Optimisation par algorithme génétique sous contraintes

Nicolas Barnier — Pascal Brisset

École Nationale de l'Aviation Civile
7, avenue Édouard Belin
31055 Toulouse Cedex 4
{barnier, brisset}@recherche.enac.fr

RÉSUMÉ. *Dans le cadre de la programmation logique par contraintes sur les domaines finis CLP($\mathcal{FD}$), nous proposons une nouvelle méthode d'optimisation reposant sur un algorithme génétique. L'idée de base est de faire manipuler par l'algorithme génétique des sous-domaines des variables du CSP. La population de l'algorithme génétique est ainsi constituée de chaînes de sous-domaines dont l'adaptation est calculée par la résolution des « sous-CSP » correspondants qui sont plus simples que le problème original. Nous présentons des opérateurs de croisement et de mutation basiques puis spécifiques, dotés de divers degrés de robustesse. Les premières expérimentations de la méthode ont été menées sur des formulations CSP naïves du problème de tournées (VRP) et d'affectation de fréquences radio (RLFAP). Les résultats sont encourageants et la méthode est plus efficace que les techniques CLP($\mathcal{FD}$) ou les algorithmes génétiques seuls sur ces problèmes.*

ABSTRACT. *Within the framework of Constraint Logic Programming on finite domains CLP($\mathcal{FD}$), we introduce a new optimization method based on a Genetic Algorithm. The main idea is the handling of sub-domains of the CSP variables by the genetic algorithm. The population of the genetic algorithm is made up of strings of sub-domains whose fitness are computed through the resolution of the corresponding "sub-CSPs" which are somehow much easier than the original problem. We provide basic and dedicated recombination and mutation operators with various degrees of robustness. The first set of experimentations adresses a naïve formulation of the Vehicle Routing Problem (VRP) and the Radio Link Frequency Assignement Problem (RLFAP). The results are quite encouraging as we outperform CLP($\mathcal{FD}$) techniques and genetic algorithm alone.*

MOTS-CLÉS : *Optimisation, algorithme génétique, CSP, méthode mixte.*

KEY WORDS : *Optimisation, Genetic Algorithm, CSP, Hybridization.*

1. Introduction

La résolution d'un problème d'optimisation consiste à explorer un espace de recherche afin de maximiser (ou minimiser) une fonction donnée. Les complexités (en taille ou en structure) relatives de l'espace de recherche et de la fonction à maximiser conduisent à utiliser des méthodes de résolutions radicalement différentes. En première approximation, on peut dire qu'une méthode déterministe est adaptée à un espace de recherche petit et complexe et qu'un espace de recherche grand nécessite plutôt une méthode de recherche stochastique (recuit simulé, algorithme génétique, ...).

Dans la plupart des cas, un problème d'optimisation se divise naturellement en deux phases : recherche des solutions admissibles puis recherche de la solution à coût optimal parmi ces dernières. Suivant la méthode employée, ce découpage est plus ou moins apparent dans la résolution.

L'usage d'un algorithme génétique [GOL 89] est adapté à une exploration rapide et globale d'un espace de recherche de taille importante et est capable de fournir plusieurs solutions. Dans le cas où l'ensemble des solutions admissibles est complexe (i.e. il est difficile d'isoler une solution admissible), l'admissibilité peut être rendue intrinsèque à la représentation choisie ou intégrée à la génération des chromosomes (mutation, croisement) ou à la fonction à optimiser (on attribue une mauvaise adaptation à une solution non admissible).

L'utilisation d'une technique de satisfaction de contraintes (CSP) est adaptée aux problèmes très contraints où une exploration exhaustive de l'espace de recherche est envisageable. La méthode fournit naturellement des solutions admissibles. En ajoutant une contrainte (dynamique) portant sur le coût d'une solution, la résolution peut produire une solution optimale (c'est le prédicat `minimize` de CHIP [DIN 88]). Cette méthode garantit l'optimalité de la solution.

Mais il n'existe évidemment pas de dichotomie simple au sein de l'ensemble des problèmes d'optimisation : de nombreux problèmes sont fortement contraints et possèdent un vaste espace de recherche. Ces deux caractéristiques excluent l'usage direct d'un algorithme génétique ou d'un CSP.

Nous proposons de profiter des avantages des deux approches en les hybridant : nous utilisons un CSP pour calculer des solutions admissibles sur un sous-espace de l'espace de recherche et l'intégrons à un algorithme génétique explorant cet espace.

Nous décrivons dans cet article une méthode générique pour réaliser cette hybridation[1] pour un CSP quelconque. Dans un premier temps, nous rappelons brièvement ce que sont CSP et algorithme génétique. Dans la seconde partie, nous décrivons l'algorithme génétique : fabrication des sous-espaces, opérations sur ces derniers et évaluation. Nous concluons par l'analyse des résultats obtenus sur des problèmes classiques et comparons notre méthode avec les approches similaires.

[1] Le projet à été baptisé GASCON pour *Genetic AlgorithmS + CONstraint programming*.

2. Contexte

Nous présentons ici les deux techniques d'optimisation dont nous proposons l'hybridation dans la section suivante.

2.1. *Problème de satisfaction de contraintes (CSP)*

Nous considérons ici un CSP défini par un triplet (X, D, C) où X est un ensemble de n variables (X_1, X_2, ..., X_n) , D leurs domaines finis respectifs ($D(X_1)$, $D(X_2)$, ...,$D(X_n)$) et C un ensemble de relations ou *contraintes* entre ces variables (une contrainte portant sur $X_{i_1}, ..., X_{i_k}$ est un sous-ensemble du produit cartésien $D(X_{i_1}) \times ... \times D(X_{i_k})$). Pour un problème d'optimisation (ici de maximisation d'une fonction entière), on considère en outre une fonction de coût f et une contrainte sur ce coût $f(X_1, X_2, ...) > c$ où c est une constante que la stratégie d'optimisation fait évoluer.

Nous exprimons notre problème en programmation logique avec contraintes (CLP) [VAN 89] et utilisons le système ECL^{i}PS^e [ECL 92] qui implémente toutes les contraintes classiques, linéaires (`#=`, `#>`, ...) et autres (`alldistinct`, `element`, ...), et permet également d'en définir simplement de nouvelles (opérations directes sur les domaines, contrôle précis du *coroutining*, ...). Le prédicat `min_max` (`minimize`, `maximize`, ...) permet d'optimiser une expression linéaire en intégrant le but de résolution du problème (en général l'instantiation des variables, *labeling*) au sein d'un *Branch & Bound* (i.e. parcours de l'arbre de recherche avec élagage par limitation de la fonction de coût).

2.2. *Algorithmes génétiques*

Les algorithmes génétiques tentent de simuler le processus d'évolution naturelle suivant le modèle darwinien dans un environnement donné. Ils utilisent un vocabulaire similaire à celui de la génétique naturelle. Cependant, les processus naturels auxquels ils font référence sont beaucoup plus complexes. On parlera ainsi d'individu dans une population. L'individu est représenté par un chromosome constitué de gènes qui contiennent les caractères héréditaires de l'individu. Les principes de *sélection*, de *croisement*, de *mutation* s'inspirent des processus naturels de même nom.

Pour un problème d'optimisation donné, un individu représente un point de l'espace d'états, une solution potentielle. On lui associe la valeur du critère à optimiser, son *adaptation*. On génère ensuite de façon itérative (figure 1) des populations d'individus sur lesquelles on applique des processus de sélection, de croisement et de mutation. La sélection a pour but de favoriser les meilleurs éléments de la population pour le critère considéré (les mieux *adaptés*), le croisement et la mutation assurent l'exploration de l'espace d'états.

On commence par générer une population aléatoire d'individus. Pour passer d'une

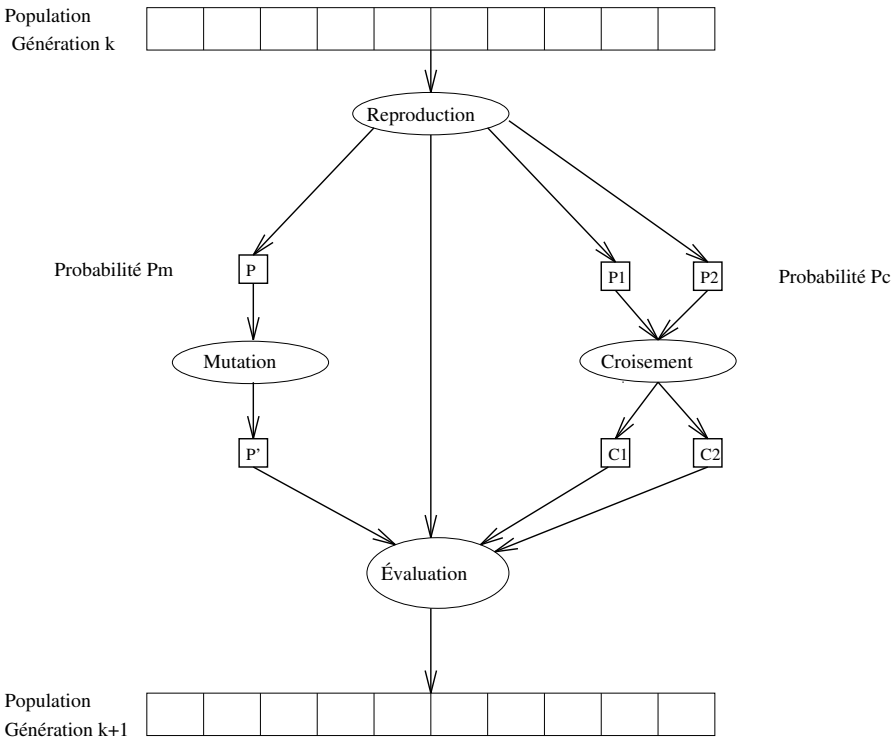


Figure 1. *Principe général des algorithmes génétiques*

génération k à la génération $k+1$, les opérations suivantes sont effectuées. Dans un premier temps, la population est reproduite par *sélection* où les bons individus se reproduisent mieux que les mauvais. Ensuite, on applique un *croisement* aux paires d'individus (les parents) d'une certaine proportion de la population (probabilité P_c, généralement autour de 0.6) pour en produire des nouveaux (les enfants). Un opérateur de *mutation* est également appliqué à une certaine proportion de la population (probabilité P_m, généralement très inférieure à P_c). Enfin, les nouveaux individus sont évalués et intégrés à la population de la génération suivante[2].

Plusieurs critères d'arrêt de l'algorithme sont possibles : le nombre de générations peut être fixé *a priori* (temps constant) ou l'algorithme peut être arrêté lorsque la population n'évolue plus suffisamment rapidement.

Pour utiliser un algorithme génétique sur un problème d'optimisation on doit donc disposer d'un principe de codage des individus, d'un mécanisme de génération de la population initiale et d'opérateurs permettant de diversifier la population au cours des générations et d'explorer l'espace de recherche.

[2]Classiquement, on fait en sorte de conserver une population de taille constante.

3. Une approche mixte

Nous présentons dans cette section les composants d'un algorithme génétique générique pour l'optimisation d'un CSP portant sur des variables à domaines finis.

L'idée est illustrée par la figure 2 (dans le cas d'un problème à deux variables continues) : les zones grisées sont les individus de l'algorithme génétique qui correspondent chacun à un sous-espace ; pour chaque sous-espace est calculée une solution à l'aide du CSP. On peut noter qu'à un individu ne correspond pas nécessairement une solution et deux individus distincts peuvent correspondre à une même solution. La rapport de la taille d'un sous-espace à la taille totale de l'espace de recherche (noté ρ par la suite) est le paramètre essentiel de l'hybridation : on peut passer continûment d'une résolution purement CSP ($\rho = 1$) à une résolution purement stochastique ($\rho = 0$, un sous-espace est réduit à une valeur).

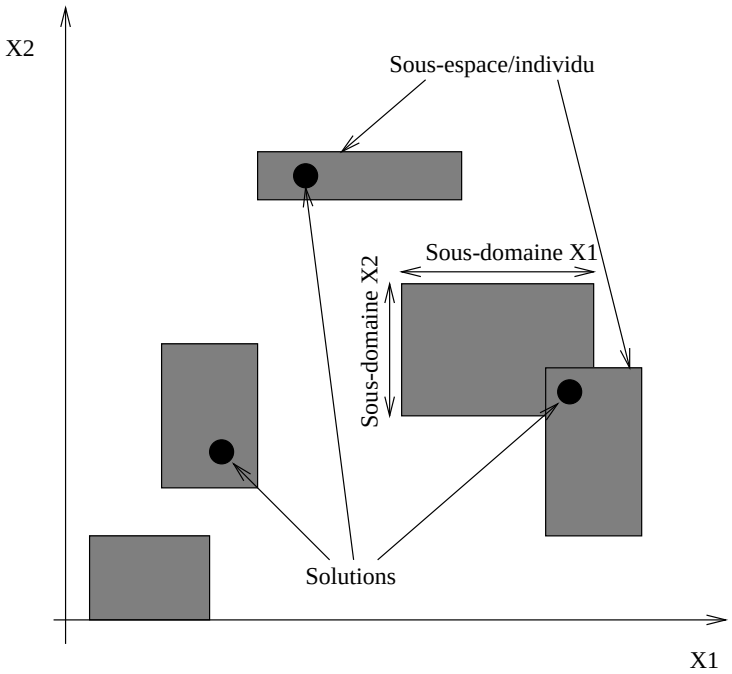


Figure 2. *Un espace de recherche à deux dimensions*

3.1. *Chromosomes*

Considérons un CSP à n variables X_1, X_2, ..., X_n et, selon les notations de la section 2.1, leurs domaines finis respectifs $D(X_1)$, $D(X_2)$, ..., $D(X_n)$.

Un individu de l'algorithme génétique est représenté par un chromosome consti-

tué d'une chaîne de gènes $G_1G_2...G_n$. Chaque gène de l'individu est associé à une variable du CSP et l'individu représente à la fois un sous-problème et éventuellement une solution. Le gène G_i ($\forall i \in [1..n]$) correspondant à une variable X_i est un sous-domaine, *i.e.* un sous-ensemble, de $D(X_i)$ et on note $|G_i|$ son cardinal. Le paramètre essentiel de l'algorithme ρ ($0 \leq \rho \leq 1$) est défini par le rapport des cardinaux respectifs du sous-espace G_i et du domaine $D(X_i)$:

$$\rho = \frac{|G_i|}{|D(X_i)|} \quad [1]$$

On peut passer de façon continue d'un algorithme génétique pur en choisissant ρ petit tel que tout les G_i ne contiennent qu'une seule valeur, à un CSP pur en prenant $\rho = 1$, c'est-à-dire $G_i = D(X_i)$. Pour des raisons pratiques, le cardinal de G_i est toutefois calculé dans l'algorithme par la formule suivante :

$$|G_i| = \max\{1, \text{round}(\rho\,|D(X_i)|)\} \quad [2]$$

où round(x) est l'entier le plus proche de x. En effet, si on choisit ρ tel que

$$\rho < \frac{1}{\min\limits_{i\in[1..n]} |D(X_i)|}$$

alors G_i est vide ($\forall i \in [1..n]$) selon la définition 1, ce qui présente peu d'intérêt. Avec la formule 2, chaque gène d'un individu contient au moins un élément du domaine associé et on garde la propriété de passage « continu » de l'algorithme génétique pur au CSP pur :

$$(\rho = 0 \implies |G_i| = 1) \wedge (\rho = 1 \implies G_i = D(X_i))$$

Il peut être utile d'avoir des valeurs de ρ distinctes pour chaque variable si les tailles initiales des domaines sont très différentes [3]. Il faut alors choisir n valeurs ρ_i qui s'appliqueront individuellement à chaque détermination de cardinal d'un gène. De même, rien n'indique *a priori* qu'il faille garder le paramètre d'hybridation ρ constant au cours des générations de l'algorithme génétique. Cependant, pour des raisons de clarté et de simplicité du modèle d'hybridation, nous supposerons ρ (ou les ρ_i) constant(s) tout au long de l'algorithme. Nous verrons ultérieurement que cet invariant sera utile pour l'élaboration des opérateurs de l'algorithme génétique.

4. Initialisation de la population

C'est lors de l'initialisation de l'algorithme génétique que les individus originaux sont générés. Avant cette étape, les domaines des variables sont déjà substantiellement réduits (si la nature et le nombre des contraintes s'y prêtent) par la propagation

[3] Si le problème comporte à la fois des variables booléennes et des variables entières par exemple, il peut être plus efficace de choisir $\rho = 0$ ou $\rho = 1$ pour les booléens indépendamment de la valeur de ρ pour les autres variables.

des contraintes posées qui assure l'arc-consistance (ou une approximation de l'arc-consistance) du problème. Puis pour chaque gène de chaque chromosome, un sous-domaine aléatoire (de la taille adéquate) du domaine de la variable correspondante est construit. Traditionnellement, les individus de la « soupe primitive » d'un algorithme génétique sont uniformément répartis sur l'espace de recherche. Pour notre algorithme hybride, l'espace de recherche est défini par le produit cartésien de parties de taille fixe (déterminée par le paramètre ρ) des domaines de chaque variable.

4.1. *Valuation des individus*

L'adaptation d'un individu est calculée lors de la résolution du CSP limité aux sous-domaines générés pour chaque variable : pour l'individu $G_1G_2...G_n$, les contraintes $X_i \in G_i$ sont ajoutées ; la résolution est alors réalisée de façon standard par étiquetage des variables. Si une solution est trouvée, l'adaptation est calculée par simple application de la fonction de coût aux valeurs des variables instanciées ($f(X_1, ..., X_n)$). Si aucune solution n'est trouvée (*i.e.* le sous-espace $G_1G_2...G_n$ n'en contient pas), on attribue une adaptation nulle à l'individu (ceci peut être raffiné pour chaque problème particulier). On choisit de ne pas chercher la meilleure solution dans le sous-espace, ce qui signifie que la partie « optimisation » de l'algorithme hybride est entièrement laissée à l'algorithme génétique[4]. Notons que les performances de l'algorithme peuvent être affectées par la qualité de la solution trouvée qui dépend directement de la stratégie de recherche dans le sous-espace; un individu peut être considéré comme mauvais alors qu'il contient une bonne solution. Cependant, cette potentielle sous-évaluation ne remet pas en cause le fonctionnement de l'algorithme génétique.

En pratique, prouver qu'un sous-espace ne contient pas de solution par la résolution du CSP associé peut être très coûteux en temps de calcul. De même, les solutions d'un sous-espace peuvent être toutes très éloignées dans l'arbre de recherche induit par l'étiquetage choisi, et déterminer une solution devient alors également très coûteux en temps de calcul. Or le temps mis par l'évaluation d'un individu est une donnée critique pour un algorithme génétique. C'est pourquoi il est indispensable de borner le temps passé à effectuer l'évaluation d'un individu. On rajoute alors un nouveau paramètre τ (*timeout*) à notre algorithme, qui désigne le temps imparti à l'étiquetage pour fournir une solution. À l'issue de ce délai et si aucune solution n'a été trouvée, on peut alors procéder de manière analogue au traitement des individus qui ne contiennent pas de solution (adaptation nulle ou évaluation ad hoc au problème).

[4]Dans notre implémentation, l'évaluation d'un individu est une procédure « utilisateur » qui peut donc choisir de réaliser une optimisation locale s'il en existe une pour le problème considéré.

4.2. *Opérateurs*

Nous présentons dans cette section des opérateurs de croisement et mutation adaptés à la manipulation de chromosomes constitués de sous-domaines. Nous proposons d'abord des opérateurs inspirés plus ou moins directement des opérateurs classiques utilisés au sein des algorithmes génétiques, puis de nouveaux opérateurs ensemblistes sur les gènes. Enfin, nous présentons quelques opérateurs qui intègrent des heuristiques en utilisant la valuation et/ou la fonction de coût.

4.2.1. *Opérateurs classiques*

Les opérateurs classiques pour les chaînes de bits [GOL 89] peuvent être utilisés avec notre codage (tableau 1).

Mutation

L'opérateur de mutation classique prend en entrée un individu P sélectionné pour la mutation et renvoie un individu mutant P' obtenu par transformation locale de l'un des gènes de P. Par exemple, si le chromosome d'un individu est codé avec une chaîne de bits, on peut le muter en complémentant l'un de ses gènes/bits.

Dans le cas présent, un gène est codé par un sous-domaine des valeurs possibles de la variable correspondante ; par analogie, la mutation d'un individu consiste à remplacer l'un de ses gènes/sous-domaines par un autre sous-domaine choisi aléatoirement. Un gène est ainsi capable par mutation seule de parcourir exhaustivement son espace de recherche, c'est-à-dire l'ensemble des parties de cardinal k du domaine de la variable associée (si on suppose que le sous-domaine de chaque gène a un cardinal k).

On aurait pu également interpréter de manière plus sémantique l'opérateur de mutation en rapprochant bit *complémenté* et *complémentaire* d'un ensemble. En effet, si on considère notre algorithme appliqué à un problème où les variables sont booléennes, on obtient exactement le même opérateur pour $\rho = 1/2$. Cependant, nous souhaitons garder constante la taille des sous-domaines générés, ce qui imposerait systématiquement $\rho = 1/2$, une restriction jugée trop forte pour notre approche.

Croisement

L'opérateur de croisement classique prend en entrée un couple d'individus parents P_1 et P_2 et renvoie un couple d'individus enfants C_1 et C_2 obtenus en choisissant aléatoirement un point de croisement (ou éventuellement plusieurs points de croisement pour éviter certains effets de bord du codage) dans les chromosomes et en recopiant dans le fils C_1 les gènes de P_1 jusqu'au point de croisement puis en complétant avec les gènes de P_2. On effectue l'opération symétrique pour C_2.

Avec notre codage, cette méthode de croisement est utilisable directement : le croisement explore ainsi l'espace des solutions en essayant de mélanger deux sous-espaces différents représentés par les individus parents.

Ce croisement a un sens uniquement avec les hypothèses classiques de séparabilité

X_1	X_2	X_3	Variables
1..7	1..9	1..5	Domaines initiaux
1,2,3	4,7,8	1,3,4	Individu P_1
1,5,7	1,4,8	1,2,5	Individu P_2
1,2,3	1,7,8	1,3,4	Individu P_1 muté
1,2,3	4,7,8	1,2,5	1^{er}fils classique de P_1 et P_2
1,5,7	1,4,8	1,3,4	$2^{\text{ème}}$fils classique de P_1 et P_2
1,2,3,5,7	1,4,7,8	1,2,3,4,5	Union de P_1 et P_2
2,3,5	1,4,8	2,3,4	1^{er}fils ensembliste de P_1 et P_2
1,7	7	1,5	Complémentaire du 1^{er}fils
1,7,5	7,4,8	1,5,2	$2^{\text{ème}}$fils ensembliste de P_1 et P_2

Figure 2. *Opérateurs classiques et ensemblistes*

des variables au sein des contraintes et de la fonction de coût. Une analyse de connexité du graphe des contraintes permettrait de choisir a priori un « bon » point de croisement (ou plusieurs).

4.2.2. *Opérateurs ensemblistes*

Les opérateurs triviaux calqués sur ceux utilisés avec les chaînes de bits ne sont pas très efficaces en général [GOL 89] et il est préférable d'utiliser des opérateurs plus sémantiques, c'est-à-dire en rapport avec le codage des chromosomes. Par exemple dans le cas d'un codage par une chaîne de réels, la mutation consiste à ajouter un bruit gaussien à l'un des gènes et le croisement est une combinaison linéaire des deux parents (un chromosome est alors vu comme un vecteur). Pour notre codage, il est également naturel d'envisager des opérateurs travaillant sur les gènes.

Les gènes étant des ensembles finis, ils sont candidats aux opérations ensemblistes telles que l'union, l'intersection, le complémentaire... Mais si l'on fixe la taille des sous-domaines, i.e. leur cardinal, l'union et l'intersection seules de deux sous-domaines ne produisent pas des gènes de taille adéquate.

Croisement

Nous proposons un nouveau croisement dont le principe est illustré également sur le tableau 1. On suppose que des gènes situés sur le même locus (position dans le chromosome) pour deux individus distincts ont toujours la même taille ; le croisement s'effectue alors de la manière suivante :

1. pour chaque locus i ($i \in [1..n]$), on réalise d'abord l'union $G_{i_{Père}}^{\cup} = G_{i_{Père}}^{1} \cup G_{i_{Père}}^{2}$

des sous-domaines des pères $G_{i_{Père}}^{1}$ et $G_{i_{Père}}^{2}$;

2. puis on choisit aléatoirement une partie de $G_{i_{Père}}^{\cup}$ de taille appropriée (selon la valeur de ρ) qui constitue le $i^{\text{ème}}$ gène $G_{i_{Fils}}^{1}$ du premier fils ;

3. on prend ensuite les valeurs restantes de l'union des gènes des pères, *i.e.* $G_{i_{Père}}^{\cup} - G_{i_{Fils}}^{1}$ pour constituer la première partie des gènes du deuxième fils $G_{i_{Fils}}^{2}$;

4. enfin, si l'intersection des gènes des pères n'est pas vide, *i.e.* $G_{i_{Père}}^{1} \cap G_{i_{Père}}^{2} \neq \emptyset$, on complète $G_{i_{Fils}}^{2}$ avec une partie de taille adéquate choisie aléatoirement dans $G_{i_{Fils}}^{1}$.

On peut montrer que cet opérateur de croisement vérifie les « bonnes » propriétés nécessaire à la convergence stochastique de l'algorithme [CER 94] :

— lorsque deux individus identiques s'accouplent, ils engendrent des individus identiques à eux-mêmes ;

— la probabilité que rien ne se passe pendant un croisement est toujours non nulle ;

— les deux individus qui s'accouplent jouent des rôles identiques (la population est asexuée).

4.2.3. *Opérateurs guidés par la fonction de coût*

Une heuristique peut parfois être déduite de la fonction de coût et intégrée aux opérateurs de l'algorithme génétique. Si la fonction de coût est croissante avec certaines variables du problème, par exemple, on peut essayer d'améliorer la qualité des solutions en appliquant un opérateur de mutation qui fait croître la moyenne des valeurs des sous-domaines correspondants.

On peut ainsi concevoir des opérateurs de croisement et de mutation sophistiqués pour beaucoup de problèmes, mais c'est au détriment de la robustesse de l'algorithme. D'autre part des opérateurs trop déterministes peuvent limiter l'exploration de l'espace de recherche en guidant les individus trop systématiquement vers une même région éloignée des optima globaux.

4.2.4. *Opérateurs guidés par la valuation*

L'opérateur de croisement ensembliste conserve les valeurs des sous-domaines des gènes parents sans tenir compte de la solution effective calculée par la résolution du CSP (une seule valeur pour chaque sous-domaine). On peut éventuellement accroître l'efficacité des opérateurs en conservant ces valeurs « héréditairement » : on peut appliquer aux individus le croisement ensembliste en insérant en priorité dans les sous-domaines des enfants les valeurs solutions des gènes des deux pères et compléter les gènes des fils de la même manière qu'avec le croisement ensembliste décrit précédemment. On peut appliquer la même idée à l'opérateur de mutation classique : on fabrique les gènes mutants en y insérant d'abord la valeur solution correspondante du père puis en complétant aléatoirement.

La même réserve que celle de la section précédente 4.2.3 s'impose pour ce type d'opérateurs : l'exploration peut se trouver limitée par trop de déterminisme. Par exemple lorsque les sous-domaines sont de petite taille (3 ou 4 éléments), ces opéra-

teurs conservateurs produisent des individus proches des individus initiaux et l'exploration de l'espace de recherche risque d'en être handicapée.

4.3. *Le prédicat* `ga_minimize`

L'hybridation proposée est générique : aucune hypothèse n'est faite sur le problème traité. Le processus d'optimisation nécessite uniquement un ensemble de variables à domaine fini et une procédure (c.-à-d. en Prolog un but) effectuant la résolution du problème CSP. L'algorithme d'optimisation peut-être donc présenté à l'utilisateur sous la forme d'un prédicat analogue au `minimize` de CHIP :

`ga_minimize(` *Goal* `,` *Variables* `,` *Eval* `,` *Rho*`)`

où *Goal* est le but de résolution, *Variables* la liste des variables à domaine fini définissant l'espace de recherche de l'algorithme génétique, *Eval* l'évaluation de la solution calculée par *Goal* et *Rho* notre paramètre d'hybridation. De même que pour le `minimize` classique, le *Goal* est en général simplement le *labeling* des variables.

4.3.1. *Implémentation*

L'originalité de l'implémentation de l'algorithme génétique réside uniquement dans l'évaluation des éléments de population : un individu étant constitué de sous-domaines, le domaine de chaque variable est restreint avant que le but *Goal* soit résolu. En cas de succès du but, *Eval* vaut l'évaluation de l'individu et en cas d'échec, l'évaluation de l'individu est nulle (cela peut être raffiné pour chaque problème particulier par un calcul ad hoc).

On peut noter qu'il est facile de distribuer ces calculs sur un réseau de processeurs : une machine *maître* exécute l'algorithme génétique (initialisation, opérateurs, ...) et sous-traite l'évaluation des individus à des machines *esclaves* qui implémentent uniquement le CSP[5].

4.3.2. *Paramétrage et utilisation*

L'une des difficultés d'utilisation d'un algorithme génétique réside dans le choix des nombreux paramètres qui le contrôlent : nombre d'individus dans la population, nombre de génération et/ou critère d'arrêt, probabilités de croisement et mutation. Dans notre cas s'ajoute à tous ces paramètres classiques le paramètre ρ qui spécifie la taille relative des sous-domaines et le paramètre de *timeout* τ. Dans notre implémentation, tous ces paramètres ont des valeurs par défaut et peuvent être modifiés pendant la phase d'initialisation.

---

[5]Un autre type de parallélisation *par îlots* inspiré de la programmation multi-agents permet également d'améliorer les performances d'un algorithmes génétique, mais en en changeant le comportement.

5. Autres approches

D'autres approches d'hybridation entre programmation par contraintes et algorithme génétique ont été expérimentées :

— Sur des problèmes d'optimisation combinatoire, [KUE 93] propose d'intégrer la CLP au sein de l'initialisation de la population et de l'opérateur de croisement de l'algorithme génétique pour ne générer que des individus admissibles. L'opérateur de croisement reprend le principe du *slicing crossover* : la première partie du chromosome de l'un des pères est recopiée dans le chromosome du fils, puis celui-ci est complété avec les valeurs de la partie du second père qui ne violent pas de contraintes ; les valeurs restantes sont déterminées par un étiquetage classique. De bons résultats sont présentés pour des problèmes de TSP contraints avec un opérateur de mutation spécifique effectuant une optimisation locale.

— Dans le même esprit, [BUR 95] intègre aux opérateurs de l'algorithme génétique une plate-forme fondée sur des techniques de satisfaction de contraintes par coloration de graphes développée pour la résolution de problèmes d'emploi du temps.

— Dans un tout autre style, [KOK 96] utilise le solveur de contraintes sur les rationnels du système ECL^{i}PS^e et l'algorithme génétique GENOCOP [6] [MIC 92] dans le système CoCo (COmputational intelligence plus COnstraint logic programming) pour effectuer l'apprentissage de réseaux de neurones contraints. Les contraintes inhérentes à la topologie et au modèle du réseau de neurones sont exprimée en CLP, ainsi que celles qui assurent l'unicité de la représentation ; les contraintes subséquemment générées par le système ECL^{i}PS^e et la fonction de coût sont ensuite fournies à GENOCOP qui se charge de l'optimisation.

Cette dernière solution propose un couplage très « lâche » de la programmation par contraintes et des algorithmes génétiques. Notre approche est plus directement comparable à celles de [KUE 93] et [BUR 95] qui font coopérer « intimement » les deux types de solveurs, bien que ces méthodes restent assez dépendantes de leur domaine d'application car elles utilisent des opérateurs très spécifiques.

Des hybridations impliquant soit la programmation par contraintes, soit les algorithmes génétiques, et un autre type de solveur ont également été introduites pour résoudre des problèmes d'optimisation présentant des aspects combinatoires :

— [MéD 94] hybride un algorithme génétique et un Simplexe pour traiter des problèmes de trafic aérien (la résolution de conflits). Les dimensions combinatoires du problème sont prises en charge par l'algorithme génétique, et l'évaluation d'un individu par un Simplexe résout le problème d'optimisation linéaire (et continu) associé ; si ce problème est inconsistant, les contraintes incriminées sont relaxées et une pénalité est infligée dans l'adaptation.

— D'une manière analogue à [KOK 96], le couplage assez faible de la CLP sur les domaines finis et d'un Simplexe muni d'un *Branch & Bound* est présenté dans

[6] GEnetic algorithm for Numerical Optimization for COnstrained Problems : il s'agit d'un algorithme génétique développé par Zbigniew MICHALEWICZ destiné à résoudre des problèmes d'optimisation sujets à des contraintes linéaires.

[HAJ 95] sur un problème d'affectation pour la flotte de British Airways. Le système ECL^{i}PS^e est utilisé pour générer une première solution qui sert de point de départ au Simplexe. Le Simplexe optimise alors la solution en relaxant les contraintes d'intégrité ; si l'optimum satisfait ces dernières, le problème est résolu, sinon le Simplexe est intégré à un *Branch & Bound*.

— [PES 96] utilise la programmation par contraintes au sein d'un algorithme de recherche locale ; l'idée est de spécifier le voisinage d'une solution, au sein duquel est effectué le mouvement local, en CSP et d'utiliser une recherche déterministe pour déterminer un voisin admissible, éventuellement le meilleur des voisins. Cette méthode est donc à rapprocher de celles citées précédemment utilisant un CSP au sein des opérateurs d'un algorithme génétique.

6. Applications

Nous avons expérimenté notre algorithme hybride sur des problèmes d'optimisation combinatoire très contraints possédant un vaste espace de recherche afin de valider notre approche. Nous avons ainsi exprimé en CSP de manière très directe (naïve) deux problèmes issus de situations réelles : le problème de tournées (VRP) et le problème d'affectation de fréquences radio (RLFAP). On pourra se reporter à [BAR 97] pour une analyse plus précise des résultats obtenus.

L'objet de ces expérimentations n'est pas d'évaluer la méthode dans l'absolu mais uniquement relativement aux approches GA et CSP pures.

6.1. *Problème de tournées (VRP)*

Le problème de tournées (VRP, *Vehicle Routing Problem*) est en quelque sorte un mélange de problème de voyageur de commerce (TSP, *Traveling Salesman Problem*) et de problème d'ordonnancement (*Scheduling Problem*).

Des tâches doivent être effectuées dans des sites distincts (repérés par leurs coordonnées) et dans des intervalles de temps fixés (*time window*). Chaque tâche peut être exécutée par un certain nombre d'ingénieurs qualifiés. De plus, certaines tâches doivent être réalisées avant ou en même temps que d'autres (ordonnancement). Les ingénieurs partent d'une base repérée par ses coordonnées et voyagent à une certaine vitesse pour rejoindre le lieu des travaux. Le problème est de trouver pour chaque tâche l'ingénieur qui va l'effectuer et sa date de début en minimisant le temps de voyage et d'attente des ingénieurs

6.1.1. *Formulation*

Nous avons formulé ce problème en CSP directement de la façon suivante :

— À chaque tâche i sont associées deux variables, l'une correspondant à l'ingénieur réalisant cette tâche Eng_i et l'autre correspondant à l'heure à laquelle la tâche

sera effectivement réalisée T_i. Les domaines de ces variables sont fixés : ingénieurs qualifiés pour une tâche $E_i \in \mathcal{E}_i$ et *time window* pour la date de début d'une tâche $T_i \in [TW_i^{start}..TW_i^{end}]$.

— Les contraintes d'ordonnancement de précédence et de synchronisation (inégalités et égalités) sont exprimées sur les T_i.

— Pour chaque paire de tâche (i, j), nous posons la contrainte suivante :

$$M_{ij} \text{ si } E_i == E_j \quad [3]$$

$$\text{et} \quad T_i + d_i + t_{ij} \leq T_j + \infty B_{ij}^1 \quad [4]$$

$$\text{et} \quad T_j + d_j + t_{ij} \leq T_i + \infty B_{ij}^2 \quad [5]$$

$$\text{et} \quad M_{ij} + B_{ij}^1 + B_{ij}^2 = 2 \quad [6]$$

où M_{ij} est une variable booléenne vraie (valeur 1, 0 sinon) si les deux tâches sont effectuées par le même ingénieur (contrainte 3), d_i (resp. d_j) est la durée de la tâche i (resp. j), t_{ij} est la durée de parcours entre les sites de chaque tâche, B_{ij}^1 et B_{ij}^2 sont des booléens permettant l'expression de la contrainte disjonctive (ordre des tâches) par une conjonction (contraintes 4 et 5). La contrainte 6 exprime que soit les tâches ne sont pas effectuées par le même ingénieur ($M_{ij} = 0, B_{ij}^1 = B_{ij}^2 = 1$), soit une (et une seule) des contraintes 4 et 5 est vérifiée ($M_{ij} = 1, B_1 + B_2 = 1$).

— Le coût d'une solution correspond à la somme des temps de trajet des ingénieurs et à la date de la dernière tâche effectuée.

6.1.2. *Étiquetage et opérateurs*

L'étiquetage choisi est complètement standard : on commence par étiqueter les dates des tâches T_i, puis les ingénieurs E_i et enfin les variables auxiliaires introduites pour rendre compte du fait qu'un ingénieur ne peut effectuer deux travaux en même temps M_{ij}, B_{ij}^1 et B_{ij}^2.

Les deux opérateurs de croisement classique et ensembliste ont été testés sur le problème de VRP, ainsi que la mutation classique qui change aléatoirement le sous-domaine d'un gène.

6.1.3. *Résultats*

Nous donnons ici quelques résultats qui montrent l'influence des différents paramètres de l'algorithme sur ses performances. Nous donnons également des éléments de comparaison avec les méthodes pures, AG et CSP.

La figure 3 illustre l'influence du paramètre ρ sur la résolution d'un problème de VRP à 10 ingénieurs et 30 tâches avec une population de 60 individus évoluant sur 50 générations et des probabilités de croisement et de mutation de 0.4 et 0.2 respectivement. La courbe pleine a été obtenue avec l'opérateur de croisement ensembliste et la courbe en pointillés avec l'opérateur de croisement classique. Elles représentent le coût du meilleur individu obtenu (l'objectif est de minimiser ce coût).

Les coûts infinis correspondant aux valeurs de ρ inférieures à 0.15 pour les deux

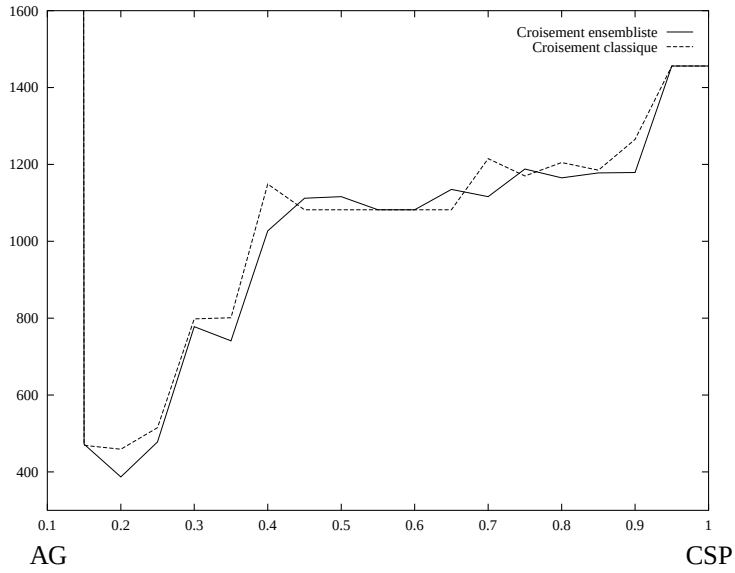


Figure 3. *Influence de ρ sur la qualité de la solution*

courbes traduisent l'incapacité de l'algorithme génétique seul (les sous-domaines sont ici réduits à une seule valeur) à fournir une solution admissible au problème. On remarque également que la valeur de ρ associée au coût minimum pour ce problème se situe autour de 0.2 et que l'algorithme s'avère peu efficace pour $\rho > 0.4$ car le CSP utilisé pour l'évaluation de l'adaptation ne cherche pas à optimiser la solution obtenue : des éléments de population associés à des sous-domaines de grande taille susceptibles de contenir beaucoup de mauvaises solutions se voient ainsi attribuer un mauvais coût.

On peut d'autre part constater que l'opérateur de croisement ensembliste est plus efficace que l'opérateur trivial sur la plupart des valeurs de ρ. Nous avons aussi pu observer [BAR 97] que l'opérateur de croisement ensembliste est plus robuste que l'opérateur classique vis-à-vis des variations des autres paramètres de l'algorithme (probabilités de mutation et de croisement).

Enfin, on peut noter que la même formulation de ce problème en CSP pur avec optimisation par simple *Branch & Bound* fournit une solution de coût 1 300 en un temps[7] comparable (1956s) à celui que met notre algorithme génétique (finement paramétré) pour obtenir une solution de coût 387. De plus, en un temps environ 50 fois supérieur, le programme CLP (muni de `minimize`) n'atteint qu'un coût de 1 177 ; il est donc incapable d'optimiser les solutions obtenues.

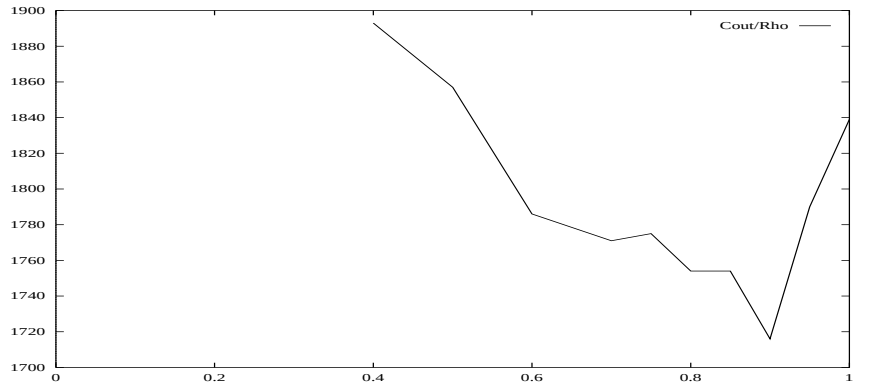


Figure 4. *Test Solomon R101 : influence de ρ (optimum pour $\rho = 0.9$)*

6.1.4. *Instances académiques « Solomon »*

Afin de comparer les performances de notre approche avec les méthodes habituelles, nous nous sommes intéressés à des instances classiques d'un problème VRP. Ces instances sont disponibles sur Internet par exemple sur le site de Y. ROCHAT[ROC 96] (`http://dmawww.epfl.ch/~rochat/rochat_data/solomon.html`).

Il s'agit d'un problème de VRP plus « pur » que celui décrit précédemment, l'équivalence entre les véhicules simplifiant considérablement la formulation : il n'est pas nécessaire d'exprimer quel véhicule réalise quelle tâche mais seulement que chaque tâche est réalisée par un et un seul véhicule.

Nous avons utilisé une variable de décision principale par ville (i) ayant pour valeur le numéro de la ville suivante ($Next_i$) dans la tournée. Des variables secondaires (distance, temps de passage, remplissage partiel du véhicule) permettent d'exprimer les contraintes (*time window* , capacité) et le coût d'une solution (somme des distances parcourues par les véhicules).

Pour la recherche d'une solution, nous avons utilisé une heuristique simple basée sur un ordonnancement statique :

— sont étiquetées en priorité les variables $Next_i$ des villes i les plus éloignées de la base ;

— les valeurs tentées en priorité pour $Next_i$ correspondent aux villes les plus proches de la ville i (dans l'espace et dans le temps en tenant compte des fenêtres de temps).

Les figures 4 et 5 illustrent des résultats obtenus pour le *benchmark* `R101` à 100 villes et 25 véhicules. Les paramètres de l'algorithme génétique sont les suivants : croisement ensembliste, population de 30 individus, 100 générations, $P_m = 0.2$ et $P_c = 0.5$. Le *timeout* pour l'évaluation d'un individu est de 6s, ce qui correspond à 1h de calcul pour le déroulement total de l'algorithme. La figure 4 montre l'influence de

[7] Les tests ont été réalisés avec ECL^{i}PS^e 3.5.2 sur un Pentium 200Mhz.

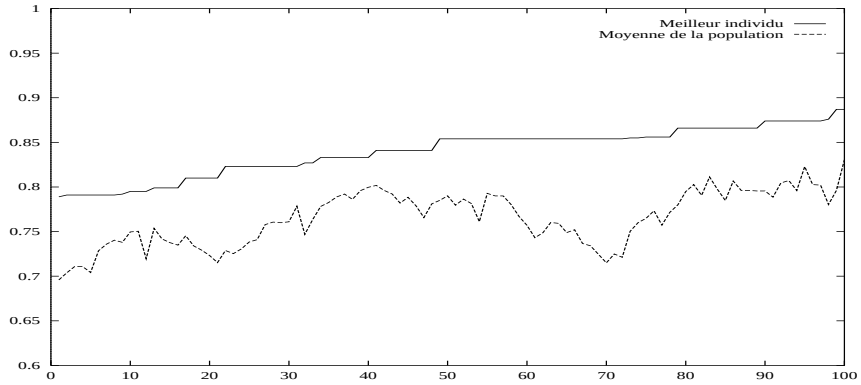


Figure 5. *Test Solomon R101 : évolution de la population*

la valeur de ρ sur l'optimum obtenu. On voit que sur ce problème des sous-domaines de grande taille sont nécessaires au bon fonctionnement de l'algorithme. En particulier, l'algorithme génétique pur n'est pas capable de construire des solutions admissibles. De même l'algorithme CSP pur ne réussit pas à améliorer la première solution trouvée (le *Branch & Bound* étant très inefficace sur une fonction de coût composée d'une somme de termes). La figure 5 montre l'évolution du meilleur individu et de la moyenne de la population au fur et à mesure des générations ($\rho = 0.85$, l'évaluation, qui est maximisée, est une mise à l'échelle entre 0 et 1 du coût). La croissance du meilleur individu de la population, conservé par élitisme, est constante durant tout l'algorithme, ce qui valide les opérateurs de croisement et mutation.

Sur l'ensemble des instances SOLOMON `R101` à `R112` l'algorithme hybride, avec ces mêmes paramètres, donne des solutions à environ 10% des meilleures solutions connues[ROC 96].

Les solutions obtenues peuvent être améliorées par augmentation de la taille de la population et du nombre de générations. Pour $\rho = 0.9$ avec 50 individus et 400 générations, la solution obtenue pour `R101` est de coût 1691.82 (en 24786 s, presque

Nb clients	Tournée	Nb clients	Tournée
6	27 69 30 51 20 1	6	5 83 61 85 84 96
5	28 12 76 79 50	5	42 15 87 57 97
1	6	6	33 81 3 54 24 80
4	52 99 94 26	5	45 82 7 60 89
2	40 53	4	29 78 34 68
3	31 88 18	5	39 23 67 56 4
5	2 75 22 55 25	7	63 62 11 90 10 32 70
7	92 14 44 38 43 91 100	7	36 47 19 8 46 17 93
7	59 95 98 16 86 37 13	3	64 49 48
6	72 21 73 41 74 58	6	65 71 9 66 35 77

Figure 6. *Solution de coût 1691.82 pour l'instance Solomon* `R101`

7 h) qui est à comparer avec les meilleures solutions obtenues par des algorithmes *ad hoc*[ROC 96] : 2.5% de la solution ROCHAT (1650), 5.2% de la meilleure solution (1607) disponible en 1996 [DES 92]. La solution est donnée dans la table 6. Cette performance nous semble acceptable pour un algorithme générique avec une heuristique basique (une centaine de lignes de code en tout).

6.2. *Problème d'affectation de fréquences radio (RLFAP)*

Le problème d'affectation de fréquences radio (RLFAP, *Radio Link Frequency Assignment Problem*) [TIO 95] est un des problèmes tests issus du projet européen CALMA (*Combinatorial ALgorithms for Military Applications*). Ce problème apparaît à l'installation d'un réseau de canaux de communication radio : on doit affecter une fréquence à chaque canal de tel manière que les interférences entre les fréquences assignées soit suffisamment faibles pour que les communications ne soient pas distordues. Chaque canal possède un domaine fini de fréquences entières et certains canaux doivent avoir des fréquences suffisamment éloignées pour ne pas dépasser un certain seuil d'interférence. De plus, on doit minimiser le nombre de fréquences différentes affectées.

Le problème comporte des contraintes complexes (égalités et inégalités sur des valeurs absolues de différences de deux variables), et un vaste espace de recherche (plusieurs centaines de variables).

6.2.1. *Formulation*

Nous avons formulé ce CSP de manière très simple :

— À chaque canal correspond une variable f_i qui représente sa fréquence et dont le domaine est $\mathcal{F}_i$ ($f_i \in \mathcal{F}_i$). Certains canaux ont déjà une valeur « pré-affectée » p_i ($f_i = p_i$).

— Les contraintes sont définies sur des paires de fréquences (f_i, f_j) qui doivent respecter l'une ou l'autre des contraintes suivantes :

$$|f_i - f_j| > d_{ij} \quad [7]$$

$$|f_i - f_j| = d_{ij} \quad [8]$$

avec d_{ij} qui représente une certaine distance entre les fréquences. Deux canaux de fréquences i et j impliqués dans une contraintes de type 7 sont des canaux *interfèrents* et d_{ij} leur *distance d'interférence* ; s'ils sont liés par une contrainte de type 8, ce sont des canaux *parallèles*.

— La fonction de coût à minimiser est le nombre de fréquences différentes attribuées aux canaux.

6.2.2. *Étiquetage et opérateurs*

Nous avons utilisé une heuristique d'étiquetage guidée par la fonction de coût : pour instancier une variable, on forme d'abord l'union des valeurs des variables déjà instanciées, puis on effectue l'intersection avec le domaine de la variable et on commence par essayer les valeurs de cette intersection ; on tente donc d'instancier les variables avec le moins de nouvelles valeurs possibles. Cet étiquetage s'est révélé bien plus efficace que le *labeling* standard sur l'ensemble des instances du problème RLFAP traitées.

Nous avons de nouveau testé les deux opérateurs de croisement classique et ensembliste sur les problèmes de RLFAP. Nous avons également testé la mutation classique ainsi qu'un opérateur de mutation spécifique pour ce problème guidé à la fois par la fonction de coût et par la valuation des variables. Cet opérateur fonctionne de la manière suivante :

— on choisit aléatoirement un gène G_i du chromosome de l'individu à muter ;

— on réalise ensuite l'union des valeurs des variables instanciées lors de l'évaluation de l'individu correspondant à tous les *autres* gènes : $\mathcal{G}_i^{\cup} = \bigcup_{j \neq i} X_j$;

— on en effectue alors l'intersection avec le domaine D_i associé au gène mutant : $\mathcal{I}_i^{\cap} = \mathcal{G}_i^{\cup} \cap D_i$;

— on choisit ensuite aléatoirement une partie de $\mathcal{I}_i^{\cap}$ pour construire le sous-domaine du gène muté ;

— on complète éventuellement avec des valeurs de $D_i - \mathcal{I}_i^{\cap}$ pour obtenir la taille adéquate.

Cette mutation tente de faire coïncider les valeurs du sous-domaine correspondant au gène muté avec les valeurs associées aux autres gènes lors de l'évaluation de l'individu, pour faire diminuer le nombre de fréquences attribuées.

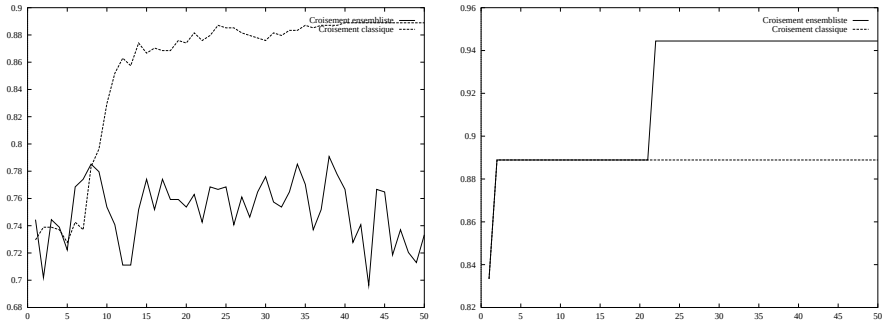
6.2.3. *Résultats*

Nous présentons les résultats obtenus pour un problème de RLFAP à 200 canaux et $1\,235$ contraintes, avec une population de 30 individus évoluant pendant 50 générations.

La figure 7 présente l'évolution de la population de l'algorithme génétique au cours des générations pour les deux croisements classique et ensembliste, avec $\rho = 0.8$, $P_c = 0.5$ et $P_m = 0.0$ [8]. La courbe de gauche correspond à la moyenne des adaptations [9] des individus de la population et celle de droite à l'adaptation du meilleur individu. On remarque que pour le graphe des moyennes, le croisement ensembliste est inférieur au croisement classique et plus irrégulier ; en effet, l'absence de mutation handicape l'exploration de l'espace de recherche et l'opérateur de croisement ensem-

[8] La probabilité de mutation a volontairement été mise à 0 pour comparer seulement l'influence des croisements.

[9] Le problème RLFAP est un problème de *minimisation* mais un algorithme génétique *maximise* les adaptations des individus (l'adaptation est l'opposée de la fonction de coût mise à l'échelle). Les meilleurs individus correspondent donc ici à des adaptations élevées.



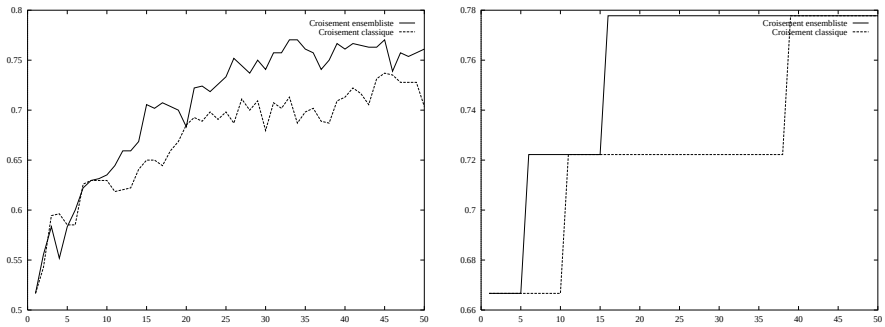
Adaptation moyenne de la population Adaptation du meilleur individu

Figure 7. *Comparaison des deux opérateurs de croisement*

bliste essaye de créer des fils les plus distincts possibles, ce qui est plus destructeur que le croisement classique. Cependant, la courbe du meilleur individu montre que le croisement ensembliste est, comme pour le VRP, plus efficace que le croisement classique.

La figure 8 montre l'évolution de la population de l'algorithme génétique au cours des générations pour les deux mutations classique et spécifique, avec $\rho = 0.6$, $P_c = 0.0$ (la probabilité de croisement a volontairement été mise à 0 pour comparer seulement l'influence des mutations) et $P_m = 0.3$. De même que pour la figure 7, les courbes de gauche correspondent à la moyenne des adaptations des individus de la population et celles de droite à l'adaptation du meilleur individu. L'opérateur de mutation spécifique est globalement plus efficace que l'opérateur classique, tant pour la courbe moyenne que pour celle de la meilleure solution.

De même que pour le VRP, nous avons comparé l'efficacité de notre algorithme hybride avec celle de la même formulation CSP du problème RLFAP. Le CSP seul obtient une solution de coût 18 en quelques secondes et ne parvient pas à l'améliorer en



Adaptation moyenne de la population Adaptation du meilleur individu

Figure 8. *Comparaison des deux opérateurs de mutation*

moins de 135 heures, alors que l'algorithme génétique hybride (finement paramétré : $\rho = 0.9$, $P_c = 0.2$ et $P_m = 0.4$) fournit une solution de coût 14 (l'optimum en l'occurrence) en moins de 17 minutes (et à la $17^{\text{ème}}$génération). Le CSP muni de `minimize` est donc, comme pour le VRP, incapable d'optimiser les solutions obtenues sur le problème RLFAP.

Même si ces résultats ne sont pas « bons » comparés à l'état de l'art [CAB 96], ils illustrent la supériorité de l'algorithme hybride devant les méthodes simples.

6.3. *Validation de l'approche*

Les résultats obtenus sont encourageants car l'algorithme hybride trouve de meilleures solutions que le CSP avec *Branch & Bound* pur, et l'algorithme génétique seul est en général incapable de générer suffisamment d'individus admissibles pour résoudre les problèmes traités. Cependant, nous avons utilisé des formulations CSP naïves des problèmes de VRP et RLFAP, ce qui ne permet pas de conclure à l'infériorité des techniques de satisfaction de contraintes qui peuvent être bien plus sophistiquées. De même, l'algorithme génétique utilisé est un algorithme simple qui n'utilise pas tous les raffinements communément intégrés au sein des algorithmes génétiques « modernes ». Néanmoins, les versions « basiques » de la formulation CSP des problèmes traités et la simplicité de notre algorithme génétique montrent la robustesse de la méthode : très peu d'efforts sont nécessaires pour obtenir des résultats sur des problèmes qui résistent aux deux méthodes classiques seules. Bien entendu, un algorithme d'optimisation spécifiquement dédié à la résolution de l'un ou l'autre de ces problèmes est susceptible de fournir de meilleurs résultats, mais avec un investissement bien plus important et sans la possibilité de le réutiliser pour d'autres problèmes.

7. Conclusion

Nous avons présenté une nouvelle méthode d'optimisation où une résolution CSP est plongée dans un algorithme génétique. L'approche se veut générique et indépendante du problème traité. Pour l'algorithme génétique, nous avons proposé des opérateurs originaux adaptés à la sémantique des chromosomes manipulés. La méthode a été appliquée à deux problèmes d'optimisation classiques et difficiles et l'analyse des résultat obtenus valide l'algorithme.

Les perspectives du projet GASCON (Genetic AlgorithmS + CONstraint programming) sont diverses :

— L'étude théorique analysant les propriétés de complexité, de convergence, « dynamique » de l'algorithme doit être poursuivie afin de permettre la conception d'opérateurs plus efficaces ainsi que la caractérisation des domaines d'application (variables à grand domaine, solutions admissibles complexes, ...).

— Les améliorations « classiques » pour les algorithmes génétiques doivent être

intégrées à l'implémentation. En particulier, le *sharing* (pénalisation des individus trop voisins les uns des autres) a déjà été étudié et une distance entre individus a été définie [BAR 97].

— Le concept d'hybridation entre algorithmes génétiques et CLP(FD) peut être étendu à d'autres domaines de calcul : CLP($\mathbb{Q}$), CLP($\mathbb{R}$), ...

— Les résultats obtenus sur les problèmes traités doivent être comparés à ceux obtenus avec les méthodes autres que celles utilisées par l'hybridation.

8. Bibliographie

[BAR 97] BARNIER N., « Optimisation par hybridation d'un algorithme génétique avec la programmation par contraintes ». Rapport de DEA, Institut National Polytechnique de Toulouse, 1997.

[BUR 95] BURKE E. K., ELLIMAN D. G. et WEARE R. F., « A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems ». Rapport technique, University of Nottingham, 1995.

[CAB 96] CABON B., « *Problèmes d'optimisation combinatoire : évaluation des méthodes de la physique statistique* ». PhD thesis, ENSAE, Toulouse, 1996.

[CER 94] CERF R., « Une théorie asymptotique des algorithmes génétiques ». In ALLIOT J.-M., ÉVELYNE LUTTON, RONALD E. et SCHOENAUER M., Eds., *Évolution artificielle 94*, p. 11–18, Toulouse, 1994. Cépaduès Éditions.

[DES 92] DESROCHERS M., DESROSIERS J. et SOLOMON M., « A new optimization algorithm for the vehicle routing problem with time windows ». *Operations Research*, vol. 40, p. 342–354, 1992.

[DIN 88] DINCBAS M., HENTENRYCK P. V., SIMONIS H., AGGOUN A. et GRAF T., « The Constraint Logic Programming Language CHIP ». In *Int. Conf. Fifth Generation Computer Systems*, vol. 1, p. 693–702, Tokyo, Japan, 1988.

[ECL 92] *ECL^{i}PS^e User Manual (ECRC Common Logic Programming System)*. 1992.

[GOL 89] GOLDBERG D., *Genetic Algorithms*. Addison Wesley, 1989.

[HAJ 95] HAJIAN M. T., EL-SAKKOUT H. H., WALLACE M. G., RICHARDS E. B. et LEVER J. M., « Towards a Closer Integration of Finite Domain Propagation and Simplex-Based Algorithms ». In *AI Maths 96 Florida Atlantic University*, 1995.

[KOK 96] KOK J. N., MARCHIORI E., MARCHIORI M. et ROSSI C., « Evolutionary Training of CLP-Constrained Neural Networks ». In *Proceedings of the Second International Conference on the Practical Application of Constraint Technology (PACT'96)*, p. 129–142, London, U.K., 1996. The Practical Application Company Ltd.

[KUE 93] KUECHENHOFF V., « Novel Search and Constraints – an Integration ». Technical Report ECRC–CORE–93–9, European Computer-Industry Research Centre, 1993.

[MéD 94] MÉDIONI F., « Optimisation de la résolution de conflits par algorithmes stochastiques et programmation linéaire ». Master's thesis, École Nationale de l'Aviation Civile, 1994.

[MIC 92] MICHALEWICZ Z., *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, 1992.

[PES 96] PESANT G. et GENDREAU M., « A View of Local Search in Constraint Programming ». In *Proc. of the 2nd International Conference on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, p. 353–366, Cambridge, MA, USA, October 1996.

[ROC 96] ROCHAT Y.-N., « *Modélisation et résolution de problèmes de distribution et d'ordonnancement* ». PhD thesis, École Polytechnique Fédérale de Lausanne, 1996.

[TIO 95] TIOURINE S., HURKENS C. et LENSTRA J. K., « An overview of algorithmic approaches to frequency assignment problems ». Rapport technique, Eindhoven University of Technology, 1995.

[VAN 89] VAN HENTENRYCK P., *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, MA, 1989.

Nicolas Barnier *est titulaire du DEA « informatique fondamentale et parallélisme » de l'*ENSEEIHT*, Toulouse et ingénieur diplomé de l'*ENAC *où il prépare actuellement une thèse au laboratoire d'optimisation globale.*

Pascal Brisset *est enseignant-chercheur à l'ENAC depuis 1994 où il est membre du laboratoire d'optimisation globale. Il est titulaire d'une thèse en informatique de l'Université de Rennes 1. Il s'intéresse aux extensions de la programmation logique et en particulier aux applications de la programmation logique avec contraintes.*