Optimisation par hybridation d'un CSP avec un algorithme génétique

Nicolas Barnier, Pascal Brisset

Optimisation par hybridation d'un CSP avec un algorithme génétique

Nicolas Barnier & Pascal Brisset

École Nationale de l'Aviation Civile
7, avenue Édouard Belin
B.P. 4005
F–31055 Toulouse Cedex 4, France
E-mail : barnier,brisset@recherche.enac.fr

Résumé : *Dans le cadre de la programmation logique avec contraintes, et plus généralement des CSP (Constraint Satisfaction Problem) sur des domaines finis, 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, cet ensemble de sous-domaines constituant un chromosome,* i.e. *un individu de l'algorithme génétique. L'évaluation d'un chromosome, c.-à-d. la fonction optimisée par l'algorithme génétique, est alors calculée par résolution du CSP sur le sous-espace correspondant. Dans ce cadre nous présentons plusieurs opérateurs de mutation et croisement, triviaux, puis adaptés au problème. Le résultat de cet étude est résumé en un nouveau prédicat d'optimisation pouvant remplacer le* `minimize` *de* CHIP. *Cette méthode d'optimisation hybride est destinée aux problèmes dont l'espace de recherche est trop vaste pour être exploré par un CSP simple et trop complexe pour qu'un algorithme génétique découvre des solutions admissibles. Les premiers tests de la méthode sur une formulation CSP naïve de VRP (Vehicle Routing Problem) sont encourageants.*

Mots-clés : *Optimisation, algorithme génétique, CLP, méthode mixte.*

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 minimal 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 intrinsèque à la représentation choisie[1] 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 (éventuellement à un pourcentage près) 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 (naïf) d'un algorithme génétique ou d'un CSP.

Nous proposons de profiter des avantages des deux approches en les hybridant :

- utilisation d'un CSP pour calculer des solutions admissibles sur un sous-espace de l'espace de recherche ;
- utilisation d'un algorithme génétique pour explorer cet espace.

L'idée est illustrée par la figure 1 (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. 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).

1. C'est par exemple possible pour le TSP.

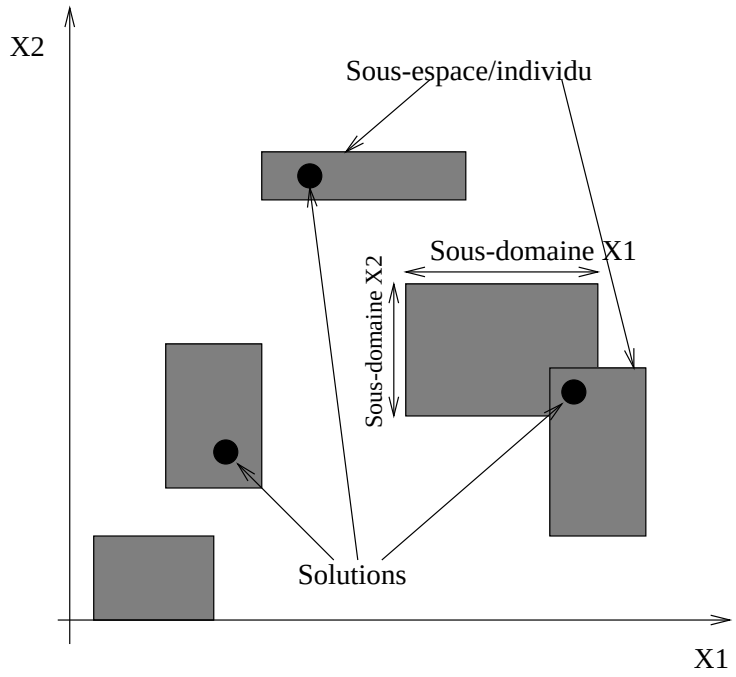

Figure 1 : *Un espace de recherche à deux dimensions*

Nous décrivons dans cet article une méthode générique pour réaliser cette hybridation 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 (mutation et croisement) sur ces derniers et évaluation. Nous terminons par un exemple prometteur et comparons notre méthode avec les approches similaires.

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, ...) , D leurs domaines finis respectifs ($D(X_1)$, $D(X_2)$, ...) et C un ensemble de relations entre ces variables. Pour un problème d'optimisation, 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 CSP 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*.

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 composé d'un chromosome lui-même 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. On lui associe la valeur du critère à optimiser, son *adaptation*. On génère ensuite de façon itérative (figure 2) 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, le croisement et la mutation assurent une exploration efficace de l'espace d'états.

On commence par générer une population d'individus. Pour passer d'une génération k à la génération $k+1$, les trois opérations suivantes sont répétées pour tous les éléments de la population k. Des couples de parents P_1 et P_2 sont sélectionnés en fonction de leurs adaptations. L'opérateur de croisement leur est appliqué avec une probabilité P_c (généralement autour de 0.6) et génère des couples d'enfants C_1 et C_2. D'autres éléments P sont sélectionnés en fonction de leur adaptation. L'opérateur de mutation leur est appliqué avec la probabilité P_m (P_m est généralement inférieure à

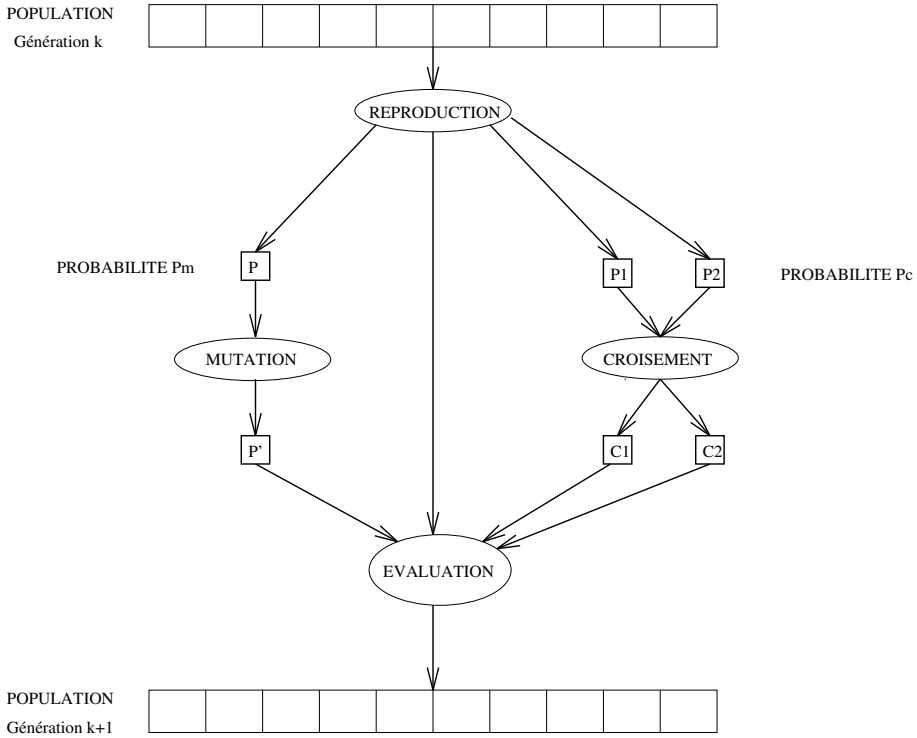

Figure 2 : *Principe général des algorithmes génétiques*

P_c) et génère des individus mutés P'. L'adaptation des enfants (C_1,C_2) et des individus mutés P' est ensuite évaluée avant insertion dans la nouvelle population.

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.

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.

3.1 *Chromosome*

Chaque variable du CSP est associée à un gène du chromosome[2] qui le représente. Le gène G_i correspondant à une variable X_i est un sous-domaine, *i.e.* un sous-ensemble, de $D(X_i)$ de cardinal K_i. Le cardinal de ces sous-domaines est un paramètre de l'algorithme. On note ρ le rapport $|G_i|/|D(X_i)|$. On peut passer continûment 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 $\rho = 1$. On peut aussi envisager d'avoir des valeurs de ρ distinctes pour chaque variable : cela peut-être par exemple nécessaire si les tailles initiales des domaines sont très différentes.

C'est lors de l'initialisation de l'algorithme génétique que les individus originaux sont générés : pour chaque chromosome et pour chaque gène un sous-domaine aléatoire (de la taille adéquate) du domaine de la variable correspondante est construit.

3.2 *Valuation*

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).

Dans le cas d'une réelle hybridation ($\rho < 1$), il n'est pas nécessaire d'effectuer l'optimisation durant cette phase (cela serait possible avec le prédicat `minimize`) car elle est réalisée par l'algorithme génétique.

3.3 *Opérateurs triviaux*

Les opérateurs classiques pour les chaînes de bits [GOL 89] peuvent être utilisés avec notre codage (figure 3).

2. Nous nommons indifféremment individu ou chromosome les éléments de la population de l'algorithme génétique.

3.3.1 *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).

3.3.2 *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 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.

3.4 *Opérateurs sur les gènes*

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.-à-d. 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.

	X_1	X_2	X_3	
X :				Variables
D :	1..7	1..9	1..4	Domaines initiaux
P_1 :	1,2,3	4,7,8	1,3,4	Individu
P_2 :	1,5,7	1,4,8	1,2,3	Individu
P_1' :	1,2,3	<u>1,7,8</u>	1,3,4	Individu P_1 muté
C_1 :	1,2,3	4,7,8	1,2,3	Premier fils de P_1 et P_2
C_2 :	1,5,7	1,4,8	1,3,4	Second fils de P_1 et P_2

Figure 3 : *Mutation et croisement classiques*

On suggère donc un nouvel opérateur de croisement : on réalise d'abord l'union UP des sous-domaines des pères, puis on choisit aléatoirement une partie de UP de taille appropriée qui constitue le gène GC_1 du premier fils ; on prend ensuite pour constituer GC_2 les valeurs restantes de l'union des gènes des pères (i.e. $UP - GC_1$) que l'on complète si nécessaire par des valeurs de GC_1 pour obtenir le cardinal désiré. Par exemple, pour le croisement du premier gène de P_1 et P_2 de la figure 3, l'union vaut 1,2,3,5,7 parmi laquelle on choisit au hasard 3 valeurs pour le premier fils (1,3,7), il reste pour le second 2,5 qu'on complète avec des éléments du premier (pour obtenir par exemple 1,2,5).

3.5 *Opérateurs guidés par la valuation*

L'opérateur de croisement précédent conserve les valeurs des sous-domaines des gènes parents sans tenir compte de la solution effective calculée par le CSP (une seule valeur pour chaque sous-domaine). On peut accroître l'efficacité des opérateurs en conservant ces valeurs déterminées par la résolution du CSP : pour la mutation, on garde la valeur solution du gène à muter dans le gène mutant, et pour le croisement, on insère en premier dans les sous-domaines des fils les valeurs solutions des gènes des deux pères et on complète comme précédemment.

L'inconvénient majeur d'opérateurs par trop déterministes au sein d'un algorithme génétique est de limiter le facteur d'exploration. Par exemple lorsque les sous-domaines sont de petite taille (3 ou 4 éléments), ces opérateurs conservateurs produisent des individus proches des individus initiaux et l'exploration de l'espace de recherche risque d'en être handicapée.

3.6 *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* `)`

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

3.6.1 *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 (prédicat `::`) avant que le but *Goal* soit appelé (`call`). 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[3].

3.6.2 *Paramétrage et utilisation*

L'une des difficulté 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. Dans notre implémentation, tous ces paramètres ont des valeurs par défaut et peuvent être modifiés par des variables globales Prolog (`setval`).

4 Application

Nous avons testé notre algorithme mixte sur un problème de VRP (Vehicle Routing Problem). Le VRP est une extension du TSP (Travelling Salesman Problem) :

3. Cela peut être raffiné pour chaque problème particulier par un calcul ad hoc.

des tâches doivent être effectuées dans des sites distincts et dans des intervalles de temps fixés (*time window*) ; chaque tâche peut être exécutée par un certain nombre d'ingénieurs ; certaines tâches doivent être réalisées avant ou en même temps que d'autres (ordonnancement) ; le problème est de produire un emploi du temps pour chaque ingénieur qui minimise le temps passé par chacun d'entre eux (temps de parcours lié à la distance entre deux tâche successives et temps d'attente). Nous avons choisi ce problème pour sa complexité intrinsèque et la taille de son espace de recherche[4].

4.1 *Formulation*

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

- À chaque tâche (i) sont associées deux variables à domaine, l'une correspondant à l'ingénieur réalisant cette tâche (E_i) et l'autre correspondant à l'heure à laquelle la tâche sera effectivement réalisée (T_i) ; les domaines de ces variables sont donnés (ingénieurs qualifiés pour une tâche et *time window* pour une tâche) ;

- Les contraintes d'ordonnancement (#= et #<=) sont exprimées sur les T_i's ;

- Avec cette formulation, c'est le fait qu'un ingénieur ne peut pas effectuer deux tâches à la fois qui est délicat à exprimer. Pour chaque paire de tâche (i, j), nous avons posé la contrainte suivante :

$$
\begin{aligned}
& M \text{ si } E_i = E_j && [1]\\
\& \quad & T_i + d_i + t \leq T_j + \infty B_1 && [2]\\
\& \quad & T_j + d_j + t \leq T_i + \infty B_2 && [3]\\
\& \quad & M + B_1 + B_2 = 2 && [4]
\end{aligned}
$$

  où M est une variable booléenne vraie (valeur 1) si les deux tâches sont effectuées par le même ingénieur (contrainte 1), d_i (resp. d_j) est la durée de la tâche i (resp. j), t est la durée de parcours entre les sites de chaque tâche, B_1 et B_2 sont des booléens permettant le codage classique de la contrainte disjonctive (ordre des tâche) par une conjonction (contraintes 2 et 3). La contrainte 4 exprime que soit les tâches ne sont pas effectuées par le même ingénieur ($M = 0, B_1 = B_2 = 1$), soit une (et une seule) des contraintes 2 et 3 est vérifiée ($B_1 + B_2 = 1$).

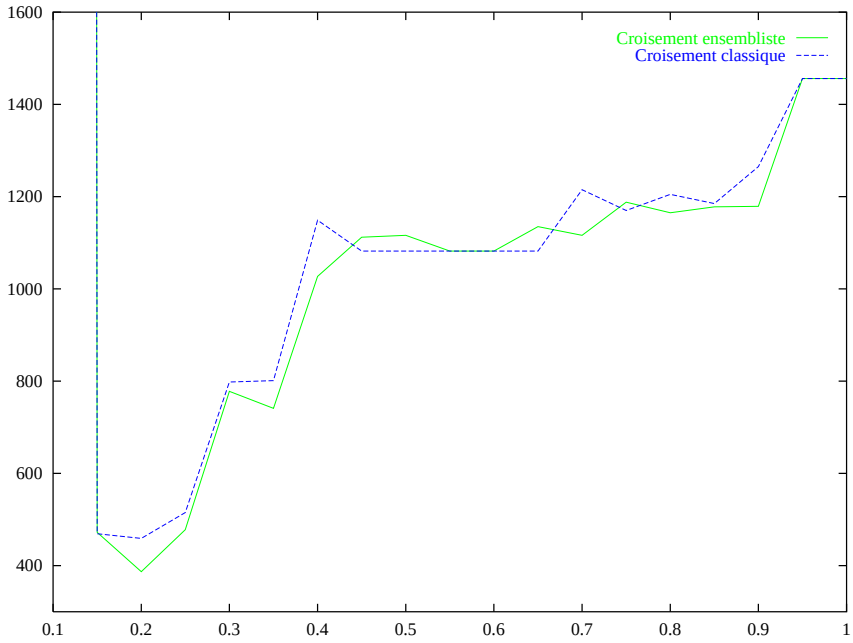

Figure 4 : *Influence de ρ sur la qualité de la solution*

4.2 *Résultats*

La figure 4 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 "trivial" et la courbe en pointillés avec le deuxième opérateur de croisement utilisant des opérations ensemblistes présenté ci-dessus ; elles représentent le coût du meilleur individu obtenu.

Les coûts infinis correspondant aux valeurs de ρ inférieures à 0.15 pour les deux 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 recherche pas la meilleure solution

4. Merci à M. WALLACE pour le jeu de données.

contenue dans les sous-domaines d'un individu : 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 le deuxième opérateur de croisement ensembliste est un peu plus efficace que l'opérateur trivial sur l'ensemble des valeurs de ρ.

La figure 5 montre l'influence des probabilités de croisement P_c et de mutation P_m sur le même problème que précédemment avec le second croisement et $\rho = 0.2$. La moitié seulement de la courbe limitée par $P_c + P_m \leq 0.7$ doit être prise en compte, le reste étant dû à des valeurs fictives résultats de l'extrapolation des données.

La partie correspondant à une probabilité de mutation nulle est tronquée car les coûts associés sont trop élevés : la population évolue très peu lorsqu'il n'y a pas de mutation. On constate que les probabilités des opérateurs pour lesquelles on obtient les coûts les plus faibles se situent autour de valeurs élevées pour la mutation ($0.3 \leq P_m \leq 0.6$) et assez faible pour le croisement ($0.0 \leq P_c \leq 0.3$).

Par ailleurs, l'étude comparée de l'influence des probabilités des opérateurs avec les deux croisements présentés confirme que l'opérateur ensembliste est globalement plus efficace que le croisement trivial.

Enfin, on peut noter que le CSP pur avec le `minimize` classique produit une solution de coût 1300 dans un temps comparable à celui de l'algorithme génétique et la meilleure solution atteinte (en un temps envion 50 fois supérieur) est seulement 1177 (à comparer avec 387, meilleure solution obtenue).

5 Autres approches

D'autres approches d'hybridation entre CSP et algorithme génétique ont été expérimentées. La solution proposée était alors d'intégrer le CSP aux opérateurs de l'algorithme génétique afin que tous les individus de la population correspondent à des solutions admissibles au problème. Dans ce cadre, [KUE 93] obtient des bons résultats pour le problème du TSP avec un opérateur de mutation effectuant une optimisation locale (à l'aide d'un CSP) et un opérateur de croisement générant des individus entièrement nouveaux en cas d'échec du CSP (bonne exploration de l'espace de recherche). [BUR 95] résout des problème d'emploi du temps (*timetabling*) à l'aide d'une méthode analogue en utilisant des opérateurs très spécifiques.

Notre approche est plus directement comparable à celle de [HAJ 95] qui hybride un CSP avec un Simplex : la méthode est générique et indépendante du problème à traiter.

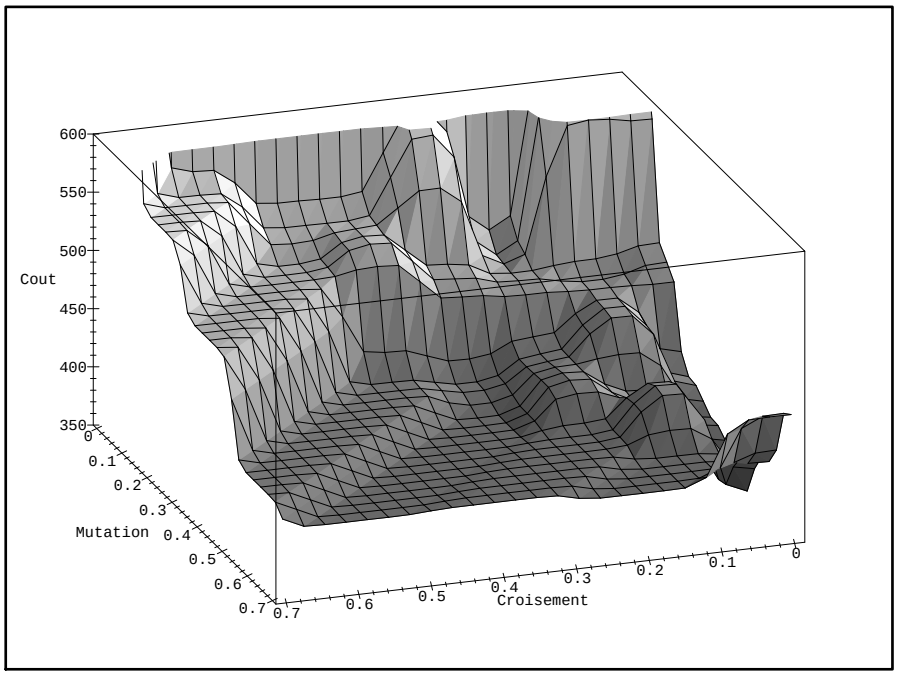

Figure 5 : *Influence des probabilités de croisement et mutation*

6 Conclusion

Nous avons présenté une nouvelle méthode d'optimisation où une résolution CSP est plongée dans un algorithme génétique. L'algorithme proposé peut être appliqué à n'importe quel problème formulé en CSP. Nous donnons les premiers résultats expérimentaux sur un problème VRP, résultats qui valident l'approche.

Cette étude est poursuivie en appliquant la méthode à divers problèmes d'opti-

misations classiques (*scheduling*, *timetabling*, ...) et en particulier à divers problèmes d'optimisation liés au trafic aérien [ALL 96]. L'étude sera également étendue à d'autres moyens d'hybridation :

- division de l'espace suivant ses "dimensions", l'une étant traitée par un algorithme génétique et l'autre par un CSP (par exemple pour un problème d'emploi du temps, choix des horaires par algorithme génétique et choix des salles par un CSP) ;
- intégration d'une méthode CSP de "réparation" [MIN 94] au sein des opérateurs de l'algorithme génétique pour fabriquer des solutions admissibles.

Bibliographie

[ALL 96] ALLIOT J.-M., Techniques d'optimisation stochastique appliquées aux problèmes du contrôle aérien, Thèse d'habilitation, Université de Toulouse Paul Sabatier, 1996.

[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, Technical Report NOTTCS-TR-95-8.

[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*, volume 1, pp. 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 », 1995, AI Maths 96 Florida Atlantic University.

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

[MIN 94] MINTON S., JOHNSTON M., PHILIPS A., et LAIRD P., Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems, In FREUDER E. C. et MACKWORTH A. K., éditeurs, *Constraint-based Reasoning*, MIT Press, 1994.

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