnary Computation*, 3(2):124, July 1999.

[Eib01] A. E. Eiben. Evolutionary algorithms and constraint satisfaction: Definitions, survey, methodology, and research directions. In Leila Kallel, Bart Naudts, and Alex Rogers, editors, *Theoretical Aspects of Evolutionary Computing*, pages 13–30. Springer, 2001.

[ER96] A. E. Eiben and Zsofia Ruttkay. Self-adaptivity for constraint satisfaction: Learning penalty functions. In *International Conference on Evolutionary Computation*, pages 258–261, 1996.

[ERR94] A.E. Eiben, P-E Raue, and Zs. Ruttkay. Solving constraint satisfaction problems using genetic algorithms. In *Proc. of the First IEEE Conf on Evolutionary Computation*, pages 542–547, Orlando, 1994.

[ES91] L.J. Eshelman and J.D. Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In *Fourth International Conference on Genetic Algorithms (ICGA)*, pages 115–122, San Mateo, 1991. Morgan Kaufmann,.

[Ev97] A. E. Eiben and J. K. van der Hauw. Adaptive penalties for evolutionary graph coloring. In J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, editors, *Artificial Evolution – Third European Conference*, pages 95–106. Springer, 1997. LNCS 1363.

[EvKK95] A. E. Eiben, C. H. M van Kemenade, and J. N. Kok. Orgy in the computer: Multi-parent reproduction in genetic algorithms. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *Proceedings of the Third European Conference on Artificial Life : Advances in Artificial Life*, volume 929 of *LNAI*, pages 934–945. Springer Verlag, June 1995.

[Evv98] Agoston E. Eiben, J. K. van der Hauw, and J. I. van Hemert. Graph coloring with adaptive evolutionary algorithms. *Journal of Heuristics*, 4(1):25–46, 1998.

[FD95] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI'95*, pages 572–578, Montreal, Canada, 1995.

[Fog89] Terence C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *Proceedings of the Third International Conference on Genetic Algorithms,*, pages 104–109, 1989.

[FW92] E. Freuder and R. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[Gal99] P. Galinier. *Recherche locale et hybridation pour le problème de satisfaction de contraintes et la coloration de graphes.* Thèse de doctorat, Université de Montpellier II, Janvier 1999.

[Gee92] P. A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Tenth European Conference on Artificial Intelligence (ECAI92)*, pages 31–35, 1992.

[Gin93] M. Ginsberg. Dynamic backtracking. *Journal of Aertificial Intelligence Research*, 1:25–46, 1993.

[GL97] F. Glover and M. Laguna. *Tabu Search.* University of Colorado at Boulder Kluwer Academic Publishers, Boston, July 1997. 408 pp.

[Glo86] F. Glover. Future paths for integer programming and links to artificial intelligence. In *Computers and Operation Research*, volume 13, pages 533–549, 1986.

[GM00] D. Greenhalgh and S. Marshall. Convergence criteria for genetic algorithms. *SIAM Journal of Computing*, 30(1):269–282, 2000.

[GMP+96] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, Barbara M. Smith, and Toby Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In E. C. Freuder, editor, *Principles and Practice of Constraint Programming - CP96*, volume 1118, pages 179–193. Springer-Verlag, 1996.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading. Addison-Wesley, Massachusetts, 1989.

[Goo95] E.D. Goodman. GALOPPS: The "genetic algorithm optimized for portability and parallelism" system, 1995. Michigan State University, http://isl.msu.edu/GA.

[GS00] Matt Glickman and Katia Sycara. Reasons for premature convergence of self-adapting mutation rates. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 62–69, Piscataway, NJ, 2000. IEEE Service Center.

[GV89] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, 2nd edition, 1989.

[HB91] Frank Hoffmeister and Thomas Back. Genetic algorithms and evolution strategies: Similarities and differences. In Hans-Paul Schwefel and Reinhard Manner, editors, *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 455–470. Springer, 1991.

[HE80] R. Haralick and G. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[HG95] W. Harvey and M. Ginsberg. Limited discrepancy search. In Chris Mellish, editor, *IJCAI'95: Proc. of International Joint Conference on Artificial Intelligence*, pages 607–613, Montreal, 1995.

[HGM99] J.K. Hao, P. Galinier, and M.Habib. Métaheuristiques pour l'optimisation combinatoire et l'affectation sous contraintes. *Revue d'Intelligence Artificielle*, 13(2):283–324, 1999.

[HM90] Jurgen Hesser and Reinhard Manner. Towards an optimal mutation probability for genetic algorithms. In *Parallel Problem Solving from Nature*, pages 23–32, 1990.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems, An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* The University of Michigan, 1st edition, 1975.

[ILO98] ILOG. Ilog solver reference manual. Technical report, ILOG, 1998.

[Jeg90] P. Jegou. Cyclic clustering: a compromise between tree-clustering and the cyclecutset method for improving search efficiency. In *European Conference on Artificial Intelligence (ECAI-90)*, pages 369–371, 1990.

[JT93] D.S. Johnson and M.A Trick. Cliques, coloring and satifiability. In *Discr. Math. and Theoretical Comput. Sci., 2nd DIMACS Implementation Challenge*, volume 26 of *DIMACS Series*. 1993.

[KGV83] S. Kirpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[KKD96] S. Koakutsu, M. Kang, and W.W.-M. Dai. Genetic simulated annealing and application to non-slicing floorplan design. In *5th ACM/SIGDA Physical Design Workshop*, pages 134–141, Virginia, USA, 1996.

[Kor96] R.E. Korf. Improved limited discrepancy search. In *AAAI/IAAI, Vol. 1*, pages 286–291, 1996.

[Koz92] J.R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, Massachusetts, 1992. ISBN 0-262-11170-5.

[Lan98] M. Land. *Evolutionary Algorithms with Local Search for Combinatorial Optimization.* PhD thesis, Computer Science and Engr. Dept. - University of California, San Diego, 1998.

[LGX97] Y. Leung, Y. Gao, and Z. B. Xu. Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis. *IEEE Transactions on Neural Networks*, 8(5):1165–1176, September 1997.

[LKH93] F. T. Lin, C. Y. Kao, and C. C. Hsu. Applying the genetic approach to simulated annealing in solving some NP-hard problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(6):1752–1767, - 1993.

[LMS99] J. Larrosa, P. Meseguer, and T. Schiex. Maintaining reversible DAC for MAX-CSP. *Artificial Intelligence*, 107(1):149–163, 1999.

[Mes97] P. Meseguer. Interleaved Depth-First Search. In *15th International Joint Conference on Artificial Intelligence*, pages 1382–1387, Nagoya, Japan, 1997.

[MFH92] M. Mitchell, S. Forrest, and J.H. Holland. The royal road function for genetic algorithms: Fitness landscapes and ga performance. In F. J. Varela and P. Bourgine, editors, *First European Conference on Artificial Life*, pages 245–254, Cambridge, MA, 1992. MIT Press.

[Müh92] H. Mühlenbein. How genetic algorithms really work: I. mutation and hill-climbing. In R. Manner and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pages 15–25, Amsterdam. Elsevier., 1992.

[Mic94] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1994.

[MJ91] Zbigniew Michalewicz and Cezary Z. Janikow. Handling constraints in genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Fourth International Conference on Genetic Algorithms*, pages 151–157, San Mateo, California, 1991. Morgan Kaufmann.

[MJPL92] S. Minton, M. Johnston, A. Philips, and P. Laird. Minimizing conflict: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[MN89] P. Moscato and M. Norman. A competitive-cooperative approach to complex combinatorial search. Technical Report 790, Caltech Concurrent Computation Program, 1989. (Expanded version published in the Proceedings of the 20th JAIIO Conference,pp 3.15–3.29, Buenos Aires, Argentina (August 1991)).

[MPSW98] E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. Random constraint satisfaction : Theory meet practice. In *Principles and Practice of Constraint Programming CP'98*, number 1520 in LNCS, pages 325–339. Springer, 1998.

[NV92] Allen E. Nix and Michael D. Vose. Modelling genetic algorithms with markov chains. *Annals of Mathematics and Articial Intelligence*, 5:79–88, 1992.

[Prc98] N. Prcovic. *Recherche arborescente parallèle et séquentielles pour les problèmes de satisfaction de contraintes.* Thèse de doctorat, École Nationale des Ponts et Chaussée, 1998.

[Pro93] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[Rad91] Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–205, 1991.

[Rec73] I. Rechenberg. Evolutionsstrategie werkstatt bionik und evolutionstechnik band 1, 1973.

[RFR99] Olivier Roux, Cyril Fonlupt, and Denis Robilliard. Co-operative improvement for a combinatorial optimization algorithm. In *Artificial Evolution*, pages 231–241, 1999.

[RG93] Connie Loggia Ramsey and John J. Grefenstette. Case-based initialization of genetic algorithms. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 84–91, San Mateo, CA, 1993. Morgan Kaufmann.

[Rif97] M.C. Riff Rojas. *Résolution de problèmes de satisfaction de contraintes avec des algorithmes évolutionnistes.* Thèse de doctorat, École Nationale des Ponts et Chaussées, 1997.

[Rif98] M.-C. Riff Rojas. A network-based adaptive evolutionary algorithm for constraint satisfaction problems. In S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, *Meta-heuristics : Advances and Trends in Local Search Paradigms for Optimization*, pages 269–283. Kluwer Academic Publishers, 1998.

[Rud94] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1):96–101, 1994.

[RW91] Yuri Rabinovich and Avi Wigderson. An analysis of a simple genetic algorithm. In Rick Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 215–221, San Mateo, CA, 1991. Morgan Kaufman.

[SBHW95] Barbara Smith, Sally Brailsford, Peter Hubbard, and H. Paul Williams. The Progressive Party Problem: Integer Linear Programming and Constraint Programming Compared. In *CP95: Proceedings 1st International Conference on Principles and Practice of Constraint Programming)*, Marseilles, 1995.

[SD91] William M. Spears and Kenneth A. De Jong. An analysis of multi-point crossover. In G. J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 301–315, San Mateo, CA, 1991. Morgan Kaufmann.

[SF94] Daniel Sabin and Eugene C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In Alan Borning, editor, *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, PPCP'94*, volume 874, pages 10–20, Rosario, Orcas Island, Washington, USA, 1994.

[SF96] Jim E. Smith and Terence C. Fogarty. Adaptively parameterised evolutionary systems: Self adaptive recombination and mutation in a genetic algorithm. In H. Voigt, W. Ebeling, and I. Rechenberg, editors, *Parallel Problem Solving from Nature IV*, pages 441–450. Springer, 1996.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. *CP'98, Lecture Notes in Computer Science*, 1520:417–431, 1998.

[SK90] H. Hirata S. Koakutsu, Y. Sugai. Block placement by improved simulated annealing based on genetic algorithm. In *Institute of Electronics,Information and Communication Engineers of Japan*, volume J73-A, pages 87–94, 1990.

[SLM92] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.

[SMM+91] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley. A comparison of genetic sequencing operators. In Rick Belew and Lashon Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 69–76, San Mateo, CA, 1991. Morgan Kaufman.

[Spe98] W. Spears. *The Role of Mutation and Recombination in Evolutionary Algorithms*. PhD thesis, George Mason University, Virginia, 1998.

[SS96] Michèle Sebag and Marc Schoenauer. Contrôle d'un algorithme génétique. *Intelligence artificielle*, 1996.

[SSR97] M. Sebag, M. Schoenauer, and C. Ravise. Revisiting the memory of evolution. In Thomas Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, San Francisco, CA, 1997. Morgan Kaufmann.

[SV94] T. Schiex and G. Verfaillie. Stubbornness: a possible enhancement for backjumping and no-good recording, 1994.

[SX93] Marc Schoenauer and Spyros Xanthakis. Constrained GA optimization. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 573–580, San Mateo, CA, 1993. Morgan Kaufmann.

[Sys89] Gilbert Syswerda. Uniform crossover in genetic algorithms. In J. D. Shaeffer, editor, *Proceeding of Third International Conference on Genetic Algorithms and Their Applications*, pages 2–9, 1989.

[TRK00] R. Thomsen, P. Rickers, and T. Krink. A religion-based spatial model for evolutionary algorithms. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, Hans-Paul Schwefel, editor, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, 2000. Springer Verlag.

[Tsa90] E. Tsang. Applying genetic algorithms to constraint satisfaction optimization problems. In *European conference on artificial intelligence*, pages 649–654. Pitman Publishing, 1990.

[vH02] J.I. van Hemert. Comparing classical methods for solving binary constraint satisfaction problems with state of the art evolutionary computation. In S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, editors, *Applications of Evolutionary Computing*, number 2279 in LNCS, pages 82–91. Springer-Verlag, 2002. ISBN:3-540-43432-1.

[VMB99] Gerard Verfaillie, David Martinez, and Christian Bessiere. A generic customizable framework for inverse local consistency. In *AAAI/IAAI*, pages 169–174, 1999.

[Wal96] M. Wall. GALib: A C++ library of genetic algorithm components, 1996. Massachusetts Institute of Technology, Mechanical Engineering Department,`http://lancet.mit.edu/ga/`.

[Wal97] Toby Walsh. Depth-bounded discrepancy search. In *IJCAI*, pages 1388–1395, 1997.

[WP00] R. A. Watson and J. B. Pollack. Symbiotic combination as an alternative to sexual recombination in genetic algorithms. In Hans-Paul Schwefel Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan Julian Merelo, editor, *Parallel Problem Solving from Nature - PPSN VI 6th International Conference*, Paris, France, 16-20 2000. Springer Verlag.

[WRP00] R.A. Watson, T. Reil, and J.B. Pollack. Mutualism, parasitism, and evolutionary adaptation. In *Artificial Life VII*, 2000.

## Résumé

Cette thèse traite de l'utilisation des algorithmes évolutionnaires (AE) pour résoudre des problèmes de satisfaction de contrainte (CSP) en domaines finis sans spécialisation ni hybridation particulière. Après avoir présenté les CSP et les méthodes couramment utilisées pour les résoudre (chapitres 1 et 2), nous présentons le paradigme évolutionnaire et ses applications (chapitres 3 et 4). Ensuite, nous proposons une comparaison entre les méthodes de recherche arborescente et les métaheuristiques sur des coloriages de graphe sur-contraints, dans un contexte de réglage des paramètres minimal (chapitre 5). Nous étudions le paysage de recherche pour comprendre les raisons des différences d'efficacité des méthodes. Enfin, nous proposons de nouveaux opérateurs génétiques (croisement, mutation, diversification) dont le paramétrage est moins fastidieux qu'avec les opérateurs classiques (chapitre 6). Nous concluons sur l'intérêt d'exploration des réseaux de neutralité.

**Mots clés :** Algorithmes Génétiques, Algorithmes Evolutionnaire, Problèmes de satisfaction de contraintes, Coloriage de Graphes.

---

## Abstract

This thesis deals with the use of evolutionary algorithms (EA) to solve constraint satisfaction problems (CSP) in finite domains, without any particular specialization nor hybridization. Having presented the CSP and general methods used to solve them (chapters 1 and 2), we present the evolutionary paradigm and its applications (chapters 3 and 4). Then, we propose a comparison between the tree search methods and metaheuristics on over-constrained graph coloring, in a context of minimal regulation of the parameters (chapter 5). We study the research landscape for understanding the reasons of the differences in efficiency of methods. Finally, we propose new genetic operators (crossover, mutation, diversification), the parameter setting of which is less difficult than with the classical operators (chapter 6). We conclude on the interest of investigating the neutrality networks.

**Keywords :** Genetic Algorithms, Evolutionary Algorithms, Constraint Satisfaction Problems, Graph Coloring.