Hybridation de GWW avec de la recherche locale

Bertrand Neveu et Gilles Trombettoni
Projet COPRIN, CERMICS-INRIA-I3S
2004 route des lucioles
06902 Sophia Antipolis Cedex BP 93
email: `neveu,trombe@sophia.inria.fr`

Résumé

Une nouvelle métaheuristique pour l'optimisation combinatoire, appelée *"Go With the Winners"* (`GWW`) a été proposée par Dimitriou et Impagliazzo en 1996. `GWW` gère plusieurs configurations en même temps, utilise un seuil pour éliminer les pires configurations, et inclut une étape de marche aléatoire qui permet une distribution uniforme des configurations visitées dans chaque sous-problème.

Cet article propose de remplacer l'étape de marche aléatoire de `GWW` par de la recherche locale pour favoriser les meilleures solutions.

Nous proposons une instance de cet algorithme hybride, appelée `GWW-grw`. Une attention particulière a été portée sur l'efficacité pour des problèmes de taille réaliste. Nous comparons différentes manières de baisser le seuil. Nous ajoutons un seul paramètre pour intégrer la recherche locale dans le schéma existant. Nous montrons comment et dans quel ordre, régler les paramètres.

Une étude expérimentale a été menée sur des instances difficiles de coloriage de graphe issues du challenge DIMACS et sur des problèmes d'affectation de fréquences du CELAR. Les meilleures bornes connues ont été trouvées sur toutes les instances.

1 Introduction

Notre travail est basé sur l'algorithme *"Go With the Winners"* (`GWW`) présenté dans [Dimitriou et Impagliazzo, 1996]. `GWW` explore différentes configurations de l'espace de recherche en même temps en gérant une population de solutions appelées *particules*. Au début, les particules sont distribuées aléatoirement et un seuil est placé au niveau du coût de la plus mauvaise particule. Les phases suivantes sont effectuées itérativement jusqu'à qu'il n'y ait plus de particule en dessous du seuil (nous considérons un problème de minimisation) :

1. **Redistribution :** Les particules au dessus du seuil sont "redistribuées" sur les autres (d'où le nom de l'algorithme) : une particule redistribuée est remplacée par une copie d'une autre particule sous le seuil choisie aléatoirement.

2. **Exploration aléatoire :** Cette phase est réalisée pour chaque particule et tend à séparer les particules qui ont été regroupées par la phase de redistribution. Elle est basée sur une marche aléatoire de longueur spécifiée. Chaque étape de cette marche fait passer une particule sur un état voisin qui reste sous le seuil.
3. La valeur du seuil est décrémentée de 1.

Quand le problème a de bonnes propriétés (voir Section 2), `GWW` est théoriquement capable de trouver une solution optimale en temps polynomial avec une forte probabilité. Cependant, en pratique, de premières expérimentations réalisées sur les instances du CELAR et de coloriage ont donné de plus mauvaises performances qu'un algorithme de Metropolis standard [Connolly, 1990](recuit simulé à température constante).

Dans la phase d'exploration aléatoire de l'algorithme `GWW`, les meilleures configurations ne sont pas favorisées et une particule peut indifféremment monter ou descendre. Ainsi, la progression dans `GWW` est due uniquement à la gestion du seuil. Dans cet article, nous montrons de manière expérimentale que `GWW` peut être rendu plus efficace en remplaçant la phase de marche aléatoire par de la recherche locale. Cela donne l'algorithme `GWW-LS` qui est décrit par la suite. Les principales contributions sont les suivantes :

– Précédemment, `GWW` a été appliqué sur des problèmes de théorie des graphes (la clique maximum et la bissection [1]). Toutes les versions correspondantes de `GWW` supposent que les coûts possibles sont des entiers petits, justifiant la baisse du seuil de 1 à chaque itération. Ce n'est pas le cas pour de nombreux problèmes comme les MAX-CSP pondérés présentés dans cet article. Par exemple, l'instance `celar7` comprend des configurations dont le coût varie entre 343000 et 2.10^7. Nous proposons donc différentes manières de baisser le seuil et les comparons entre elles.
– `GWW-LS` gère plusieurs paramètres : le nombre B de particules, un paramètre T utilisé pour baisser le seuil, la longueur S de la marche, et les paramètres additionnels de la recherche locale sélectionnée (la longueur de la liste taboue, ou une température permettant de s'échapper des minimums locaux). Nous proposons une instance de ce schéma `GWW-LS`, appelé `GWW-grw` (`GWW` avec *marche aléatoire gloutonne*), qui limite le nombre de paramètres additionnels à 1. Pour minimiser l'effort requis pour régler les paramètres, les expérimentations ont validé une manière heuristique d'effectuer ce réglage.
– Nous avons mené des expérimentations sur des instances difficiles de problèmes d'affectation de fréquences et de coloriage de graphe. `GWW-grw` trouve les meilleures bornes (connues) pour tous ces problèmes et nous le comparons avec `GWW` et d'autres algorithmes de recherche locale.

La partie 2 donne un bref historique de `GWW`. La partie 3 détaille le schéma `GWW-LS`. L'instance `GWW-grw` de `GWW-LS` est décrite dans la partie 4. Les expérimentations sont exposées dans la partie 5. La partie 5.2 présente divers moyens de gérer le seuil dans des problèmes réels. La partie 5.3 donne une première manière heuristique de régler les quatre paramètres de `GWW-grw`. Les parties 5.4 et 5.5 montrent les performances de `GWW-grw` et le comparent à `GWW` et à des méthodes de recherche locale.

1. Couper un graphe en deux sous graphes de même nombre de nœuds avec un minimum d'arêtes entre les deux sous-graphes.

2 De GWW à GWW-LS

Avant de présenter l'algorithme GWW, nous définissons un *graphe de recherche* qui est une abstraction d'un problème d'optimisation combinatoire. Les nœuds du graphe de recherche sont les différentes configurations (c.à.d. les solutions potentielles) et les arêtes relient deux configurations qui sont "voisines", c.à.d , pour lesquelles un changement élémentaire permet de passer d'une configuration à l'autre. Nous supposons que la valeur, ou coût, de chaque nœud est un entier. Le but de l'optimisation combinatoire est de trouver un nœud de coût minimum.

L'algorithme "Go with the winners" a été introduit dans [Aldous et Vazinari, 1994]. Cet algorithme a été conçu pour trouver une feuille de profondeur maximum dans un arbre. Au début, plusieurs particules sont placées à la racine de l'arbre. Dans la phase d'exploration, chaque particule est déplacée sur un nœud fils choisi aléatoirement. Dans la phase de redistribution, les particules qui ont atteint une feuille sont redistribuées : elles sont remplacées par une copie d'une autre particule non feuille choisie aléatoirement. Le processus s'arrête quand toutes les particules sont sur une feuille. Les auteurs ont déterminé le nombre de particules nécessaire pour trouver le nœud le plus profond avec une forte probabilité dépendant d'une mesure d'équilibrage de l'arbre.

L'algorithme GWW dont il est question dans cet article est présenté dans [Dimitriou et Impagliazzo, 1996]. C'est une modification de la première version pour l'optimisation combinatoire. L'arbre de la première version apparaît dans tout problème d'optimisation en gérant un seuil. En effet, le graphe de recherche peut être divisé hiérarchiquement en plusieurs sous-graphes selon leurs coûts. Plus précisément, un nœud de l'arbre est constitué d'un sous-graphe de recherche dont les sommets ont un coût inférieur ou égal au seuil. Le sommet de l'arbre contient le graphe de recherche entier (quand le seuil est au plus haut). Baisser le seuil divise le graphe de recherche en composantes connexes qui forment une hiérarchie entre un nœud de l'arbre et ses fils. Le fait que deux régions deviennent déconnectées quand le seuil décroît signifie qu'aucune marche aléatoire (restant au niveau du seuil ou sous le seuil) ne peut déplacer une particule d'une région à l'autre.

Dimitriou et Impagliazzo analysent dans [Dimitriou et Impagliazzo, 1998] les performances de GWW par rapport aux propriétés combinatoires du problème. Ils démontrent en particulier que deux conditions suffisantes rendent un problème soluble en temps polynomial avec une forte probabilité :

- Le nombre de fils d'un nœud de l'arbre doit être borné polynomialement. Cela signifie que baisser le seuil (de 1) ne doit pas faire apparaître un nombre exponentiel de régions déconnectées. Cela garantit une convergence en temps polynomial si chaque région déconnectée possède une particule.
- L'*expansion locale* est vérifiée, c.à.d chaque nœud d'un sous-graphe de recherche a un nombre suffisant de voisins. Ainsi, une marche suffisamment longue dans une région donnée assure une distribution uniforme des particules à l'intérieur et ne les confine pas dans une petite portion de la région. Les phases de redistribution et de marche aléatoire assurent que chaque composante connexe reçoit un nombre de particules proportionnel à sa taille.

La première condition souligne l'importance du nombre B de particules. La seconde est liée à la longueur S d'une marche aléatoire.

Différentes marches aléatoires pour GWW sont décrites dans la littérature [2] :

- Dans [Dimitriou et Impagliazzo, 1996], une marche aléatoire de longueur 1 est effectuée à chaque étape : pour chaque particule de coût i, tous les voisins sont visités et l'un d'entre eux est choisi parmi ceux ayant un coût $i-1$.
- Dans [Dimitriou et Impagliazzo, 1998], une marche aléatoire de longueur S est proposée. Cette marche alterne des nœuds de coût i et de coût $i-1$.
- [Carson et Impagliazzo, 1999] et [Dimitriou, 2002] présentent une version simplifiée où la marche aléatoire de longueur S doit rester au seuil ou sous le seuil.

Cet article introduit l'algorithme GWW-LS où la marche aléatoire de GWW est remplacée par de la recherche locale, chaque étape de cette marche restant sous ou au niveau du seuil. La différence entre les deux approches est qu'une recherche locale tend à faire progresser la particule, ce qu'une marche aléatoire ne fait pas. La population descend donc plus vite mais la distribution uniforme des particules dans une région connexe n'est plus respectée. Cependant, les bons résultats expérimentaux obtenus avec GWW-LS nous laissent penser que d'autres conditions de convergence en temps polynomial avec forte probabilité existent. Une telle hypothèse est évoquée dans la conclusion.

3 Description de `GWW-LS`

Les différences entre GWW et GWW-LS résident dans les deux points suivants :

- *La descente du seuil est adaptative.*
  La partie 5.2 compare différentes façons de définir la fonction `Baisserseuil` utilisée pour baisser le seuil. Malheureusement, une gestion réaliste implique l'ajout d'un nouveau paramètre T dans le schéma GWW-LS.
- *La marche n'est plus simplement aléatoire.*
  Les deux boucles `for` de l'algorithme 1 décrivent la marche : chaque particule effectue une marche de longueur S. La différence principale avec les marches utilisées dans les versions précédentes [Carson et Impagliazzo, 1999; Dimitriou, 2002] réside dans la fonction `Cherchervoisin`.

Dans GWW, cette fonction renvoie un voisin de la configuration de la particule p. Plus précisément, tous les voisins (ou un nombre limité) sont visités jusqu'à ce qu'un soit trouvé sous ou au niveau du seuil. Si aucun n'est trouvé, la configuration courante est renvoyée, c.à.d. la particule ne bouge pas. Dans GWW-LS, cette fonction essaye de faire progresser la particule, tout en s'échappant des minimums locaux.

2. Dans la version initiale [Aldous et Vazinari, 1994], une particule est simplement déplacée sur un fils.

```
algorithme GWW-LS (B : nombre de particules; S : longueur de la marche, T : paramètre
seuil; WP : paramètres de la marche) : configuration
  Initialisation : chaque particule est aléatoirement placée sur une configuration et rangée
  dans le tableau Particules
  seuil ← coût plus mauvaise particule
  Boucle
    seuil ← Baisserseuil(seuil, T, Particules)
    si Meilleur(Particules) > seuil alors retourner(Meilleur-trouvé) /* la
    meilleure configuration trouvée durant la recherche */
    Redistribuer(Particules) /* Place chaque particule ayant un coût supérieur au
    seuil sur la configuration d'une particule vivante choisie aléatoirement */
    pour tout particule p dans Particules faire
      pour i de 1 à S faire
        voisin ← Cherchervoisin(p, WP)
        Déplacer(p,voisin) /* Mouvement de p sur une nouvelle configuration */
      fin.
    fin.
  fin.
fin.
```

Algorithme 1: L'algorithme GWW-LS

4 De GWW-LS à GWW-grw

Afin de valider le schéma GWW-LS, nous devons concevoir au moins un algorithme déduit de GWW-LS et montrer son efficacité. Nous avons essayé divers algorithmes classiques de recherche locale pour améliorer GWW. Nous avons obtenu de bons résultats avec un algorithme de Metropolis (acceptant toujours un voisin d'un coût inférieur ou égal, et acceptant un voisin plus mauvais avec une probabilité dépendant d'une "température"). Cependant, un important travail nous a conduits à le simplifier en diminuant le nombre de paramètres et en trouvant comment les régler plus rapidement.

L'algorithme GWW gère deux paramètres : le nombre B de particules et la longueur S de la marche aléatoire. En pratique, un troisième paramètre T doit être ajouté pour que le seuil ne baisse pas trop lentement. Notre nouvel algorithme GWW-grw (GWW avec *marche aléatoire gloutonne*) ajoute seulement un paramètre N pour la recherche locale. Cet unique paramètre est utilisé par une particule pour progresser vers les meilleures solutions tout en étant capable de s'échapper des minimums locaux. Le paramètre N est le nombre maximum de voisins qui sont explorés par une particule pendant l'exécution de la fonction Cherchervoisin,en vue de bouger d'une configuration à une autre. A partir d'une configuration courante x, la détermination de la nouvelle configuration retournée par la fonction Cherchervoisin de GWW-grw s'effectue de la manière suivante :

1. un voisin x' parmi les N explorés a un coût inférieur ou égal au coût de x $\Rightarrow$ **Retourner** x'
2. aucun voisin parmi les N explorés n'a un coût inférieur ou égal au coût de x **et** un voisin x'' parmi les N est au niveau du seuil ou en dessous $\Rightarrow$ **Retourner** x''
3. Les N voisins explorés sont au dessus du seuil $\Rightarrow$ **Retourner** x

Le premier cas diffère radicalement de GWW standard. Il reproduit le comportement des algorithmes de descente comme GSAT [Selman *et al.*, 1992] : un voisin est accepté seulement si son coût est meilleur ou le même que celui de la configuration courante. Le deuxième cas indique comment choisir une configuration quand aucun voisin visité n'a été accepté. Il propose une sélection aléatoire d'un voisin qui permet de sortir des minimums locaux. Certaines particules peuvent ainsi remonter assez haut, mais cette remontée est limitée par le seuil.

Ainsi le paramètre N évite de visiter tous les voisins et réalise un compromis entre progresser et sortir des minimums locaux. Cette recherche locale, appelée `grw` [Neveu et Trombettoni, 2003], peut se voir comme une généralisation de la *candidate list strategy Acceptation+* [Glover et Laguna, 1997]. Pour résumer, `GWW-grw` gère les paramètres suivants :

- Un nombre B de particules utilisé pour explorer l'espace de recherche en parallèle : les particules doivent être uniformément distribuées entre les composantes connexes apparaissant dans le graphe de recherche quand le seuil est baissé.
- Une longueur S pour les marches utilisée pour séparer les particules regroupées, et qui permet aux particules de progresser vers de meilleures configurations.
- Un paramètre T utilisé pour gérer le seuil dans les problèmes réels (voir partie 5.2).
- Un paramètre additionnel N qui a un rôle crucial. D'abord, il est nécessaire de limiter le nombre de voisins visités quand le voisinage est grand. Par ailleurs, N doit être assez grand pour permettre aux particules de progresser. Enfin, N doit être suffisamment petit pour permettre aux particules de sortir rapidement des minimums locaux.

Complexité de `GWW-grw`

La complexité en pire cas de `GWW-LS` ou `GWW-grw` reste la même que celle de `GWW`, c.à.d, $O(B \times S \times N \times nT)$, où nT est le nombre de fois où le seuil est baissé, B le nombre de particules, S la longueur d'une marche et N le nombre maximum de voisins explorés par pas d'une marche.

5 Expérimentations

Cette partie présente quelques résultats expérimentaux. Nous sélectionnons un moyen de baisser le seuil. Nous proposons une méthode pour régler les quatre paramètres. Enfin, nous comparons `GWW-grw` avec `GWW` standard d'une part, et des algorithmes de recherche locale Metropolis, recuit simulé, liste taboue et `grw` d'autre part.

5.1 Bancs d'essais, codage, implantation

Nous avons réalisé des expérimentations sur des instances difficiles issues de deux catégories de problèmes modélisés comme des problèmes MAX-CSP pondérés : des instances de coloriage de graphe proposées dans le challenge DIMACS [Morgenstern, 1996], et des problèmes d'affectation de fréquences du CELAR (Centre d'électronique de l'Armement).

Coloriage de graphe

Nous avons sélectionné trois instances difficiles de coloriage de graphes du catalogue DIMACS : `le450_15c` avec 450 sommets et 16680 arêtes, `le450_25c` avec 450 sommets et 17425 arêtes, et l'instance plus dense `flat300_28` avec 300 sommets et 21695 arêtes. Dans toutes ces instances, une solution coloriable en respectivement, 15, 25 et 28 couleurs est cachée. Dans cet article, les instances de coloriage sont modélisées en MAX-CSP : les variables sont les sommets du graphe à colorier; le nombre d de couleurs donne des domaines allant de 1 à d; les sommets reliés par une arête doivent prendre des couleurs différentes. Colorier un graphe en d couleurs revient à minimiser le nombre de contraintes non satisfaites et à trouver une solution de coût 0.

Affectation de fréquences du CELAR

Nous avons aussi sélectionné les trois instances les plus difficiles des problèmes d'affectation de fréquences du CELAR `celar6`, `celar7` et `celar8` Ces instances sont réalistes car elles ont été construites à partir de différentes parties d'un problème réel. `celar6` a 200 variables et 1322 contraintes; `celar7` a 400 variables et 2865 contraintes; `celar8` a 916 variables et 5744 contraintes.

Les variables sont les fréquences à affecter avec une valeur appartenant à un ensemble discret prédéfini (domaines de taille d'environ 40). Les contraintes sont de la forme : $|x_i - x_j| = \delta$ ou $|x_i - x_j| > \delta$ (pour éviter les interférences). Notre modélisation est standard et crée seulement les variables paires dans le CSP avec les inégalités uniquement [3].

La fonction objectif à minimiser est une somme pondérée des violations de contraintes. Les contraintes appartiennent à quatre catégories avec différents coûts de violation :

- Les contraintes de `celar8` ont un poids 1, 2, 3 ou 4.
- Les contraintes de `celar6` ont un poids 1, 10, 100 ou 1000.
- Les contraintes de `celar7` ont un poids 1, 100, 10000 ou 10^6.

Voisinage

Nous avons choisi la définition habituelle de voisinage pour les CSP : une nouvelle configuration x' est un voisin de la configuration courante x si les deux ont les mêmes valeurs, sauf pour une variable. Ce voisinage est réduit en utilisant en partie l'heuristique *Min-conflicts* de Minton [Minton *et al.*, 1992] : on ne considère que les variables dont la valeur courante participe à un conflit dans la configuration x.

Implantation

Tous les algorithmes ont été développés dans la même bibliothèque logicielle INCOP [Neveu et Trombettoni, 2003]. Notre plate-forme est implantée en C++ et toutes les expérimentations ont été effectuées sur `PentiumIII 935 Mhz` sous Linux. Tous les algorithmes appartiennent à une hiérarchie de classes partageant du code, ce qui permet des comparaisons équitables.

3. Une bijection existe entre variables paires et impaires. Une simple propagation des égalités permet de déduire les valeurs des variables impaires.

5.2 Gestion du seuil

Nos premières expérimentations ont montré que la baisse du seuil de 1 à chaque itération ne convenait pas pour les problèmes du CELAR, pour lesquels l'intervalle des coûts possibles est très grand. Même pour le coloriage où le paysage est plus “plat”, nous avons obtenu de meilleurs résultats en gérant avec attention la baisse du seuil. Plusieurs gestions du seuil ont été conçues. Elles diffèrent l'une de l'autre par le delta (en terme de coût) à soustraire au seuil courant à chaque itération :

1. $\Delta_{aveugle} = 1 + \text{seuil} \times T_{aveugle}$
2. $\Delta_{pire} = 1 + \text{dist}(B) + \text{seuil} \times T_{pire}$
3. $\Delta_{adaptatif} = 1 + \text{dist}(i) \quad (= 1 + \text{dist}(B) + 100\%(\text{coût}(B) - \text{coût}(i)))$
4. $\Delta_{meilleur} = 1 + \text{dist}(B) + T_{meilleur}(\text{coût}(B) - \text{coût}(1))$
5. $\Delta_{borneinf} = 1 + \text{dist}(B) + (\text{seuil} - B_{borneinf}) \times T_{borneinf}$

$T_{aveugle}$, T_{pire}, $T_{meilleur}$, $T_{borneinf}$ ou i est le paramètre à régler pour rendre la baisse du seuil plus ou moins rapide. Les quatre premiers paramètres sont des taux; dist(i) est la distance (c.à.d., la différence de coût) entre le seuil et la $i^{\text{ème}}$ meilleure particule; ainsi, B est la particule la plus mauvaise et dist(B) est la distance (en coût) entre le seuil et cette particule B. coût(j) est le coût de la $j^{\text{ème}}$ meilleure particule.

Le premier delta essayé a été $\Delta_{aveugle}$. Les autres deltas sont adaptatifs et tiennent compte de la manière dont les particules descendent. Ils essayent de remédier à deux mauvaises caractéristiques de $\Delta_{aveugle}$: $\Delta_{aveugle}$ est souvent trop petit au début et trop grand à la fin. En effet :

- Lors des premières marches, les particules progressent facilement et même la plus mauvaise particule est très en dessous du seuil. Le composant dist(B) des 4 derniers deltas est surtout utile dans ce cas.
- A la fin, quand le seuil se rapproche de la meilleure configuration trouvée, le delta reste grand si le coût de la meilleure solution est supérieur à 0 (c'est le cas pour les problèmes du CELAR).

  $B_{borneinf}$ est un second paramètre nécessaire pour régler $\Delta_{borneinf}$. Il correspond à une borne inférieure dépendant du problème (par exemple 3389 pour `celar6` puisque cette borne a été trouvée par des méthodes complètes). $\Delta_{borneinf}$ a été testé pour vérifier l'intérêt d'avoir un seuil baissant lentement à la fin.

Les conclusions des tests décrits plus bas sont les suivantes. Premièrement, il est généralement intéressant de forcer le seuil à descendre sous la plus mauvaise particule (pour effectuer au moins un regroupement). La seconde conclusion contredit nos premières intuitions : faire attention à la façon dont le seuil décroît à la fin n'est pas utile. C'est pourquoi nous avons adopté la simple gestion du seuil Δ_{pire}.

$\Delta_{adaptatif}$ tend à converger rapidement vers 1, même quand on est encore loin des meilleures solutions. En effet, quand progresser devient difficile, les particules ne sont pas uniformément distribuées (en terme de coût) entre le seuil et la meilleure particule, de plus en plus de particules restent au seuil et dist(i) devient nul.

Pour pallier cette difficulté, $\Delta_{meilleur}$ utilise la plus grande distance adaptative possible puisque coût(B) − coût(1) ne devient nul que quand `GWW-LS` termine.

Finalement, les relativement mauvais résultats obtenus par une gestion avec $\Delta_{meilleur}$ par rapport au simple $\Delta_{aveugle}$ sur des instances avec des paysages de recherche chahutés nous ont conduits à simplement corriger ce dernier en prenant en compte la particule la pire pour accélérer le début de l'algorithme et nous avons conçu et adopté Δ_{pire}. Les séries de tests conduisant à ce choix sont rassemblés dans le tableau 1.

	temps	**aveugle**	**adaptatif**	**meilleur**	**pire**	**borneinf**
`le450_15c`	103	61 (44)	4.2 (42)	0.9 (42)	**0.2 (41)**	–
`le450_25c`	70	11.4 (29)	10.2 (32)	10.1 (31)	10.1 (31)	–
`flat300_28`	112	3.7 (48)	3.4 (48)	3.6 (48)	3.7 (49)	–
`celar8`	140	303 (103)	335 (123)	306 (99)	**289 (106)**	303 (104)
`celar6`	102	3504 (49)	3515 (64)	3557 (43)	**3451 (49)**	3537 (41)
`celar7`	135	372125 (138)	627774 (152)	422616 (135)	377214 (141)	**370784 (138)**

TAB. 1 – *Gestion du seuil. Les cases donnent le coût moyen d'une solution obtenu par* `GWW-grw` *pour un temps d'exécution donné en secondes (deuxième colonne); entre parenthèses, le nombre moyen de modifications de seuil. Les meilleurs résultats sont en gras.*

Pour chaque instance, 10 essais ont été réalisés et les moyennes sont inscrites sur le tableau. Les instances du CELAR ont été traitées avec $B = 50$ ($B = 200$ pour `celar6`), $S = 200$, $N = 40$. Les instances de coloriage ont été résolues avec $B = 20$, $S = 2000$ et différentes valeurs pour N.

Les tests sur `le450_25c` et `flat300_28` ne sont pas significatifs (n'importe quel schéma de baisse du seuil convient). $\Delta_{adaptatif}$ est souvent le pire et $\Delta_{borneinf}$ donne généralement de plus mauvais résultats que Δ_{pire}. Enfin, Δ_{pire} n'est jamais mauvais et est souvent le meilleur.

5.3 Réglage des paramètres

Un avantage de `GWW-grw` est que chacun de ses quatre paramètres semble avoir un rôle spécifique. Cependant, trois de ces quatre paramètres ne sont pas faciles à régler a priori. En effet, réduire T ou augmenter B ou S donne toujours des résultats de meilleure qualité en allongeant le temps de calcul. Il n'est pas évident de sélectionner quelle augmentation de paramètre améliorera beaucoup la solution sans coûter trop de temps. Cependant, un réglage très fin semble inutile et nous avons obtenu des procédures de réglage prometteuses. Nos expérimentations et la littérature nous ont conduits à régler les paramètres de la manière heuristique suivante :

1. **T:** Le comportement asymptotique de T est le plus simple à observer. Dans nos expérimentations, les taux T_{pire} et $T_{aveugle}$ sont toujours choisis entre 0.5% et 2%; le taux $T_{meilleur}$ entre 0.5% et 10%. Des taux plus forts conduisent à de très mauvais résultats et des taux plus faibles n'améliorent pas les résultats tout en perdant beaucoup de temps. Ainsi, en commençant avec un taux de 1%, quelques essais (typiquement 3 ou 4) sont suffisants pour sélectionner une valeur acceptable[4].
2. **N:** Les expérimentations décrites ci-après montrent que la valeur de N a un impact significatif sur la qualité des résultats. De nouveau, quelques essais avec de petites valeurs pour B et S suffisent à déterminer une valeur de N acceptable pour une instance donnée.

4. De petites valeurs pour B, S et N peuvent être choisies à cette étape, par exemple $B = 20$, $S = 200$, $N = 50$.

3. **S:** Suivant l'étude faite dans [Dimitriou, 2002], les tests décrits plus bas nous permettent de déterminer une longueur de marche suffisamment longue pour explorer les sous-composantes connexes du graphe de recherche
4. **B:** Une fois les paramètres précédents réglés, B peut être augmenté jusqu'à ce qu'une bonne solution soit obtenue.

Bien sûr, cette étude doit être considérée comme une première approche. Nos expérimentations nous laissent à penser que T et N sont faciles à régler. Une meilleure compré/-hension du comportement de `GWW-grw` nous permettrait de conforter ou de modifier les moyens de régler S et B.

Réglage de N

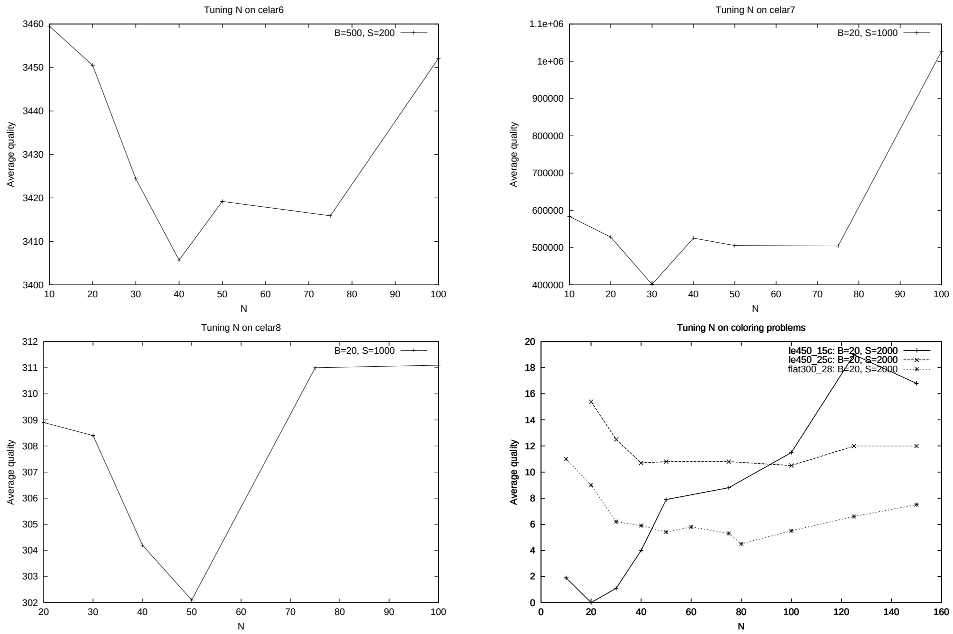

FIG. 1 – *Réglage du paramètre N. Les valeurs choisies pour les paramètres B et S sont indiquées sur les figures.*

La figure 1 montre les courbes obtenues pour le réglage du paramètre N sur les différentes instances. La valeur de N est donnée en abscisse et les ordonnées donnent le coût de la meilleure solution (moyenne sur 10 essais). Les conclusions sont les suivantes :

- Il n'est pas difficile de régler N puisque les courbes ont un minimum. En effet, si N est trop petit, les particules ne peuvent pas progresser; si N est trop grand, les particules ne peuvent pas sortir des minimums locaux.
- Régler N a un impact très significatif sur la performance. En particulier, il permet à `GWW-grw` de colorier toujours en 15 couleurs `le_450_15c` en 2 minutes. Seule l'instance `celar7` ne semble pas très sensible à ce paramètre.

Nous devons mentionner que, pour toutes les instances testées, les courbes obtenues avec différents paramètres B et S ont à peu près la même forme et donnent la même meilleure valeur pour N. Cela semble signifier que pour l'algorithme `GWW-grw`, la valeur de N est intrinsèque à un paysage de recherche donné (instance, voisinage) et ne dépend pas des autres paramètres.

Régler S

Nous avons appliqué la technique décrite dans [Dimitriou, 2002] pour `GWW`. Dans cet article, des tests avaient été effectués avec différentes valeurs de S mais avec une valeur constante de $B \times S$: la valeur de S donnant la meilleure solution (en moyenne) est choisie. Cette approche est justifiée par l'intuition suivante. Dans `GWW`, S doit être suffisamment long pour permettre aux particules d'être uniformément distribuées sur les sous-composantes du graphe de recherche. Ainsi, quand cette longueur "minimum" de marche a été atteinte, il n'y a plus d'intérêt à continuer la marche.

Cette situation est plus compliquée pour `GWW-grw` où S est utilisé pour diffuser les particules dans les composantes connexes mais aussi pour les faire progresser vers de meilleures configurations. De plus, la technique décrite ci-dessus a été appliquée pour `GWW` qui a seulement deux paramètres B et S, alors que notre gestion du seuil introduit un troisième paramètre (monotone) T (si on suppose que N a déjà été réglé et fixé).

Nous avons malgré tout appliqué cette approche de réglage à `GWW-grw`, et obtenu les résultats suivants :

1. Pour les instances du CELAR, le meilleur coût est obtenu quand S est compris entre 200 et 400, et la même valeur est obtenue pour différentes valeurs de $B \times S$. Ceci conduit à adopter une grande valeur de B (plusieurs milliers) et une petite valeur de S (300) dans les tests suivants.
2. Pour les instances de coloriage de graphe, le comportement opposé a été observé. Les meilleurs résultats sont toujours obtenus avec un petit nombre de particules, par exemple 20, et une longue marche.

Ces tests sont une bonne indication de la pertinence de l'utilisation de `GWW-grw`. L'intérêt de gérer plusieurs particules en parallèle est clair dans le premier cas, alors qu'une recherche locale peut obtenir de meilleurs résultats dans le second.

5.4 Performance de `GWW-grw`

Le tableau 2 rend compte des meilleurs résultats obtenus par `GWW-grw` sur les instances étudiées.

	Meilleur algo	**borne**	**temps**	`GWW-grw`	**temps**	**succès**	**B, S, N**
`le450_15c`	Mo93,CO99	15 coul	mn	15 coul (0.2)	1.1 mn	8/10	20,2000,20
`le450_25c`	Mo93	25 coul	mn	25 coul (1.4)	2.3 h	1/10	20,800000,100
`flat300_28`	Mo93,Do98,Ga99	31 coul	heures	31 coul (1.3)	5.9 mn	2/10	20,20000,80
`celar6`	Gi97,Vo98,Ko99	3389	mn/h	3389 (3405.7)	9.2 mn	4/10	500,200,40
`celar7`	Vo98,Ko99	343592	mn	343596 (343657)	4.5 h	1/10	5000,300,50
`celar8`	Vo98,Ko99	262	mn	262 (267.4)	33.7 mn	2/10	1000,200,30

TAB. 2 – *Meilleurs résultats. Les résultats des meilleurs algorithmes connus sont inscrits sur la partie gauche du tableau; les résultats de* `GWW-grw` *sur la partie droite, les coûts moyens étant entre parenthèses.*

Le graphe `flat300_28` n'a jamais été colorié en 28 couleurs. A notre connaissance, seulement trois algorithmes incomplets ont réussi à le colorier en 31 couleurs : la version distribuée de l'algorithme `Impasse` [Morgenstern, 1996], une recherche avec liste taboue [Dorne et Hao, 1998], et une heuristique hybridant un algorithme génétique avec de la recherche locale [Galinier et Hao, 1999].

[de Givry *et al.*, 1997] et [Koster *et al.*, 1999] décrivent des algorithmes complets basés une décomposition du graphe de contraintes et de la programmation dynamique qui ont pu résoudre `celar6`. Les deux autres instances sont réellement difficiles et n'ont jamais été résolues par un algorithme complet [5]. L'algorithme décrit dans [Kolen, 1999] est basé sur un algorithme génétique spécialisé avec un opérateur de mutation codant de la programmation linéaire en nombres entiers. L'algorithme décrit dans [Voudouris et Tsang, 1998] est une recherche locale prenant en compte les conflits pour sélectionner les voisins.

Le tableau 2 montre les très bons résultats obtenus par `GWW-grw`. Les meilleures bornes connues sont obtenues pour toutes les instances testées, sauf pour `celar7` où elle est approchée de près. De plus, le temps requis par `GWW-grw` est souvent très satisfaisant. A notre connaissance, il est parmi les meilleurs algorithmes pour résoudre `flat300_28`, `le450_15c` et `celar6`. Pour les trois autres instances, le temps requis par `GWW-grw` est plus grand (des heures contre des minutes pour les meilleurs algorithmes). Cependant, on peut noter que `GWW-grw` n'utilise aucune technique ou modélisation spécifiques au type de problème, contrairement aux algorithmes décrits dans [Morgenstern, 1996; Kolen, 1999].

5.5 Comparaisons entre `GWW-grw`, `GWW` et la recherche locale

Le tableau 3 rassemble les comparaisons entre `GWW-grw`, `GWW` et différents algorithmes de recherche locale (Metropolis, le recuit simulé `RS`, la méthode de la liste taboue `tabou`[Glover et Laguna, 1997]) sur les instances choisies. La ligne `GWW'` rend compte des résultats de `GWW` avec la même gestion du seuil que `GWW-grw`. La ligne `grw` donne les résultats obtenus avec l'algorithme de recherche locale grw utilisé seul (en dehors du schéma GWW et donc sans notion de seuil).

Rappelons que tous les algorithmes de recherche locale ont été implantés dans le même logiciel [Neveu et Trombettoni, 2003] et partagent la plupart du code avec `GWW-grw` et `GWW`.

Notre Metropolis est standard. Il commence avec une configuration aléatoire et une marche de longueur S est effectuée comme suit :

- Un nouveau voisin est accepté si son coût est inférieur ou égal au courant.
- Un nouveau voisin dont le coût est plus mauvais que le courant est accepté avec une probabilité dépendant d'une température donnée, constante pour Metropolis [Connolly, 1990].
- Quand aucun voisin n'est accepté, la configuration n'est pas changée dans l'itération courante.

Le recuit simulé `RS` suit le même schéma, avec une température diminuant au cours de la recherche. Il a été implanté avec une baisse linéaire de température à partir d'une température initiale donnée en paramètre.

La méthode taboue a été implantée de la manière classique suivante : une liste taboue des derniers mouvements est constituée et on s'interdit de réaffecter une variable

5. Ceci peut être expliqué par la taille de `celar8` et par le critère de `celar7`.

avec une valeur qu'on vient d'enlever. Le critère d'aspiration est appliqué quand on a trouvé une configuration meilleure que la meilleure trouvée jusqu'à présent. On choisit comme mouvement le meilleur non tabou dans la portion du voisinage exploré. Les deux paramètres de cet algorithme sont donc la longueur de la liste taboue (fixe) et la taille du voisinage exploré.

L'algorithme de recherche locale `grw` est le même que celui utilisé par chaque particule dans l'hybridation `GWW-grw`. La seule différence est qu'il n'y a plus de seuil et que tous les mouvements sont donc "faisables". On choisit le premier voisin d'un coût inférieur ou égal à la configuration courante si on en trouve un dans une portion du voisinage de taille N, un voisin quelconque sinon.

	le15c	**le25c**	**flat28**	**celar6**	**celar7**	**celar8**
Nb coul	15	25	31			
Temps	2	14	9	14	6	50
Metrop.	5.9 (2)	**3.1** (2)	0.9 **(0)**	5048 (3906)	$6\,10^6$ ($2.9\,10^6$)	410 (300)
RS	9.6 **(0)**	5.8 (4)	1.8 **(0)**	4167 (3539)	$1.2\,10^6$ (456893)	281 (264)
tabou	1.5 **(0)**	3.7 (3)	2.5 (1)	4183 (3935)	$1.2\,10^6$ (620159)	373 (315)
`grw`	0.5 **(0)**	**3.1 (1)**	**0.8 (0)**	3447 **(3389)**	373334 (343998)	291 (273)
`GWW'`	536 (410)	17.1 (14)	6.6 (6)	3648 (3427)	583278 (456968)	276 (265)
`GWW-grw`	**0 (0)**	4 (3)	1.3 **(0)**	**3405 (3389)**	**368452 (343600)**	**267 (262)**

TAB. 3 – *Comparaisons entre algorithmes. La première ligne indique le nombre de couleurs essayées pour les problèmes de coloriage, la deuxième ligne le temps (en minutes) alloué par essai. Chaque case contient le coût moyen des solutions (sur 10 ou 20 essais) ; le meilleur coût sur ces essais apparaît entre parenthèses.*

Toutes les instances ont été résolues avec le même voisinage. Bien sûr, changer aussi le voisinage peut changer les performances relatives des algorithmes sur une instance donnée, par exemple suivre l'heuristique de Minton de minimisation des conflits pour le choix de valeur améliore le comportement de Metropolis sur `flat300_28`.

Les résultats de nos tests peuvent être interprétés comme suit. D'abord, les résultats de `GWW` standard (avec un seuil baissé de 1 à chaque itération) sont très mauvais et n'ont pas été inscrits dans le tableau. Ensuite, `GWW'` donne de mauvais résultats sur les instances de coloriage de graphe et est toujours plus mauvais que `GWW-grw`. Ses résultats sont moins mauvais sur les problèmes du CELAR, spécialement pour `celar8`.

Les algorithmes de recherche locale ont été utilisées avec leurs propres paramètres (la température pour Metropolis, la température initiale pour le recuit simulé et la taille du voisinage pour `grw`, la taille de la liste taboue et la taille du voisinage pour la méthode taboue) qui ont été réglés pour obtenir les meilleurs résultats possibles.

Metropolis a obtenu de mauvais résultats sur les problèmes du CELAR. Pour `celar6` et `celar7`, ceci peut être expliqué par les poids différents associés aux contraintes et pour lesquels une température unique constante ne serait pas adéquate. Le recuit simulé avec une baisse linéaire de température donne de meilleurs résultats sur ces problèmes. Metropolis semble meilleur sur le coloriage de graphes. Il n'est pas mauvais sur `le450_15c` et bon sur `le450_25c` et `flat300_28`, où il surpasse légèrement `GWW-grw`.

L'algorithme de recherche locale `grw` donne de bons résultats sur l'ensemble des problèmes étudiés. Ceci confirme le fait que pour les problèmes de coloriage de graphe traités, un tel algorithme suffit et la mécanique de `GWW-grw` est alors inutile. Par contre, pour les problèmes du CELAR, `GWW-grw` améliore de manière significative les résultats de `grw`.

Sur ces problèmes, nous avons également vérifié que plusieurs marches `grw` assez courtes menées sans aucune interaction (comme pour GSAT) ne donnaient pas de bons résultats. Ces essais sont dénommés `grw-restart` dans le tableau 4. Nous avons pour cela effectué K essais de longueur L avec $K = 20 * B$ et $L = S * Nb\_changement\_seuil$ pour donner la même longueur de marche totale par particule et le même temps global que les 20 essais de `GWW-grw` (10 pour celar8). La seule comparaison valide est alors la meilleure solution obtenue durant la recherche. Le paramètre de voisinage de `grw-restart` est celui optimal pour `grw`.

	temps total (mn)	`grw-restart`	`GWW-grw`
`celar6`	280	3420	**3389**
`celar7`	120	364522	**343600**
`celar8`	500	294	**262**

TAB. 4 – *Comparaisons entre* `grw-restart` *et GWW-grw. La première colonne indique le temps total alloué. Chaque case contient le meilleur coût trouvé.*

Pour conclure, `GWW-grw` est le meilleur pour 4 des 6 instances et semble toujours donner de bons résultats (sur les 6 instances étudiées). L'écart type sur les coûts obtenus par `GWW-grw` est généralement très faible, ce qui indique que l'algorithme est robuste et justifie le fait de n'avoir effectué que 10 essais par jeu de paramètres. Finalement, les résultats semblent confirmer qu'utiliser `GWW-grw` est pertinent quand l'utilisateur peut facilement trouver une longueur de marche S minimisant le coût en maintenant constant $B \times S$ (voir fin de 5.3).

6 Travaux connexes

`GWW` peut être vu comme une version simplifiée (sans opérateur de croisement) d'un algorithme génétique. Cet opérateur est souvent très coûteux et ne permet pas une évaluation incrémentale des coûts dans le cas des problèmes MAX-CSP. Au contraire, les mouvements élémentaires dans `GWW-LS` sont de la recherche locale si bien qu'une évaluation incrémentale des coûts peut facilement être implantée. Certaines études comme [Baluja et Caruana, 1995] avaient déjà essayé d'enlever l'opérateur de croisement dans les algorithmes génétiques. L'hybridation avec de la recherche locale a aussi été essayée avec les algorithmes génétiques et a souvent grandement amélioré leurs résultats : on parle alors d'algorithmes mémétiques [Moscato, 1999].

L'algorithme d'acceptation à seuil [Dueck et Scheuer, 1990] et l'algorithme σ [Carson et Impagliazzo, 1999] sont des métaheuristiques de recherche locale qui utilisent un seuil. Dans ces deux algorithmes, une marche aléatoire est effectuée et un mouvement est accepté si la nouvelle configuration ou le delta ne dépassent pas le seuil courant, qui est réduit tout au long de l'algorithme.

La technique de regroupement *clustering* est principalement utilisée pour de l'optimisation globale sur les réels [Törn, 1973], où elle obtient de très bons résultats. Elle gère une population et des mécanismes existent pour éviter le regroupement de “particules”.

7 Conclusion

Nous avons conçu un algorithme hybride nommé `GWW-LS` introduisant de la recherche locale dans l'algorithme Go With the Winners. Une instance de ce schéma, nommé `GWW-grw`, a été défini. `GWW-grw` gère 4 paramètres et une première tentative de réglage a été décrite. La gestion du seuil a été étudiée sur des problèmes réalistes. Le nombre maximum N de voisins visités permet à `GWW-grw` de progresser vers les meilleures solutions tout en sortant des minimums locaux. Ce paramètre est crucial en pratique et semble pouvoir être réglé de manière indépendante.

`GWW-grw` a obtenu de très bons résultats sur le coloriage de graphe et sur les problèmes du CELAR. Seules, quelques autres heuristiques d'optimisation peuvent atteindre de telles bornes sur les instances testées. `GWW-grw` n'utilise par ailleurs aucune technique spécifique au type de problème.

Bien sûr, `GWW-grw` devrait être essayé sur d'autres types de problèmes, par exemple des instances contenant des contraintes dures.

Les bons résultats expérimentaux obtenus par `GWW-grw` confirment que l'idée utilisée par les algorithmes génétiques consistant à explorer l'espace de recherche en parallèle peut être intéressante. Cependant, le fait que `GWW-grw` ne possède pas d'opérateur de croisement apporte plusieurs avantages :

- L'algorithme est plus facile à régler.
- Puisque le concept de voisinage de la recherche locale est préservé, des structures de données incrémentales peuvent être utilisées quand on passe d'une configuration à son voisin.
- L'algorithme peut être développé sur la même plate-forme logicielle que les algorithmes de recherche locale.

Comme mentionné à la fin de la partie 2, la descente gloutonne de `GWW-grw` viole la distribution uniforme des particules dans une région connexe (distribution sur laquelle est bâtie la propriété d'expansion). Ainsi, un point intéressant serait d'établir de manière théorique ou expérimentale une autre condition suffisante de succès de `GWW-grw`, par exemple, la distribution uniforme des particules *à chaque niveau de coût* des composantes connexes. Ceci indiquerait l'intérêt d'utiliser une recherche locale dans `GWW-LS` où les particules ne seraient pas piégées dans les minimums locaux.

Références

[Aldous et Vazinari, 1994] D. Aldous et U. Vazinari. Go with the winners algorithms. Dans *Proc. STOC*, 1994.

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

[Carson et Impagliazzo, 1999] Ted Carson et Russell Impagliazzo. Experimentally determining regions of related solutions for graph bisection problems. Dans *IJCAI, workshop ML1: machine learning for large scale optimization*, 1999.

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

[de Givry *et al.*, 1997] S. de Givry, G. Verfaillie, et T. Schiex. Bounding the optimum of constraint optimization problems. Dans *Proc. CP97*, number 1330 in LNCS, 1997.

[Dimitriou et Impagliazzo, 1996] Tassos Dimitriou et Russell Impagliazzo. Towards an analysis of local optimization algorithms. Dans *Proc. STOC*, 1996.

[Dimitriou et Impagliazzo, 1998] Tassos Dimitriou et Russell Impagliazzo. Go with the winners for graph bisection. Dans *Proc. SODA*, pages 510–520, 1998.

[Dimitriou, 2002] Tassos Dimitriou. Characterizing the space of cliques in random graphs using "go with the winners". Dans *Proc. AMAI*, 2002.

[Dorne et Hao, 1998] Raphaël Dorne et Jin-Kao Hao. Tabu search for graph coloring, t-colorings and set t-colorings. Dans I.H. Osman S. Voss, S.Martello et C. Roucairol, editors, *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pages 77–92. Kluwer Academic Publishers, 1998.

[Dueck et Scheuer, 1990] G. Dueck et T. Scheuer. Threshold accepting : a general purpose algorithm appearing superior to simulated annealing. *Journal of Computational Physics*, 90:161–175, 1990.

[Galinier et Hao, 1999] Philippe Galinier et Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[Glover et Laguna, 1997] F. Glover et M. Laguna. *Tabu Search.* Kluwer Academic Publishers, 1997.

[Kolen, 1999] A. Kolen. A genetic algorithm for frequency assignment. Technical report, Universiteit Maastricht, 1999.

[Koster *et al.*, 1999] A. Koster, C. Van Hoesel, et A. Kolen. Solving frequency assignment problems via tree-decomposition. Technical Report 99-011, Universiteit Maastricht, 1999.

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

[Morgenstern, 1996] Craig Morgenstern. Distributed coloration neighborhood search. Dans David S. Johnson et Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26, pages 335–357. American Mathematical Society, 1996.

[Moscato, 1999] Pablo Moscato. Memetic algorithms : a short introduction. Dans Marco Dorigo David Corne et Fred Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw Hill, 1999.

[Neveu et Trombettoni, 2003] Bertrand Neveu et Gilles Trombettoni. Incop: An Open library for INcomplete Combinatorial OPtimization. Dans *Proc. of CP'03*, LNCS, 2003.

[Selman *et al.*, 1992] B. Selman, H. Levesque, et D. Mitchell. A new method for solving hard satisfiability problems. Dans *Proc. AI'92*, pages 440–446, 1992.

[Törn, 1973] A.A. Törn. Global optimization as a combination of global and local search. Dans *Proc. of Computer Simulation vs Analytical Solutions for Business and Economic Models*, pages 191–206, 1973.

[Voudouris et Tsang, 1998] Christos Voudouris et Edward Tsang. Solving the radio link frequency assignment problem using guided local search. Dans *Nato Symposium on Frequency Assignment,Sharing and Conservation in Systems(AEROSPACE)*, 1998.