

Évaluation d'approches complètes pour le problème de somme coloration

Maël Minot, Samba Ndojh Ndiaye, Christine Solnon

▶ **To cite this version:**

Maël Minot, Samba Ndojh Ndiaye, Christine Solnon. Évaluation d'approches complètes pour le problème de somme coloration : Version étendue. Douzièmes Journées Francophones de Programmation par Contraintes (JFPC 2016), Jun 2016, Montpellier, France. pp.1-10. hal-01309350

HAL Id: hal-01309350
https://hal.science/hal-01309350

Submitted on 29 Apr 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Évaluation d'approches complètes pour le problème de somme coloration

Version étendue

Maël Minot[1,2] **Samba Ndojh NDIAYE**[1,3] **Christine SOLNON**[1,2]

[1] Université de Lyon - LIRIS
[2] INSA-Lyon, LIRIS, UMR5205, F-69621, France
[3] Université Lyon 1, LIRIS, UMR5205, F-69622 France

{mael.minot,samba-ndojh.ndiaye,christine.solnon}@liris.cnrs.fr

Résumé

Le problème de somme coloration est un problème de coloration de graphes dans lequel la somme des couleurs utilisées pour chaque sommet doit être minimisée. Peu d'approches complètes ont été proposées à ce jour, et les résultats sont encore largement perfectibles. Cet article évalue les capacités de la programmation par contraintes pour ce problème. Nous étudions différents paramètres et proposons des améliorations visant à mieux adapter la programmation par contraintes à la somme coloration.

Abstract

The sum colouring problem is a graph colouring problem in which the sum of vertex colours must be minimized. Only a few complete approaches were evaluated thus far. This paper evaluates the capabilities of constraint programming to solve this problem. We study several parameters and suggest improvements to better adapt constraint programming to this problem.

1 Introduction

Le problème de somme coloration minimale (MSCP) est une variante $\mathcal{NP}$-difficile du problème de coloration de graphes qui consiste à trouver une coloration minimisant *la somme des couleurs* plutôt que le nombre de couleurs. Ce problème survient notamment dans les contextes d'allocation de ressources et de planification. Le MSCP n'a, à ce jour, pas beaucoup été étudié, et les méthodes proposées sont principalement des métaheuristiques. Seules quelques méthodes complètes ont été avancées : la programmation linéaire entière [18], le *Branch and Bound*, SAT, et la programmation par contraintes (CP) [11]. Cependant, le modèle CP jusqu'alors utilisé est assez direct et n'a pas donné de résultats probants. La programmation par contraintes souples n'a encore jamais été appliquée à ce problème, bien qu'intuitivement elle semble plus indiquée que CP, puisqu'elle est naturellement capable de gérer la préférence entre solutions.

Dans cet article, nous menons une évaluation plus poussée de CP et de sa contrepartie souple, dans le but d'estimer plus justement leur aptitude à résoudre le MSCP. Nous en profitons pour souligner certains avantages de telles approches, ainsi que pour leur apporter plusieurs améliorations dans ce cadre.

2 Définition du MSCP

Un graphe non orienté G est défini par un ensemble de sommets N_G et un ensemble $E_G \subseteq N_G \times N_G$ d'arêtes. Chaque arête est un couple non orienté de sommets. On note $\deg(v)$ le degré d'un sommet v, *i.e.*, $\deg(v) = |\{u \in V, \{u,v\} \in E\}|$, et $\Delta(G)$ le plus grand degré présent dans le graphe G, *i.e.*, $\Delta(G) = \max\{\deg(v), v \in N_G\}$. Une *clique* est un sous-ensemble de sommets tous liés deux à deux. Elle est dite *maximale* si elle n'est pas strictement incluse dans une autre clique. Un *ensemble de sommets indépendants*, a contrario, est un sous-ensemble de sommets au sein duquel on ne trouve aucune arête.

Une k-coloration *propre* ou *légale* de G est une fonction $\mathrm{c} : V \to \{1, \ldots, k\}$ telle que $\forall \{x,y\} \in E, \mathrm{c}(x) \neq \mathrm{c}(y)$. Une telle coloration peut aussi être définie comme

une partition de V en k ensembles indépendants.

Le problème de coloration de graphe classique vise à trouver une k-coloration propre qui minimise k. Cet article, cependant, traite du problème $\mathcal{NP}$-difficile de *somme coloration*, dont l'objectif est de trouver une k-coloration propre qui minimise la somme des couleurs assignées aux sommets ($\sum_{x \in V} c(x)$) plutôt que k. La plus faible somme que l'on peut obtenir pour un graphe G est appelée *somme chromatique de* G et notée $\Sigma(G)$.

Dans [17], des bornes théoriques pour la somme chromatique sont proposées. Elles sont définies de la manière suivante : $\lceil \sqrt{8|E|} \rceil \leq \Sigma(G) \leq \lfloor \frac{3(|E|+1)}{2} \rfloor$. Par la suite, [10] a également démontré qu'une somme coloration optimale n'utilise jamais $\Delta(G)+1$ couleurs. Une autre borne, parfois utile bien que généralement plus faible, est $\Sigma(G) \leq |V| + |E|$.

La *dominance* de certaines solutions par d'autres a été abordée dans [11]. Concrètement, une coloration peut être vue comme une partition ordonnée des sommets du graphe : le k^{e} ensemble S_k correspond aux sommets utilisant la couleur k. Une solution est *dominée* lorsque la somme des couleurs des sommets peut être réduite en réordonnant les indices des ensembles. En d'autres termes, il est alors possible d'améliorer la solution en échangeant des couleurs. Notez qu'une coloration propre le restera lors d'une telle opération. Une partition donnée peut aisément être associée à une coloration que nulle ne domine : il suffit d'utiliser la couleur 1 pour le plus grand ensemble de sommets, la 2 pour le second, et ainsi de suite.

3 Approches existantes pour le MSCP

3.1 Approches incomplètes

De nombreuses approches incomplètes ont été employées pour trouver des solutions approchées au MSCP. La plupart sont passées en revue dans [7], et sont réparties en trois classes : les algorithmes gloutons [19, 20], les heuristiques de recherche locale [2, 6] et les algorithmes évolutionnaires [8, 13, 21].

Notons qu'aucun de ces algorithmes n'est capable d'atteindre toutes les meilleures bornes inférieures ou supérieures. Cependant, le pourcentage de bornes atteintes se situe entre 46 % et 90 %. [21] Ces approches parviennent à prouver l'optimalité d'une solution dès lors que la borne inférieure la plus basse atteint la borne supérieure la plus haute. Cependant, de telles preuves n'ont été possibles que sur un tiers des instances, même en utilisant conjointement les bornes fournies par toutes les méthodes citées par [7].

3.2 Approches complètes

Programmation par contraintes Un problème de satisfaction de contrainte, ou CSP, est défini par un triplet (X, D, C). X est un ensemble fini de variables, D associe un domaine $D(x_i)$ à chaque variable $x_i \in X$, et C est un ensemble de contraintes. Une solution est une affectation de toutes les variables qui satisfait toutes les contraintes.

Un modèle CSP basique pour le MSCP a été proposé dans [11]. Pour un graphe $G = (V, E)$, ce modèle s'obtient en associant une variable x_k à chaque sommet v_k, auxquelles s'applique une contrainte binaire de différence pour chaque arête du graphe :

— $X = \{x_1, \ldots, x_{|V|}\}$
— $\forall i \in \{1, \ldots, |V|\}, D(x_i) = \{1, \ldots, K\}$
— $C = \cup_{\{v_a, v_b\} \in E} \{x_a \neq x_b\}$

où $K = \Delta(G) + 1$. L'objectif est de minimiser $\sum_{x_i \in X} x_i$.

La programmation par contraintes a été évaluée pour le MSCP dans [11], avec le solveur *Choco* [9]. Les résultats étaient assez décevants, avec notamment un temps de résolution de près de 2 000 secondes sur l'instance simple `myciel4`, que les méthodes compétitives résolvent presque instantanément.

Branch and Bound L'approche *Branch and Bound* décrite dans [11] intègre l'idée présentée dans [14, 20], qui consiste à décomposer le graphe en cliques afin de calculer une borne inférieure. Cette technique est appliquée à chaque nœud de l'arbre, en ne considérant que le sous-graphe induit par les sommets qui n'ont pas encore été coloriés. Ainsi est obtenue une borne inférieure globale sur la branche courante de l'arbre. Si cette borne dépasse la borne supérieure courante, cette branche est abandonnée. Pour chaque coloration trouvée, un équivalent dominant est utilisé pour améliorer la borne, comme expliqué à la fin de la section 2. Cette approche donne de meilleurs résultats que le modèle basique CP, mais ne peut être appliquée avec succès qu'à des instances de taille réduite.

SAT Des solveurs SAT ont été employés pour résoudre le MSCP dans [11]. Les résultats rapportés sont très différents de ceux de CP et *Branch and Bound* et sont de qualité inégale : grande efficacité sur des classes d'instances telles que `miles` et `myciel`, mais incapacité à résoudre `queen5_5` en moins d'une heure, malgré une difficulté très raisonnable.

Programmation linéaire Dans [18], un programme linéaire est proposé pour le MSCP. Il associe une variable binaire x_{uk} à chaque paire $(u, k) \in V \times [1, K]$, où u est un sommet, k une couleur et $K = \Delta(G) + 1$, de sorte que $x_{uk} = 1$ si u est colorié avec k, et 0 sinon.

L'objectif est de minimiser $f(x) = \sum_{u=1}^{|V|} \sum_{k=1}^{K} k \cdot x_{uk}$ sous les contraintes :

— $c_1 : \sum_{k=1}^{K} x_{uk} = 1, \forall u \in \{1, \ldots, |V|\}$

— $c_2 : x_{uk} + x_{vk} \leq 1, \forall (u, v) \in E, \forall k \in \{1, \ldots, K\}$

La contrainte c_1 force les sommets à n'utiliser qu'une couleur, et c_2 empêche les sommets voisins d'employer la même couleur.

L'article proposant ce modèle rapporte de bons résultats mais également des problèmes dus à un manque de mémoire sur certaines instances. [18]

4 Nouveaux modèles pour le MSCP

Notre but est d'explorer de nouvelles possibilités de modèles pour le MSCP. En section 4.1, nous décrivons un modèle CP souple. La section 4.2 considère la possibilité de remplacer certaines contraintes binaires par des contraintes globales *AllDifferent*. La spécification des domaines initiaux des variables est remise en question dans la section 4.3, tandis que les choix d'heuristiques sont évoqués en 4.4 et 4.5. Une méthode de filtrage est décrite en section 4.6, tandis qu'une technique d'échange de couleurs est présentée en 4.7.

4.1 Modèle WCSP

À ce jour, aucun modèle WCSP n'a été proposé pour le MSCP.

Un WCSP est définit par un triplet (X, D, C) tel que X est un ensemble fini de variables, D une fonction associant à chaque variable $x_i \in X$ son domaine $D(x_i)$, et C un ensemble de fonctions de coût chacune définie sur un sous-ensemble de X. Chaque fonction de coût associe un coût à chaque tuple, un tuple étant une combinaison possible de valeurs pour les variables concernées par la fonction de coût.

Afin de traduire le MSCP en WCSP, les contraintes de différences peuvent être représentées par des fonctions de coût binaires, qui associent un coût infini à tout tuple donnant la même valeur aux deux variables impliquées. Il n'y a généralement pas de fonction objectif explicite dans un WCSP : la somme des fonctions de coûts joue ce rôle. Nous pouvons donc ajouter une fonction de coût unaire par variable, qui associe un coût de n au tuple qui donne la couleur n au sommet associé à cette variable.

Plus formellement, pour un graphe $G = (V, E)$, ce modèle WCSP est défini comme suit, en supposant qu'à chaque variable x_k correspond un sommet k :

— $X = \{x_1, \ldots, x_{|V|}\}$

— $\forall k \in \{1, \ldots, |V|\}, D(x_k) = \{1, \ldots, K\}$

— $C = \{f_1(x_u, x_v), \{u, v\} \in E\} \cup \{f_2(x_u), x_u \in X\}$ avec :

  — $f_1(x_u, x_v) = +\infty$ si $x_u = x_v$, et 0 sinon

  — $f_2(x_u) = x_u$

où $K = \Delta(G) + 1$.

L'objectif est de minimiser la somme des fonctions de coût. Elles garantissent que le coût d'une coloration propre équivaut à la somme des couleurs des sommets et interdisent via un coût infini toute coloration invalide.

4.2 Contraintes globales AllDifferent

Les modèles introduits précédemment utilisent des contraintes de différence binaires pour empêcher les sommets voisins d'utiliser la même couleur. Des modèles légèrement plus élaborés peuvent être obtenus en trouvant des ensembles de contraintes binaires de différence correspondant à des cliques et en les remplaçant par des contraintes globales *AllDifferent*. Il existe de nombreuses manières de sélectionner de telles cliques : elles peuvent être maximales ou non, plus ou moins redondantes, et peuvent être construites par diverses heuristiques. Rechercher la plus grande clique, cependant, n'est pas un choix pertinent, car il s'agit d'un problème $\mathcal{NP}$-difficile.

Nous avons évalué plusieurs méthodes de construction de cliques au cours d'une étude non présentée ici. Les résultats nous ont clairement montré qu'il suffisait de construire de nombreuses cliques de manière gloutonne pour obtenir des résultats difficilement perfectibles. Le choix précis d'heuristiques pour la construction de ces cliques n'a qu'un impact réduit sur les temps de résolution. La méthode de construction que nous avons finalement employée est présentée par l'algorithme 1. Pour chaque sommet v, nous construisons une clique maximale de manière gloutonne (lignes 1–8) : nous partons d'une clique composée uniquement de v et y ajoutons itérativement le candidat de plus haut degré (ligne 5). La procédure se termine par l'élimination des cliques incluses dans d'autres et l'ajout de cliques binaires pour toute contrainte non couverte (lignes 9–12).

La contrainte *AllDifferent* permet une représentation bien plus concise des instances. De plus, un solveur peut l'interpréter comme bon lui semble, en utilisant des contraintes globales spécifiques ou des contraintes binaires, selon son implémentation. Ainsi, nous ne souffrons ici d'aucune perte de généralité.

Notre manière de représenter les *AllDifferent* dans le modèle linéaire est inspirée des contraintes qui y figurent déjà. Nous utilisons simplement des contraintes de la forme $x_{1\,c} + \ldots + x_{k\,c} \leq 1$, en citant, pour chaque couleur c, chaque variable existante associée à un sommet appartenant à la clique considérée.

Algorithme 1 : Constructions de cliques.

Data : Graphe $G = (V, E)$
Result : Ensemble S de cliques de G tel que $\forall \{x, y\} \in E, \exists C \in S$ tel que $\{x, y\} \subseteq C$

```
foreach v ∈ V do
    C ← {v}
    Candidates ← voisins de v dans G
    while Candidates ≠ ∅ do
        x ← Sommet de Candidates avec le plus grand degré dans G
        Ajouter x à C
        Retirer de Candidates les non-voisins de x
    Ajouter C à S
Retirer de S toute clique incluse dans une autre
foreach {x, y} ∈ E do
    if ∄C' ∈ S tel que x, y ∈ C' then
        Ajouter {x, y} à S
```

4.3 Réduction des domaines initiaux

Dans toutes les méthodes complètes présentées, les couleurs qui peuvent être assignées à n'importe quel sommet sont bornées par $\Delta(G) + 1$. Nous proposons d'abaisser cette borne en exploitant le fait que, dans une solution optimale, la couleur du sommet v est toujours inférieure ou égale à $\deg(v) + 1$.

Propriété 1 *Pour toute somme coloration optimale* c *d'un graphe* $G = (V, E)$, $c(v) \leq \deg(v) + 1, \forall v \in V$.

Pour démontrer cette propriété, supposons qu'elle n'est pas vraie pour une coloration optimale donnée c. Il s'ensuit qu'il existe un sommet v de V pour lequel $c(v) > \deg(v) + 1$. Dans de telles conditions, il existe une valeur x non utilisée dans $\{1, \ldots, \deg(v) + 1\}$, puisque v n'a que $\deg(v)$ voisins. Par conséquent, une meilleure coloration que c peut être obtenue en utilisant la couleur x pour v au lieu de $c(x)$. Donc, c n'est pas optimal, ce qui contredit l'hypothèse de départ.

Il s'agit en réalité d'un cas plus spécifique de la limite fixée aux domaines dans [10]. Ici, nous considérons les sommets *indépendamment*. Nous opérons cette réduction dès la génération des instances. Pour chaque sommet, au lieu d'autoriser toutes les couleurs de $\{1, \ldots, \Delta(G) + 1\}$, nous limitons le domaine de la variable associée au sommet v à $\{1, \ldots, \deg(v) + 1\}$. La différence de taille de domaine peut être conséquente, notamment dans des graphes peu denses. Cette modification des instances peut aisément être appliquée à tous les modèles mentionnés dans cet article.

Sur les 126 instances considérées (cf. section 5.2), la réduction des domaines moyenne est de 46 %. Sur quelques instances, la réduction est nulle, tandis qu'une réduction maximale de 95 % peut être observée sur une instance (`3-Insertions_5`). Des réductions de cette ampleur sont constatées dès lors qu'un petit nombre de sommets ont un très haut degré tandis que les autres ont un degré faible.

4.4 Heuristique de choix de valeur

Étant donné que le but est de minimiser la somme des variables, nous avons opté pour le choix systématique de la plus petite valeur disponible dans le domaine de la variable considérée.

4.5 Heuristiques de choix de variable

Dès lors que l'on évalue des heuristiques pour un problème d'optimisation, un problème de polyvalence survient : il est souhaitable de trouver rapidement une bonne solution afin d'être en mesure de couper des branches de l'arbre de recherche, mais les heuristiques de choix de variable qui excellent dans cette tâche sont généralement moins performantes pour effectuer des preuves d'optimalité. Pour cette raison, un utilisateur pourra porter son choix sur différentes heuristiques selon ce qu'il tente d'accomplir. Nous avons entrepris d'élargir nos expériences afin d'évaluer sept heuristiques de choix de variable :

minDom choisit la variable au plus petit domaine courant ;

minDom/wDeg choisit la variable qui minimise le rapport entre la taille de son domaine courant et son degré pondéré [3] ;

activity choisit la variable dont le domaine a le plus été réduit lors des propagations de contraintes [12] ;

minElim choisit la variable qui possède la plus petite valeur dans son domaine courant, et départage les ex-aequo en choisissant celle pour laquelle la valeur choisie sera retirée du plus petit nombre de domaines de variables voisines ;

dynDeg choisit la variable qui a le plus de voisins non colorés, et départage les ex-aequo en faveur de la variable ayant la plus petite valeur dans son domaine courant, puis de celle ayant le plus petit domaine courant (notez que *minElim* et *dynDeg* tentent de sélectionner une variable ayant au moins une faible valeur dans son domaine, afin de permettre à l'heuristique de choix de valeur d'utiliser cette valeur) ;

minRemove choisit la variable pour laquelle la valeur choisie sera retirée du moins de domaines possible, et départage les ex-aequo selon les règles de *dynDeg* ; *minRemove* maintient, pour chaque variable x_i, les voisins qui ont encore dans leur domaine la valeur qui serait utilisée si x_i était choisie ;

maxDegSizeVal choisit la variable qui maximise

$degree \div (size\,of\,domain \times smallest\,value)$; les égalités sont brisées de la même manière qu'avec *dynDeg*.

Quelle que soit l'heuristique, les égalités persistantes sont brisées aléatoirement. Dans la section 5, nous comparons ces heuristiques sur deux critères : la capacité à trouver rapidement une bonne solution, et celle à faire des preuves d'optimalité.

4.6 Filtrage

Nous avons adapté l'algorithme de filtrage utilisé avec le *Branch and Bound* de [11]. Cette méthode construit une partition de cliques sur l'ensemble des sommets non coloriés et déduit une borne inférieure de cette partition. Dans [11], une telle partition est construite à chaque nœud de l'arbre de recherche, afin d'augmenter la précision des bornes résultantes. Dans des expériences préliminaires, cependant, nous avons constaté que ces constructions répétées avaient un coût non négligeable, qui, au bout du compte, dégradaient les performances. En réaction à cela, nous avons décidé de ne calculer une partition de cliques qu'une seule fois, au démarrage de notre programme. Cette partition est construite comme indiqué dans l'algorithme 2.

Algorithme 2 : Construction d'une partition de cliques.

Data : Graphe $G = (V, E)$
Result : Partition P de cliques de G telle que $\forall v \in V, \exists! C \in P$ telle que $v \in C$

```
1  V_unused ← V
2  while V_unused ≠ ∅ do
3      v ← sommet de V_unused avec le plus haut
       degré dans le graphe induit par V_unused
4      C ← {v}
5      Candidates ← voisins de v dans G
6      while Candidates ≠ ∅ do
7          x ← sommet de Candidates avec le plus
           haut degré dans G
8          Ajouter x à C
9          Retirer de Candidates tout non-voisin de x
10     Ajouter C à P
11     V_unused ← V_unused \ C
```

Pour déduire une borne inférieure d'une partition de cliques, il faut d'abord trouver une borne pour chaque clique qui la compose. Pour une clique C de taille k, cette borne correspond à la somme des k plus petites valeurs présentes dans l'union des domaines des variables de C. L'idée sous-jacente est que dans le meilleur des cas il sera possible d'utiliser ces valeurs. Notez qu'il est nécessaire de choisir k valeurs *différentes* puisque, dans une clique, toute paire de variables est soumise à une contrainte de différence. La solution optimale réelle implique généralement des valeurs plus élevées du fait de la présence de contraintes de différence *entre* les cliques, qui sont ici ignorées.

Dans [11], la partition est calculée sur l'ensemble des sommets non colorés. Dans cet article, en revanche, puisque la partition est calculée au début de la recherche, les sommets n'ont pas encore de couleur. Nous tirons cependant profit des sommets déjà colorés au moment du calcul de la borne puisque nous calculons l'union des domaines *courants* lors du choix des k plus petites valeurs. De ce fait, tout sommet coloré – et même simplement tout domaine réduit – contribue à rendre la borne calculée plus précise.

Tenter trop tôt de couper des branches à l'aide d'une telle méthode de filtrage cause bien souvent une perte d'efficacité : lorsque seuls quelques sommets sont colorés, nous ne disposons pas d'assez d'informations pour estimer la qualité des colorations à venir. Concrètement, la borne alors calculée est de piètre qualité. Afin d'éviter des calculs inutiles, nous fixons une limite empêchant le déclenchement de cet algorithme de filtrage. Nous calculons la somme de toutes les valeurs des variables déjà affectées. Si la distance entre cette somme et et la borne supérieure courante équivaut à plus de 20 % de cette dernière, nous estimons trop peu probable que la borne inférieure que donnerait l'algorithme de filtrage atteigne la borne supérieure, et refusons d'utiliser cet algorithme.

En plus de cette limite visant à ne pas *commencer trop tôt* à utiliser ce filtrage, nous utilisons une limite pour ne pas l'employer *trop tard*, au sens de la profondeur dans l'arbre de recherche. En effet, lorsque seuls peu de sommets restent à colorer, il peut être plus rapide de finir l'exploration de la branche courante de l'arbre de recherche plutôt que de passer du temps à calculer des bornes avec l'algorithme de filtrage. Des expériences qui ne sont pas détaillées ici nous ont permis de constater qu'il est convenable de se retenir d'utiliser ce filtrage lorsque cinq sommets ou moins ne sont pas colorés.

4.7 Échange de couleurs

Comme relevé par [11], certaines solutions peuvent être *dominées* : on peut les améliorer en se contentant d'échanger des couleurs, comme rappelé en section 2. Dans les présents travaux, nous utilisons notamment cette méthode pour abaisser plus rapidement la borne supérieure, afin d'accélérer la résolution. Ainsi, toute solution dominée produite lors de la recherche sera transformée en solution non dominée. Par ailleurs, l'ajout d'une contrainte imposant de trouver une solution meilleure que la précédente permet de ne plus produire une solution dominée par celle-ci.

Il est à noter que la technique d'échange de couleurs a un coût computationnel exceptionnellement bas : le surcoût n'opère que lorsqu'une coloration complète du graphe est trouvée, et les vérifications sur la taille des ensembles représentés par les différentes couleurs sont très simples à effectuer.

5 Évaluation expérimentale

5.1 Procédure expérimentale

Nous avons exécuté les programmes évalués sur un processeur Intel® Xeon® E5-2670 0 à 2,60 Ghz, disposant de 20 480 KB de mémoire cache et de 4 GB de RAM. La réduction des domaines décrite en section 4.3 a été utilisée pour chaque modèle et solveur.

5.2 Benchmark

Nous considérons 126 instances couramment utilisées pour le MSCP [18, 7]. Certaines ont été ajoutées par COLOR02/03/04 [1], mais la plupart sont simplement les instances DIMACS conçues pour le problème de coloration classique [2]. Pour certaines de ces instances, la littérature fournit des bornes inférieures et supérieures [18, 7]. Pour les autres, lorsque nous utilisons une borne supérieure initiale, il s'agit de la meilleure trouvée lors d'expériences précédentes.

Afin d'économiser du temps de calcul pour conduire de plus larges expérimentations, nous avons choisi d'effectuer certaines d'entre elles sur un ensemble réduit d'instances. Lorsque le but était de rapidement trouver une bonne solution avec une limite de 30 minutes et sans borne initiale, nous avons opéré sur une sélection d'instances de toutes les classes, en nous concentrant sur celles qui n'étaient pas triviales et en privilégiant celles où la borne supérieure connue était très haute, afin de pouvoir observer de grandes différences entre les bornes trouvées par les différentes méthodes. Pour les expériences portant sur les preuves d'optimalité, avec 24 heures et une borne supérieure initiale donnée, les résultats sont montrés sur les instances pour lesquelles au moins une méthode a pu améliorer la borne ou prouver son optimalité. Les dernières expériences, ainsi que celles visant à comparer les modèles, ont été menées sur le *benchmark* complet.

Lorsque des contraintes *AllDifferent* sont utilisées, les cliques sont construites en dehors des expériences, lors de la création des instances.

1. `http://mat.gsia.cmu.edu/COLOR02`
2. `ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/color/`

Tableau 1 – Un résumé des résultats obtenus sans et avec contraintes *AllDifferent*, en utilisant Toulbar, CPLEX et Gecode. Les trois premières lignes donnent, respectivement, les distances minimale, moyenne et maximale entre la meilleure solution trouvée par la méthode et la meilleure solution connue, en pourcentages. « Preuves » (resp. « Pb mem. ») correspond au nombre d'instances pour lesquelles l'optimalité a été prouvée (resp. pour lesquelles l'exécution s'est terminée prématurément par manque de mémoire). « Meill. UB » donne le nombre d'instances sur lesquelles la meilleure solution connue a été atteinte, et « Nb meill. » le nombre de fois où la méthode a été la meilleure sur une instance. La « meilleure » méthode est désignée par le statut (preuve > pas de preuve), puis par la qualité de la solution, et enfin par le temps nécessaire pour la trouver.

		Toulbar		CPLEX		Gecode	
		Binaire	AllDiff	Binaire	AllDiff	Binaire	AllDiff
Dist	Min	0	0	0	0	0	0
	Moy	174,3	465,5	70,2	68,7	**9,2**	**9,2**
	Max	1 143,6	2 863,3	1 119,5	1 119,5	**78,5**	**78,5**
Preuves		19	15	**65**	64	10	10
Pb Mém.		42	97	41	43	**0**	**0**
Meill. UB		22	18	71	**73**	31	32
Nb meill.		5	4	41	**49**	13	22

5.3 Implémentation

Nous avons implémenté le modèle CSP dans Gecode 4.2.1 [16]. Le codage de ce modèle est direct et ne sera pas détaillé ici. Le moteur de recherche *Branch and Bound* (BAB) a été sélectionné, et les contraintes *AllDifferent* ont été représentées avec la contrainte « `distinct` » de Gecode. Le niveau de cohérence est GAC (« `ICL_DOM` »).

Nous avons implémenté le modèle WCSP avec Toulbar2 0.9.7.0-R [1]. Toulbar a principalement été conçu pour résoudre des WCSP et utilise des algorithmes de consistance souple de haut niveau (EDAC par défaut [5]) [15, 1].

Pour la programmation linéaire, nous avons utilisé ILOG CPLEX 12.6.2 [4]. ILOG CPLEX traite des programmes linéaires, ce qui impose une représentation purement arithmétique des contraintes de différence, qu'elles soient binaires ou globales.

5.4 Comparaison de Gecode, Toulbar et CPLEX

Le tableau 1 résume les résultats obtenus avec les trois solveurs considérés. Chacun a été évalué sur un modèle sans contraintes *AllDifferent* (« Binaire ») et sur un modèle les incluant (« AllDiff »).

Un problème flagrant dans ces premiers tests est que CPLEX, de même que Toulbar, viennent souvent à manquer de mémoire. Les *AllDifferent* n'ont pas un

impact significatif sur cet aspect pour CPLEX. En revanche, ils augmentent nettement la consommation de mémoire pour la version de Toulbar utilisée.

Si on se limite aux instances pour lesquelles il n'y a pas eu de problème de mémoire, on constate que CPLEX est très efficace, et résout le plus grand nombre d'instances (65) tout en trouvant la meilleure solution connue pour 73 d'entre elles. Ses besoins en mémoire sont donc le principal facteur limitant ; il arrive même que CPLEX soit incapable de fournir la moindre solution intermédiaire. Il en résulte que les solutions trouvées par CPLEX sont en moyenne plus grande de 70 % que les meilleures solutions connues.

Grâce à son haut niveau de filtrage, Toulbar fait presque deux fois plus de preuves que Gecode. Cependant, comme pour CPLEX, la quantité de mémoire utilisée devient problématique sur les instances les plus grandes. En effet, Toulbar utilise principalement des contraintes données en extension, ce qui explique au moins partiellement cette consommation de mémoire. Utiliser des contraintes *AllDifferent* souples induit tant de problèmes de mémoires que cette méthode en devient presque inutilisable dans notre contexte.

Gecode n'est capable de résoudre que dix instances, mais ses besoins en mémoire sont raisonnables, ce qui lui permet de ne jamais en manquer. Il en résulte des bornes plus fines, avec une distance moyenne aux meilleures bornes connues de seulement 9,2 %, tandis que CPLEX et Toulbar obtiennent des moyennes respectives de 68,7 % et 174,3 %. Utiliser des contraintes *AllDifferent* permet à Gecode d'atteindre la meilleure solution connue une fois de plus qu'avec le modèle binaire. Ces contraintes aident Gecode à filtrer, et donc à trouver rapidement des solutions intéressantes.

Puisque CPLEX et Toulbar manquent souvent de mémoire, nous nous concentrerons sur Gecode pour la suite de cet article. Nous considérerons le modèle utilisant les contraintes *AllDifferent*, du fait des résultats sensiblement meilleurs qu'il permet d'obtenir.

5.5 Heuristiques de choix de variable

Le tableau 2 présente les résultats d'une comparaison d'heuristiques pour Gecode, sur 30 minutes sans bornes, tandis que le tableau 3 contient les résultats pour 24 h avec une borne supérieure initiale donnée.

Ces expériences montrent que, contrairement à ce que notre intuition tendrait à nous faire penser, *minDom/wDeg* n'est pas extrêmement efficace, que ce soit sur les tests de 30 minutes ou sur ceux de 24 heures. Une autre heuristique couramment utilisée, *activity*, souffre de difficultés similaires. Les meilleures heuristiques sont *minElim* (pour ce qui est de la recherche rapide d'une bonne solution) et *dynDeg* (pour les preuves). Ces deux heuristiques seront donc celles considérées dans la suite de l'article.

5.6 Intégration du filtrage, de restarts et d'échanges de couleurs

La plupart des heuristiques que nous avons utilisées sont plutôt efficaces quand il s'agit de trouver une bonne première solution, mais peinent à s'éloigner de cette première solution dans l'espace de recherche. Un manque de diversité critique en résulte, et la qualité des solutions finales s'en ressent. Ce phénomène est particulièrement présent avec *minElim*.

Afin d'aider Gecode à trouver des solutions plus variées, nous pouvons l'employer avec des *restarts*. Les restarts ont leur propre ensemble de paramètres ; nous avons effectué une série d'expériences afin de trouver des valeurs adéquates pour ces paramètres. Notre choix s'est porté sur des politiques déjà définies dans Gecode : les restarts géométriques et la politique Luby, associée à une valeur d'échelle de 500.

Afin d'évaluer les différentes améliorations proposées, nous les avons ajoutées une à une et avons mené deux séries d'expériences. La première s'est déroulée avec une limite de 30 minutes et sans borne initiale (tableau 4) ; pour la seconde, une limite de 24 heures était imposée, et une borne supérieure initiale était donnée (tableau 5). Ces bornes viennent de la littérature lorsqu'elles existent, et de nos expériences précédentes dans le cas contraire.

Le filtrage apparaît comme désavantageux pour trouver une bonne solution, mais aide Gecode à faire des preuves d'optimalité. Le paramétrage sélectionné, qui consiste à commencer à utiliser le filtrage lorsque la solution courante est à 20 % de la borne supérieure et à cesser de l'employer quand seules cinq variables ou moins n'ont pas encore de valeur affectée, semble être un compromis intéressant. Il facilite les preuves tout en gardant le surcoût en termes de calculs bas.

Comme prévu, l'utilisation de restarts permet à Gecode de trouver de meilleures bornes, grâce à une plus grande diversité dans les solutions formées. On notera cependant qu'utiliser des restarts dans le cadre des preuves d'optimalité n'est pas un bon choix.

Les échanges de couleurs ayant un coût négligeable, ils ne perturbent jamais le processus de résolution, et parviennent à accélérer la baisse de la borne supérieure sur plusieurs instances.

5.7 Évaluation complète

Nous avons évalué l'impact de ces améliorations sur des exécutions de 24 heures. Les premiers tests ont été effectués avec une borne supérieure initiale, sur les 92 instances pour lesquelles une telle borne est fournie

TABLEAU 2 – Comparaison d'heuristiques pour Gecode, sur 30 minutes, sans borne initiale. Les bornes supérieures « UB_L » sont celles de la littérature, et « UB » celles obtenues en fin d'exécution, avec « * » en cas de preuve. « t_{UB} » sont les secondes nécessaires à Gecode pour trouver la solution associée à « UB ».

		minDom/wDeg		*minDom*		*dynDeg*		*minRemove*		*minElim*		*activity*		*maxDegSizeVal*	
Instance	UB_L	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}	UB	t_{UB}
DSJC500.5	10 886	15913	133	15736	1694	16261	731	13289	6	**13059**	**1349**	15125	898	15995	1747
DSJR500.1	2 156	2510	4	2477	59	2712	0	2339	1	2331	0	**2279**	**36**	2617	37
flat1000_50_0	25 500	56294	88	54578	34	55567	1119	**45217**	**592**	45519	141	55235	1452	56051	361
4-FullIns_5		12062	113	9231	221	8385	255	6707	739	**6680**	**222**	11506	392	8389	221
games120	443	466	0	467	0	497	0	**451**	**0**	456	0	474	0	492	658
ash958GPIA		4794	220	4753	1044	5180	323	**4324**	**46**	4345	16	4580	10	5474	356
3-Insertions_5		2875	309	2721	997	3855	141	2289	8	**2289**	**7**	2841	3	3808	375
latin_square_10	41 444	49885	813	48955	91	58561	301	46847	56	**46847**	**48**	48934	353	59136	1629
le450_5a	1 350	2337	632	2244	0	2313	416	1863	0	**1752**	**1**	2188	483	2312	41
mug88_25	178	182	189	183	0	185	0	181	195	**180**	**0**	181	0	185	0
myciel7	381	499	547	438	1100	722	33	381	0	**381**	**0**	382	0	625	38
qg.order30	13 950	14283	1701	13996	468	14324	477	13953	2	**13950***	**1**	13981	1044	14327	858
r1000.1		9040	86	8383	37	9009	2	7803	115	7703	2	**7579**	**334**	8964	3
fpsol2.i.3	1 636	1959	1	1695	1	8135	1	1677	882	**1670**	**119**	1809	1	7738	0
inithx.i.3	1 986	2518	1	2033	1	13122	1	2034	2	**2019**	**1**	2053	1	13085	1
mulsol.i.5	1 160	**1163**	**120**	1165	0	3734	0	1165	0	1166	0	1164	0	3694	0
zeroin.i.3	998	1018	11	**999**	**0**	3192	301	1080	0	1034	0	1047	0	3129	54
school1_nsh	2 392	4842	845	4749	181	4900	64	3958	352	**3880**	**818**	4478	1426	4821	156
david	237	257	0	254	1436	372	0	255	11	**245**	**727**	257	0	333	3
homer	1 150	1229	10	1228	581	1392	0	1240	1	**1219**	**17**	1227	0	1301	221
miles500	705	861	229	806	0	864	1154	835	0	761	0	**751**	**0**	824	253
queen11_11	733	886	599	837	35	934	7	821	11	**812**	**1701**	853	1310	943	762
wap06a	13 773	16872	1	16791	575	20282	883	**15374**	**3**	15427	818	15882	1102	16902	1708

TABLEAU 3 – Comparaison d'heuristiques pour Gecode, sur 24 heures, avec borne supérieure initiale donnée. Les bornes supérieures « UB_L » sont celles de la littérature, et « UB » celles obtenues en fin d'exécution. « t_{UB} » est nombre de secondes nécessitées par Gecode pour trouver la solution associée à « UB ».

		minDom/wDeg			*minDom*			*dynDeg*			*minRemove*			*minElim*			*activity*			*maxDegSizeVal*		
Instance	UB_L	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t	UB	t_{UB}	t
1-FullIns_5		554	0	T	521	2006	T	554	0	T	505	0	T	**499**	**0**	**T**	554	0	T	530	1479	T
2-FullIns_3		93*	0	35964	93*	0	29227	**93***	**0**	**47**	93*	0	49505	93*	0	3819	93*	0	618	93*	0	520
2-FullIns_4		417	0	T	417	0	T	417	0	T	372	2272	T	**363**	**105**	**T**	417	0	T	416	9	T
3-FullIns_3		151	0	T	**150**	**42452**	**T**	151	0	T	151	0	T	151	0	T	151	0	T	151	0	T
3-FullIns_4		735	0	T	735	0	T	735	0	T	**683**	**0**	**T**	683	0	T	735	0	T	722	63886	T
4-FullIns_3		241	0	T	239	0	T	**205**	**328**	**T**	220	55	T	213	37	T	241	0	T	241	0	T
ash331GPIA		1695	8	T	1575	3	T	1736	0	T	**1487**	**1**	**T**	1511	309	T	1608	306	T	1736	0	T
1-Ins._4		119	0	T	119	0	T	**119***	**2997**	**5685**	119	0	T	119	0	T	125	0	T	125	0	T
2-Ins._3	62	62*	0	19	62*	0	14	**62***	**0**	**1**	62*	0	24	62*	0	9	62*	0	13	62*	0	1
3-Ins._3	92	92*	0	29466	92*	0	31770	**92***	**0**	**120**	92*	0	42607	92*	0	4206	92*	0	20847	92*	0	158
4-Ins._3		127	21	T	127	0	T	136	53627	T	127	0	T	**127**	**0**	**T**	130	1	T	136	49990	T
myciel5	93	93	0	T	93	0	T	93*	0	26910	93	0	T	93*	0	29980	**93***	**0**	**2689**	93*	0	4384
qg.order30	13 950	13950*	0	1	13950*	0	1	13950*	0	0	13950*	0	1	**13950***	**0**	**0**	13950*	0	0	13950*	0	0
queen5_5	75	75*	0	964	75*	0	917	75*	0	63	75*	0	20	75*	0	13	75*	0	122	**75***	**0**	**12**

TABLEAU 6 – La colonne « Simple » présente les résultats d'une version de *minElim* sans filtrage, ni restarts, ni échanges de couleurs. La version dite « Améliorée » inclut quant à elle toutes ces améliorations.

		Simple	Améliorée
Dist	Min	0	0
	Moy	9	5,2
	Max	78,5	72
Preuves		10	11
Meill. UB		30	57
Nb meill.		16	118

par la littérature ; les tests suivants, sans borne initiale, utilisent le *benchmark* entier.

Le tableau 6 montre que les améliorations proposées sont bénéfiques pour la quasi-intégralité du *benchmark*. Une preuve supplémentaire a pu être faite, et la distance moyenne est presque divisée par deux.

Neuf preuves ont pu être effectuées par *dynDeg* en employant toutes les améliorations. Puisque les restarts ne sont pas adaptés aux preuves d'optimalité, nous avons tenté de les retirer, ce qui a permis une preuve supplémentaire et accéléré les preuves existantes (tableau 7).

6 Conclusion et travaux futurs

Dans cet article, nous avons considéré le problème de la somme coloration minimale, et démontré que même si la programmation par contraintes semble à première vue désavantagée, elle possède quelques atouts, comme notamment sa consommation nettement réduite de

TABLEAU 4 – Ajout progressif des améliorations présentées, pour la recherche d'une bonne solution en 30 minutes, sans borne initiale. Les bornes présentées sur la gauche sont celles de l'état de l'art. « f20-5 » signifie que le filtrage à base de partitions de cliques a été utilisé, avec les paramètres mentionnés en section 5.6 ; « rl500 » indique l'utilisation de restarts Luby avec une échelle de 500 ; « éch. » signifie que les échanges de couleurs étaient autorisés pour améliorer les solutions trouvées. Les « * » mettent les preuves en évidence.

			minElim				*minElim* f20-5				*minElim* f20-5 rl500				*minElim* f20-5 rl500 éch.			
Instance	LB_L	UB_L	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$
DSJC500.5	2 923	10 886	13059	20	1349	T	13068	20	296	T	12798	17.6	1029	T	**12734**	**17**	**392**	**T**
DSJR500.1	2 069	2 156	2331	8.1	0	T	2331	8.1	0	T	**2287**	**6.1**	**85**	**T**	2287	6.1	92	T
flat1000_50_0	6 601	25 500	45519	78.5	141	T	45520	78.5	53	T	44587	74.9	1280	T	**44243**	**73.5**	**365**	**T**
4-FullIns_5			6680	0	222	T	6680	0	225	T	**6680**	**0**	**205**	**T**	6680	0	238	T
games120	443	443	456	2.9	0	T	456	2.9	0	T	**443**	**0**	**1189**	**T**	444	0.2	722	T
ash958GPIA			4345	2	16	T	4345	2	18	T	4267	0.2	470	T	**4259**	**0**	**513**	**T**
3-Insertions_5			2289	0	7	T	2289	0	6	T	**2289**	**0**	**6**	**T**	2289	0	7	T
latin_square_10	40 950	41 444	46847	13	48	T	46847	13	41	T	45533	9.9	918	T	**45372**	**9.5**	**895**	**T**
le450_5a	1 193	1 350	1752	29.8	1	T	1752	29.8	17	T	1496	10.8	1611	T	**1424**	**5.5**	**1090**	**T**
mug88_25	162	178	180	1.1	0	T	180	1.1	0	T	179	0.6	4	T	**179**	**0.6**	**4**	**T**
myciel7	286	381	**381**	**0**	**0**	**T**	381	0	0	T	**381**	**0**	**0**	**T**	381	0	0	T
qg.order30	13 950	13 950	13950*	0	1	1	13950*	0	1	1	13950*	0	1	1	**13950***	**0**	**1**	**1**
r1000.1			7703	1.4	2	T	7703	1.4	2	T	7607	0.1	801	T	**7597**	**0**	**1359**	**T**
fpsol2.i.3	1 636	1 636	1670	2.1	119	T	1670	2.1	170	T	**1641**	**0.3**	**473**	**T**	1641	0.3	521	T
inithx.i.3	1 986	1 986	2019	1.7	1	T	2019	1.7	1	T	1991	0.3	874	T	**1986**	**0**	**584**	**T**
mulsol.i.5	1 160	1 160	1166	0.5	0	T	1166	0.5	0	T	1164	0.3	3	T	**1164**	**0.3**	**2**	**T**
zeroin.i.3	998	998	1034	3.6	0	T	1033	3.5	35	T	1019	2.1	551	T	**1012**	**1.4**	**0**	**T**
school1_nsh	2 176	2 392	3880	62.2	818	T	3883	62.3	154	T	3361	40.5	714	T	**3149**	**31.6**	**1127**	**T**
david	237	237	245	3.4	727	T	244	3	103	T	**240**	**1.3**	**812**	**T**	240	1.3	867	T
homer	1 129	1 150	1219	6	17	T	1219	6	88	T	1191	3.6	1043	T	**1191**	**3.6**	**932**	**T**
miles500	705	705	761	7.9	0	T	752	6.7	0	T	**727**	**3.1**	**1155**	**T**	730	3.5	869	T
queen11_11	726	733	812	10.8	1701	T	818	11.6	345	T	775	5.7	774	T	**772**	**5.3**	**1654**	**T**
wap06a	12 454	13 773	15427	12	818	T	15427	12	661	T	15098	9.6	581	T	**15041**	**9.2**	**1101**	**T**

TABLEAU 7 – « UB_L » sont les bornes supérieures de la littérature données à Gecode comme bornes initiales. « Toutes » indique que toutes les améliorations ont été employées. « rNon » est la même méthode, mais sans restarts. Si la preuve a pu être effectuée en 24 h, nous donnons le nombre de secondes nécessaires.

Instance	UB_L	Toutes	rNon
2-Insertions_3	62	**0**	1
3-Insertions_3	92	129	**38**
myciel3	21	**0**	**0**
myciel4	45	**0**	**0**
myciel5	93		**41 279**
qg.order30	13 950	0	**0**
qg.order40	32 800	2	**1**
qg.order60	109 800	7	**6**
queen5_5	75	17	**10**
queen8_12	624	**0**	**0**

mémoire, comparée à celle des solveurs de WCSP ou de programmation linéaire. De ce fait, un solveur tel que Gecode peut être appliqué à des instances de très grande taille, et pourra passer plus de temps à trouver des solutions de meilleure qualité. Les compétences d'un tel solveur peuvent être améliorées de plusieurs manières, comme en utilisant des restarts convenablement paramétrés, en employant avec modération un algorithme de filtrage, et en veillant à améliorer les solutions via des échanges de couleurs lorsque cela s'avère possible.

Du fait de l'existence de solutions dominées, comme expliqué en section 5.3, nous avons affaire à de nombreuses redondances dans la procédure de recherche, et ce quelque soit le solveur. Ces redondances peuvent être supprimées en exprimant le problème sous la forme d'une construction de partition de sommets, mais quelques expériences nous ont montré qu'il était difficile d'employer une telle méthode sans pénaliser le solveur, car il devient alors difficile d'avoir une évaluation précise de la qualité d'une solution en cours de construction. Par ailleurs, il faut noter que les échanges de couleurs, combinés avec la contrainte imposant de trouver à chaque fois une meilleure solution, permet déjà d'éviter de multiples énumérations de solutions dominées.

Il sera important d'inclure à l'avenir des modèles SAT dans cette étude. Dans [11], ces modèles ont donné de très bon résultats sur certaines instances (classes `miles` et `myciel`), mais se montraient très insuffisant pour des instances raisonnables telles que `queen5_5`.

Ces travaux ont été soutenus par l'ANR SoLStiCe (ANR-13-BS02-0002-01).

Références

[1] D Allouche, S de Givry, and T Schiex. Toulbar2, an open source exact cost function network solver. Technical report, Technical report, INRIA, 2010.

[2] U. Benlic and J.-Kao Hao. A study of breakout local search for the minimum sum coloring problem. In *Simulated Evolution and Learning*, pages 128–137. Springer, 2012.

TABLEAU 5 – Ajout progressif des améliorations présentées, pour la preuve d'optimalité avec une borne supérieure initiale donnée. Les bornes présentées sur la gauche sont celles de l'état de l'art. « f20-5 » signifie que le filtrage à base de partitions de cliques a été utilisé, avec les paramètres mentionnés en section 5.6 ; « rl500 » indique l'utilisation de restarts Luby avec une échelle de 500 ; « éch. » signifie que les échanges de couleurs étaient autorisés pour améliorer les solutions trouvées. Les « * » mettent les preuves en évidence.

			dynDeg				*dynDeg* f20-5				*dynDeg* f20-5 rl500				*dynDeg* f20-5 rl500 éch.			
Instance	LB_L	UB_L	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$
1-FullIns_5			554	11	0	T	554	11	0	T	499	0	1	T	**499**	**0**	**1**	**T**
2-FullIns_3			93*	0	0	47	93*	0	0	9	**93***	**0**	**0**	**8**	93*	0	0	10
2-FullIns_4			417	13	0	T	417	13	0	T	369	0	2025	T	**369**	**0**	**953**	**T**
3-FullIns_3			151	4.1	0	T	**145***	**0**	**15083**	**29781**	145*	0	85	61805	145*	0	84	54774
3-FullIns_4			735	4.4	0	T	735	4.4	0	T	704	0	14346	T	**704**	**0**	**13963**	**T**
4-FullIns_3			205	0	328	T	205	0	315	T	**205**	**0**	**1**	**T**	205	0	2	T
5-FullIns_3			294	5	0	T	294	5	0	T	287	2.5	54992	T	**280**	**0**	**0**	**T**
ash331GPIA			1736	4.7	0	T	1736	4.7	0	T	**1658**	**0**	**8131**	**T**	1668	0.6	881	T
1-Insertions_4			**119***	**0**	**2997**	**5685**	119*	0	3579	6291	123	3.4	56	T	123	3.4	63	T
2-Insertions_3	55	62	62*	0	0	1	62*	0	0	0	62*	0	0	0	**62***	**0**	**0**	**0**
3-Insertions_3	84	92	92*	0	0	120	**92***	**0**	**0**	**50**	92*	0	0	137	92*	0	0	96
4-Insertions_3			136	7.1	53627	T	**127***	**0**	**14804**	**19122**	127	0	60472	T	127	0	3362	T
myciel5	93	93	**93***	**0**	**0**	**26910**	93	0	0	T	93	0	0	T	93	0	0	T
qg.order30	13 950	13 950	**13950***	**0**	**0**	**0**	13950*	0	0	1	13950*	0	0	1	13950*	0	0	0
r125.1			257	0	1	T	**257***	**0**	**0**	**21069**	257	0	0	T	257	0	1	T
queen5_5	75	75	75*	0	0	63	**75***	**0**	**0**	**15**	75*	0	0	17	75*	0	0	16

[3] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004.

[4] I. CPLEX. High-performance software for mathematical programming and optimization, 2005.

[5] S. De Givry, F. Heras, M. Zytnicki, and J. Larrosa. Existential arc consistency: Getting closer to full arc consistency in weighted csps. In *IJCAI*, volume 5, pages 84–89, 2005.

[6] A. Helmar and M. Chiarandini. A local search heuristic for chromatic sum. In *Proceedings of the 9th metaheuristics international conference*, volume 1101, pages 161–170, 2011.

[7] Y. Jin, J.-P. Hamiez, and J.-K. Hao. Algorithms for the minimum sum coloring problem: a review. *arXiv preprint arXiv:1505.00449*, 2015.

[8] Y. Jin, J.-Kao Hao, and J.-Philippe Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43:318–327, 2014.

[9] N. Jussien, G. Rochart, and X. Lorca. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, 2008.

[10] M. Kubale. *Graph colorings*, volume 352. American Mathematical Soc., 2004.

[11] C. Lecat, C. Min Li, C. Lucet, and Y. Li. Comparaison de méthodes de résolution pour le problème de somme coloration. In *JFPC'15: Journées Francophones de Programmation par Contraintes*, 2015.

[12] L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems*, pages 228–243. Springer, 2012.

[13] A Moukrim, K Sghiouer, C Lucet, and Y Li. Upper and lower bounds for the minimum sum coloring problem, submitted for publication.

[14] A. Moukrim, K. Sghiouer, C. Lucet, and Y. Li. Lower bounds for the minimal sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:663–670, 2010.

[15] M Sanchez, S Bouveret, S De Givry, et al. Max-csp competition 2008: toulbar2 solver description. *Proceedings of the Third International CSP Solver Competition*, pages 63–70, 2008.

[16] G. Team. Gecode: Generic constraint development environment, 2006, 2008.

[17] C. Thomassen, P. Erdös, Y. Alavi, P. J Malde, and A. J Schwenk. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13(3):353–357, 1989.

[18] Y. Wang, J.-Kao Hao, F. Glover, and Z. Lü. Solving the minimum sum coloring problem via binary quadratic programming. *arXiv preprint arXiv:1304.5876*, 2013.

[19] Q. Wu and J.-Kao Hao. An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600, 2012.

[20] Q. Wu and J.-Kao Hao. Improved lower bounds for sum coloring via clique decomposition. *arXiv preprint arXiv:1303.6761*, 2013.

[21] J.-K. Hao Y. Jin. Hybrid evolutionary search for the minimum sum coloring problem of graphs, accepted to information sciences feb 2016, 2015.

Les résultats complets de certaines expérimentations sont donnés dans ces annexes, et ne figurent que dans la version étendue de l'article.

Annexes

Résultats complets, en deux parties, pour la comparaison des modèles et solveurs :

			Toulbar Bin		Toulbar AD		CPLEX Bin		CPLEX AD		Gecode Bin		Gecode AD	
Instance	LB_L	UB_L	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$
DSJC125.1	247	326	371	T	391	#M#	**334**	**T**	337	T	353	T	353	T
DSJC125.5	549	1 012	1227	T	9501	#M#	1178	T	1173	T	1162	T	**1162**	**T**
DSJC125.9	1 691	2 503	3164	T	15126	#M#	2982	#M#	3155	#M#	**2819**	**T**	2819	T
DSJC250.1	570	970	1190	T	1209	#M#	1161	T	1154	#M#	1112	T	**1110**	**T**
DSJC250.5	1 287	3 210	15918	#M#	15918	#M#	4587	T	4587	#M#	3943	T	**3942**	**T**
DSJC250.9	4 311	8 277	28147	#M#	28147	#M#	28147	#M#	28147	#M#	**9686**	**T**	9686	T
DSJC500.1	1 250	2 836	3741	T	34501	#M#	4110	T	4110	T	3323	T	**3323**	**T**
DSJC500.5	2 923	10 886	63124	#M#	63124	#M#	63124	#M#	63124	#M#	13052	T	**13052**	**T**
DSJC500.9	11 053	29 862	112937	#M#	112937	#M#	112937	#M#	112937	#M#	**33717**	**T**	33717	T
DSJC1000.1	2 762	8 991	50629	#M#	50629	#M#	50629	#M#	12892	#M#	10565	T	**10565**	**T**
DSJC1000.5	6 708	37 575	250826	#M#	250826	#M#	250826	#M#	250826	#M#	45748	T	**45748**	**T**
DSJC1000.9	26 557	103 445	450449	#M#	450449	#M#	450449	#M#	450449	#M#	**122562**	**T**	122562	T
DSJR500.1	2 069	2 156	2557	T	2493	#M#	**2145**	**T**	2152	#M#	2331	T	2331	T
DSJR500.1c	15 398	16 286	121775	#M#	121775	#M#	121775	#M#	121775	#M#	**17815**	**T**	17815	T
DSJR500.5	22 974	25 440	59362	#M#	59362	#M#	59362	#M#	59362	#M#	**29214**	**T**	29214	T
flat300_20_0	1 531	3 150	21675	#M#	21675	#M#	6334	#M#	6334	#M#	4992	T	**4992**	**T**
flat300_26_0	1 548	3 966	21933	#M#	21933	#M#	6078	#M#	6078	#M#	5181	T	**5181**	**T**
flat300_28_0	1 547	4 238	21995	#M#	21995	#M#	6237	#M#	6237	#M#	5065	T	**5063**	**T**
flat1000_50_0	6 601	25 500	246000	#M#	246000	#M#	246000	#M#	246000	#M#	**45515**	**T**	45516	T
flat1000_60_0	6 640	30 100	246830	#M#	246830	#M#	246830	#M#	246830	#M#	45560	T	**45560**	**T**
flat1000_76_0	6 632	37 164	247708	#M#	247708	#M#	247708	#M#	247708	#M#	45132	T	**45132**	**T**
1-FullIns_3			**54***	**0**	54*	0	54*	0	54*	0	54*	1	54*	1
1-FullIns_4			166*	182	166*	174	**166***	**2**	166*	4	166	T	166	T
1-FullIns_5			556	T	556	T	499*	871	**499***	**602**	499	T	499	T
2-FullIns_3			93*	1	93*	3	**93***	**0**	93*	0	93*	21947	93*	8971
2-FullIns_4			363	T	400	T	**363***	**22**	363*	22	363	T	363	T
2-FullIns_5			13053	#M#	13053	#M#	**1433**	**T**	1446	T	1435	T	1435	T
3-FullIns_3			145*	51	1601	#M#	145*	2	**145***	**1**	159	T	159	T
3-FullIns_4			734	T	758	#M#	**683***	**178**	683*	437	683	T	683	T
3-FullIns_5			35781	#M#	35781	#M#	6725	#M#	6725	#M#	**3343**	**T**	3343	T
4-FullIns_3			205*	1250	205	#M#	205*	0	**205***	**0**	213	T	213	T
4-FullIns_4			1223	T	1277	#M#	**1138***	**6572**	1138*	7015	1145	T	1145	T
4-FullIns_5			81451	#M#	81451	#M#	81451	#M#	81451	#M#	6680	T	**6679**	**T**
5-FullIns_3			280	T	289	#M#	**280***	**2**	280*	3	300	T	300	T
5-FullIns_4			12480	#M#	12480	#M#	**1776***	**48434**	1776*	49177	1824	T	1824	T
games120	443	443	484	T	472	#M#	443*	1	**443***	**1**	456	T	456	T
ash331GPIA			1479	T	1504	T	1452	#M#	**1446**	**#M#**	1511	T	1511	T
ash608GPIA			2801	T	2785	T	2675	#M#	**2660**	**#M#**	2758	T	2758	T
ash958GPIA			4429	T	4429	T	**4294**	**#M#**	4295	#M#	4345	T	4345	T
will199GPIA			2242	T	27340	#M#	**1889**	**#M#**	1930	#M#	2034	T	2034	T
1-Insertions_4			119*	2	119*	2	**119***	**1**	119*	1	119	T	119	T
1-Insertions_5			431	T	431	T	357*	121	**357***	**109**	357	T	357	T
1-Insertions_6			13282	#M#	13282	#M#	1078	T	1078	T	1068	T	**1068**	**T**
2-Insertions_3	55	62	**62***	**0**	**62***	**0**	62*	0	62*	0	62*	9	62*	10
2-Insertions_4			249*	2370	249*	2455	249*	17	**249***	**7**	249	T	249	T
2-Insertions_5			1183	T	1183	T	996	#M#	996	#M#	996	T	**996**	**T**
3-Insertions_3	84	92	**92***	**0**	92*	0	92*	1	92*	0	92*	4714	92*	5012
3-Insertions_4			460	T	460	T	459*	57	**459***	**54**	459	T	459	T
3-Insertions_5			11101	#M#	11101	#M#	2297	T	2297	T	2289	T	**2289**	**T**
4-Insertions_3			127*	0	**127***	**0**	127*	1	127*	1	127	T	127	T
4-Insertions_4			819	T	819	T	**761***	**373**	761*	415	761	T	761	T
latin_square_10	40 950	41 444	308250	#M#	308250	#M#	308250	#M#	308250	#M#	46844	T	**46844**	**T**
le450_5a	1 193	1 350	2049	T	2118	#M#	**1366**	**#M#**	1373	T	1749	T	1749	T
le450_5b	1 189	1 350	2178	T	2190	#M#	1399	#M#	**1381**	**#M#**	1733	T	1733	T
le450_5c	1 278	1 350	2238	T	30151	#M#	**1350***	**4614**	1350*	12742	1416	T	1416	T
le450_5d	1 282	1 350	2170	T	31051	#M#	1350*	15345	**1350***	**11317**	1429	T	1429	T
le450_15a	2 331	2 632	3272	T	45001	#M#	2819	#M#	**2715**	**#M#**	2946	T	2946	T
le450_15b	2 348	2 632	3245	T	42751	#M#	2773	#M#	**2752**	**#M#**	2901	T	2896	T
le450_15c	2 610	3 487	17130	#M#	17130	#M#	5302	T	4665	T	4616	T	**4615**	**T**
le450_15d	2 628	3 505	17200	#M#	17200	#M#	5421	T	**4457**	**T**	4499	T	4499	T
le450_25a	3 003	3 153	3780	T	58051	#M#	3188	#M#	**3181**	**#M#**	3390	T	3390	T
le450_25b	3 305	3 365	4055	T	50401	#M#	3374	#M#	**3354**	**#M#**	3565	T	3565	T
le450_25c	3 657	4 515	17793	#M#	17793	#M#	5157	#M#	**4983**	**#M#**	5221	T	5221	T
le450_25d	3 698	4 544	17875	#M#	17875	#M#	**5169**	**#M#**	5284	#M#	5297	T	5297	T
mug88_1	164	178	178*	4335	178*	4534	178*	1	**178***	**1**	180	T	180	T

			Toulbar Bin		Toulbar AD		CPLEX Bin		CPLEX AD		Gecode Bin		Gecode AD	
Instance	LB_L	UB_L	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$	UB	$t(s)$
mug88_25	162	178	178*	286	178*	304	**178***	**1**	178*	1	180	T	180	T
mug100_1	188	202	202*	20320	202*	20316	202*	2	**202***	**2**	206	T	206	T
mug100_25	186	202	202*	10642	202*	10692	**202***	**5**	202*	5	208	T	208	T
myciel3	21	21	**21***	**0**	**21***	**0**	21*	0	21*	0	**21***	**0**	**21***	**0**
myciel4	45	45	**45***	**0**	**45***	**0**	45*	0	45*	0	45*	0	45*	0
myciel5	93	93	93*	6	93*	6	**93***	**2**	93*	2	93*	32583	93*	27573
myciel6	189	189	189	T	189	T	**189***	**16**	189*	20	189	T	189	T
myciel7	286	381	480	T	480	T	381*	11781	**381***	**11036**	381	T	381	T
qg.order30	13950	13950	14160	T	14160	T	13950*	505	13950*	526	**13950***	**1**	13950*	1
qg.order40	32800	32800	64000	#M#	64000	#M#	32800*	58036	32800*	58331	**32800***	**5**	32800*	5
qg.order60	109800	109800	216000	#M#	216000	#M#	216000	#M#	116520	T	**109804**	**T**	109804	T
r125.1			257*	75097	1126	#M#	**257***	**0**	**257***	**0**	258	T	258	T
r125.1c			3044	T	15626	#M#	**2184***	**38848**	7626	#M#	2336	T	2336	T
r125.5			2259	T	12501	#M#	1830	T	**1826**	**T**	2168	T	2168	T
r250.1			793	T	757	#M#	**704***	**1**	704*	1	744	T	744	T
r250.1c			30477	#M#	30477	#M#	30477	#M#	30477	#M#	**6318**	**T**	6318	T
r250.5			15099	#M#	48001	#M#	**6598**	**T**	6918	T	7708	T	7708	T
r1000.1			9038	T	50001	#M#	7350	T	**7328**	**T**	7703	T	7703	T
fpsol2.i.1	3403	3403	12150	#M#	12150	#M#	3403*	15	**3403***	**10**	3464	T	3464	T
fpsol2.i.2	1668	1668	9142	#M#	9142	#M#	**1668***	**7**	1668*	7	1730	T	1730	T
fpsol2.i.3	1636	1636	9113	#M#	9113	#M#	1636*	7	**1636***	**7**	1670	T	1670	T
inithx.i.1	3676	3676	19571	#M#	19571	#M#	3676*	26	**3676***	**23**	3685	T	3685	T
inithx.i.2	2050	2050	14624	#M#	14624	#M#	**2050***	**13**	2050*	16	2131	T	2133	T
inithx.i.3	1986	1986	14590	#M#	14590	#M#	**1986***	**12**	1986*	15	2019	T	2019	T
mulsol.i.1	1957	1957	3282	T	24035	#M#	1957*	2	**1957***	**1**	1958	T	1958	T
mulsol.i.2	1191	1191	3172	T	29517	#M#	1191*	2	**1191***	**1**	1195	T	1195	T
mulsol.i.3	1187	1187	3102	T	29073	#M#	1187*	2	**1187***	**1**	1191	T	1191	T
mulsol.i.4	1189	1189	3281	T	29416	#M#	1189*	2	**1189***	**1**	1194	T	1194	T
mulsol.i.5	1160	1160	3248	T	29761	#M#	1160*	2	**1160***	**1**	1166	T	1166	T
zeroin.i.1	1822	1822	2790	T	23633	#M#	1822*	2	**1822***	**2**	1869	T	1869	T
zeroin.i.2	1004	1004	2482	T	29752	#M#	1004*	2	**1004***	**1**	1055	T	1055	T
zeroin.i.3	998	998	2474	T	29047	#M#	998*	2	**998***	**1**	1033	T	1034	T
school1	2439	2674	19480	#M#	19480	#M#	5757	T	19480	#M#	4610	T	**4610**	**T**
school1_nsh	2176	2392	14964	#M#	14964	#M#	**2425**	**T**	4971	#M#	3874	T	3872	T
anna	276	276	298	T	280	#M#	**276***	**0**	276*	0	284	T	284	T
david	237	237	275	T	250	#M#	237*	0	**237***	**0**	245	T	245	T
homer	1129	1150	1184	T	1204	#M#	**1150***	**25**	1150*	31	1215	T	1215	T
huck	243	243	273	T	248	#M#	**243***	**0**	**243***	**0**	243	T	243	T
jean	217	217	224	T	222	#M#	**217***	**0**	217*	0	217	T	217	T
miles250	325	325	358	T	358	#M#	**325***	**0**	325*	0	335	T	340	T
miles500	705	705	806	T	4993	#M#	**705***	**24**	705*	38	752	T	752	T
miles750	1145	1173	1435	T	1280	T	1173*	163	**1173***	**144**	1294	T	1294	T
miles1000	1623	1666	1915	T	11137	#M#	**1666***	**113**	1666*	243	1874	T	1874	T
miles1500	3239	3354	3689	T	13697	#M#	3354*	21	**3354***	**9**	3603	T	3603	T
queen5_5	75	75	75*	101	75	#M#	**75***	**0**	**75***	**0**	75*	15	75*	15
queen6_6	138	138	142	T	144	#M#	**138***	**451**	138*	503	142	T	138	T
queen7_7	196	196	226	T	217	#M#	196*	1	**196***	**1**	207	T	206	T
queen8_8	291	291	323	T	336	#M#	291*	32690	**291***	**29286**	314	T	314	T
queen8_12	624	624	633	T	633	T	624*	107	**624***	**2**	632	T	630	T
queen9_9	405	409	465	T	492	#M#	410	#M#	**409**	**#M#**	443	T	443	T
queen10_10	550	553	646	T	653	#M#	**559**	**#M#**	565	#M#	619	T	619	T
queen11_11	726	733	878	T	860	#M#	739	#M#	**736**	**#M#**	809	T	809	T
queen12_12	936	943	1109	T	1084	#M#	**962**	**#M#**	965	#M#	1052	T	1051	T
queen13_13	1183	1191	1402	T	8282	#M#	1209	#M#	**1207**	**#M#**	1273	T	1273	T
queen14_14	1470	1482	1743	T	1755	#M#	1565	#M#	**1514**	**T**	1625	T	1621	T
queen15_15	1800	1814	2065	T	2031	#M#	1947	T	**1933**	**#M#**	2004	T	2004	T
queen16_16	2176	2193	2588	T	15361	#M#	**2259**	**T**	2293	T	2352	T	2352	T
wap05a	12449	13656	43986	#M#	43986	#M#	15300	T	**14323**	**T**	15195	T	15195	T
wap06a	12454	13773	44518	#M#	44518	#M#	16319	T	**14416**	**T**	15427	T	15427	T
wap07a	24800	28617	105177	#M#	105177	#M#	105177	#M#	36355	T	32365	T	**32365**	**T**
wap08a	25283	28885	106046	#M#	106046	#M#	106046	#M#	36484	T	32496	T	**32496**	**T**

Résultats complets, en deux parties, pour la comparaison entre la version simple de *minElim* et sa version améliorée :

			minElim				*minElim* f20-5 rl500 éch.			
Instance	LB_L	UB_L	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$
DSJC125.1	247	326	353	8.3	72	T	**338**	**3.7**	**72455**	**T**
DSJC125.5	549	1 012	1162	14.8	42	T	**1076**	**6.3**	**10900**	**T**
DSJC125.9	1 691	2 503	2819	12.6	0	T	**2566**	**2.5**	**44785**	**T**
DSJC250.1	570	970	1110	14.4	79024	T	**1049**	**8.1**	**23063**	**T**
DSJC250.5	1 287	3 210	3942	22.8	68694	T	**3591**	**11.9**	**3023**	**T**
DSJC250.9	4 311	8 277	9686	17	3	T	**8992**	**8.6**	**2478**	**T**
DSJC500.1	1 250	2 836	3323	17.2	32448	T	**3163**	**11.5**	**63279**	**T**
DSJC500.5	2 923	10 886	13052	19.9	46965	T	**12661**	**16.3**	**15167**	**T**
DSJC500.9	11 053	29 862	33717	12.9	23	T	**32814**	**9.9**	**133**	**T**
DSJC1000.1	2 762	8 991	10565	17.5	206	T	**10328**	**14.9**	**37648**	**T**
DSJC1000.5	6 708	37 575	45748	21.8	654	T	**44887**	**19.5**	**72**	**T**
DSJC1000.9	26 557	103 445	122562	18.5	212	T	**119347**	**15.4**	**453**	**T**
DSJR500.1	2 069	2 156	2331	8.1	0	T	**2258**	**4.7**	**64802**	**T**
DSJR500.1c	15 398	16 286	17815	9.4	27	T	**17218**	**5.7**	**1165**	**T**
DSJR500.5	22 974	25 440	29214	14.8	2	T	**28179**	**10.8**	**13980**	**T**
flat300_20_0	1 531	3 150	4992	58.5	18690	T	**4304**	**36.6**	**21570**	**T**
flat300_26_0	1 548	3 966	5181	30.6	20326	T	**4843**	**22.1**	**16287**	**T**
flat300_28_0	1 547	4 238	5063	19.5	81249	T	**4821**	**13.8**	**1596**	**T**
flat1000_50_0	6 601	25 500	45516	78.5	11951	T	**43872**	**72**	**10353**	**T**
flat1000_60_0	6 640	30 100	45560	51.4	1914	T	**44206**	**46.9**	**3529**	**T**
flat1000_76_0	6 632	37 164	45132	21.4	43721	T	**44149**	**18.8**	**14253**	**T**
1-FullIns_3			54*	0	0	1	**54***	**0**	**0**	**0**
1-FullIns_4			166	0	0	T	**166**	**0**	**0**	**T**
1-FullIns_5			**499**	**0**	**0**	**T**	**499**	**0**	**0**	**T**
2-FullIns_3			93*	0	5739	8971	**93***	**0**	**0**	**98**
2-FullIns_4			363	0	106	T	**363**	**0**	**0**	**T**
2-FullIns_5			1435	0.1	2	T	**1433**	**0**	**14**	**T**
3-FullIns_3			159	9.7	177	T	**145**	**0**	**0**	**T**
3-FullIns_4			**683**	**0**	**0**	**T**	**683**	**0**	**0**	**T**
3-FullIns_5			3343	0.2	22	T	**3335**	**0**	**62**	**T**
4-FullIns_3			213	3.9	54	T	**205**	**0**	**0**	**T**
4-FullIns_4			1145	0.6	2795	T	**1138**	**0**	4	**T**
4-FullIns_5			6679	0	72927	T	**6679**	**0**	**1999**	**T**
5-FullIns_3			300	7.1	0	T	**280**	**0**	**0**	**T**
5-FullIns_4			1824	2.7	3	T	**1776**	**0**	**13**	**T**
games120	443	443	456	2.9	0	T	**443**	**0**	**12771**	**T**
ash331GPIA			1511	4.9	308	T	**1440**	**0**	**48289**	**T**
ash608GPIA			2758	4.4	5	T	**2641**	**0**	**25077**	**T**
ash958GPIA			4345	2.6	18	T	**4236**	**0**	**26698**	**T**
will199GPIA			2034	4.1	26687	T	**1953**	**0**	**21157**	**T**
1-Insertions_4			**119**	**0**	**0**	**T**	**119**	**0**	**0**	**T**
1-Insertions_5			357	0	0	T	**357**	**0**	**0**	**T**
1-Insertions_6			1068	0	1	T	**1068**	**0**	**1**	**T**
2-Insertions_3	55	62	62*	0	0	10	**62***	**0**	**0**	**2**
2-Insertions_4			**249**	**0**	**0**	**T**	**249**	**0**	**0**	**T**
2-Insertions_5			**996**	**0**	**1**	**T**	996	0	1	T
3-Insertions_3	84	92	**92***	**0**	**0**	**5012**	92*	0	0	15386
3-Insertions_4			459	0	0	T	**459**	**0**	**0**	**T**
3-Insertions_5			**2289**	**0**	**7**	**T**	2289	0	7	T
4-Insertions_3			**127**	**0**	**0**	**T**	**127**	**0**	**0**	**T**
4-Insertions_4			**761**	**0**	**0**	**T**	761	0	0	T
latin_square_10	40 950	41 444	46844	13	6425	T	**45372**	**9.5**	**1054**	**T**
le450_5a	1 193	1 350	1749	29.6	14401	T	**1424**	**5.5**	**998**	**T**
le450_5b	1 189	1 350	1733	28.4	0	T	**1509**	**11.8**	**16520**	**T**
le450_5c	1 278	1 350	1416	4.9	0	T	**1367**	**1.3**	**4937**	**T**
le450_5d	1 282	1 350	1429	5.9	0	T	**1350**	**0**	**389**	**T**
le450_15a	2 331	2 632	2946	11.9	44789	T	**2852**	**8.4**	**16018**	**T**
le450_15b	2 348	2 632	2896	10	69205	T	**2857**	**8.5**	**7654**	**T**
le450_15c	2 610	3 487	4615	32.3	58079	T	**4406**	**26.4**	**45409**	**T**
le450_15d	2 628	3 505	4499	28.4	4875	T	**4381**	**25**	**33148**	**T**
le450_25a	3 003	3 153	3390	7.5	71030	T	**3287**	**4.2**	**41983**	**T**
le450_25b	3 305	3 365	3565	5.9	14405	T	**3502**	**4.1**	**34895**	**T**
le450_25c	3 657	4 515	5221	15.6	26206	T	**5009**	**10.9**	**32000**	**T**
le450_25d	3 698	4 544	5297	16.6	1712	T	**5042**	**11**	**39090**	**T**

			minElim				*minElim* f20-5 rl500 sw			
Instance	LB_L	UB_L	UB	Dist	$t_{UB}(s)$	$t(s)$	UB	Dist	$t_{UB}(s)$	$t(s)$
mug88_1	164	178	180	1.1	71922	T	**179**	**0.6**	**69565**	**T**
mug88_25	162	178	180	1.1	0	T	**178**	**0**	**29320**	**T**
mug100_1	188	202	206	2	596	T	**204**	**1**	**2149**	**T**
mug100_25	186	202	208	3	245	T	**204**	**1**	**1**	**T**
myciel3	21	21	**21***	**0**	**0**	**0**	**21***	**0**	**0**	**0**
myciel4	45	45	45*	0	0	0	**45***	**0**	**0**	**0**
myciel5	93	93	**93***	**0**	**0**	**27573**	93	0	0	T
myciel6	189	189	**189**	**0**	**0**	**T**	**189**	**0**	**0**	**T**
myciel7	286	381	**381**	**0**	**0**	**T**	381	0	0	T
qg.order30	13950	13950	13950*	0	1	1	**13950***	**0**	**1**	**1**
qg.order40	32800	32800	**32800***	**0**	**5**	**5**	32800*	0	6	6
qg.order60	109800	109800	109804	0	60	T	**109800***	**0**	**157**	**157**
r125.1			258	0.4	14145	T	**257***	**0**	**1726**	**5347**
r125.1c			2336	3.9	0	T	**2249**	**0**	**5091**	**T**
r125.5			2168	6.8	0	T	**2030**	**0**	**12944**	**T**
r250.1			744	3.2	8086	T	**721**	**0**	**25061**	**T**
r250.1c			6318	6.2	4	T	**5951**	**0**	**3223**	**T**
r250.5			7708	4.5	0	T	**7376**	**0**	**20002**	**T**
r1000.1			7703	1.9	2	T	**7557**	**0**	**41807**	**T**
fpsol2.i.1	3403	3403	3464	1.8	1	T	**3405**	**0.1**	**62**	**T**
fpsol2.i.2	1668	1668	1730	3.7	1	T	**1670**	**0.1**	**13926**	**T**
fpsol2.i.3	1636	1636	1670	2.1	103	T	**1638**	**0.1**	**60110**	**T**
inithx.i.1	3676	3676	3685	0.2	2	T	**3676**	**0**	**58**	**T**
inithx.i.2	2050	2050	2133	4	3748	T	**2061**	**0.5**	**8264**	**T**
inithx.i.3	1986	1986	2019	1.7	1	T	**1986**	**0**	**583**	**T**
mulsol.i.1	1957	1957	1958	0.1	0	T	**1957**	**0**	**0**	**T**
mulsol.i.2	1191	1191	1195	0.3	0	T	**1195**	**0.3**	**0**	**T**
mulsol.i.3	1187	1187	**1191**	**0.3**	**0**	**T**	1191	0.3	0	T
mulsol.i.4	1189	1189	1194	0.4	0	T	**1193**	**0.3**	**0**	**T**
mulsol.i.5	1160	1160	1166	0.5	0	T	**1164**	**0.3**	**2**	**T**
zeroin.i.1	1822	1822	1869	2.6	0	T	**1822**	**0**	**10**	**T**
zeroin.i.2	1004	1004	1055	5.1	0	T	**1014**	**1**	**3**	**T**
zeroin.i.3	998	998	1034	3.6	0	T	**1012**	**1.4**	**0**	**T**
school1	2439	2674	4610	72.4	75	T	**3647**	**36.4**	**78077**	**T**
school1_nsh	2176	2392	3872	61.9	58392	T	**3031**	**26.7**	**2978**	**T**
anna	276	276	284	2.9	1	T	**276**	**0**	**10172**	**T**
david	237	237	245	3.4	540	T	**240**	**1.3**	**827**	**T**
homer	1129	1150	1215	5.7	8551	T	**1182**	**2.8**	**38408**	**T**
huck	243	243	**243**	**0**	**0**	**T**	**243**	**0**	**0**	**T**
jean	217	217	217	0	8	T	**217**	**0**	**0**	**T**
miles250	325	325	340	4.6	6	T	**332**	**2.2**	**533**	**T**
miles500	705	705	752	6.7	7410	T	**726**	**3**	**13187**	**T**
miles750	1145	1173	1294	10.3	8299	T	**1236**	**5.4**	**14471**	**T**
miles1000	1623	1666	1874	12.5	0	T	**1755**	**5.3**	**142**	**T**
miles1500	3239	3354	3603	7.4	0	T	**3431**	**2.3**	**117**	**T**
queen5_5	75	75	75*	0	0	15	**75***	**0**	**0**	**0**
queen6_6	138	138	138	0	60436	T	**138**	**0**	**0**	**T**
queen7_7	196	196	206	5.1	61564	T	**196**	**0**	**3**	**T**
queen8_8	291	291	314	7.9	41641	T	**300**	**3.1**	**298**	**T**
queen8_12	624	624	630	1	78089	T	**626**	**0.3**	**3026**	**T**
queen9_9	405	409	443	8.3	1066	T	**422**	**3.2**	**6533**	**T**
queen10_10	550	553	619	11.9	9182	T	**575**	**4**	**4671**	**T**
queen11_11	726	733	809	10.4	49125	T	**763**	**4.1**	**83421**	**T**
queen12_12	936	943	1051	11.5	49058	T	**988**	**4.8**	**777**	**T**
queen13_13	1183	1191	1273	6.9	965	T	**1242**	**4.3**	**22461**	**T**
queen14_14	1470	1482	1621	9.4	46819	T	**1533**	**3.4**	**32647**	**T**
queen15_15	1800	1814	2004	10.5	4505	T	**1885**	**3.9**	**307**	**T**
queen16_16	2176	2193	2352	7.3	38551	T	**2269**	**3.5**	**3528**	**T**
wap05a	12449	13656	15195	11.3	7198	T	**14906**	**9.2**	**28423**	**T**
wap06a	12454	13773	15427	12	817	T	**14992**	**8.9**	**51609**	**T**
wap07a	24800	28617	32365	13.1	1637	T	**31923**	**11.6**	**3121**	**T**
wap08a	25283	28885	32496	12.5	3354	T	**32094**	**11.1**	**66946**	**T**