# Chapitre 3

# Fonction d'Évaluation

Dans le chapitre précédent, nous avons présenté les composantes d'un algorithme évolutionniste. Mais la question suivante demeure: Qu'est-ce que peut apporter de plus un algorithme évolutionniste, par rapport aux méthodes stochastiques traditionnelles pour la résolution des CSP? En premier lieu, un algorithme évolutionniste permet une recherche multiple (population de pré-solutions), on peut donc chercher en parallèle dans différents espaces de recherche. Au contraire, les méthodes stochastiques traditionnelles ne cherchent qu'avec une seule pré-solution. Une différence fondamentale, à notre avis, des algorithmes évolutionnistes par rapport aux méthodes stochastiques classiques (telles que GSAT ou *min-conflits* et leurs variantes) réside dans la coopération possible entre les pré-solutions de la population pour trouver une solution au problème. Une pré-solution n'évolue plus individuellement mais peut se combiner avec d'autres. Dans un algorithme évolutionniste qui fait évoluer de manière aléatoire les différents individus de la population (exploration), on ajoute de l'exploitation qui cherche à profiter de la connaissance contenue dans les différentes pré-solutions, en les combinant. Puisque notre but est de concevoir un algorithme évolutionniste pour résoudre des problèmes de satisfaction de contraintes, nous allons donc dans ce chapitre aborder la définition d'une fonction d'évaluation, en essayant de prendre en compte quelques caractéristiques des CSP. Pour profiter du mécanisme que fournissent les algorithmes évolutionnistes, il nous faut aussi de bons opérateurs qui soient capables de mettre en valeur la connaissance des pré-solutions par une bonne combinaison. Ce

sujet sera traité en détail dans les chapitres suivants.

Nous commencerons ce chapitre avec un exemple montrant les limitations de la fonction d'évaluation standard (voir définition 2.5.1), afin de motiver l'introduction de notre nouvelle fonction, après avoir défini quelques concepts. Ensuite, nous incorporerons cette fonction, comme heuristique dans la méthode de *min-conflits* et puis la comparerons à l'aide de quelques résultats. Enfin, nous evaluerons le comportement de cette fonction dans un algorithme génétique qui résout le problème de coloriage de graphe avec 3 couleurs. Finalement, nous proposerons une extension de cette fonction pour les problèmes n-aires.

## 3.1 Motivation: un exemple

Nous rappelons que les méthodes stochastiques telles que l'escalade, le recuit simulé, la recherche tabou (voir Chapitre 1), consistent à réparer une instanciation pour arriver à trouver une solution d'un CSP. Pour ce faire, elles utilisent quelques heuristiques pour guider le "parcours stochastique" dans l'espace de recherche. La fonction d'évaluation est utilisée pour mesurer l'amélioration de l'instanciation modifiée et pour conduire la recherche vers des instanciations plus prometteuses en termes de satisfaction des contraintes. Souvent, cette fonction est la fonction d'évaluation standard $Z_{std}(\mathbf{I})$ de la définition 2.5.1, qui correspond au nombre de contraintes violées. [Fre95]. Par exemple, considérons le CSP suivant dont le graphe et la matrice de contraintes sont illustrés par la figure 3.1:

*variables* $V=X_1, X_2, X_3, X_4$

*domaines*

- $D_1 = (10, 20, 35, 50)$ avec $m_1 = 4$
- $D_2 = (30, 25, 20, 10)$ avec $m_2 = 4$
- $D_3 = (25, 30, 20)$ avec $m_3 = 3$
- $D_4 = (40, 50, 60)$ avec $m_4 = 3$

- *L'ensemble* $\zeta$ *avec 4 contraintes*
  - $C_1 : X_1 \geq X_2$
  - $C_2 : X_3 \geq X_2$
  - $C_3 : X_4 \geq X_3$
  - $C_4 : X_4 \geq X_2$

| | $X_1$ | $X_2$ | $X_3$ | $X_4$ |
|---|---|---|---|---|
| $C_1$ | 1 | 1 | 0 | 0 |
| $C_2$ | 0 | 1 | 1 | 0 |
| $C_3$ | 0 | 0 | 1 | 1 |
| $C_4$ | 0 | 1 | 0 | 1 |

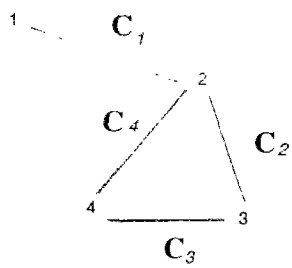



FIG. 3.1 – *Exemple: Graphe de Contraintes et Matrice de Contraintes*

L'ordre des variables dans l'instanciation est le suivant: $X_1$,$X_2$,$X_3$,$X_4$. Supposons que nous puissions choisir parmi les deux instanciations suivantes pour notre problème:

$\mathbf{I}_1$= | 35 | 30 | 25 | 40 | qui ne satisfait pas la contrainte $C_2$

$\mathbf{I}_2$= | 10 | 25 | 30 | 40 | qui ne satisfait pas la contrainte $C_1$

$Z_{std}(\mathbf{I})$ sera pour les deux égal à 1, car chacune ne viole qu'une seule contrainte. Regardons maintenant le graphe de contraintes: la contrainte $C_2$ est liée aux trois autres contraintes, ce qui veut dire que si nous changeons quelques valeurs des variables pertinentes pour $C_2$, cela pourrait éventuellement se traduire par une violation d'autres contraintes satisfaites à présent. En revanche, $C_1$ est liée à $C_2$ et $C_4$. On peut donc penser qu'une réparation de cette contrainte aurait moins de conséquences vis à vis des autres contraintes qui sont déjà satisfaites dans le réseau. C'est cette idée que nous allons incorporer dans la définition de la nouvelle fonction d'évaluation.

## 3.2 Fonction d'évaluation pour CSP binaire

A part $Z_{std}(\mathbf{I})$, d'autres fonctions d'évaluation, plus spécifiques au type de problème à traiter, ont été proposées dans la littérature pour les CSP binaires. Par exemple, pour le problème de N-reines, Eiben et al. [ERR94] ont utilisé comme fonction d'évaluation le nombre de contraintes non satisfaites sur la diagonale. Pour un problème de dessin mécanique, Thorton a proposé de minimiser $Y = \sum d_i^2$ où $d_i$ est l'erreur normalisée pour la contrainte i, [Tho94]. Notre objectif ici est de définir une fonction indépendante du problème à résoudre. D'autre part, le fait de prendre en compte le seul nombre de contraintes violées signifie:

- D'un coté, qu'on considère que toutes les contraintes sont également facilement (difficilement) réparables dans la recherche d'une solution. Néanmoins, dans la plupart des problèmes de satisfaction de contraintes, des contraintes sont plus dures que d'autres à satisfaire.

  D'un autre coté qu'il n'y a pas de préférence parmi les contraintes. Avoir une préférence permet de relâcher les problèmes sous contraints pour obtenir une "bonne solution", quand on n'a pas le temps d'obtenir une solution satisfaisant toutes les contraintes. C'est notamment le cas des PCSP (Partial Constraint Satisfaction Problems) [FW92].

On se place dans le premier cas, c'est-à-dire, nous voulons obtenir une solution du CSP (et non une solution à un problème relâché). Nous souhaitons définir une fonction d'évaluation qui représente une sorte de "distance" entre une instanciation et une solution, en termes du nombre de changements effectués par un algorithme de recherche locale. Une fonction qui considère donc la difficulté d'obtenir une solution. Cela nous a conduite à la définition de la nouvelle fonction qui prend en compte la structure du réseau de contraintes, plus précisément le degré de liaison entre les différentes variables.

Pour cela, nous commençons avec un nouveau concept: nous voulons montrer que dans le cas où une contrainte est violée, les variables qui sont impliquées dans cette violation ne sont pas seulement les deux variables connectées par cette contrainte.

En effet, en plus des variables pertinentes, la réparation d'une contrainte violée peut affecter d'autres variables qui leur sont connectées à travers d'autres contraintes.

**Définition 3.2.1** *(Ensemble de Variables Impliquées)*
*Étant donné un CSP binaire* $P = (V, D, \zeta)$, *et une instanciation* $\mathbf{I}$, *pour chaque contrainte* $C_\alpha(\alpha = 1, \ldots, \eta)$ *un ensemble de variables impliquées* $\mathbf{E}_{\alpha(\mathbf{I})} \subseteq V$ *est défini par:*

- $\mathbf{E}_{\alpha(\mathbf{I})} = \phi$ *ssi* $C_\alpha$ *est satisfaite*

  $X_i \in \mathbf{E}_{\alpha(\mathbf{I})}$ *si* $X_i \rhd C_\alpha$ *et* $C_\alpha$ *n'est pas satisfaite sous* $\mathbf{I}$

- $X_j \in \mathbf{E}_{\alpha(\mathbf{I})}$ *si* $\exists X_i \rhd C_\alpha$ *et* $\exists C_\beta \neq C_\alpha$ *tel que* $X_i$ *et* $X_j \rhd C_\beta$, *et* $C_\alpha$ *n'est pas satisfaite sous* $\mathbf{I}$

Cette définition montre qu'en essayant de réparer une contrainte $C_\alpha$ sous une certaine instanciation $\mathbf{I}$, le changement de valeur d'une variable sera perçu par d'autres contraintes dans le réseau. C'est justement cet effet que nous voulons incorporer dans la fonction d'évaluation. Pour une instanciation $\mathbf{I}$ donnée, nous allons quantifier l'erreur attribuée à une contrainte $C_\alpha$ par:

**Définition 3.2.2** *(Évaluation de l'erreur)*
*Étant donné un CSP binaire* $P = (V, D, \zeta)$ *avec une matrice de contraintes* $\mathbf{R}$ *et une instanciation* $\mathbf{I}$, *la fonction de l'évaluation de l'erreur* $\mathbf{e}(C_\alpha, \mathbf{I})$ *pour une contrainte* $C_\alpha$ *est définie par:*

$$\mathbf{e}(C_\alpha, \mathbf{I}) = \textit{Nombre de variables dans } \mathbf{E}_{\alpha(\mathbf{I})}$$

Si une contrainte binaire non-satisfaite $C_\alpha$ a $X_k$ et $X_l$ comme variables pertinentes (elle en a juste deux, exactement ces deux), alors nous pouvons exprimer $\mathbf{e}(C_\alpha, \mathbf{I})$ de la manière suivante:

$\mathbf{e}(C_\alpha, \mathbf{I})$ = (Nombre de variables pertinentes pour $C_\alpha$) + (Effet Propagé par $X_k$ et $X_l$)

où l'effet propagé par $X_k$ et $X_l$ dans un réseau de contraintes binaires est défini comme le nombre de contraintes $C_\beta$, $\beta = 1, \ldots, \eta, \beta \neq \alpha$ qui ont comme variables pertinentes $X_k$ ou $X_l$

Donc, en termes de la matrice de contraintes cela se traduit par:

$$\mathbf{e}(C_\alpha, \mathbf{I}) = \left( \sum_{j=1}^{n} \mathbf{R}[\alpha, j] \right) + \left( \sum_{\beta \neq \alpha, \beta=1}^{\eta} \mathbf{R}[\beta, k] + \sum_{\beta \neq \alpha, \beta=1}^{\eta} \mathbf{R}[\beta, l] \right) \tag{3.1}$$

**Remarque 3.2.1** *Si $C_\gamma$ est satisfaite alors $\mathbf{e}(C_\gamma, \mathbf{I}) = 0$*

**Remarque 3.2.2** *La complexité du calcul de $\mathbf{e}(C_\alpha, \mathbf{I})$ pour chaque contrainte $C_\alpha$ est égal à $O(\eta)$.*

Chaque contrainte dans notre fonction contribue différemment à la valeur de la fonction d'évaluation, on peut définir donc la contribution d'une contrainte comme:

**Définition 3.2.3** *(Contribution de $C_\alpha$)*
*Étant donné un CSP binaire $P = (V, D, \zeta)$, on dira que la Contribution de $C_\alpha$ à la fonction d'évaluation $\mathbf{c}(C_\alpha)$ sera:*

$$\mathbf{c}(C_\alpha) = \mathbf{e}(C_\alpha, I_j), \text{ quand } C_\alpha \text{ est violée sous } I_j.$$

Cette fonction quantifie la contribution de $C_\alpha$ à la fonction d'évaluation quand elle n'est pas satisfaite. On remarque que cette contribution ne dépend pas des valeurs effectives de l'instanciation, mais seulement du fait que la contrainte est violée.

Finalement, la valeur de la fonction d'évaluation pour une instanciation $\mathbf{I}$ donnée sera la somme des évaluations des erreurs (équation 3.1) sur toutes les contraintes dans le CSP.

**Définition 3.2.4** *(Fonction d'Évaluation pour CSP binaire)*
*Étant donné un CSP binaire avec une matrice de contraintes $\mathbf{R}$, une instanciation $\mathbf{I}$ et la fonction d'Évaluation de l'erreur $\mathbf{e}(C_\alpha, \mathbf{I})$ pour chaque contrainte $C_\alpha$, $(\alpha = 1, \ldots, \eta)$, la Fonction d'Évaluation $\mathbf{Z}(\mathbf{I})$ est définie par:*

$$\mathbf{Z}(\mathbf{I}) = \sum_{\alpha=1}^{\eta} \mathbf{e}(C_\alpha, \mathbf{I}) \tag{3.2}$$

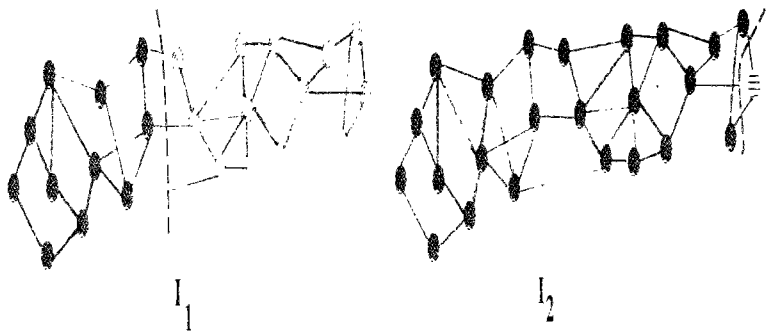



FIG. 3.2 – *Deux instanciations pour le même graphe*

Le but de la recherche d'une solution est de minimiser cette fonction $Z(\mathbf{I})$, laquelle sera égale à zéro quand toutes les contraintes seront satisfaites.

$Z(\mathbf{I})$ peut être interprétée comme une façon de quantifier notre préférence pour les instanciations qui satisfont plus de contraintes et de liaisons. Pour illustrer cela, regardons la figure 3.2. Il s'agit d'un problème de coloriage de graphe. Nous avons deux instanciations pour le même graphe: $I_1$ et $I_2$. Pour chaque instanciation, l'ensemble de nœuds à gauche de la ligne en pointillés représente un coloriage consistant, et l'ensemble à droite un autre coloriage consistant. Mais les deux coloriages sont inconsistants entre eux. À première vue, les deux instanciations peuvent paraître comme aussi bonnes, car dans les deux cas nous n'avons que trois contraintes violées. Par contre, en utilisant la fonction $Z(\mathbf{I})$ nous préférons la seconde instanciation, parce qu'elle a moins de variables impliquées (voir définition 3.2.1).

### Fonction d'Évaluation et CSP aléatoires

Considérons les CSP générés aléatoirement, [Smi95], [Pro94] qui sont caractérisés par 4 paramètres: n le nombre de variables, m le nombre de valeurs de chaque domaine, égal pour toutes les variables, $p_1$ la probabilité qu'il existe une contrainte entre deux variables et $p_2$ la probabilité conditionnelle qu'une paire de valeurs soit inconsistante pour une paire de variables, étant donné qu'il y a une contrainte entre les variables. Pour ce type de CSP aléatoire, l'espérance du nombre de contraintes violées pour une instanciation quelconque $\mathbf{I}$ sera:

$$E[Z_{std}(\mathbf{I})] = \left(p_1 \frac{n(n-1)}{2}\right) p_2 \tag{3.3}$$

Le nombre maximum de contraintes pour un problème avec n variables est $\frac{n(n-1)}{2}$. Le CSP aléatoire aura donc $p_1 \frac{n(n-1)}{2}$ contraintes ou arêtes. C'est-à-dire que le nombre espéré de contraintes violées sera égal au nombre de contraintes sur le réseau, multiplié par la probabilité d'incompatibilité de valeurs des domaines $p_2$ pour chaque contrainte.

La valeur de notre fonction sera:

$$E[Z(\mathbf{I})] = 2Z_{std}(\mathbf{I})p_1(n-1) \tag{3.4}$$

Pour expliquer l'équation 3.4, voyons la figure 3.3. Chaque contrainte a deux variables pertinentes. Chaque variable du problème a une arité (voir définition 1.1.3) de $p_1(n-1)$. Alors, pour une contrainte $C_a$ violée, nous allons compter ses deux variables pertinentes plus l'effet de propagation à partir d'elles, donc $2 + (2p_1(n-1) - 2)$. Maintenant nous devons faire la somme sur toutes les contraintes qui sont violées sur le reseau donc sur $Z_{std}(\mathbf{I})$.

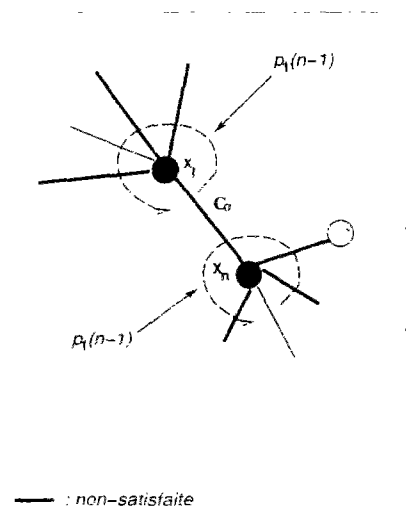



FIG. 3.3 – *Exemple:équation 3.4*

Nous avons généré les deux fonctions: $E[Z_{std}(\mathbf{I})]$ et $E[Z(\mathbf{I})]$ pour des graphes avec 8 variables (n=8) et leur taille de domaine égale à 3 (m=3), pour différentes valeurs

de $p_1$ et $p_2$. Sur la figure 3.4, calculée à partir de l'équation 3.3, nous observons que la variation de valeur de la fonction $E[Z_{std}(\mathbf{I})]$ est directement proportionnelle à $p_1$ et $p_2$, par contre sur la figure 3.5 déduite de la formule 3.4, notre fonction est directement proportionnelle à $p_1^2$ et à $p_2$. Cela veut dire que notre fonction dépend plus fortement de $p_1$ (e.g. connectivité) que $E[Z_{std}(\mathbf{I})]$.

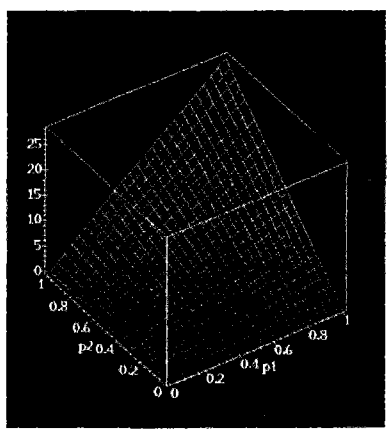



FIG. 3.4 – $E[Z_{std}(\mathbf{I})]$ *en fonction de* $p_1$ *et* $p_2$

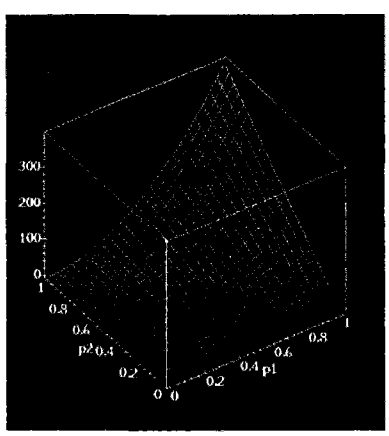



FIG. 3.5 – $E[Z(\mathbf{I})]$ *en fonction de* $p_1$ *et* $p_2$

## 3.3 Minimum-réseau-conflits

Nous voulons inclure l'idée développée dans le calcul de la fonction $Z(\mathbf{I})$ comme une nouvelle heuristique dans la méthode *min-conflits*, [MJPL92]. Pour cela, nous commençons avec la définition de l'ensemble de contraintes qui ne sont pas satisfaites et qui ont la même variable pertinente $X_j$, noté par $N_j$:

**Définition 3.3.1** *($N_j(\mathbf{I})$)*
*Étant donné un CSP binaire $P = (V, D, \zeta)$ avec une instanciation $\mathbf{I}$, pour une variable $X_j$, un ensemble de contraintes $N_j(\mathbf{I}) \subseteq \zeta$ est défini par $C_\alpha \in N_j(\mathbf{I})$ ssi:*

- $X_j \rhd C_\alpha$
- *$C_\alpha$ est violée sous $\mathbf{I}$*

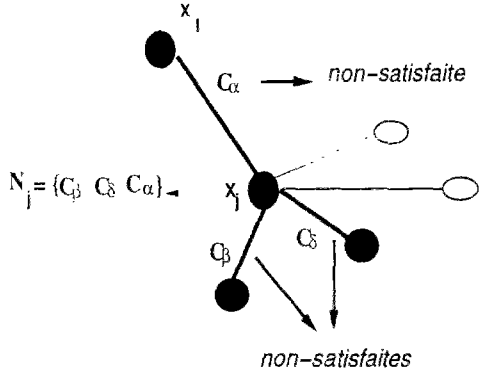



FIG. 3.6 – *Définition 3.3.1*

En utilisant cette définition, nous pouvons décrire l'heuristique de min-conflits (voir définition 1.3.1) du point de vue des contraintes comme:

**Définition 3.3.2** *(Fonction de min-conflits)*
*Soit un CSP binaire $P = (V, D, \zeta)$ avec une instanciation $\mathbf{I}$, et l'ensemble de contraintes $N_j(\mathbf{I})$ pour chaque variable $X_j$. La fonction de min-conflits,* **mc** *pour $X_j$ est définie par:*

$$\mathbf{mc}(X_j, \mathbf{I}) = \textit{Nombre de contraintes en } N_j(\mathbf{I})$$

En d'autres mots, $\mathbf{mc}(X_j, \mathbf{I})$ quantifie le nombre de conflits (contraintes violées) dans lesquels participe $X_j$.

Pour introduire l'idée de la connectivité dans l'heuristique, nous allons remplacer la fonction **mc** par la fonction suivante:

**Définition 3.3.3** *(Fonction de Min-réseau-conflits)*
*Étant donné un CSP binaire $P = (V, D, \zeta)$ avec une instanciation* $\mathbf{I}$*, et l'ensemble de contraintes $N_j(\mathbf{I})$ pour chaque variable $X_j$, et les fonctions $\mathbf{e}(C_\alpha, \mathbf{I})$ pour toute contrainte $C_\alpha$, la fonction de min-réseau-conflits,* **nc** *pour $X_j$ est définie par:*

$$\mathbf{nc}(X_j, \mathbf{I}) = \sum_{C_\gamma \in N_j(\mathbf{I})} \mathbf{e}(C_\gamma, \mathbf{I}) \tag{3.5}$$

**Remarque 3.3.1** *Cette fonction est calculée en prenant en compte seulement les arêtes impliquées (liées à $X_j$).*

Nous allons utiliser cette fonction dans l'algorithme de réparation *min-conflits*, décrit sur la figure 1.4, en introduisant une nouvelle heuristique que nous définissons dans la section suivante:

### 3.3.1 Heuristique: min-réseau-conflits

Comme pour *min-conflits*, l'heuristique commence avec une instanciation $\mathbf{I}$ complète devant être réparée. D'abord, la variable à changer est choisie aléatoirement parmi les variables en conflits, ensuite nous sélectionnons une valeur pour elle. La sélection de la valeur est faite en utilisant la *fonction min-réseau-conflits* **nc** (équation 3.5). Pour faire cela, la procédure calcule **nc** pour toutes les valeurs dans le domaine de la variable $X_j$ et elle sélectionne la valeur $x_j$ qui a une valeur de la *fonction min-réseau-conflits* la plus faible. Cela est possible parce que nous travaillons sur des réseaux de contraintes avec des domaine finis.
Cette heuristique guide le choix de la valeur de la variable sélectionnée en cherchant la plus grande descente, en termes des conflits directs et indirects du graphe

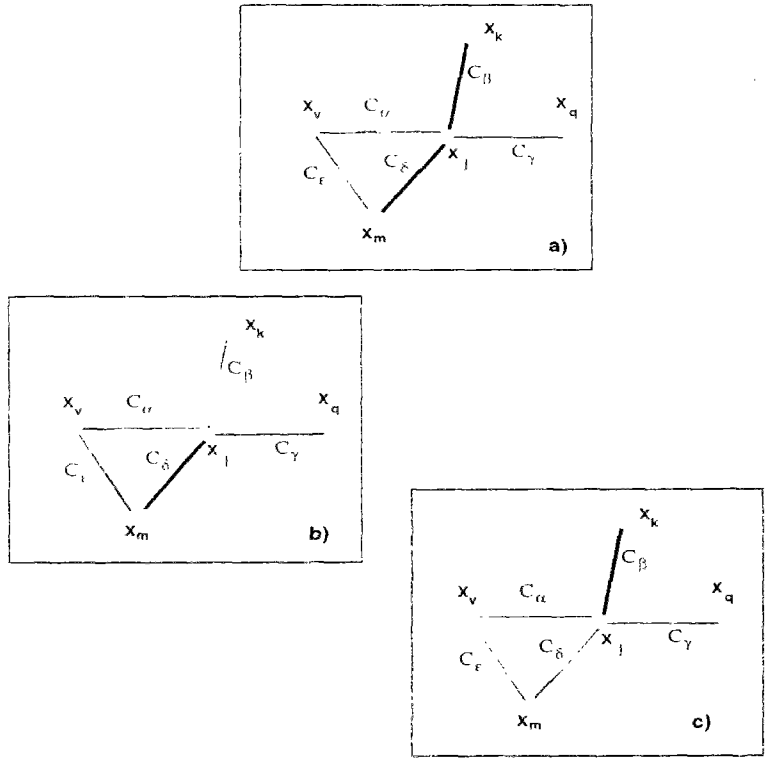



FIG. 3.7 – *Recherche d'une valeur pour* $X_j$

de contraintes. Pour montrer ce que cela veut dire regardons la figure 3.7. Supposons que le graphe a) représente la première pré-solution pour un problème donné. L'instanciation représentée viole $C_\beta$ et $C_\delta$ (dessinées par une ligne noire). Soit $X_j$ la variable en conflit sélectionnée d'une façon aléatoire pour être réparée. Le graphe b) et le graphe c) correspondent à deux choix de valeurs possibles pour cette variable. Pour l'heuristique *min-conflits*, les deux valeurs sont également bonnes, parce qu'après avoir modifié la valeur de $X_j$, dans les deux cas il ne reste qu'une contrainte violée. Mais pour *min-réseau-conflits* le graphe c) est meilleur que le graphe b) parce que l'instanciation représentée en c) pourrait être plus facilement réparée du point de vue de l'effet de propagation sur le réseau. Il suffirait que $X_k$ soit sélectionnée et que l'algorithme instancie une valeur satisfaisant $C_\beta$.

**Remarque 3.3.2** *Si le CSP est représenté par un graphe complètement connecté,*

*avec chaque variable appartenant à* n-1 *contraintes, alors min-réseau-conflits et min-conflits ont le même comportement. Plus précisément, pour un CSP où chaque variable appartient exactement à c contraintes, leur recherche est équivalente.*

*Min-conflits* a pour chaque contrainte la valeur 1 ou 0, selon si elle est satisfaite ou non-satisfaite, quel que soit le type du réseau de contraintes. Quand il s'agit d'un graphe où chaque variable est connectée à un ensemble $c$ de contraintes, *min-réseau-conflits* ajoute la valeur $2c$ ou 0 selon que la contrainte est satisfaite ou non-satisfaite. Pour un graphe complètement connecté, *min-réseau-conflits* $\mathbf{nc}(X_j, \mathbf{I})$ vaudra pour le pire des cas $2(n-1)^2$. La priorité de contraintes implicite de *min-réseau-conflits*, dans ce cas n'existe pas, car elle est faite en prenant en compte la connectivité de chaque variable.

### 3.3.2 Algorithme de réparation en deux étapes

Nous voulons incorporer cette heuristique dans un algorithme de réparation. L'idée est de faire de l'escalade guidée par le réseau de contraintes. Nous avons réalisé plusieurs tests avec *min-conflits* et *min-réseau-conflits* et nous avons trouvé une sorte de complémentarité entre les deux heuristiques. Plus précisément, certains problèmes qui étaient faciles pour *min-conflits* ne l'étaient pas pour *min-réseau-conflits* et vice-versa. Cela peut être apprecié sur la fig 3.10. Cela a motivé la proposition d'algorithme en deux étapes. Dans la première étape, il utilise l'heuristique de *min-conflits*, donc avec la fonction **mc**, (définition 3.3.2). Dans le cas où il n'est pas capable de trouver une solution à la fin d'un nombre maximum d'itérations, il continue avec l'heuristique de *min-réseau-conflits*, donc avec la fonction **nc** (définition 3.3.3), jusqu'à trouver une solution ou avoir réalisé un nombre maximum d'itérations. Nous avons donc deux paramètres de contrôle, le nombre maximum d'itérations pour *min-conflits* et pour *min-réseau-conflits*. Nous pouvons interpréter cet algorithme comme une façon d'explorer d'abord les conflits directs. Si nous n'avons pas de succès dans la première étape, alors l'algorithme continue son exploration en utilisant les conflits propagés dans le réseau. La structure de l'algorithme est montrée sur la figure 3.8.

Pour évaluer notre algorithme, nous allons utiliser des problèmes générés d'une

**Procédure** Deux-étapes $(V, D, \zeta)$
**Début**
Générer aléatoirement une pre-solution **I**
**si** **I** n'est pas une solution **alors**
  **Répéter**
    Sélectionner aléatoirement $X_j \in K(\mathbf{I})$ [a]
    $\mathbf{I}(X_j) = argmin_{x_j \in D_j}\{\mathbf{nc}(X_j = x_j, ((\mathbf{I} - x_{j_a}{}^{b}) \cup x_j))\}$ [c]
    itérations-conflits++
  **Jusqu'à** ((**I** soit solution) ou (max itérations-conflits))
**si** **I** n'est pas une solution **alors**
  **Répéter**
    Sélectionner aléatoirement $X_j \in K(\mathbf{I})$
    $\mathbf{I}(X_j) = argmin_{x_j \in D_j}\{\mathbf{nc}(X_j = x_j, ((\mathbf{I} - x_{j_a}) \cup x_j))\}$
    itérations-réseau-conflits++
  **Jusqu'à** ((**I** soit solution) ou (max itérations-réseau-conflits))
Fin /* procédure deux-étapes*/

[a] l'ensemble des variables en conflit
[b] sa valeur courante
[c] $argmin_{l \in S}\{a_l\}$ donne la valeur $l^*$ tel que $a_{l^*} \leq a_l, \forall l \in S$

FIG. 3.8 – *Structure de la procédure deux étapes*

façon aléatoire, spécifiquement le problème de coloriage de graphe avec trois couleurs, à partir de maintenant dénommé par *3-coloriage*. La procédure utilisée est décrite dans la section suivante. Elle sera aussi rappelée dans les prochains chapitres.

### Ensemble de problèmes: 3-coloriage

Le problème du coloriage de graphe est un problème NP-complet bien connu, qui est utilisé pour modéliser certains types de problèmes d'ordonnancement et d'allocation de ressources. Le but de nos essais est d'observer l'effet d'incorporer notre approche dans la performance d'un algorithme de réparation. Les problèmes de 3-coloriage sont générés de façon aléatoire, la contrainte est toujours la même: un nœud ne doit pas avoir la même couleur qu'un nœud voisin. La procédure de génération des graphes a été proposée par Adorf et al. en [AJ90]. Cette procédure génère des problèmes, qui ont au moins une solution, de la façon montrée sur la figure 3.9.

Les graphes générés par cette procédure auront $n$ nœuds (variables) et $m$ arêtes (contraintes).

**Procédure** Génération des Problèmes 3-Coloriage $(n, m)$
**Début**
Affecter $\frac{n}{3}$ des $n$ nœuds à chacun des trois ensembles (A,B,C)
**Répéter**
    $\text{Ensemble}_1$ = random-entre(A, B, C)
    $\text{nœud}_1$ = random-dans($\text{Ensemble}_1$)
    $\text{Ensemble}_2$ = random-entre(A,B,C - $\text{Ensemble}_1$)
    $\text{nœud}_2$ = random-dans($\text{Ensemble}_2$)
    **si** ($\nexists C_\alpha$ tel que $\text{nœud}_1 \rhd C_\alpha$ et $\text{nœud}_2 \rhd C_\alpha$) **alors**
      Créer $C_\alpha$ entre $\text{nœud}_1$ et $\text{nœud}_2$
      Nombre-d'arêtes++
**Jusqu'à** Nombre-d'arêtes = $m$
**Fin** /* Procédure Génération des Problèmes 3-Coloriage */

FIG. 3.9 – *Procédure de Génération aléatoire des Problèmes de 3-Coloriage avec solution*

### Évaluation de l'heuristique

Pour évaluer l'algorithme en deux étapes nous avons généré des graphes aléatoires, qui sont appelés "graphes peu denses[1]", c'est à dire, avec $\eta = 2n$, d'après leur structure. Ce type de graphes est connu pour être celui qui donne le plus de difficulté aux algorithmes de réparation stochastique (voir [MJPL92]). Ces graphes ont plusieurs solutions donc l'algorithme a plus de choix pour effectuer la sélection de valeurs, donc le nombre de combinaisons "prometteuses" de valeurs est plus important que pour les graphes denses. Nous avons essayé d'abord de résoudre chaque graphe en utilisant l'heuristique, définie par Brelaz [Bre79], laquelle est assez performante pour les problèmes de 3-coloriage. Il s'agit d'un algorithme glouton qui affecte une couleur à chaque nœud.

**Définition 3.3.4** *(Heuristique de Brelaz)*
*Trouver le nœud qui n'est pas colorié qui a le moins de couleurs consistantes avec*

1. en anglais: sparse graphs

*ses voisins. S'il y en a plusieurs, alors en choisir un qui a le degré maximum dans le sous-graphe non-colorié.*

Nous allons faire les expériences seulement avec les problèmes que l'heuristique de Brelaz n'a pas pu résoudre, donc des problèmes particulièrement difficiles. La pré-solution trouvée par l'algorithme qui utilise cette heuristique, sera utilisée comme pré-solution initiale dans nos tests.

Pour chaque $\eta = [60, 90, 120, 150, 180]$ nous avons 100 graphes différents qui n'ont pas pu être résolus en appliquant l'heuristique de Brelaz (définition 3.3.4). Nous comparons trois algorithmes d'escalade, le premier utilise l'heuristique *min-conflits* avec un nombre maximum d'itérations égal à 5000. Le deuxième utilise l'algorithme en deux étapes. Pour chaque étape, le nombre maximum d'itérations est fixé à 2500. Le troisième utilise l'algorithme avec l'heuristique *min-réseau-conflits* pur, avec un maximum de 5000 itérations. Les résultats sont montrés sur la figure 3.10. En abscisse, nous avons le nombre de contraintes du graphe, en ordonnée nous avons le pourcentage de convergence de l'algorithme. Par exemple, pour les graphes avec 60 contraintes (30 variables), notre algorithme a trouvé une solution pour 70% des graphes, en revanche *min-conflits* a trouvé la solution seulement pour 20% des graphes. Au fur et à mesure que la taille du problème augmente, la probabilité de trouver une solution diminue, mais notre algorithme présente toujours un meilleur comportement que *min-conflits* pur. Il est important de remarquer que ce type de problèmes est particulièrement difficile pour les méthodes stochastiques. Ces expériences montrent que le fait de prendre en compte la structure du réseau de contraintes peut être un facteur important pour guider la recherche des méthodes stochastiques.

Maintenant que nous avons constaté que la fonction d'évaluation peut apporter une amélioration de la performance d'une recherche stochastique par réparation locale, nous allons introduire cette fonction dans un algorithme évolutionniste.

## 3.4 Z(I) dans un algorithme évolutionniste

Nous avons conçu un jeu de tests pour mesurer l'avantage d'utiliser Z(**I**) au lieu de $Z_{std}(\mathbf{I})$ dans un algorithme évolutionniste (EA). Dans le chapitre précédent, nous

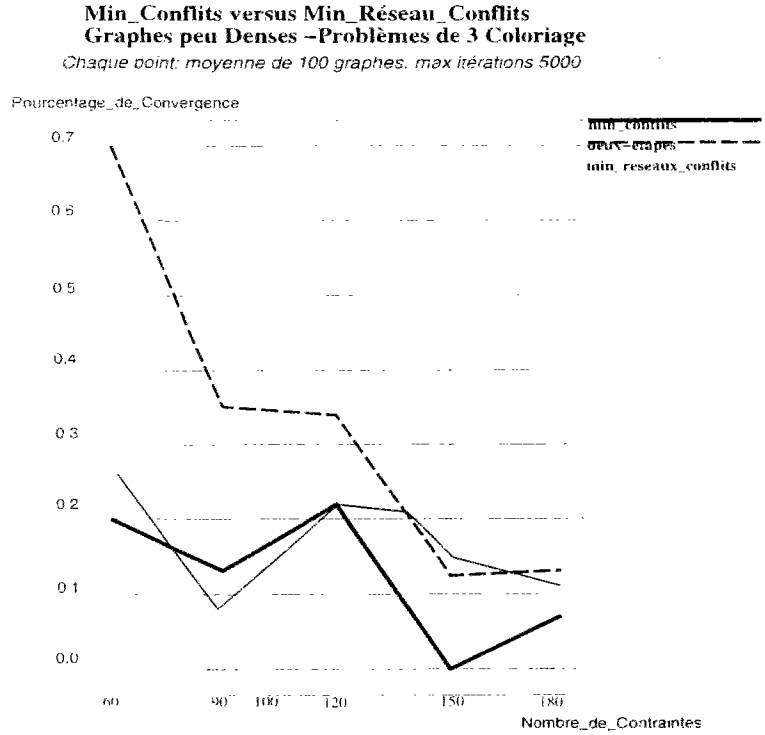



FIG. 3.10 – *Comparaison: Min-conflits v/s Min-réseau-conflits pour graphes peu denses*

avons affirmé que la conception d'un algorithme évolutionniste a besoin de six composantes (voir figure 2.1). Dans les sections suivantes, nous décrirons les différentes parties de notre EA.

### 3.4.1 Population initiale

La population initiale est générée aléatoirement. Chaque instanciation **I** est composée par des valeurs des variables qui sont choisies à partir de leurs domaines avec une distribution de probabilité uniforme.

### 3.4.2 Représentation génétique

Nous avons choisi une représentation génétique non-binaire, car ce type de représentation s'adapte naturellement aux CSP. Les domaines de variables dans un CSP

peuvent être de différents types, nous pouvons donc avoir dans un même CSP des variables booléennes, réelles, entières. Alors, la représentation génétique choisie a la structure de la figure 3.11

| X1 | X2 | X3 | ..... | Xi | ..... | Xn |
|---|---|---|---|---|---|---|
| A | 1 | 1024 | ..... | 0 | ..... | Z |

FIG. 3.11 - *Représentation du Chromosome*

### 3.4.3 Algorithme de sélection

La fonction d'évaluation Z(**I**) nous a permis de concevoir un nouvel algorithme pour réaliser la sélection à partir de la population. Nous avons présenté dans la section 2.2.1 la méthode de sélection standard de la roue biaisée pour le problème de maximisation, son adaptation pour le calcul de la probabilité de sélection pour notre problème de minimisation est présentée dans l'annexe A. Nous souhaitons avantager les chromosomes qui ont une valeur de la fonction d'évaluation inférieure, plus fortement que ce que fait la méthode de sélection standard. Les mauvais individus auront dans notre algorithme quand même une probabilité d'être choisis, parce que pour obtenir des bons individus on a souvent besoin de produire de mauvais individus comme structures intermédiaires, [Mic94].

Nous avons conçu la stratégie de sélection de la figure 3.12. Nous divisons la population en trois sous-populations, la première est constituée par les individus qui ont une fonction d'évaluation inférieure à la moyenne, la deuxième ceux qui ont une fonction d'évaluation comprise entre la moyenne et la moyenne plus l'écart type, dans la dernière sous-population nous avons les individus qui ont une valeur supérieure à la moyenne plus l'écart type.

Pour déterminer $\alpha$ et $\beta$, nous avons comparé, pour différents réseaux de contraintes, le nombre de générations nécessaire pour trouver une solution. Pour chaque combinaison nous avons résolu 100 problèmes de 3-coloriage avec 30 variables générés de manière aléatoire. La table 3.1 montre le nombre de générations d'un des ensembles de test (des graphes avec une connectivité égale à 4) pour différentes valeurs de $\alpha$ et

| $\alpha$ | $\beta - \alpha$ | Générations |
|---|---|---|
| 0.05 | 0.8 | 124.4 |
| 0.1 | 0.75 | 93.4 |
| 0.2 | 0.65 | 79.2 |
| 0.3 | 0.55 | 69.2 |
| 0.4 | 0.45 | 67.7 |
| 0.5 | 0.35 | 58.1 |
| 0.6 | 0.25 | 63.1 |
| 0.7 | 0.15 | 70.0 |
| 0.8 | 0.05 | 62.2 |

TAB. 3.1 - *Nombre de générations pour différentes valeurs de $\alpha$ et de $\beta - \alpha$, pour 3-coloriage avec une connectivité moyenne de 4.0*

de $\beta - \alpha$.

Parmi tous les tests générés, nous avons obtenu les meilleurs résultats avec $\alpha = 0.5$ et $\beta = 0.85$.

### Propriétés statistiques de l'algorithme de sélection

En utilisant les valeurs $\alpha$ et $\beta$, la population est divisée en trois régions A, B, C. Cela est montré dans la figure 3.13. Supposons que nous ayons une population de taille N, avec $n_1$ chromosomes dans la region A, $n_2$ dans la région B et $n_3$ dans la région C, tel que $n_1 + n_2 + n_3 = N$, alors nous avons les probabilités de sélection suivantes pour un chromosome:

- à partir de la région $A = \alpha + (\beta - \alpha) * \frac{n_1}{n_1+n_2} + (1 - \beta) * \frac{n_1}{N}$
- à partir de la région $B = (\beta - \alpha) * \frac{n_2}{n_1+n_2} + (1 - \beta) * \frac{n_2}{N}$
- à partir de la région $C = (1 - \beta) * \frac{n_3}{N}$

La figure 3.13 montre que nous avons une préférence pour les chromosomes de la région A, c'est à dire, nous préférons les individus dont la fonction d'évaluation est inférieure ou égale à la moyenne. Néanmoins, la probabilité de sélectionner des individus à partir de la région C existe. Pour illustrer cela, regardons le problème de coloriage de graphe sur la figure 3.14. La population est composée par quatre

**Procédure** sélection(Population)
**Début**
Choisir j à partir de $U^{a}[0..1]$
**si** (j $< \alpha$) **alors**
  Choisir chromosome tel que $\mathbf{Z}(chromosome) \leq F^{b}$
**sinon**
  **si** (j $< \beta$) **alors**
  Choisir chromosome tel que $\mathbf{Z}(chromosome) < F + \sigma^{c}$
  **sinon**
    Choisir chromosome à partir $U[1..taille - population]$
**Fin** /* sélection */

[a] distribution uniforme
[b] moyenne de la population
[c] écart-type de la population

FIG. 3.12 – *Algorithme de Sélection*

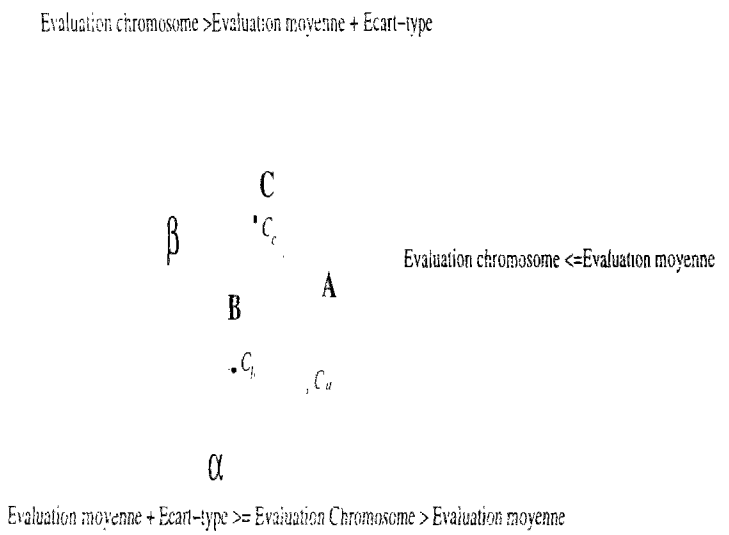
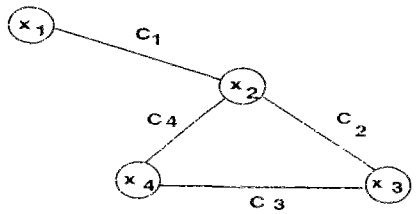



FIG. 3.13 – *Régions de Sélection*




| | | Fonction d'evaluation | | Probabilite de sélection | | |
|---|---|---|---|---|---|---|
| | contraintes violées | $Z_{std}(I)$ | Z(I) | $P_r$ | P | Région |
| Pré–solution 1 | C 4 | 1 | 5 | 0.28 | 0.41 | A |
| Pré–solution 2 | C 1 | 1 | 4 | 0.30 | 0.41 | A |
| Pré–solution 3 | C 4 C 3 | 2 | 9 | 0.20 | 0.03 | C |
| Pré–solution 4 | C 1 C 3 | 2 | 8 | 0.21 | 0.15 | B |

FIG. 3.14 - *Exemple de Sélection*

chromosomes ou pré-solutions. Chacun des individus viole au moins une contrainte. Nous pouvons regarder la valeur de leur fonction d'évaluation $Z_{std}(\mathbf{I})$ et $Z(\mathbf{I})$. Les probabilités de sélection montrées sur la figure sont calculées en utilisant $Z(\mathbf{I})$ pour les deux algorithmes de sélection. $P_r$ est la probabilité pour la méthode de la roue biaisée et $P$ pour notre méthode. Nous pouvons conclure que notre algorithme est plus strict avec les mauvais chromosomes, par contre les bons chromosomes qui sont classés dans la région A sont traités avec égalité, il y a aussi plus de chance de choisir un chromosome de la région A qu'avec la méthode de la roue biaisée.

Dans l'annexe A, nous calculons la probabilité de sélection en utilisant la roue biaisée, pour le traitement des meilleurs individus (ceux qui sont classés par notre méthode dans la sous-population A).

### 3.4.4 Tests sur différents ensembles de problèmes de 3-coloriage

Nous avons choisi, pour évaluer notre fonction dans l'algorithme évolutionniste, le problème de 3-coloriage sur des graphes générés d'une façon aléatoire. Nous avons conçu différents tests qui sont classés selon les opérateurs génétiques utilisés. Le premier ensemble de problèmes utilise les opérateurs standards: *croisement à un point*

```
Procédure Algorithme Évolutionniste pour CSP
Début
t = 0
initialiser population P(t)
évaluer les individus en P(t) en utilisant Z(I)
tant que (non condition de fin) faire
      t=t+1
     tant que (P(t) n'est pas complète) faire
           Parent1 = sélection[a](P(t-1))
           Parent2 = sélection(P(t-1))
           Enfants = transformation(Parent1, Parent2)
           P(t) = Enfants ∪ P(t)
      évaluer P(t) en utilisant Z(I)
Fin /* procédure Algorithme Évolutionniste pour CSP */
```

[a] algorithme de sélection défini sur la figure 3.12

FIG. 3.15 – *Structure de l'Algorithme Évolutionniste proposé pour résoudre les CSP*

et *mutation*. Les résultats obtenus avec ces opérateurs nous ont motivée pour définir un nouvel opérateur nommé *permutation* qui aidera à améliorer ces résultats. Enfin, le dernier ensemble de problèmes utilise un algorithme avec l'opérateur asexué(n,p,g) avec les paramètres (#,r,b) spécialement défini par Eiben et al. en [ERR95] pour le problème de 3-coloriage (voir chapitre 2). Nous avons comparé les résultats obtenus en utilisant $Z_{std}(\mathbf{I})$ et $Z(\mathbf{I})$ comme fonctions d'évaluation. Pour tous les tests, nous avons utilisée une taille de la population de 20 individus, une probabilité de croisement de 0.9 et une probabilité de mutation de 0.1. La structure générale de l'algorithme proposé est montré sur la figure 3.15. Il utilise la fonction d'évaluation $Z(\mathbf{I})$ et l'algorithme de sélection défini par la figure 3.12. La transformation dépend des opérateurs utilisés sur les différents ensembles de problèmes.

### Tests avec opérateurs standards: *croisement à un point, mutation*

Considérons d'abord le petit graphe pour 3-coloriage avec seulement sept variables et onze contraintes illustré par la figure 3.16. Ce graphe en particulier pourrait être réduit au sous-graphe composé par les nœuds 2, 4 et 6, en appliquant l'algorithme

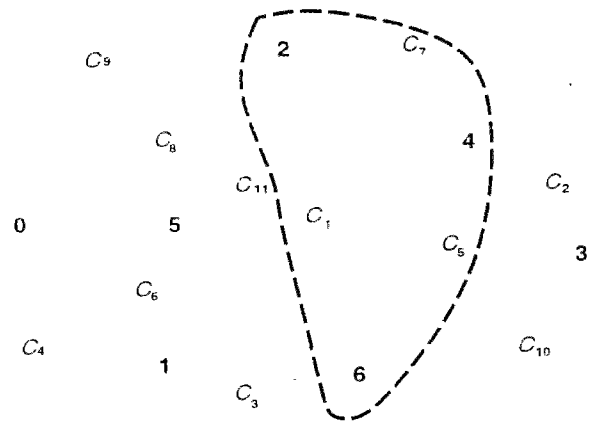



FIG. 3.16 – *Graphe 3-coloriage*

de réduction proposé par Cheeseman [CKT91]. Si nous avons une instanciation qui satisfait les nœuds 2,4 et 6 (lignes coupées), il sera facile de trouver une valeur pour les autres variables qui satisferont toutes les contraintes. Cependant, pour illustrer notre algorithme, la recherche d'une solution a été faite avec tous les nœuds en utilisant la nouvelle fonction d'évaluation. Avec cette fonction d'évaluation, nous donnons une certaine priorité aux contraintes pour les satisfaire. Par exemple dans le graphe, la contrainte $C_1$ entre le nœud 2 et le nœud 6 est plus importante à satisfaire que la contrainte $C_{10}$ qui lie le nœud 3 avec le nœud 6, même en partageant le même nœud 6. En effet, la contrainte $C_1$ à travers le nœud 2 est connectée à $C_9$, $C_8$, $C_{11}$ et $C_7$, par contre la contrainte $C_{10}$ à travers le nœud 3 est seulement connectée à la contrainte $C_2$. Cela confirme la réduction qu'on pourrait faire en faisant la pré-analyse proposée par Cheeseman. Cet exemple est trivial, mais son but est de montrer que la structure est une chose importante à considérer pour améliorer la performance de la recherche stochastique faite par un algorithme évolutionniste. Nous avons généré six différentes topologies de graphes avec 7 variables et 11 contraintes. Nous avons résolu le 3-coloriage de ces graphes en utilisant un algorithme avec $Z_{std}(\mathbf{I})$, l'*Algorithme A*, et un algorithme avec $Z(\mathbf{I})$, l'*Algorithme B*, toutes les autres composantes de l'EA étant évidemment les mêmes pour les deux. Pour chaque graphe généré, nous avons changé l'ordre des variables dans la représentation du chromosome.

Les résultats sont montrés sur la figure 3.17 pour l'*Algorithme A* et sur la figure 3.18 pour l'*Algorithme B*.

En observant ces résultats, on peut conclure les faits suivants:

- Le nombre de générations en utilisant $Z(\mathbf{I})$ est inférieur à celui obtenu en utilisant $Z_{std}(\mathbf{I})$. En fixant le nombre maximum d'itérations à 100, $Z(\mathbf{I})$ arrive à trouver la solution avec un nombre de générations moyen de 20, en revanche avec $Z_{std}(\mathbf{I})$ le nombre d'itérations moyen est de 48.

- Le nombre de générations utilisé pour les deux algorithmes est influencé par l'ordre dans lequel les variables sont organisées dans le chromosome, c'est à dire, qu'il y a des ordres qui permettent de trouver plus facilement la solution que d'autres. Par exemple, pour la topologie 4, l'ordre 19 a empêché l'*Algorithme A* avec $Z_{std}(\mathbf{I})$ de trouver une solution. Pour le même problème et avec le même ordre des variables, l'*Algorithme B* a trouvé une solution.

- Le surcoût de l'utilisation de Z(**I**) à la place de $Z_{std}(\mathbf{I})$ est négligeable, car on détermine la valeur de la *contribution* (voir def 3.2.3) de chaque contrainte $C_\alpha$ au début de l'algorithme. Pour calculer la valeur de Z(**I**), on vérifie si la contrainte est violée ou pas (comme pour $Z_{std}(\mathbf{I})$). La seule différence reside dans le cas où la contrainte n'est pas satisfaite, car on ajoute à la fonction d'évaluation la valeur de sa *contribution* au lieu de 1 (comme le fait $Z_{std}(\mathbf{I})$).

Ensuite, nous avons généré un problème avec 30 variables et 40 contraintes, les résultats des deux algorithmes sont comparés sur la figure 3.19, toujours pour 30 ordres différents des variables dans le chromosome, un nombre maximum d'itérations égal à 100, donc nous pouvons conclure:

- Pour l'*Algorithme B*, en moyenne, il a été plus facile de trouver une solution. Néanmoins, pour l'*Algorithme A*, le "meilleur ordre" a permis de trouver une solution avec 12 itérations, en revanche pour l'*Algorithme B* l'ordre 11 a permis de trouver la solution en faisant 16 itérations.

- L'*Algorithme A* n'a pas pu trouver la solution pour un ensemble plus important d'ordres que l'*Algorithme B*, qui a échoué seulement avec l'ordre 13.

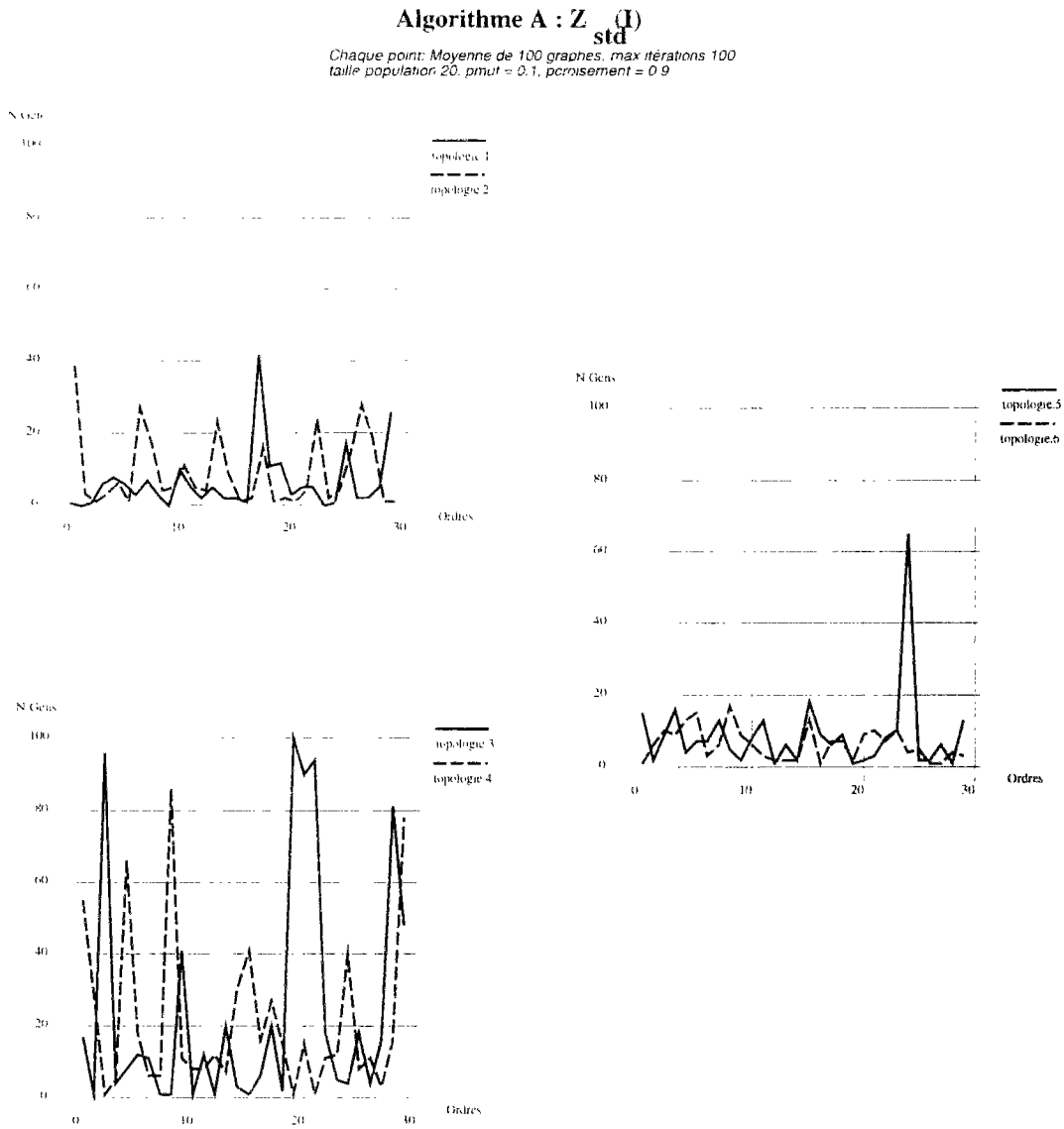



FIG. 3.17 – *Algorithme A:Graphes avec 7 variables et 11 contraintes. Echelle du nombre de générations entre 0 et 100*

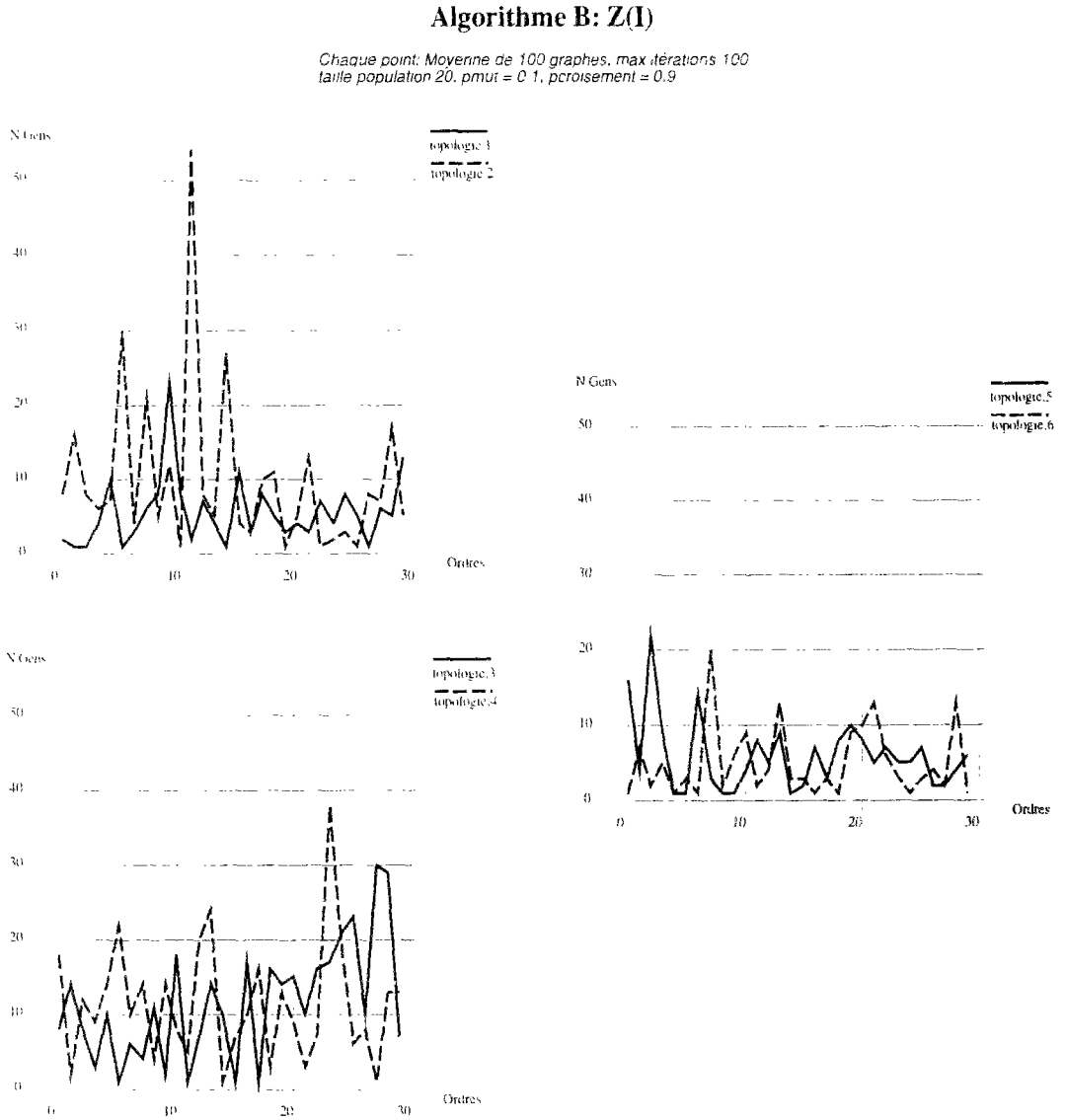



FIG. 3.18 – *Algorithme B: Graphes avec 7 variables et 11 contraintes. Echelle du nombre de générations entre 0 et 50*

Il est important de remarquer que nous n'avons pas une connaissance a priori du bon ordre des variables dans le chromosome. À partir de ces résultats, nous pouvons donc conclure finalement que considérer la structure dans la fonction d'évaluation peut conduire à une amélioration de la performance de l'algorithme, même en choisissant un mauvais ordre des variables dans le chromosome. L'ordre des variables dans le chromosome joue un rôle important aussi dans la performance de l'algorithme. Cela a été considéré par Goldberg [Gol89] et plus récemennt par Kargupta [Kar95] pour la conception des algorithme génétiques "messy". Cela motive l'introduction d'un opérateur de "permutation" qui nous permettra de modifier la valeur de la *longueur utile d'un schéma*, $\delta(Sch)$, (voir définition 2.2.7) en changeant de position certaines variables dans le chromosome, en conséquence la probabilité de destruction du schéma (équation 2.6) changera aussi. L'objectif de cet opérateur de "permutation" est d'augmenter la potentialité de l'opérateur de croisement à un point.

**Opérateur de permutation** L'opérateur de permutation sera activé seulement une fois que l'algorithme réalise que l'ordre des variables dans le chromosome n'est pas bon. S'il a choisi au début un bon ordre, l'algorithme converge naturellement sans une aide supplémentaire. Nous incorporons un autre paramètre, la *probabilité de permutation*, qui correspond à la probabilité de changer l'ordre des variables dans le chromosome. Contrairement aux opérateurs de croisement et mutation, pour cet opérateur, c'est toute la population qui est concernée, c'est-à-dire, si l'algorithme décide de réaliser une permutation, il changera de la même manière la position des variables dans la population entière. Si l'opérateur n'est pas activé, la probabilité de permutation est nulle. Nous avons conçu une simple heuristique pour identifier un mauvais ordre des variables. Avant de l'introduire, nous définissons le concept de stabilité:

**Définition 3.4.1** *(Stabilité dans la Recherche)*
*On dit qu'un ordre est stable si l'algorithme évolutionniste a trouvé le même meilleur chromosome pendant les S dernières générations.*

La figure 3.20 montre l'instant pendant le processus d'évolution où l'opérateur pourrait être activé. La structure de l'algorithme de l'opérateur de permutation est

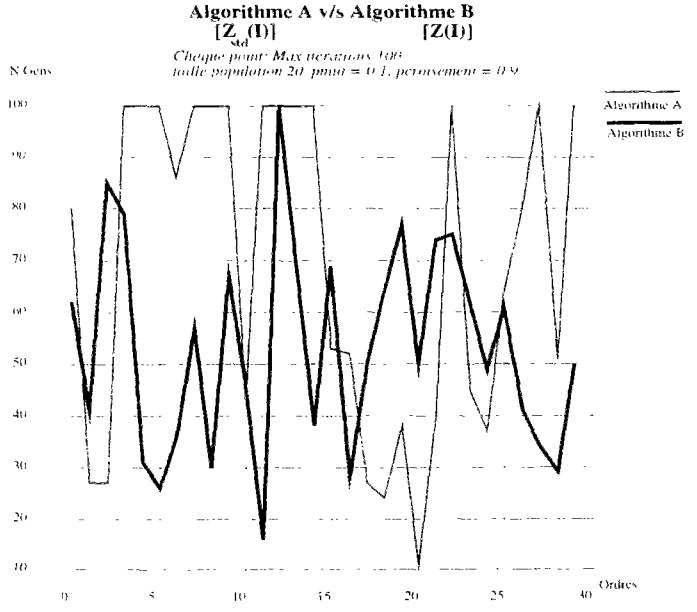



FIG. 3.19 – *Graphe avec 30 variables, 40 contraintes: Algorithme A v/s Algorithme B*

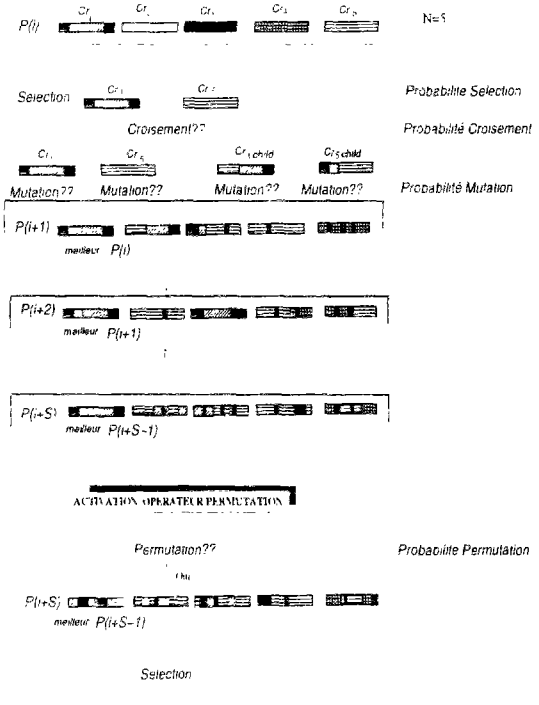



FIG. 3.20 – *Activation de l'opérateur de permutation*

**Procédure** Opérateur Permutation (Population)
**Début**
si l'opérateur de permutation est activé **alors**
Générer un nombre aléatoire $r$ à partir $U[0..1]$
si $r$ <probabilité de permutation **alors**
Générer un nouvel ordre du chromosome pour $[1..n]$
**Pour** tous les chromosomes dans la population
Positionner $(X_j, x_j)$ suivant le nouvel ordre, $\forall j = 1, \ldots, n$
**Fin** /* opérateur permutation*/

FIG. 3.21 – *Structure de l'opérateur permutation*

montrée sur la figure 3.21. Une fois que l'opérateur est activé, l'algorithme l'applique pendant toutes les futures générations selon la *probabilité de permutation*. La valeur de la probabilité de permutation est faible pour ne pas perturber la recherche réalisée avec un nouvel ordre.

Nous avons incorporé cet opérateur dans les algorithmes $A$ et $B$ de la section précédente. Nous avons fait des tests avec S=25 et une probabilité de permutation de 0.3. Les résultats ont montré une augmentation de la performance pour les pires ordres de 20%, donc l'*algorithme B* cette fois-ci a été capable de trouver une solution pour tous les problèmes.

On pourrait aussi utiliser d'autres critères plus complexes et coûteux du point de vue du temps de calcul pour mesurer la stabilité, tels que l'entropie définie en [FF95] pour le 3-coloriage comme:

**Définition 3.4.2** *(Entropie)*
*Soit $n_{ij}$ le nombre de fois que le nœud $i$ prend la couleur $j$ dans la population $P$. On définit la mesure de la diversité de la population* entropie *par:*

$$E = \frac{-\sum_{i=1}^{|V|}\sum_{j=1}^{k}\frac{n_{ij}}{|P|}log(n_{ij}|P|)}{|V|logk} \tag{3.6}$$

*Cette entropie $E \in [0, 1]$ et elle indique à quel point les couleurs sont uniformément affectées aux nœuds dans la population.*

| Algorithme | Fonction d'évaluation | Sélection |
|---|---|---|
| A | $Z_{std}(\mathbf{I})$ | Roue Biaisée |
| B | Z(**I**) | Roue Biaisée |
| C | Z(**I**) | Nouvelle sélection |

TAB. 3.2 – *Trois algorithmes qui diffèrent par leur fonction d'évaluation et par leur algorithme de sélection*

Nous pouvons décider que dans le cas où la population n'est pas assez diversifiée (petite entropie), alors on active la permutation.

### Tests avec l'opérateur spécialisé: (#,r,b)

Dans cet ensemble de tests, nous avons comparé trois algorithmes, lesquels diffèrent par leur fonction d'évaluation et par leur algorithme de sélection, comme cela est montré dans le tableau 3.2. Tous les trois utilisent l'opérateur asexué (n,p,g) avec les paramètres spécifiques pour le problème de 3-coloriage (#,r,b) (pour sa définition voir section 2.5.2). Pour cela, nous avons généré 1000 CSP aléatoires avec différentes topologies avec un degré de connectivité entre 4 et 6 pour 30 variables. Pour chaque connectivité, nous avons généré 100 différents graphes aléatoires.

La figure 3.22 montre le pourcentage de solutions trouvées pour les trois algorithmes. Les meilleurs résultats ont été trouvés en utilisant $Z(\mathbf{I})$ avec le nouvel algorithme de sélection. Il a trouvé dans le pire des cas 70% de solutions contrairement à l'*Algorithme A* qui lui a trouvé seulement 20% de solutions. La figure 3.23 montre le nombre moyen de générations pour chaque connectivité. Le nombre moyen d'itérations réalisé par l'*Algorithme A* est supérieur à celui des deux autres algorithmes. De plus, on peut observer que la nouvelle fonction donne de meilleurs résultats quand elle est couplée avec le nouvel algorithme de sélection.

Les problèmes qui ont été les plus difficiles à résoudre pour les trois algorithmes se trouvent parmi les connectivités [4.5, 5.0], ce qui correspond à la zone connue des problèmes difficiles de 3-coloriage [CKT91], [Smi95].

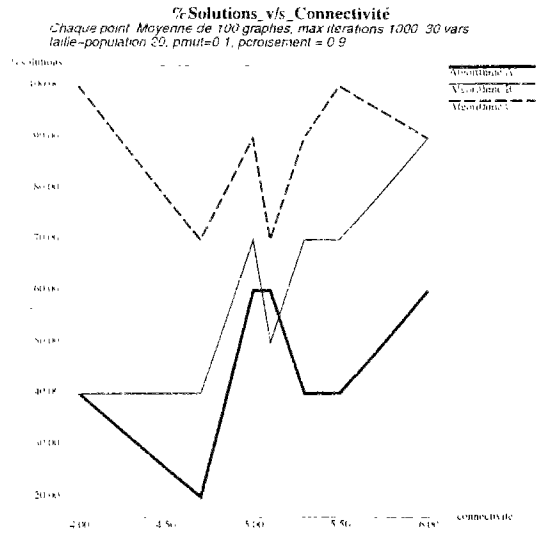



FIG. 3.22 – *Pourcentage de solutions trouvées par les trois algorithmes pour différentes connectivités*

## 3.5 Fonction d'évaluation pour CSP n-aire

Pour faire l'extension de la fonction d'évaluation présentée dans les sections précédentes aux CSP n-aires, il faut prendre en compte la réflexion suivante:

Si nous avons un chromosome dont les allèles ne satisfont pas une contrainte, pour le réparer, dans le pire de cas, nous devrions changer toutes les valeurs des variables qui appartiennent à cette contrainte, et les valeurs des variables qui leur sont connectées par d'autres contraintes dans le réseau. L'effet de propagation donc utilise la même idée que pour les réseaux binaires, comme on le montre dans les définitions suivantes:

**Définition 3.5.1** *(Évaluation de l'erreur n-aire)*
*Étant donné un CSP* $P = (V, D, \zeta)$ *avec une matrice de contraintes* $\mathbf{R}$ *et une instanciation* $\mathbf{I}$, *la fonction de l'évaluation de l'erreur n-aire* $\mathbf{e_n}(C_\alpha, \mathbf{I})$ *pour une contrainte* $C_\alpha$ *qui a comme variables pertinentes* $X_{k_i}, i \in [1, \ldots, l], l \leq n$, $(\mathbf{R}[\alpha, k_i] = 1, \forall i)$, *est définie par:*

$$\mathbf{e_n}(C_\alpha, \mathbf{I}) = (\textit{Nombre de variables en } C_\alpha) + \sum_i (\textit{Effet de propagation par } X_{k_i})$$

*où l'effet de propagation par* $X_{k_i}$ *dans un réseau de contraintes n-aires est défini comme le nombre de contraintes* $C_\beta, \beta = 1, \ldots, \eta, \beta \neq \alpha$ *qui ont aussi comme variable*

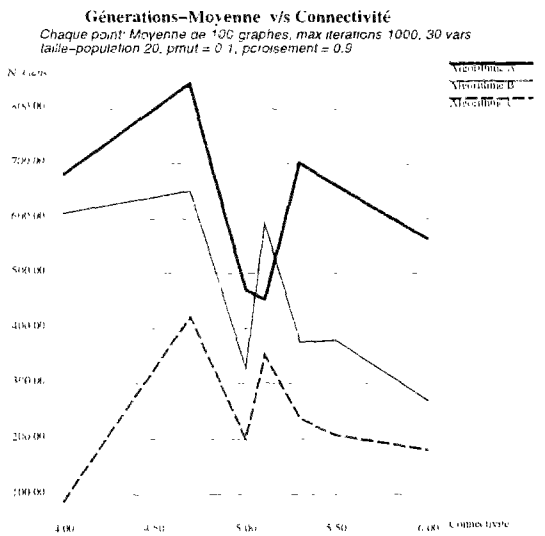



FIG. 3.23 – *Comparaison: Nombre moyen de générations des trois algorithmes pour différentes connectivités*

*pertinente* $X_{k_i}$. *Cela s'exprime en utilisant la matrice de contraintes par:*

$$\mathbf{e_n}(C_\alpha, \mathbf{I}) = \left(\sum_{w=1}^{n} \mathbf{R}[\alpha, w]\right) + \left(\sum_{w=1}^{n} \mathbf{R}[\alpha, w] \left[\sum_{\beta \neq \alpha, \beta=1}^{\eta} \mathbf{R}[\beta, w]\right]\right) \tag{3.7}$$

**Remarque 3.5.1** *Si* $C_\gamma$ *est satisfaite* $\mathbf{e_n}(C_\gamma, \mathbf{I}) = 0$

Nous illustrons l'effet de propagation n-aire sur la figure 3.24. Chaque contrainte est représentée par un carré, par exemple les variables pertinentes pour la contrainte $C_\alpha$ sont $X_1, X_2$ et $X_3$. Leur effet de propagation dans la matrice de contraintes $\mathbf{R}$, correspond à la somme des valeurs sur leurs colonnes (sans compter ceux de la file $\alpha$).

**Définition 3.5.2** *(Contribution n-aire de* $C_\alpha$*)*
*Étant donné un CSP* $P = (V, D, \zeta)$, *on dira que la Contribution n-aire de* $C_\alpha$ *à la fonction d'évaluation* $c_n(C_\alpha)$ *sera:*

$$\mathbf{c_n}(C_\alpha) = \mathbf{e_n}(C_\alpha, I_j), \text{ quand } C_\alpha \text{ est violée sous } I_j.$$

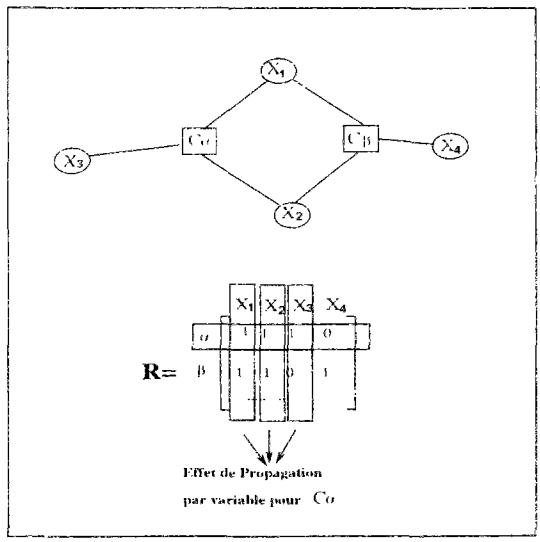



FIG. 3.24 – *Effet de Propagation pour CSP n-aire*

Finalement la fonction d'évaluation pour un CSP n-aire est définie par:

**Définition 3.5.3** *(Fonction d'Évaluation pour CSP n-aire)*
*Étant donné un CSP avec une matrice de contraintes* $\mathbf{R}$*, une instanciation* $\mathbf{I}$ *et la fonction d'Évaluation de l'erreur n-aire* $\mathbf{e_n}(C_\alpha, \mathbf{I})$ *pour chaque contrainte* $C_\alpha, (\alpha = 1, \ldots, \eta)$*, la Fonction d'Évaluation n-aire* $\mathbf{Z_n}(\mathbf{I})$ *est définie par:*

$$\mathbf{Z_n}(\mathbf{I}) = \sum_{\alpha=1}^{\eta} \mathbf{e_n}(C_\alpha, \mathbf{I}) \tag{3.8}$$

Dans le cas où nous avons un CSP binaire, cette fonction est équivalente à la fonction de la définition 3.2.4. De la même manière, nous avons une préférence pour les chromosomes dont les valeurs des variables satisfont plus d'arêtes et de liaisons. Il reste à tester cette fonction d'évaluation.

## 3.6 Conclusion du chapitre

Nous avons défini une nouvelle fonction d'évaluation $Z(\mathbf{I})$, en prenant en compte la structure du graphe de contraintes, ce qui permet de mieux guider la recherche

stochastique que ne le fait la *fonction d'évaluation standard*. Nous avons proposé un nouvel algorithme du type escalade en deux étapes, qui combine *min-conflits* et *min-réseau-conflits*, heuristiques basées sur les fonctions d'évaluation $Z_{std}(\mathbf{I})$ et $Z(\mathbf{I})$. Cet algorithme a obtenu un pourcentage de convergence vers la solution plus élevé qu'avec *min-conflits* seul, sur des exemples.

Nous avons aussi proposé un algorithme évolutionniste, guidé par $Z(\mathbf{I})$ et avec une nouvelle méthode de sélection qui a fourni de bons résultats pour le problème de 3-coloriage.

L'opérateur standard de *croisement à un point* rend l'algorithme évolutionniste influençable par l'ordre dans lequel se trouvent les variables dans le chromosome. C'est cela qui nous a motivée à proposer un nouvel opérateur, activé seulement lorsque l'algorithme n'arrive pas à trouver une meilleure solution après un certain nombre d'itérations. Ceci a permis d'améliorer les résultats de l'algorithme de 20%.

La fonction d'évaluation $Z(\mathbf{I})$ définie dans ce chapitre est bien adaptée aux problèmes où le graphe est important, par exemple pour le 3-coloriage où toutes les contraintes sont les mêmes, et aussi pour les CSP aléatoires avec toutes les contraintes ayant la même difficulté.

Pour résoudre des CSP non uniformes, il faudrait probablement définir une fonction d'évaluation qui prenne en compte aussi la difficulté des contraintes[2].

Maintenant, notre but est d'incorporer la notion de structure dans la conception de nouveaux opérateurs génétiques, qui permettraient une amélioration de la performance de l'algorithme évolutionniste et qui de plus seraient indépendants de l'ordre des variables dans la représentation.

Dans le chapitre suivant, nous définirons deux nouveaux opérateurs qui prennent en compte cette conclusion et qui sont conçus pour résoudre des problèmes de satisfaction de contraintes généraux. Les travaux exposés dans ce chapitre ont été présentés dans les références [Rif96a], [Rif96b].

2. en anglais:constraint tightness