République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieure et de la Recherche Scientifique

Centre Universitaire Belhadj Bouchaib d'Aïin-Témouchent

Faculté des Sciences

Département de Mathématiques et de l'Informatique

Mémoire

Pour l'obtention du Diplôme de Master en Informatique

Option : Réseaux et Ingénierie des Données (RID)

Présenté par :

Melle. Khadidja BOUAZZA MAROUF

Melle. Nouha BOUSSAID

SYMÉTRIES ET DOMINANCES DANS LES RÉSEAUX DE CONTRAINTES DE DIFFÉRENCE.

Encadrant :

Mr. Mohamed Réda SAIDI

Maitre Conference "B" à C.U.B.B.A.T.

Soutenu en 2019

Devant le jury composé de :

Président :	Mr. Mohammed Hakim BENDIABDALLAH (M.C.B)	C.U.B.B.A.T.
Examinateurs :	Melle. Khadidja MEGAGUE (M.A.A)	C.U.B.B.A.T.
Encadrant :	Mr. Mohamed Réda SAIDI (M.C.B)	C.U.B.B.A.T.

Remerciements

Pour la chance et le bonheur qu'il nous offre, pour le bien vers lequel il nous guide, pour le courage, la patience et la foi qu'il nous donne pour mener à bien notre projet. Merci au plus puissant ***ALLAH****.*

Au terme de ce travail nous tenons à remercier vivement notre encadreur Dr.SAIDI Mohamed Réda pour sa disponibilité, son guide et ses suggestions, orientations et remarques fructueuses, ses efforts déployés et ses précieux conseils, qu'il trouve ici notre profonde gratitude.

Nous tenons à exprimer notre respect et notre sincère remerciement aux membres de jury, Mr.BENDIABDALLAH Mohammed Hakim de nous avoir fait l'honneur de présider ce jury et Melle.MEGAGUE Khadidja d'avoir accepté d'examiner et dévaluer ce travail.

Sans oublier de remercier l'ensemble des enseignants de centre universitaire BELHADJ Bouchaïb d'Aïn Témouchent (CUAT) spécialement ceux du département de mathématiques et informatique pour leurs encadrements tout au long de notre cursus d'études.

Tous ceux qui ont contribué de près ou de loin à la réalisation de ce travail trouvent ici l'expression de notre sincères considérations et nos meilleures gratitudes. De notre part, nous espérons que notre conduite a laissé une bonne impression.

Dédicace

Nous dédions ce mémoire à...

À nos chers parents pour leurs soutiens indéfectibles et leurs efforts fournis jour et nuit pour nos éducations. Aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que nous avons toujours eu pour vous.
Que dieu vous protège et que la réussite soit toujours à nos portées pour que nous puissons vous combler de bonheur.

À nos chers sœurs et frères pour leurs encouragements permanents, et leurs soutiens moral.

À toutes nos familles, nos amis et nos proches pour leurs soutiens tout au long de nos parcours universitaire.

BOUAZZA MAROUF K.
BOUSSAID N.

Résumé

De nombreux problèmes pratiques (coloration de cartes géographiques, conception d'emplois du temps, allocation des ressources du réseau...) peuvent être modélisés par des problèmes de colorations de graphes qui peuvent eux-mêmes exprimer par des CSPs à contraintes de différence (NECSPs). Bien que nous ayons actuellement plusieurs méthodes pour résoudre efficacement des problèmes de taille raisonnable, il n'en reste qu'en théorie, ce problème est NP-complet, c'est-à-dire que toutes ces techniques ont une complexité exponentielle dans le pire des cas. Il existe tout de même, plusieurs façons de diminuer cette complexité, l'une d'entre elles est l'élimination de la symétrie. La détection des symétries du problème pendant la recherche de solutions permet donc de ne pas considérer certains cas redondants (symétriques), où la recherche de solutions est vouée à l'échec.

Dans ce mémoire, notre travail consiste à apporter des améliorations pour un algorithme exact afin de résoudre le problème de coloration de graphes formalisé sous la forme de NECSPs proposé en 2006 qui est basé sur la détection et l'élimination des symétries et de dominances. Les heuristiques proposées améliorent le choix de la prochaine variable à affecter. Les résultats des expérimentations sont très encourageants.

Mots-clés: CSP à contraintes de différence, problème de coloration de graphe, symétrie, dominance, heuristiques.

Abstract

Many practical problems (coloring maps, design of schedules, allocation of network resources ...) can be modeled by graphical problems of graphs that can express themselves by CSPs with constraints of difference (NECSPs). Although we currently have several methods to effectively solve problems of reasonable size, it remains only in theory, this problem is NP-complete, that is to say that all these techniques have an exponential complexity in the worst cases. There are still several ways to reduce this complexity, one of which is the elimination of symmetry. The detection of the symmetries of the problem during the search for solutions thus makes it possible not to consider certain redundant (symmetrical) cases where the search for solutions is doomed to failure.

In this report, our work consists in making improvements for an exact algorithm in order to solve the graph coloring problem formalized in the form of NECSPs proposed in 2006 that is based on the detection and elimination of symmetries and dominance. The proposed heuristics improve the choice of the next variable to be affected. The results of the experiments are very encouraging.

Keywords: Not equal Constraint satisfaction problem, graph coloring problem, symmetry, dominance, heuristics.

ملخص

العديد من المشاكل العملية (تلوين الخرائط الجغرافية، تصميم جداول الوقت, تخصيص موارد الشبكة....) يمكن ان تمثل على شكل مخططات بيانية التي يمكن أن تعبر بحد ذاتها بواسطة مشاكل تحقيق القيود CSPs)) مع قيود الفرق (NECSPs). على الرغم من أن لدينا حاليا عدة طرق فعالة لحل مشاكل ذات حجم معقول ، يبقى ذلك نظريا فقط ، هذه المشكلة تعد كاملة (NP-complet) ، وهذا يعني أن كل هذه التقنيات معقدة بشكل كبير في أسوأ الحالات. ومع ذلك، يوجد هناك عدة طرق لتقليل هذا التعقيد، أحدها إزالة التناظر. يسمح اكتشاف المتناظرات الموجودة في المشكلة أثناء البحث عن حلول بعدم الأخذ بعين الاعتبار بعض الحالات المتكررة (المتناظرة) ، حيث يؤدي إلى البحث عن حلول مصيرها الفشل.

في هذه المذكرة ، مهمتنا هي إجراء تحسينات لخوارزمية دقيقة من أجل حل مشكلة تلوين المخطط البياني المشكلة المعير عنها على شكل (NECSPs) المقترح في عام 2006 والذي يستند إلى الكشف والقضاء على المتناظرات و من المهيمنات. الاستدلال المقترح يحسن اختيار المتغير التالي المراد تعيينه. نتائج التجارب مشجعة للغاية.

المفتاحية الكلمات : مشكلة تحقيق قيود الفروق، مشكلة تلوين المخطط البياني، التناظر، الهيمنة، الاستدلال.

Table des matières

Chapitre 1
Introduction Générale **1**

Chapitre 2
Problème de Satisfaction de Contraintes (CSP) **5**

2.1 Introduction 6
2.2 Formalisme CSP 6
2.3 Méthodes de résolution du CSP 10
2.3.1 Consistances partielles et méthodes de filtrage 10
2.3.2 Méthodes complètes 10
2.3.2.1 Générer et tester (GeT) 10
2.3.2.2 Le backtrack 12
2.3.2.3 Amélioration du Backtracking 12
2.3.2.4 Heuristiques d'instanciation 13
2.3.2.5 Méthodes de décomposition 15
2.3.3 Méthodes incomplètes 15
2.4 CSPs à contraintes de différence 16
2.5 Conclusion 17

Chapitre 3
La coloration des graphes **19**

3.1 Introduction 20
3.2 Historique 20
3.3 Méthodes de résolution 20
3.3.1 Les méthodes exactes 20
3.3.2 Les méthodes approchées 21

3.3.2.1 Les méthodes constructives 21
3.3.2.2 Les méta-heuristiques 21
3.3.2.3 Les algorithmes de recherche locale 21
3.3.2.4 Les algorithmes génétiques 21
3.3.3 Les méthodes hybrides 21
3.4 Domaines d'applications 22
3.4.1 Coloration d'une carte géographique 22
3.4.2 Problème du sudoku 24
3.4.3 Problème d'allocation de fréquences 28
3.5 Conclusion 30

Chapitre 4
Symétrie et dominance dans les CSPs à contraintes de différence 31

4.1 Introduction 32
4.2 Théorie des groupes et symétrie 32
4.3 Symétrie dans les CSPs 33
4.3.1 Problème des reines : 33
4.3.2 Symétrie dans les CSPs quelconques 34
4.3.3 Une condition suffisante pour la symétrie dans les NECSPs 35
4.3.4 Affaiblissement des conditions de la symétrie dans les NECSPs 36
4.4 La dominance dans les NECSPs 37
4.4.1 Le principe de la dominance 37
4.4.2 Une condition suffisante pour la dominance dans les NECSPs 37
4.4.3 Affaiblissement de la condition de dominance 38
4.4.4 Détection de la Dominance 39
4.5 Conclusion 40

Chapitre 5
Exploitation de la symétrie et la dominance dans les NECSPs 41

5.1 Introduction 42
5.2 Principe de SFC-weak-dom 42
5.3 Améliorations 44
5.3.1 Première amélioration : 44
5.3.2 Deuxième amélioration : 45
5.4 Expérimentations 48
5.4.1 Problèmes aléatoires de coloration des graphes 48
5.4.2 Les benchmarks de Dimacs 53

5.4.2.1 Instances Dimacs 53
5.5 Conclusion 56

Chapitre 6
Conclusion **57**

Bibliographie **59**

Chapitre 1

Introduction Générale

La programmation par contraintes (PPC) apparue à la fin des années 80, c'est une technique de résolution des problèmes combinatoires complexes issus de la programmation logique et de l'intelligence artificielle. Elle consiste à modéliser un problème par un ensemble de relations logiques, des contraintes, imposant des conditions sur l'instanciation possible d'un ensemble de variables définissant une solution du problème. À partir des travaux de Montanari [Montanari, 1974], les chercheurs en intelligence artificielle ont développé un formalisme pour la modélisation et la résolution des problèmes qui satisfont des contraintes, connu sous le nom de CSP (pour **C**onstraint **S**atisfaction **P**roblem en anglais). Ce formalisme est présenté par la suite avec quelques notions et concepts de base.

Les problèmes de satisfaction de contraintes sont des ensembles de contraintes impliquant des ensembles de variables. La résolution de ces problèmes consiste à calculer une solution en attribuant à chacune des variables une valeur satisfaisant simultanément toutes les contraintes. Les CSPs ont une grande utilité au cœur de nombreux domaines en intelligence artificielle et en recherche opérationnelle. Parmi ces applications pratiques qui peuvent être modélisées par des CSPs, nous citons : l'allocation des registres en compilation et l'allocation des fréquences, la coloration de cartes géographiques, la gestion du trafic aérien et le problème de la coloration des graphes . Ce problème remonte au 19ème siècle où une question est devenue célèbre sous le nom de problème des quatre couleurs : suffit-il de quatre couleurs pour colorier n'importe quelle carte géographique ?. La coloration des sommets d'un graphe consiste à affecter à tous les sommets de ce graphe une couleur de telle sorte que deux sommets adjacents(reliés par une arête) ne portent pas la même couleur. Ce problème peut être trivialement exprimé sous forme d'un CSP.

Bien que nous ayons actuellement plusieurs méthodes pour résoudre efficacement des problèmes de taille raisonnable, il en reste qu'en théorie, les CSPs font partie de la classe des problèmes NP-complets [1] [Hartmanis, 1982], c'est-à-dire que toutes les approches complètes connues pour les résoudre ont une complexité en temps exponentielle dans le pire des cas. Plusieurs techniques existent pour diminuer cette complexité. L'une d'entre elles est l'élimination de la symétrie. La détection des symétries du problème pendant la recherche de solutions permet de ne pas considérer certains cas redondants (symétriques), où la recherche de solutions est vouée à l'échec.

À notre connaissance, le principe de symétrie a été introduit pour la première fois par Krishnamurty [Krishnamurty, 1985] pour optimiser la résolution en logique propositionnelle. La symétrie pour les contraintes booléennes a été étudiée dans [Benhamou and Sais, 1992; Benhamou and Sais, 1994; Benhamou *et al.*, 1994], les auteurs ont montré comment les détecter et ont prouvé que leur exploitation est une réelle amélioration de l'efficacité de plusieurs algorithmes de déduction automatique. La notion de la symétrie pour les CSPs a été étudiée dans[Puget, 1993; Benhamou, 1994a]. À partir de ce moment-là, plusieurs recherches sur la symétrie sont apparues. Cependant, la plus part de ces approches exploitent uniquement les symétries globales [2] du problème initial et aucune méthode permettant d'éliminer les symétries locales [3] n'est proposée.

1. Les problèmes NP-complets sont des problèmes où il n'existe aucun algorithme jusqu'à présent pour les résoudre en un temps polynomiale. Ils ont une complexité exponentielle dans le pire des cas.
2. Les symétries initiales du problème apparaissant à la racine de l'arbre de recherche.
3. Les symétries du CSP résultant à un noeud de l'arbre de recherche correspondant à une instanciation partielle.

Prenons par exemple le CSP P muni de trois variables x, y et z ayant le même domaine {0,1,2} et une seule contrainte $y + xz = 3$. Le CSP admet trois solutions : (1) $\{x = 1, y = 1, z = 2\}$, (2) $\{x = 1, y = 2, z = 1\}$, (3) $\{x = 2, y = 1, z = 1\}$. Il a aussi, une symétrie globale entre les variables x et z. On peut éliminer cette symétrie, en postant la contrainte $(x < z)$ qui évitera de générer la solution (3) qui est globalement symétrique à la solution (1). Par ailleurs, si on affecte à la variable x la valeur 1, alors, les variables y et z deviennent symétriques dans le CSP $P_{x=1}$ obtenu après la propagation de $x = 1$. C'est une symétrie locale du CSP P. On peut éliminer cette symétrie locale entre y et z en postant une contrainte dynamique $(y < z)$ qui empêcherait la génération de la solution (2) qui est localement symétrique à la solution (1). Cet exemple simple montre l'importance d'éliminer, non seulement les symétries globales, mais aussi les symétries locales, durant la résolution.

Les auteurs de [Benhamou and Saïdi, 2006] ont utilisé un algorithme exact pour la résolution du problème de coloration de graphe en exploitant la notion de la symétrie et la notion de dominance avec l'heuristique [4] *dom/deg*, leurs résultats montrent l'intérêt de ces deux notions. Nous avons fixé comme objectif : *sera-t-il possible d'améliorer les résultats qui ont été obtenus par Benhamou et Saïdi en 2006 en utilisant des techniques un petit peu plus sophistiqué ?*.

Finalement, nous montrons nos améliorations qui consistent à combiner l'algorithme *SFC-weak-dom* avec quelques heuristiques pour le rendre plus performant et nous évaluons et comparons les performances de notre travail à l'aide d'expérimentations sur des problèmes de coloration de graphe avec l'algorithme de base et $DSATUR$.

Ce travail est organisé comme suit : Nous rappelons dans le chapitre 2, certaines notions sur les CSPs où nous limitons sur les CSPs binaires. Dans le chapitre 3, nous parlons du problème de la coloration de graphe. Puis, nous abordons les notions de la symétrie et la dominance dans les NECSPs. Par la suite, nous donnons un algorithme amélioré basé sur le *SFC-weak-dom* de [Benhamou and Saïdi, 2006] et montrons sa performance par les résultats obtenus. Le dernier chapitre conclut ce travail et donne les perspectives que nous souhaitons à réaliser futurement.

4. La notion d'heuristique sera être détaillée dans le chapitre 2.

Chapitre 2

Problème de Satisfaction de Contraintes (CSP)

Sommaire

2.1 Introduction 6
2.2 Formalisme CSP 6
2.3 Méthodes de résolution du CSP 10
2.3.1 Consistances partielles et méthodes de filtrage 10
2.3.2 Méthodes complètes 10
2.3.3 Méthodes incomplètes 15
2.4 CSPs à contraintes de différence 16
2.5 Conclusion 17

2.1 Introduction

Cet état de l'art présente le formalisme CSP, nous nous limitons aux cas des CSPs binaires qui constituent le cadre théorique dans lequel nous travaillerons par la suite. Tout d'abord, nous rappelons quelques notions de base sur les CSPs quelconques, en suite, nous présentons quelques techniques et algorithmes de résolution de CSP.

2.2 Formalisme CSP

La définition qui suit correspond au formalisme introduit par Montanari [Montanari, 1974].

Définition 1 (CSP) *Un problème de satisfaction de contraintes(Constraint Satisfaction Problem en anglais) est un quadruplet $P = (V, D, C, R)$ où :*

- *$V = \{v_1, ..., v_n\}$ est un ensemble de n variables ;*
- *$D = \{D_1, ..., D_n\}$ est l'ensemble des domaines associés aux variables, D_i inclut l'ensemble des valeurs possibles pour la variable v_i ;*
- *$C = \{C_1, ..., C_m\}$ est l'ensemble de m contraintes reliant deux ou plusieurs variables ;*
- *$R = \{R_1, ..., R_m\}$ est l'ensemble des relations correspondant aux contraintes de C, R_i représente la liste des tuples permis par la contrainte C_i.*

Une fois un problème donné formalisé sous la forme d'un CSP, la résolution de ce dernier consiste à trouver une affectation de toutes les variables du CSP qui satisfaire toutes les contraintes de celui-ci. Nous présentera par la suite les différentes techniques de résolutions.

Définition 2 (Contrainte binaire) *Une contrainte binaire est une contrainte qui relie exactement deux variables. On notera C_{ij} la contrainte impliquant les deux variables v_i et v_j.*

Définition 3 (CSP binaire) *Un CSP P est appelé CSP binaire, si toutes les contraintes de P sont des contraintes binaires.*

Un CSP peut être représenté par un graphe appelé *graphe de contraintes* $G(V, E)$ dans lequel l'ensemble des sommets V est l'ensemble des variables et chaque arête de E connecte deux variables liées par une même contrainte $C_i \in C$ [Benhamou and Saïdi, 2006].

Exemple 1 *Soit $P = (V, D, C, R)$ un CSP définie par :*

- *$V = \{v_1, v_2, v_3, v_4\}$;*
- *$D = \{D_1 = \{a, b\}, D_2 = \{a, c\}, D_3 = \{a, b\}, D_4 = \{a, c\}\}$;*
- *$C = \{C_{13}(v_1 > v_3), C_{23}(v_2 = v_3), C_{34}(v_3 \neq v_4)\}$;*
- *$R = \{R_{13} = \{(a, a), (b, a)\}, R_{23} = \{(a, a), (b, a)\}, R_{34} = \{(a, c), (b, c)\}\}$.*

La Figure 2.1 donne le graphe de contraintes du CSP P.

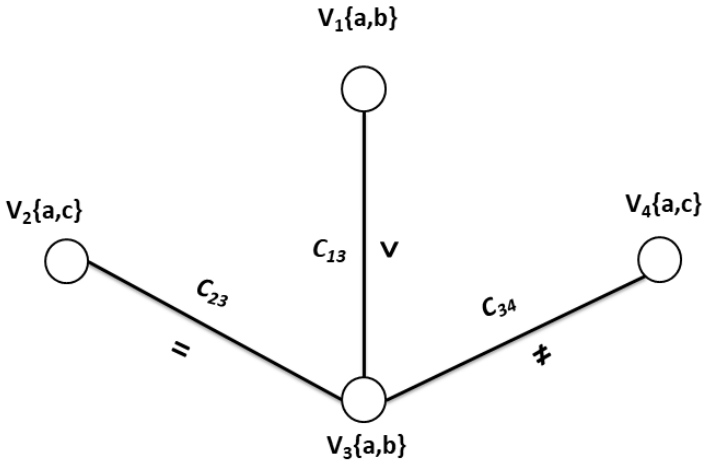


FIGURE 2.1 – Graphe de contraintes de P.

Aussi, le CSP peut être représenté par un graphe $M_P(\bigcup_{i=1}^{n}(\{v_i\} \times D_i), \acute{E})$ appelé *micro – structure* [Freuder, 1991; Jégou, 1993] où chaque sommet de ce graphe est représenté par un couple $\langle variable, valeur \rangle$ et chaque arête de $\acute{E}$ correspond soit à un tuple permis par une des contraintes du CSP, soit par un tuple permis car il n'y a pas de contraintes entre les variables correspondantes.

La micro-structure du CSP P est représentée dans la Figure 2.2.

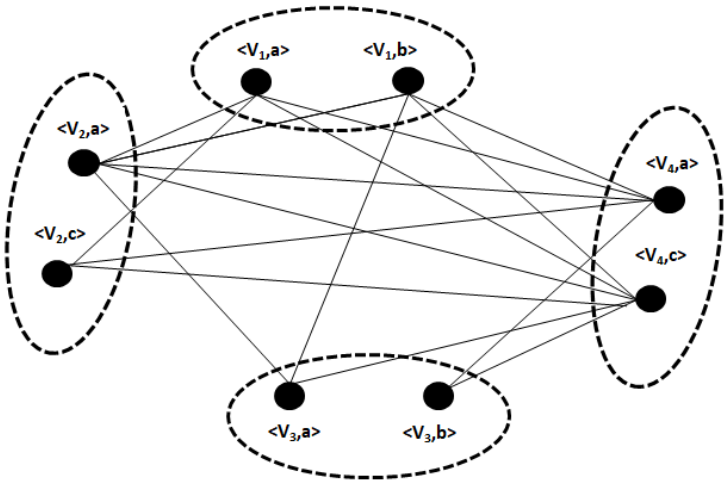


FIGURE 2.2 – Micro-structure de P.

Définition 4 (Instanciation) *Une instanciation* $I = \{\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, ..., \langle v_n, d_n\rangle\}$ *des variables d'un CSP* $P = (V, D, C, R)$ *est une affectation de ses variables* $\{v_1 = d_1, v_2 = d_2, \ldots, v_n = d_n\}$ *où chaque variable* v_i *est assignée à une valeur* d_i *de son domaine* D_i*. Une instanciation est dite* **totale** *si et seulement si elle est définie sur toutes les variables du CSP, sinon, on parle d'une instanciation* **partielle**.

Définition 5 (Solution) *L'instanciation totale I des variables d'un CSP $P = (V, D, C, R)$ est consistante si elle satisfaite toutes les contraintes de C. Alors, l'instanciation I est appelée solution du CSP P.*
Si un CSP P admet au moins une solution, il est dit consistant. Sinon, il est inconsistant.

Considérons le CSP de l'exemple 1, les instanciations suivantes :
- $I_1 = \{\langle v_1, a\rangle, \langle v_2, b\rangle\}$ est *partiellement inconsistante*, car elle viole la contrainte $C_{13}(v_1 > v_3)$;
- $I_2 = \{\langle v_1, b\rangle, \langle v_2, a\rangle\}$ est *partiellement consistante*, car elle satisfaire la contrainte $C_{13}(v_1 > v_3)$;
- $I_3 = \{\langle v_1, a\rangle, \langle v_2, c\rangle, \langle v_3, a\rangle, \langle v_4, a\rangle\}$ est *totalement inconsistante*, car elle viole toutes les contraintes ;
- $I_4 = \{\langle v_1, b\rangle, \langle v_2, a\rangle, \langle v_3, a\rangle, \langle v_4, c\rangle\}$ est *totalement consistante*, car elle satisfaite toutes les contraintes ;

Donc, I_4 est la seule solution pour le CSP P.

Jégou dans [Jégou, 1993] a montré que la résolution d'un problème CSP peut être ramenée au problème de recherche de cliques maximales dans sa micro-structure.

Définition 6 (Clique) *Dans un graphe non orienté un graphe et dit complet si chaque paire de sommets distincts est reliée par une arête.*
Une k-clique dans un graphe non orienté est un sous-ensemble de k sommets induisant un graphe complet (tous les sommets sont deux à deux adjacents).
Une clique maximale est une clique qui n'est pas un sous-ensemble propre d'une autre clique.

Exemple 2 *Le graphe G de la Figure 2.3 possède deux cliques maximales, la 1ère de taille 3 (v_4, v_5, v_6) et la 2ème de taille 4 (v_1, v_2, v_3, v_4).*

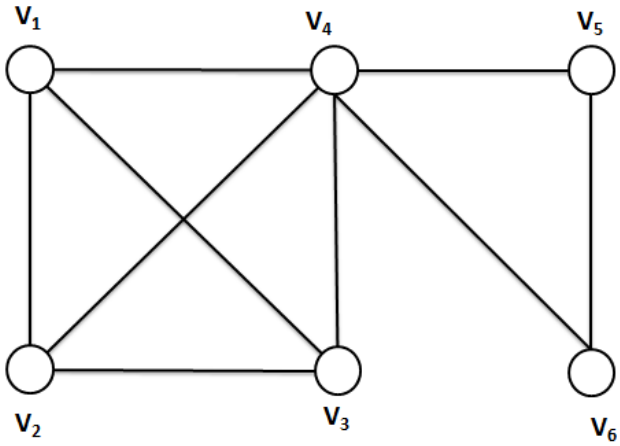


FIGURE 2.3 – Un graphe possédant deux cliques maximales (v_4, v_5, v_6) et (v_1, v_2, v_3, v_4).

La Figure 2.4 montre que la micro-structure de l'exemple 1 (Figure 2.2) possède une clique maximale de taille 4 (présentée en gras).

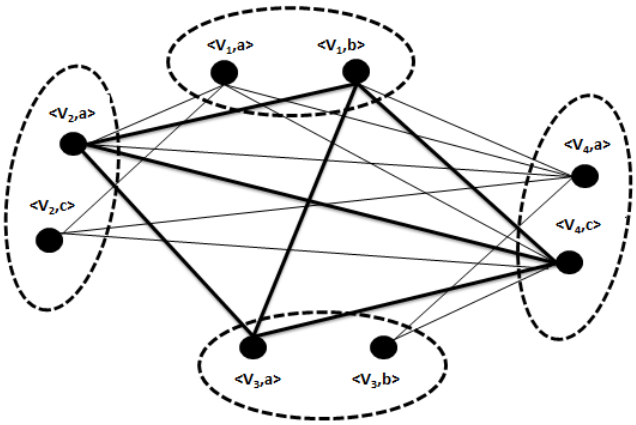


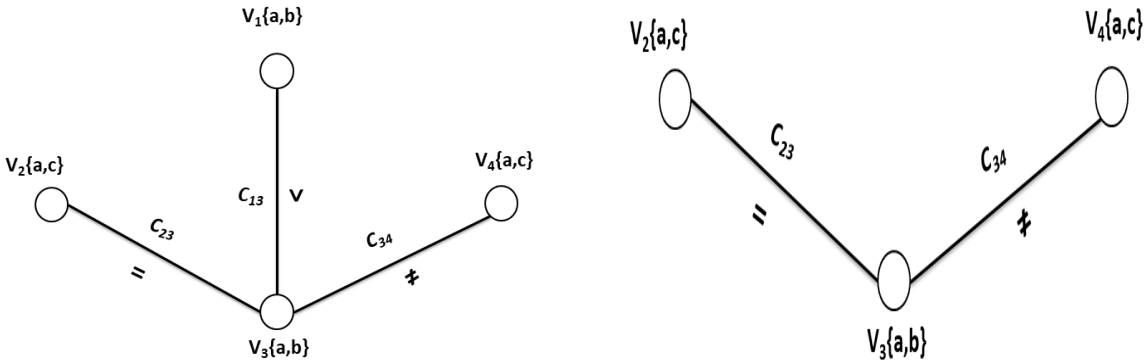
FIGURE 2.4 – La micro-structure du graphe P possédant une clique maximale de taille 4 (v_1, v_2, v_3, v_4).

Nous pouvons maintenant donner le théorème de Jégou.

Théorème 1 *[Jégou, 1993] Un CSP binaire composé de n variables possède une solution si et seulement si sa micro-structure possède une clique de taille n.*

Définition 7 (CSP local) *Soient P un CSP et I une instanciation partielle dans $P = (V, D, C, R)$, on appelle CSP local $P_I = (V', D', C', R')$ le sous-CSP de P tel que $V' \subset V$, $D' \subset D$, $C' \subset C$ et $R' \subset R$ où V' sont les variables non instanciées de P, les domaines D' sont obtenues à partir des domaines de D par les simplifications induites par l'instanciation I, C' et R' sont les restrictions de C et R aux variables de V' et aux domaines de D'.*

Exemple 3 *Prenons le graphe P de la 1ère figure, aprés avoir instancier v_1 à b, nous obtenons le graphe P'.*



(a) Graphe initial P.

(b) Graphe local P'.

FIGURE 2.5 – Graphe local P' aprés affecter b à v_1.

Définition 8 (Espace de recherche) *L'espace de recherche d'un CSP est l'ensemble des configurations possibles que l'on notera E, tel qu'il est égal au produit cartésien de l'ensemble des domaines des variables : $E = D_1 \times D_2 \times ... \times D_n$.*

Définition 9 (Arbre d'affectation) *Un arbre d'affectation d'un CSP P est définit durant la preuve de consistance, où tous les nœuds représentent les variables du CSP et les arêtes partant d'un nœud v_i sont étiquetées par les différentes valeurs utilisées pour instancier la variable correspondante v_i.*

Étant donné un CSP, on cherche à décider sa consistance ou à énumérer toutes ses solutions. Le problème de décision lié à l'existence d'une solution est NP-complet. Il existe plusieurs méthodes de résolution du CSP, elles peuvent être classées en deux grandes catégories : Les méthodes complètes qui garantissent l'obtention d'une solution, mais avec un coût élevé et les méthodes incomplètes moins coûteuses mais qui ne garantissent pas toujours l'obtention d'une solution.

2.3 Méthodes de résolution du CSP

2.3.1 Consistances partielles et méthodes de filtrage

Les techniques de filtrage permettent de réduire le problème initial en supprimant des inconsistances. Plusieurs niveaux de consistance ont été définis, la consistance d'arc (la 2-consistance) et la consistance de chemin (la 3-consistance). La consistance de base est la *consistance d'arc*, elle porte sur des couples de variables.

Définition 10 (Consistance d'arc) *Dans un CSP $P = (V, D, C, R)$, un domaine $D_i \in D$ est dit arc-consistant si et seulement si $D_i \neq \emptyset$ et pour tout $a \in D_i$, $\forall v_j$ telle que $C_{ij} \in C$, $\exists b \in D_j$ telle que $(a, b) \in R_{ij}$. Ainsi, un CSP P est dit* **arc-consistant** *si et seulement si, tous ses domaines sont arc-consistants.*

On peut associer à chaque niveau de consistance une opération de filtrage. Le calcul de ce dernier k-consistant a une complexité de $O(n^k.d^k)$ où n est le nombre de variables et d la taille du plus grand domaine [5].

2.3.2 Méthodes complètes

Nous présentons maintenant les différentes méthodes pour la résolution des CSPs par des algorithmes complets :

2.3.2.1 Générer et tester (GeT)

La méthode Générer et tester est une méthode simple pour la résolution des CSPs qui consiste à générer toutes les configurations possibles, c'est-à-dire toutes les combinaisons possibles de valeurs des variables et de tester si elles vérifient toutes les contraintes. Cette approche est connue sous le nom *Generate-and-test* (GaT). Le nombre de possibilités testées est alors le cardinal du produit cartésien des domaines des variables, ce qui pour les problèmes de grande tailles devient impossible à envisager [Lambert, 2006].

5. Dans la suite de cette étude n est le nombre des variables et d est la taille du plus grand domaine d'un CSP donné.

Dans la Figure 2.6, nous donnons le graphe du CSP Q et son arbre de recherche généré avec la méthode GaT sera illustré dans la Figure 2.7.

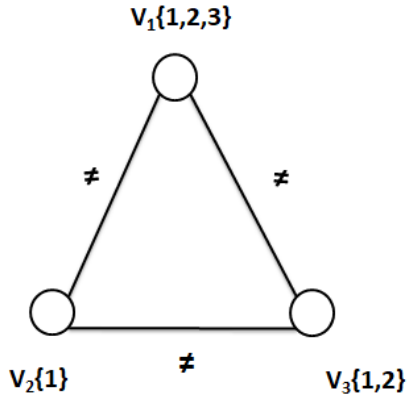


FIGURE 2.6 – Graphe de contraintes d'un CSP Q.

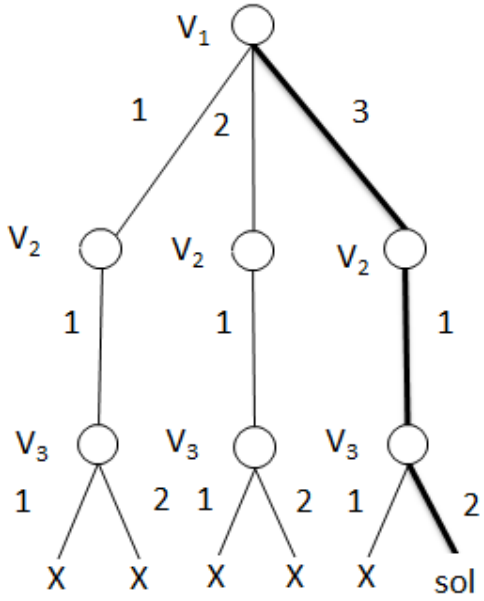


FIGURE 2.7 – Arbre de recherche de Q.

2.3.2.2 Le backtrack

L'algorithme backtrack consiste à construire progressivement une solution en affectant les variables une par une et en revenant en arrière à chaque échec rencontré. Des tests de consistance sont réalisés lors de l'instanciation d'une variable entre cette dernière et ceux réalisés sur les variables antérieures dans l'ordre, via les contraintes, afin de vérifier si l'instanciation courante est consistante. Si la nouvelle instanciation satisfait toutes les contraintes testées, alors l'instanciation courante est consistante. Si la variable que l'on vient d'instancier est la dernière variable du problème, l'instanciation courante constitue une solution au problème, sinon on passe à la variable suivante. Si une contrainte n'est pas satisfaite par la nouvelle instanciation, on choisit une nouvelle valeur pour instancier la variable courante, s'il n'y en a plus alors un retour en arrière est réalisé sur l'instanciation de la variable qui précède la variable courante pour essayer une autre valeur.

L'algorithme 1 décrit cette technique.

Algorithme 1 algorithme de Backtrack

PROCEDURE Backtrack(I, k)
Début
Entrée : $I = \{\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, ..., \langle v_k, d_k\rangle\}$ une instanciation partielle, k un entier
1: **si** (I est consistant) **alors**
2: **si** ($k = n$) {*n est le nombre de variables.*} **alors**
3: $<$ I est une solution $>$
4: **sinon**
5: **pour tout** ($d_{k+1} \in D_{k+1}$) **faire**
6: $I \leftarrow I \cup \{\langle v_{k+1}, d_{k+1}\rangle\}$
7: Backtrack($I, k+1$)
8: **fin pour**
9: **fin si**
10: **fin si**
fin

Montrer la consistance d'un CSP nécessite en général de faire un parcours exhaustif de l'espace de recherche dans le pire des cas. Or la taille de cet espace peut atteindre d^n. L'emploi de cette méthode pour la résolution d'un CSP peut s'avérer fort coûteux car exponentiel de nature. Pour palier cet inconvénient, d'autres approches ont été proposées, pour améliorer cette méthode.

2.3.2.3 Amélioration du Backtracking

Les améliorations suivent deux stratégies :

1. **Approches du type regarder devant** *Look-Ahead* [Haralik and Elliot, 1980]. Le principe des méthodes de cette catégorie est de détecter à l'avance des futures situations d'échec. Elles diffèrent selon le niveau du filtrage appliqué. Le premier niveau correspond à la procédure FC (*Forward Checking*) qui est l'un des algorithmes les plus connus (voir Algorithme 2). Il existe aussi la procédure *Partial Lookahead* et la procédure *Full Lookahead*, ou aussi la procédure *Real Full Lookahead* dont une version célèbre d'implémentation est MAC (*Maintaining Arc Consistency*) [Sabin and Freuder, 1997]. Le filtrage utilisé dans la méthode Forward Checking se fait sur les domaines des variables non encore instanciées en testant le voisinage immédiat [6] de la dernière variable instanciée. Avant de descendre

6. Ce sont les variables non encore instanciées liées à la dernière variable instanciée par une contrainte.

d'un niveau dans l'arbre de recherche nous sommes sûrs que toutes les valeurs qui restent dans les domaines des variables sont compatibles avec l'instanciation courante. Un retour arrière sera provoqué par l'épuisement du domaine d'une certaine variable future.
Le niveau de filtrage utilisé influe directement sur la taille de l'arbre de recherche développé par la méthode de résolution. En général, lorsque le niveau de filtrage augmente, la taille de l'arbre de recherche diminue. Mais, le nombre de tests de consistance est supérieur. Le problème est donc de trouver un compromis entre ces deux paramètres.

2. **Approche du type Regarder en arrière** *Look-Back*. Le principe à la base de ces méthodes est d'analyser les situations d'échec afin d'éviter qu'elles se reproduisent. On détermine la cause de l'échec et ensuite on effectue un retour directement vers la variable responsable de l'échec. Parmi ces techniques, nous pouvons citer l'algorithme BJ (BackJumping) de Gaschnig [Gaschnig, 1977] et plusieurs variantes proposées par Dechter avec le Graph-BasedJumping [Dechter, 1990] et Prosser avec le CBJ (Conflic-Directed-BackJumping) [Prosser, 1993].

Algorithme 2 Algorithme FC

PROCEDURE Forward_Checking(D, I, k)

Entrée : $I = \{\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, ..., \langle v_k, d_k\rangle\}$ une instanciation partielle, D les domaines du CSP local P_I, k un entier.

Début

si $(k = n)$ **alors**
  $<$ I est une solution $>$
sinon
  $<$filtrer les D_i, pour tout $k < i$ de telle sorte que si $C_{ki} \in C$, $\forall d_i \in D'_i$, alors $(d_i, d_k) \in R_{ki}>$
  si $(\forall i, j \ / \ D'_i \neq \emptyset)$ **alors**
    pour tout $(d_{k+1} \in D'_{k+1})$ **faire**
      $I \leftarrow I \cup \{\langle v_{k+1}, d_{k+1}\rangle\}$
      Forward_Checking($D', I, k + 1$)
    fin pour
  fin si
fin si

fin

2.3.2.4 Heuristiques d'instanciation

Les heuristiques d'instanciation sont des règles qui déterminent l'ordre suivant lequel les variables seront instanciées ainsi que l'ordre de choix des valeurs pour les variables. Pour cela, nous définissons les deux notions : Voisinage et degré.

Définition 11 (Voisinage) *Voisinage d'un sommet du graphe représente tous les sommets qui sont reliés à ce sommet par des arêtes.*

Définition 12 (Degré) *Degré d'un sommet est le nombre des sommets voisins.*

Une bonne heuristique d'instanciation peut avoir un grand impact sur la résolution d'un problème de décision et permettre d'accélérer l'obtention de solutions (la détection d'échecs en cas de problèmes insolubles) pendant l'exploration de l'espace de recherche.

En effet, les heuristiques permettent de réduire l'espace de recherche et ainsi, favoriser l'apparition d'une solution dans le cas où le CSP est consistant.

Les ordres de variables et de valeurs peuvent être statique, fixés au préalable ou dynamique, évoluent pendant la résolution. Nous présentons ci-dessous quelques exemples d'heuristiques : [Karoui, 2010]

1. **Heuristique du domaine minimal (min-domaine) :** Cette heuristique d'ordre sur les variables sélectionne en premier la variable qui a le plus petit domaine de valeurs.
2. **Heuristiques du degré (degree ou deg) :** L'heuristique basée sur les degrés [Dechter 1989] ordonne les variables de manière décroissante en fonction de leurs degré (cf. paragraphe 1.1.8).
3. **Heuristiques du degré dynamique (dynamic-degree ou ddeg) :** L'heuristique du degré dynamique ordonne les variables de manière décroissante en fonction de leur degrés dynamiques qui évoluent au cours de la recherche.

Ses différentes heuristiques sont également combinées entre elles. Dans [Benhamou and Saïdi, 2006], les auteurs proposent d'utiliser l'heuristique *dom/deg* qui est connu pour être l'une des meilleures heuristiques et qui donne de bons résultats en général.

Plusieurs travaux se sont intéressés aux heuristiques, que ce soit dans un contexte SAT [7], CSP ou optimisation combinatoire [Hooker 2000, Lecoutre 2007b].
La figure 2.8 représente l'arbre de recherche par l'heuristique min-domain et avec le *Backtrack*.

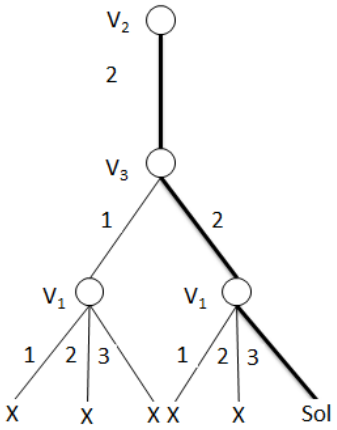


FIGURE 2.8 – Arbre de recherche par l'heuristique min-domain du CSP P'.

7. Le problème SAT est un problème de logique propositionnelle, encore appelée logique d'ordre 0 ou logique des prédicats.

2.3.2.5 Méthodes de décomposition

Les méthodes de décomposition transforment le problème en un ensemble de sous-problèmes. On peut les répartir en deux catégories :

1. **Méthode de décomposition basée sur la structure du CSP :** ces méthodes sont basées sur les propriétés structurelles du CSP. Elles reposent sur le corollaire de Freuder [Freuder, 1982], qui établit que la résolution des CSP acycliques est linéaire. Parmi ces méthodes, nous citons la méthode de l'ensemble *coupe-cycle* [Dechter and Pearl, 1988], la méthode du *regroupement en arbre* [Dechter and Pearl, 1989] et la méthode du *regroupement cyclique* [Jégou, 1990] qui s'inspire des deux techniques de décompositions précédentes.

2. **Méthode de décomposition basée sur la micro-structure :** ces méthodes exploitent la micro-structure associée au problème du départ. Parmi eux, on peut citer : la méthode de *triangulation de la micro-structure* [Jégou, 1993] qui se base sur la recherche des cliques maximales dans le graphe de la micro-structure associée au CSP une fois celle-ci triangulée. L'intérêt des cliques maximales est que chacune d'entre elles représente une décomposition des domaines (un sous problème).

Plusieurs méthodes de décomposition s'appuient sur la notion des graphes triangulés [8]. Les graphes triangulés constituent une classe particulière des graphes parfaits. Possédant des propriétés remarquables, notamment la facilité de reconnaissance ainsi que la facilité de recherche de cliques, ils rendent leur exploitation fort utile.

Il existe plusieurs algorithmes proposés, nous pouvons citer l'algorithme de Tarjan et Yannakakis [Tarjan and Yannakakis, 1984] appelé MCS (*Maximum Cardinality Search*) et LS (*Lexicographic search*) de Rose et Al. [Rose *et al.*, 1976].

2.3.3 Méthodes incomplètes

Les méthodes incomplètes (telles que la recherche locale (LS) de [Aarts and Lenstra, 1997]) reposent sur des heuristiques permettant d'étudier des zones spécifiques de l'espace de recherche dans le but d'atteindre une solution. Parmi les techniques utilisées en recherche locale on peut citer : *le recuit simulé* [Kirkpatrick *et al.*, 1983], *la recherche tabou* [Glover, 1986] et les méthodes basées sur *les algorithmes génétiques* [Goldberg, 1989; Holland, 1975a].

Le grand avantage de ces méthodes est le fait que des problèmes de taille intraitable par les méthodes complètes peuvent être rapidement résolus par les méthodes incomplètes. Par ailleur, le cas d'un CSP consistant, il n'est toujours pas garanti de trouver une solution.

Il existe d'autres formes dites méthodes *hybrides* qui combinent des approches différentes, parmi elles : la méthode *BTD* [Jégou and Terrioux, 2003] qui combine la méthode de décomposition et la méthode de Backtracking ; ou aussi *FC-CBJ* [Prosser, 1993] qui combine la méthode Forward Checking avec la méthode CBJ (*Conflic-Directed-BackJumping*).

8. Un graphe $G = (V, E)$ est triangulé si et seulement si, tout cycle de longueur 4 possède une corde, c'est-à-dire, une arête entre deux sommets non-consécutifs du cycle.

2.4 CSPs à contraintes de différence

Les CSPs à contraintes de différence font partie des CSPs binaires quelconques. Dans certains domaines réels, de nombreux problèmes peuvent être transformés à des CSPs à contraintes de différence, quelques problèmes seront représentés par la suite dans le chapitre 3.

Définition 13 (Contrainte de différence) *Une contrainte de différence est une contrainte binaire entre deux variables v_i et v_j où deux variables pertinentes prennent que des valeurs différentes.*

Définition 14 (NECSP) *Un NECSP (pour Not-Equal CSP en Anglais) est un CSP dans lequel toutes les contraintes sont des contraintes de différence.*

Les NECSPs font partie de la classe des problèmes NP-complets.

Exemple 4 *Soit $P = (V, D, C, R)$ un CSP défini par :*
- $V = \{v_1, v_2, v_3, v_4, v_5\}$;
- $D = \{D_1 = \{a, b\}, D_2 = \{a, c\}, D_3 = \{a\}, D_4 = \{a, c\}, D_5 = \{c\}\}$;
- $C = \{C_{12}(v_1 \neq v_2), C_{13}(v_1 \neq v_3), C_{14}(v_1 \neq v_4), C_{15}(v_1 \neq v_5), C_{23}(v_2 \neq v_3), C_{24}(v_2 \neq v_4), C_{25}(v_2 \neq v_5), C_{34}(v_3 \neq v_4), C_{35}(v_3 \neq v_5), C_{45}(v_4 \neq v_5)\}$;
- $R = \{R_{13} = \{(a, a), (b, a)\}, R_{23} = \{(a, a), (b, a)\}, R_{34} = \{(a, c), (b, c)\}\}$.

La Figure 2.9 donne un graphe du CSP P de contraintes de différence.

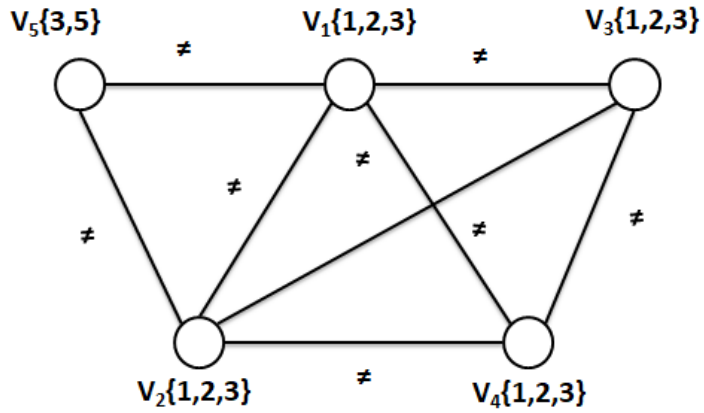


FIGURE 2.9 – Graphe d'un CSP de contraintes de différence.

Dans le cas particulier ou le graphe de contrainte d'un CSP P à contraintes de différence est complet, on peut définir une nouvelle contrainte, impliquant toutes les variables du CSP P. On parle alors de la contrainte *Alldifferent*[9].

9. Une contrainte Alldiff impose que toutes les variables de CSP ont des contraintes de différence deux à deux.

2.5 Conclusion

Le formalisme CSP est un formalisme puissant pour la représentation de nombreux problèmes. Il existe plusieurs méthodes pour la résolution de ces problèmes. Chacune de ces méthodes a des avantages et des inconvénients. Donc, le choix de la technique dépend fortement du problème que l'on souhaite résoudre.

Plus de détails sur les CSPs peuvent être trouvés dans [Montanari, 1974], [Mackworth, 1977] et [Tsang, 1993].

Dans le chapitre qui suit, nous allons voir la notion de coloration de graphe et quelques problèmes réels de domaines particuliers qui peuvent être modélisés sous forme d'un problème de coloration de graphe.

Chapitre 3

La coloration des graphes

Sommaire

3.1 Introduction 20
3.2 Historique 20
3.3 Méthodes de résolution 20
3.3.1 Les méthodes exactes 20
3.3.2 Les méthodes approchées 21
3.3.3 Les méthodes hybrides 21
3.4 Domaines d'applications 22
3.4.1 Coloration d'une carte géographique 22
3.4.2 Problème du sudoku 24
3.4.3 Problème d'allocation de fréquences 28
3.5 Conclusion 30

3.1 Introduction

Le problème de coloration de graphe est un problème important dans la théorie de graphes, son principe consiste à colorier un graphe par un nombre de couleurs donné tel que les sommets adjacents (reliés par une arête) ne soient pas porteurs de la même couleur. Le plus petit nombre de couleurs nécessaires pour colorier le graphe s'appelle le nombre chromatique et noté $\chi\ (G)$.

Dans ce chapitre, nous commençons par un petit historique de ce problème. Puis, nous citons quelques domaines d'application majeurs de la coloration des graphes.

3.2 Historique

L'origine du problème de la coloration de graphe remonte à 1852 lorsque Francis Guthrie, un étudiant en cartographie et mathématiques a remarqué que quatre couleurs suffisent pour colorier la carte des cantons d'Angleterre, sans donner la même couleur à deux cantons ayant une frontière commune. Il pose la question suivante : Es-ce-que quatre couleurs suffisent pour colorier n'importe quelle carte géographique de telle sorte que deux payes voisines n'ont pas la même couleur ?. Ce n'est qu'en 1976, que deux chercheurs Haken et Appel ont pu répondre affirmativement à cette conjecture des quatre couleurs par la réalisation d'un programme qui s'appelle *Heesch*. Pour voir comment le problème a été résolu, nous invitons à lire le livre (ENG. Four Colors Suffice : How the Map Problem Was Solved)[Wilson, 2013] écrit par Robin Wilson. Malgré la simplicité de l'énoncé, le théorème est resté une conjecture pendant plus d'un siècle, tout au long de ces années, les chercheurs ne sont bien sûr pas restés inactifs et de nombreuses démonstrations de la conjecture ont donné lieu à de nouveaux développements mathématiques ainsi, qu'à une formulation du problème en terme de graphes, ou chaque pays étant représenté par un sommet et deux pays voisins étant reliés par une arête. La preuve de ce théorème est la première preuve majeure qui utilise massivement l'ordinateur.

3.3 Méthodes de résolution

Le problème de coloration de graphes fait partie de la classe du problème NP-complet. De nombreuses méthodes ont été développées pour la résolution de ce problème. Il existe trois grandes familles de méthodes de résolution : les méthodes exactes, les méthodes approchées et les méthodes hybrides.

3.3.1 Les méthodes exactes

Les méthodes exactes ont pour but de calculer les solutions optimales pour des problèmes de petite taille. L'algorithme de Randall-Brown [Brown, 1972] qui propose de colorier les sommets du graphe à l'aide d'un algorithme d'énumération où chaque sommet est colorié par la plus petite couleur possible.

3.3.2 Les méthodes approchées

En raison du temps de calcul qui peut devenir très important, les méthodes exactes perdent toute leur efficacité dès que les graphes à traiter sont de plus en plus grands. C'est alors que des méthodes dites approchées sont proposées.

Ces méthodes trouvent souvent les colorations optimales pour des graphes de taille considérable. Ces méthodes reposent sur le principe d'une recherche non exhaustive dans l'espace de recherche.

3.3.2.1 Les méthodes constructives

Les méthodes constructives colorient le graphe un sommet à la fois, en choisissant à chaque étape celui qui semble être le meilleur selon un critère bien défini. L'algorithme $DSATUR$ [Brelaz, 1979] développé par Brélaz en 1979 est un algorithme séquentiel de coloration de sommets qui colorie successivement les sommets triés dans un ordre déterminé. L'ordonnancement est basé sur le degré de saturation du sommet, c'est-à-dire le nombre de couleurs différentes adjacentes au sommet, l'algorithme s'arrête lorsque tous les sommets sont coloriés.

3.3.2.2 Les méta-heuristiques

Les métaheuristique sont des méthodes capables de guider et d'orienter le processus de recherche dans l'espace de combinaisons.

3.3.2.3 Les algorithmes de recherche locale

Les algorithmes de recherche locale sont devenus très prisés, ils fouillent l'espace de recherche d'un problème en visitant pas à pas chaque solution . L'un des algorithmes le plus connu $TabuCol$ [Hertz and de Werra, 1987] tente de minimiser le nombre de conflits (fonction objective) à partir d'une coloration (configuration initiale) du graphe avec conflits−arêtes ayant la même couleur à leurs extrémités. Cette minimisation est faites par la modification de la couleur d'un sommet intervenant dans un conflit (voisinage). La couleur changée ne peut être assignée au même sommet qu'après un certain nombre d'itérations : cette couleur est dite *taboue* pour ce sommet.

3.3.2.4 Les algorithmes génétiques

Les algorithmes génétiques AGs (en anglais **G**enetic **A**lgorithms) ont été introduits par Holland en 1975 puis diffusés par Goldberg en 1989. Ils sont les plus populaires des algorithmes évolutionnaires. Ces algorithmes s'inspirent des principes de la génétique (sélection, croisement et parfois mutation). Les AGs sont probablement les algorithmes les plus connus et les plus utilisés dans le calcul évolutionnaire [Holland, 1975b].

3.3.3 Les méthodes hybrides

Parfois, la combinaison entre deux algorithmes ou plus peut être intéressante afin de résoudre des problèmes difficiles. Parmi ces méthodes, on peut citer :

- Une hybridation de l'algorithme génétique et la recherche tabou a été proposée dans[Galinier and Hao, 1999].
- Une hybridation de l'algorithme de recherche locale et l'algorithme génétique [Dorne and Hao, 1998].

3.4 Domaines d'applications

La coloration de graphe est liée à de nombreuses applications répandues dans des domaines variés, tels que :

— Les emplois du temps [Sabar *et al.*, 2012].
— L'ordonnancement [Laurent and Hao, 2009].
— L'allocation de ressources en réseau [Sadr and Adve, 2012].
— La gestion du trafic aérien [Barnier and Brisset, 2004].
— Le stockage de produits chimiques [BENSOUYAD, 2015].
— L'affectation de fréquences dans les réseaux cellulaires [Hale, 1980].
— Le célèbre jeu *Sudoku*.

Dans la section qui suit, nous étudions quelques exemples d'application en montrant comment ces problèmes peuvent être transformés sous la forme de problème de coloration de graphe.

3.4.1 Coloration d'une carte géographique

Comme cité en historique, l'une des premières application du problème de coloration de graphe vient du problème de coloration des cartes géographiques. Considérant la carte géographique de la wilaya d'Aïn Témouchent et ses Daïras montrée dans la Figure 3.1 (les frontières sont tracées en noir). Le problème consiste alors à colorier les différentes daïras de telles sort que les frontières entre elles restent visibles (sans tracer concrètement cette frontière).

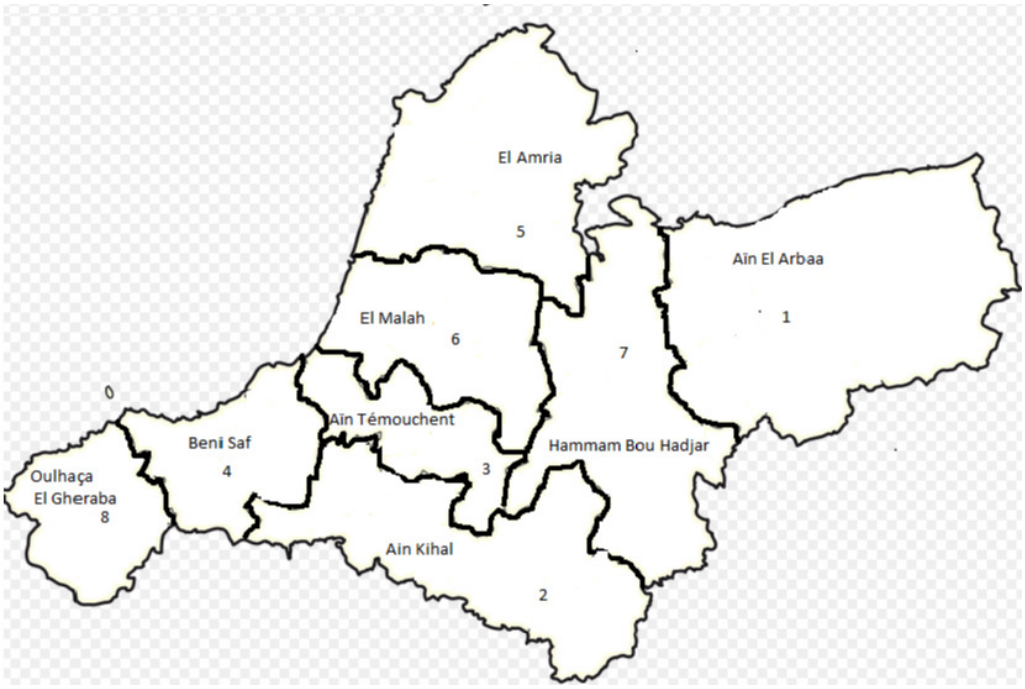


FIGURE 3.1 – La carte géographique d'Aïn Témouchent.

Nous avons traduit la carte sous forme d'un graphe, où les sommets sont les Daïras et les arêtes représentent les frontières entre deux daïras voisines.

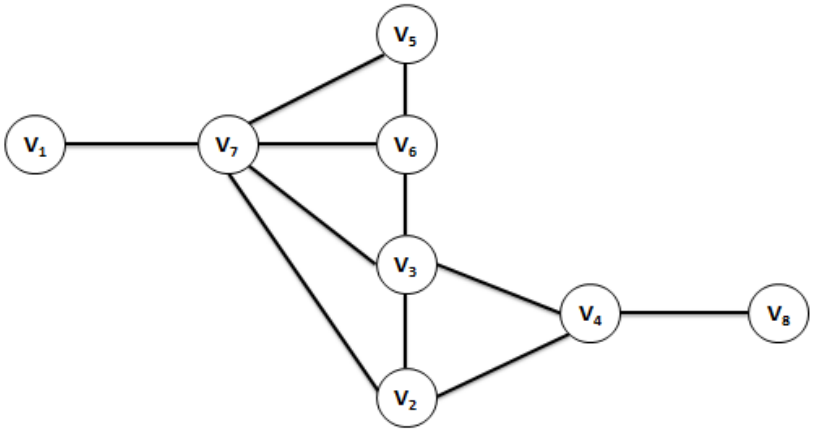


FIGURE 3.2 – Le graphe correspondant à la carte d'Aïn Témouchent.

À partir de la Figure 3.2, on peut aisément voir que ce graphe ne peut pas être 2-colorable, car il contient plusieurs cliques de taille 3. Ce qui implique que le nombre chromatique de ce graphe est strictement supérieur à 2. Donc, allons résoudre le problème suivant : Ce graphe est-il 3-colorables ?.

La Figure 3.3 montre la coloration de la carte géographique d'Aïn Témouchent (nombre chromatique est 3).

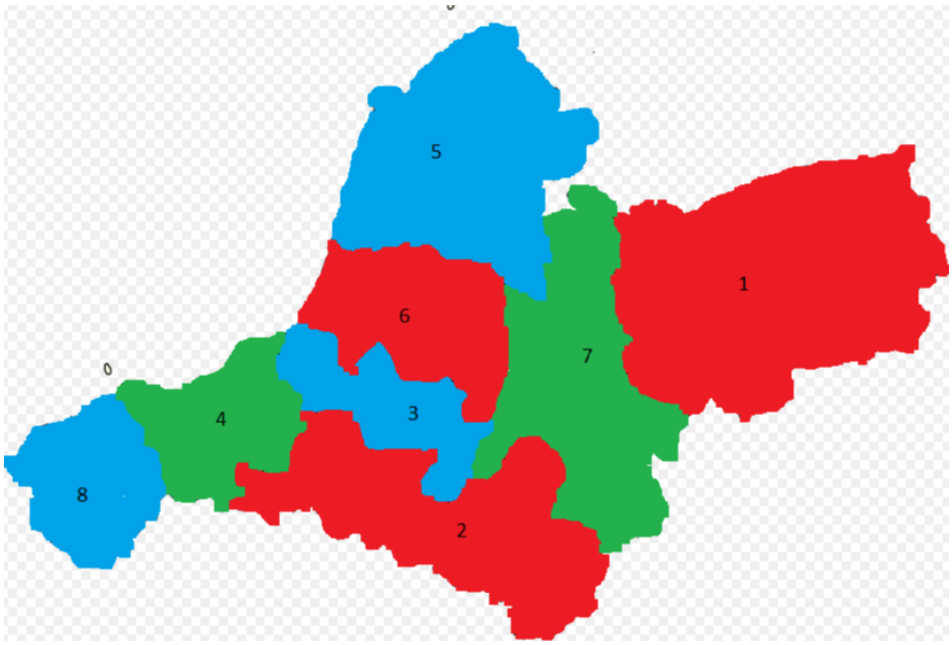


FIGURE 3.3 – La coloration de la carte géographique d'Aïn Témouchent.

3.4.2 Problème du sudoku

Le sudoku est un jeu de réflexion crée en 1979 par un américain Howard Garns [Brouwer, 2007]. Principe de ce jeu est de remplir une grille de taille 9x9 avec des chiffres allant de 1 à 9 en respectant certaines contraintes.
Il y a trois contraintes à respecter :

1. Chaque ligne doit contenir exactement chacun des 9 chiffres de 1 à 9 ;
2. Chaque colonne doit contenir exactement chacun des 9 chiffres de 1 à 9 ;
3. Chaque carré de neuf cases doit contenir exactement chacun des 9 chiffres de 1 à 9.

7								1
		6	5		9	8		
	9	4		2		3	7	
	7		4		3		9	
		3				7		
	6		8		7		1	
	4	8		9		6	3	
		7	2		8	1		
3								2

FIGURE 3.4 – Une grille de Sudoku.

Initialement certaines cases de la grille contiennent déjà des nombres. Le but alors est de compléter la grille tout en respectant les règles du Sudoku. Donc, on peut représenter un sudoku sous la forme d'un problème de coloration de graphe où les sommets sont les cases et les chiffres sont les couleurs. Les contraintes sur les lignes, colonnes et blocs sont obtenues on rajoutons les arêtes de telle sorte que chaque contrainte est représentée par un sous-graphe complet induit sur les sommets correspondant aux cases impliquées dans la contrainte.

La Figure 3.5 montre les contraintes liées à chaque sommet où les cercles pointillées représentent les contraintes sur colonnes, les cercles continues représentent les contraintes sur les lignes et les cercles continues en gras représentent les contraintes sur blocs(3x3).

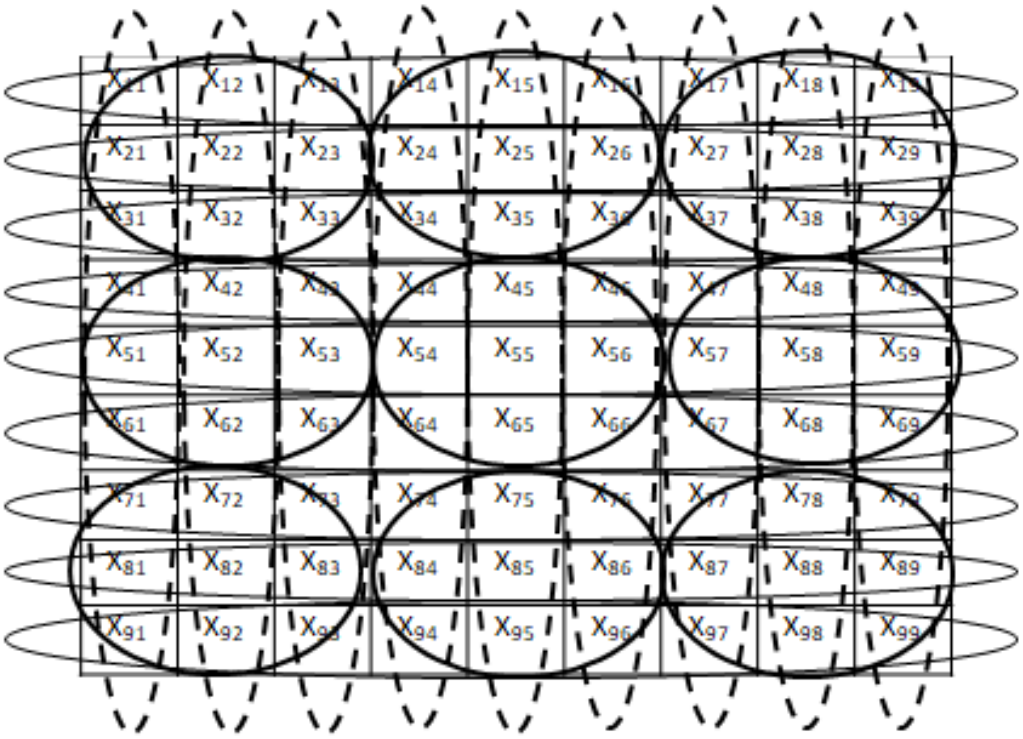


FIGURE 3.5 – La grille de Sudoku avec les contraintes liées à chaque sommet.

Donc, les sommets contenus dans chaque cercle, forment un sous graphe complet. Prenons par exemple le sommet X_{11} on peut voir qu'il participe à 3 contraintes (représentées en cercle) C_{L1}, C_{C1}, C_{B1} sur la ligne ,colonne et bloc. La Figure 3.6 montre alors comment ces cercles sont concrètement représenté dans le graphe.

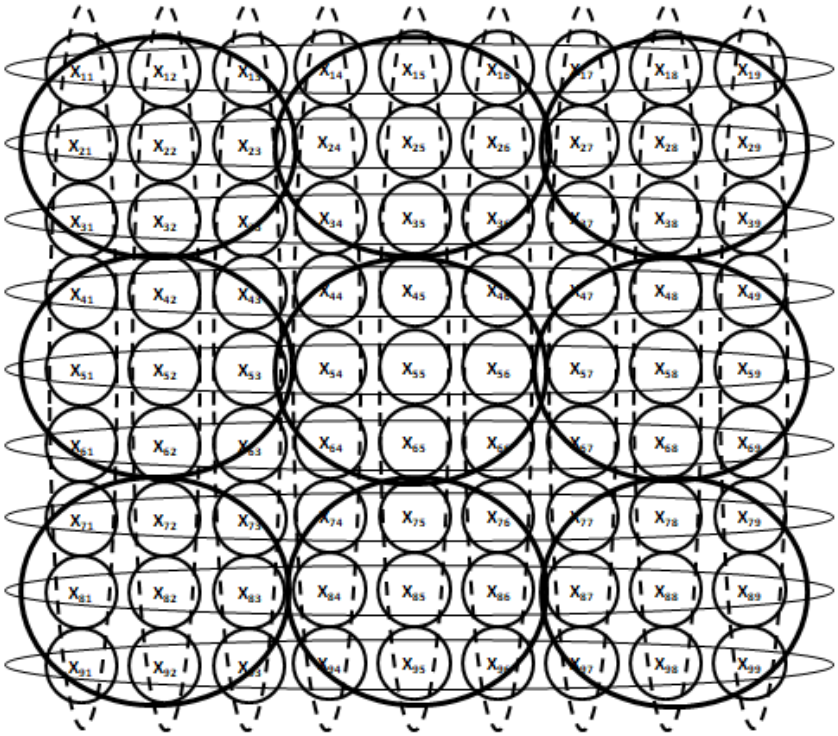


FIGURE 3.6 – Graphe de la grille sudoku.

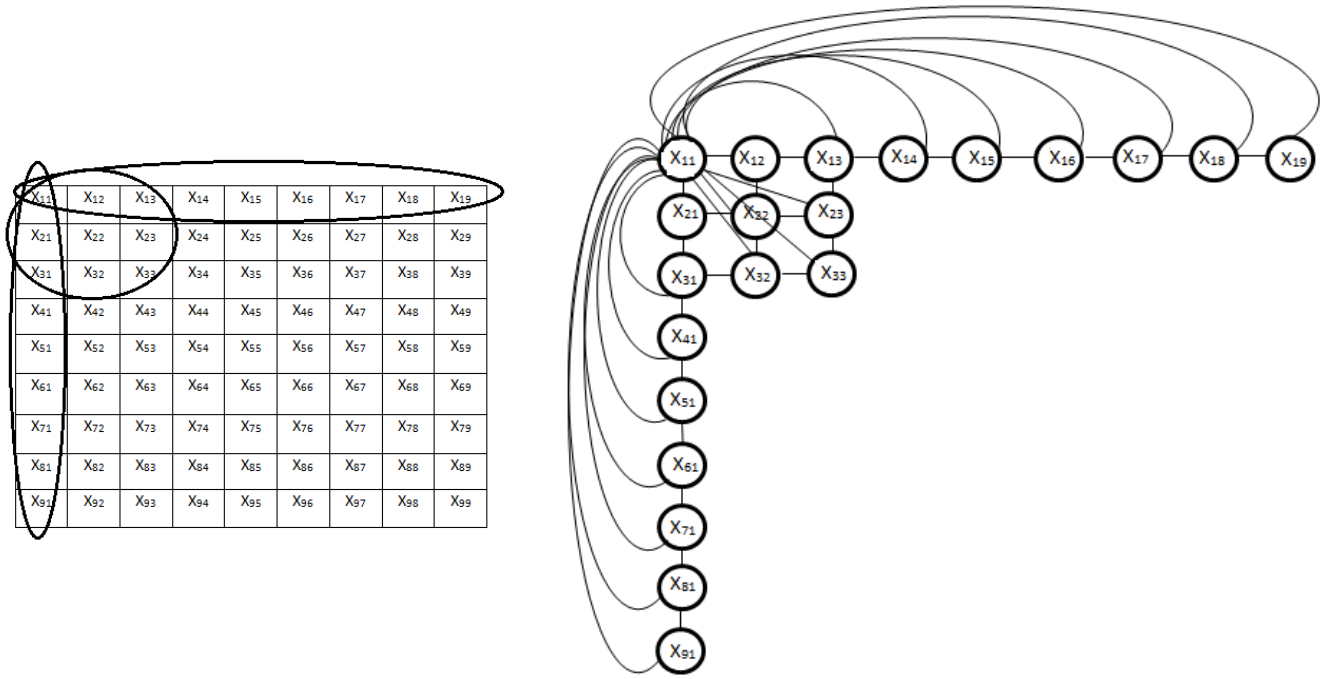


FIGURE 3.7 – La représentation graphique de contraintes du sommet X_{11}.

Tennant compte du nombre de variables, il n'est pas facile de montrer la résolution sous forme d'un arbre d'affectation, car cet arbre est trop grand pour être représentée correctement. Nous préférons alors montrer uniquement l'état de la grille après l'instanciation des variables correspondant aux celles qui étaient déjà pré-remplies.

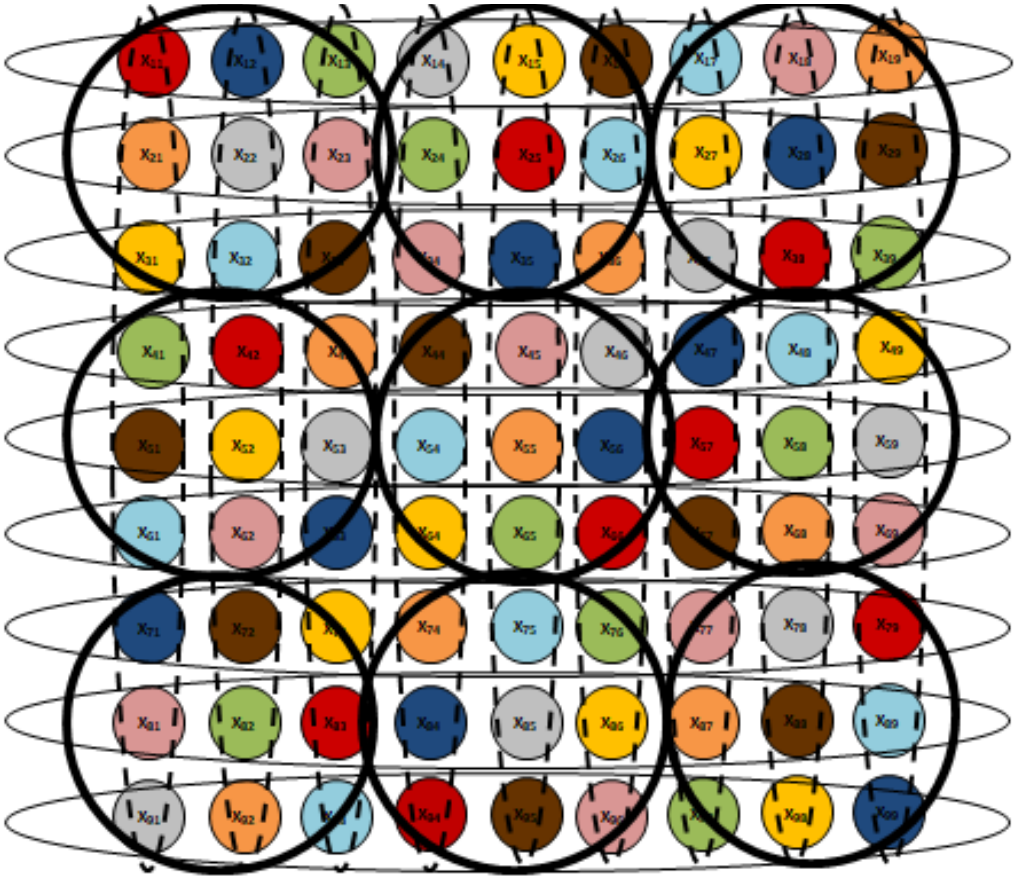


FIGURE 3.8 – La coloration du graphe de la grille Sudoku.

Les chiffres correspondant aux couleurs de graphe de Sudoku sont :

— *Rouge* → 7 ;
— *Bleu* → 2 ;
— *Vert* → 5 ;
— *Orange* → 1 ;
— *Gris* → 3 ;
— *Rose* → 6 ;
— *Jaune* → 8 ;
— *Bleuciel* → 9 ;
— *Marron* → 4 ;

7	2	5	3	8	4	9	6	**1**
1	3	**6**	**5**	7	**9**	**8**	2	4
8	**9**	**4**	6	**2**	1	**3**	**7**	5
5	**7**	1	**4**	6	**3**	2	**9**	8
4	8	**3**	9	1	2	**7**	5	6
9	**6**	2	**8**	5	**7**	4	**1**	3
2	**4**	**8**	1	**9**	5	**6**	**3**	7
6	5	**7**	**2**	3	**8**	**1**	4	9
3	1	9	7	4	6	5	8	**2**

FIGURE 3.9 – Résolution complète.

3.4.3 Problème d'allocation de fréquences

Depuis les années 80, le problème d'affectation de fréquences a fait l'objet d'études menées par différents chercheurs dans le but d'une meilleure gestion. Dans certains réseaux de télécommunication, il y a des émetteurs émettant chacun sur une fréquence particulière pour éviter les interférences sur le réseau. Si deux émetteurs sont trop proches, on ne peut pas leur allouer la même fréquence, de plus, à cause des contraintes financières, il faut minimiser le nombre des fréquences allouées.

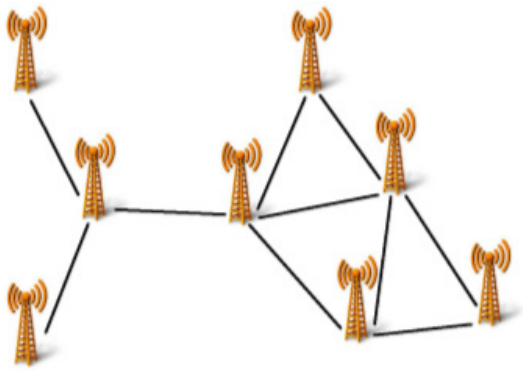


FIGURE 3.10 – Un réseaux de télécommunication.

Ce problème peut se ramener à un problème de coloration de graphe, où il suffit de mettre un sommet pour chaque émetteur et une arête entre deux sommets si et seulement si les deux émetteurs correspondants sont trop proches.

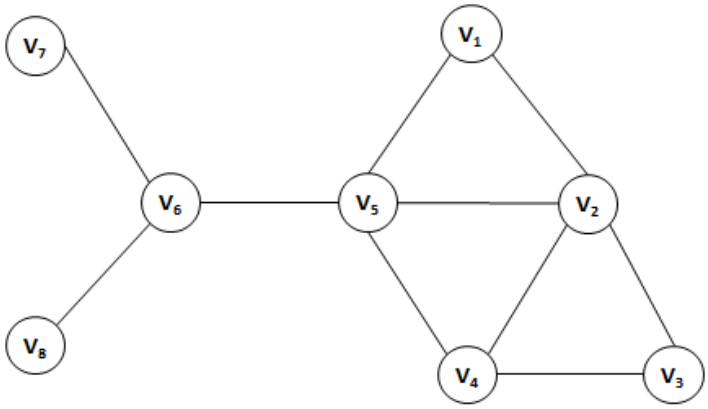


FIGURE 3.11 – Le graphe corespondant au réseaux de télécommunication.

Ce graphe est 3 colorable et c'est le nombre chromatique de ce graphe. Dans la Figure 3.12, est représenté le graphe colorié où chaque couleur définit une fréquence spécifique.

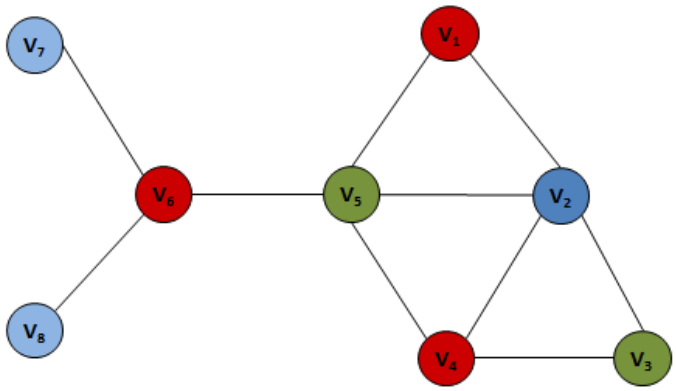


FIGURE 3.12 – La coloration du réseau de télécommunication.

3.5 Conclusion

Dans ce chapitre, nous avons présenté un aperçu court des méthodes de résolution du problème de coloration de graphe. Aussi, nous avons donné plusieurs exemples de domaines d'applications de ce problème.

Au chapitre qui suit, nous allons voir la notion de la symétrie et la notion de la dominance qui sont deux notions très intéressantes pour notre projet.

Chapitre 4

Symétrie et dominance dans les CSPs à contraintes de différence

Sommaire

4.1 Introduction 32
4.2 Théorie des groupes et symétrie 32
4.3 Symétrie dans les CSPs 33
4.3.1 Problème des reines : 33
4.3.2 Symétrie dans les CSPs quelconques 34
4.3.3 Une condition suffisante pour la symétrie dans les NECSPs 35
4.3.4 Affaiblissement des conditions de la symétrie dans les NECSPs 36
4.4 La dominance dans les NECSPs 37
4.4.1 Le principe de la dominance 37
4.4.2 Une condition suffisante pour la dominance dans les NECSPs 37
4.4.3 Affaiblissement de la condition de dominance 38
4.4.4 Détection de la Dominance 39
4.5 Conclusion 40

4.1 Introduction

Le principe de la symétrie et de la dominance dans le cas des CSPs à contraintes de différence offre la possibilité de réduire l'espace de recherche et le temps d'exécution, mais elle est une tache difficile.

Dans ce chapitre, tout d'abord, on va commencer par la notion de théorie des groupes qu'elle est nécessaire pour étudier la symétrie. Ensuite, nous parlons du principe de la symétrie dans les CSPs et pour finir, nous donnons le principe de la dominance dans les NECSPs.

4.2 Théorie des groupes et symétrie

La notion de la théorie des groupes est nécessaire pour étudier la symétrie en mathématiques. Pour cela, Nous présentons quelques concepts fondamentaux nécessaires de cette théorie.

Soit $\Omega = \{1, 2, \ldots, N\}$ pour un N donné où chaque entier représente une variable CSP, une valeur, ou une paire variable/valeur. Une permutation de Ω est une application bijective σ de Ω vers Ω. On dénote par $Perm(\Omega)$ l'ensemble de toutes les permutations de Ω et par $\circ$ la composition de permutation de $Perm(\Omega)$. La paire $(Perm(\Omega), \circ)$ forme le groupe de permutation de Ω. En effet, $\circ$ est close et associative, l'inverse d'une permutation est une permutation et la permutation identité est l'élément neutre. Une paire $(T, \circ)$ forme un sous-groupe de $(S, \circ)$ si et seulement si, T est un sous-ensemble de S et forme muni de l'opération $\circ$ un groupe.

L'orbite $\omega^{Perm(\Omega)}$ d'un élément ω de Ω sur lequel le groupe $Perm(\Omega)$ agit est $\omega^{Perm(\Omega)}=\{\omega^\sigma : \sigma \in Perm(\Omega)\}$.

Un ensemble de générateur du groupe $Perm(\Omega)$ est un sous-ensemble Gen de $Perm(\Omega)$ tel que chaque élément de $Perm(\Omega)$ peut être écrit comme composition des éléments de Gen. Nous écrivons $Perm(\Omega)=< Gen >$. Un élément de Gen est appelé générateur. L'orbite de $\omega \in \Omega$ peut être calculée en utilisant uniquement l'ensemble des générateurs Gen.

Un stabilisateur par point $Perm(\Omega)_\delta$ de $\delta \subset \Omega$ est le sous-groupe $Perm(\Omega)_\delta$ $=\{\sigma \in Perm(\Omega) : \forall \omega \in \delta,\ \omega^\sigma = \omega\}$, i.e. l'ensemble des éléments de $Perm(\Omega)$ qui fixent chaque point de δ.

Des logiciels existent pour la manipulation efficace des groupes, parmi eux, GAP, abréviation de *Groups, Algorithms and Programming* est un outil gratuit de calcul dédié à l'algèbre discrète, notamment à la théorie des groupes [GAP, 2007]. Il est important de noter que le sous-ensemble de permutations $Aut(M_P) \subset Perm(\Omega)$ préservant la microstructure M_P d'un CSP P forment le groupe d'automorphisme de M_P qui est connu pour être identique au groupe des *symétries de contraintes* du CSP P [Cohen *et al.*, 2005], qui sont aussi appelées *symétries syntaxiques* [Benhamou, 1994a]. Des algorithmes efficaces ont été proposés pour résoudre le problème d'automorphismes de graphes (Nauty [McKay, 1981], Saucy [Darga *et al.*, 2004]). Pourtant, ces algorithmes ont à la base, une complexité en temps exponentielle. En effet, on ne lui connais pas actuellement, d'algorithmes de complexité polynomiale. En plus, l'appartenance du problème de la détection d'automorphisme ou plus généralement, d'isomorphisme de graphes à la classe des problèmes NP-complet, demeure une question ouverte.

Les différentes définitions de symétries sont présentées dans la section suivante.

4.3 Symétrie dans les CSPs

Dans cette section, nous rappelons quelques propriétés et définitions importantes de la symétrie [Benhamou, 1994a] qui sont mentionnées aussi dans [Benhamou and Saïdi, 2006].

4.3.1 Problème des reines :

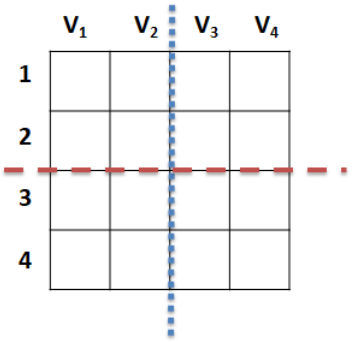


FIGURE 4.1 – La symétrie verticale et horizontale dans un échiquier de 4x4

Si on commençe par placer une reine dans la case $(V_1, 1)$, on tombe sur un échec. Dans la Figure 4.2, grâce à la symétrie on peut facilement déduire qu'ils existent trois autres cas d'échecs (illustrés dans la Figure 4.3).

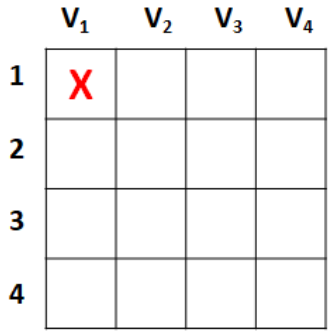


FIGURE 4.2 – Un cas d'échec.

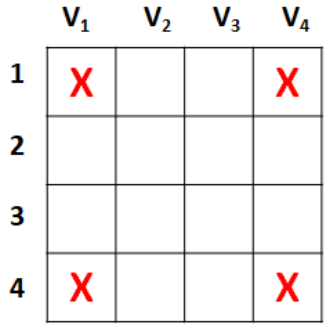


FIGURE 4.3 – Cas d'échecs symétriques.

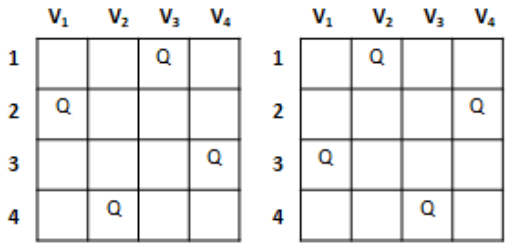


FIGURE 4.4 – Cas de solutions symétriques.

Dans la Figure 4.4 nous donnons les deux seules solutions possibles où ces deux solutions sont symétriques. Alors, on peut déduire que à l'aide de symétrie et à partir d'une solution trouvée, on peut déduire d'autres solutions (celles qui sont symétriques à cette solution).

4.3.2 Symétrie dans les CSPs quelconques

Définition 15 (Symétrie Sémantique) *Dans un CSP P, on dit que deux valeurs a_i et b_i du domaine d'une variable CSP $v_i \in V$ sont sémantiquement symétriques (notation $a_i \sim b_i$) si et seulement si [Il existe une solution du CSP qui assigne la valeur a_i à la variable $v_i \Leftrightarrow$ il existe une solution du CSP qui assigne la valeur b_i à v_i].*

Définition 16 *Dans un CSP binaire $P = (V, D, C, R)$, une permutation σ des valeurs des domaines D_i est définit comme : $\sigma : \cup_{i\in[1,n]} D_i \longrightarrow \cup_{i\in[1,n]} D_i$, tel que $\forall i \in [1, n]$ et $\forall d_i \in D_i$, $\sigma(d_i) \in D_i$.*

Définition 17 *On dit qu'une permutation des valeurs des domaines σ est une symétrie syntaxique d'un CSP binaire $P = (V, D, C, R)$ si et seulement si :*
$[\forall R_{ij} \in R, (d_i, d_j) \in tuples(R_{ij}) \Rightarrow < \sigma(d_i), \sigma(d_j) > \in tuples(R_{ij})]$

Autrement dit, une symétrie syntaxique du CSP P est une permutation laissant P invariant, c'est à dire $\sigma_R(R_{ij}) = R_{ij}$ [10].

Définition 18 *Deux valeurs b_i et c_i d'un domaine d'une variable CSP $v_i \in V$ sont syntaxiquement symétriques (notation $b_i \sim c_i$) s'il existe une symétrie syntaxique σ du CSP P tel que : $\sigma(b_i) = c_i$ ou $\sigma(c_i) = b_i$.*

10. σ_R : est la généralisation de σ aux relations de R.

Pour l'implication de la symétrie dans la consistance des CSPs, on a :
Soit I une instanciation des variables d'un CSP P, σ une symétrie de P et I/σ l'instanciation obtenue en substituant dans I chaque valeur d_i du domaine de chaque variable de CSP v_i par $\sigma(d_i)$, formellement : $I/\sigma[v_i] = \sigma(I[v_i])$, $\forall i \in [1, n]$.

On donne dans ce qui suit une propriété qui peut être utilisée pour calculer une nouvelle solution à partir d'une solution connue en utilisant la symétrie.

Proposition 1 *I est une solution de P si et seulement si I/σ est une solution P.*

Preuve *Voir [Benhamou, 1994a].*

La propriété principale reliant les symétries et la consistance de CSPs est :

Théorème 2 *Soit b_i et c_i deux valeurs du domaine d'une variable CSP $v_i \in V$. Si b_i et c_i sont syntaxiquement symétriques alors b_i participe à une solution du CSP si et seulement si la valeur c_i participe à une solution du CSP.*

Preuve *Voir [Benhamou, 1994a].*

Corollaire 1 *Si $d_i \sim \acute{d_i}$ et d_i ne participe à aucune solution de P, alors $\acute{d_i}$ ne participe à aucune solution de P.*

Le corollaire 1 nous permet de supprimer les valeurs symétriques à une valeur $d_i \in D_i$ sans affecter la consistance du CSP si on a déjà montré que d_i ne participe à aucune solution du CSP.

4.3.3 Une condition suffisante pour la symétrie dans les NECSPs

Un algorithme est proposé pour la détection des symétries dans les CSPs binaires quelconques dans [Benhamou, 1994a]. Sa complexité dans le pire des cas est exponentielle. Aussi, il est montré comment les conditions de symétrie peuvent être simplifiées dans les NECSPs et comment les valeurs symétriques peuvent être calculées efficacement avec un algorithme plus simple avec une complexité linéaire en la taille du problème. Ce résultat est dû à la propriété suivante :

Théorème 3 *Soient a_i et b_i deux valeurs d'un domaine D_i d'un NECSP P. Si a_i et b_i apparaissent dans les mêmes domaines des variables non instanciées, alors elles sont symétriques.*

Preuve *Voir [Benhamou, 2004].*

Cette propriété très simple est très utile pour le calcul des symétries entre valeurs d'un même domaine.

Exemple 5 *Soit le CSP à contraintes de différence de la figure 4.5. Les variables du problème $V = \{v_1, v_2, v_3, v_4, v_5\}$. Les domaines de chaque variable sont représentées sur la figure. Les arêtes expriment les contraintes de différence du NECSP. À cause du théorème 2, on peut déduire que les valeurs 1 et 2 du domaine de v_1 du problème sont symétriques. On remarque que les valeurs 1 et 2 apparaissent en même temps dans les domaines de v_2, v_3, v_4 et n'apparaissent pas dans le domaine de v_5. Par un raisonnement similaire, en peut déduire aussi que ces deux valeurs des domaines des variables v_2, v_3 et v_4 sont symétriques.*

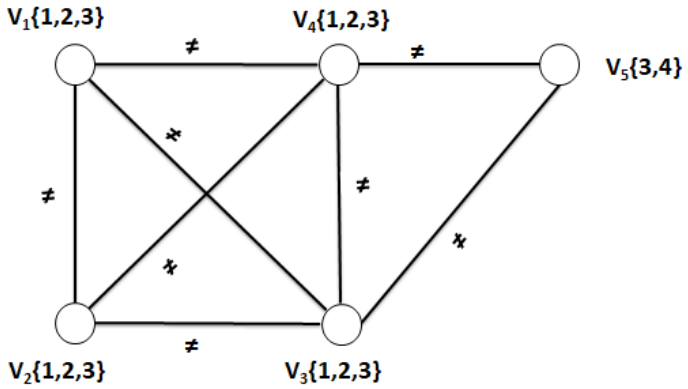


FIGURE 4.5 – Un CSP à contraintes de différence (NECSP).

4.3.4 Affaiblissement des conditions de la symétrie dans les NECSPs

Définition 19 (Arbre d'echec) *On appelle arbre d'échec d'une instanciation* $I = \{\langle v_1, a_1\rangle, \langle v_2, a_2\rangle, ..., \langle v_i, a_i\rangle\}$*, le sous-arbre d'affectation* T *noté* T_I *tel que :*

1. *La racine de l'arbre* T *et la racine du sous-arbre* T_I *sont reliées par le chemin* $\{v_1, v_2, ..., v_i\}$ *correspondant à l'instanciation* I *;*
2. *Toutes les variable de CSP correspondantes aux feuilles de* T_I *ont des domaines vides.*

Pour énoncer la propriété d'affaiblissement de la condition de symétrie, nous présentons ce théorème :

Théorème 4 *Soient un CSP* $P(V, C, D, R)$*,* $a_i \in D_i$ *et* $b_i \in D_i$ *deux valeurs du domaine* D_i *de la variable* v_i *en cours d'instanciation,* $I_0 = \{\langle v_1, a_1\rangle, ..., \langle v_{i-1}, a_{i-1}\rangle\}$ *une instanciation partielle des* $i-1$ *variables antérieures à* v_i *telle que l'extension* $I = I_0 \bigcup \{\langle v_i, a_i\rangle\} = \{\langle v_1, a_1\rangle, ..., \langle v_{i-1}, a_{i-1}\rangle \langle v_i, a_i\rangle\}$ *est inconsistante,* T_I *étant l'arbre d'échec de* I *et* $Var(T_I)$ *l'ensemble des variables correspondant aux nœuds de* T_I*. On a alors :*
Si a_i *et* b_i *apparaissent dans les mêmes domaines des variables de* $Var(T_I)$ *alors l'extension* $J = I_0 \bigcup \{\langle v_i, b_i\rangle\} = \{\langle v_1, a_1\rangle, ..., \langle v_{i-1}, a_{i-1}\rangle \langle v_i, b_i\rangle\}$ *est inconsistante.*

Preuve *Voir [Benhamou and Saïdi, 2006].*

Cette nouvelle propriété de symétrie est un affaiblissement de la condition du théorème 3 dans le cas d'instanciation menant à une inconsistance. Le cas d'inconsistance est important, car il permet de faire des coupures de symétries dans l'arbre de preuve de consistance des CSPs comme le stipule le corollaire 1. Des symétries non capturées par le théorème 3 peuvent émerger suite à cet affaiblissement.

Passant maintenant au principe de symétrie à la dominance dans les CSPs à contraintes de différence (NECSPs).

4.4 La dominance dans les NECSPs

La *dominance* peut être vue comme une symétrie à sens unique. Lorsque deux valeurs sont sémantiquement équivalentes elles sont symétriques. Mais, si ce n'est pas le cas, c'est que peut être, l'une d'entre elles *domine* l'autre.

4.4.1 Le principe de la dominance

Dans cette section, nous rappelons la notion de *Dominance* dans le cas de valeurs d'un même domaine qui a été introduite dans [Benhamou, 1994b] et dans [Benhamou and Saïdi, 2006].

Définition 20 (Dominance Sémantique) *Dans un CSP P, une valeur a_i domine une autre valeur b_i pour une variable CSP $v_i \in V$ (notation $a_i \succeq b_i$) si et seulement si [Il existe une solution du CSP qui assigne la valeur b_i à la variable $v_i \Rightarrow$ il existe une solution du CSP qui assigne la valeur a_i à v_i].*

Définition 21 (Dominance Syntaxique) *Dans un CSP $P = (V, D, C, R)$, une valeur a_i domine une autre valeur b_i pour une variable CSP $v_i \in V$ si a_i est syntaxiquement symétrique à b_i dans un CSP P' obtenu à partir de P en supprimant quelques tuples dans les $R_{ik} \in R$ contenant la valeur b_i.*

La valeur b_i participe à une solution si la valeur a_i participe à une solution ; sinon elle n'y participe pas. La valeur b_i peut être alors supprimée de D_i sans affecter la consistance du CSP.

Corollaire 2 *[Benhamou, 1994b] Si $a_i \succeq b_i$ et a_i ne participe à aucune solution du CSP P, alors b_i ne participe à aucune solution de P.*

Preuve *voir [Benhamou, 1994b]*

À partir du corollaire 2, s'il a été montré que la valeur dominante a_i ne participe à aucune solution. Alors, nous pouvons supprimer les valeurs dominées $b_i \in D_i$ sans affecter l'ensemble des solutions du CSP. La proposition suivante donne la relation entre la symétrie et la dominance.

Proposition 2 *Deux valeurs $a_i \in D_i$ et $b_i \in D_i$ sont symétriques si et seulement si a_i domine b_i et b_i domine a_i.*

Preuve *En considérant les deux définitions 20 et 21, le résultat est immédiat.*

4.4.2 Une condition suffisante pour la dominance dans les NECSPs

Tous comme Benhamou qui donne une condition suffisante de symétrie entre valeurs d'un même domaine dans le cas des NECSPs [Benhamou, 2004]. Benhamou et Saïdi dans [Benhamou and Saïdi, 2006] ont donné une condition suffisante pour la détection de la dominance entre valeurs d'un même domaine dans les NECSPs.

Théorème 5 *Deux valeurs a_i et b_i du domaine D_i correspondant à la variable v_i du NECSP P et Y l'ensemble des variables non instanciées de P. a_i domine b_i dans P si les deux conditions suivantes sont vérifiées :*

1. *$a_i \in D_j \Rightarrow b_i \in D_j$, pour tout $v_j \in Y$ tel que v_j a une contrainte avec v_i ;*
2. *$a_i \in D_j \Leftrightarrow b_i \in D_j$ pour chaque variable $v_j \in Y$ qui n'a pas de contraintes avec v_i.*

Preuve *Voir [Benhamou and Saïdi, 2006].*

Exemple 6 *Dans le domaine de la variable v_1 du problème présenté dans la figure 4.5, la valeur 1 domine la valeur 3.*

4.4.3 Affaiblissement de la condition de dominance

Le théorème 4 est un affaiblissement des conditions du théorème 3 pour la détection des valeurs symétriques quand une instanciation partielle inconsistante est générée durant la recherche. Dans ce qui suit, nous donnons le théorème 6 qui est l'affaiblissement des conditions du théorème 5.

Théorème 6 *Soient $P(V, D, C, R)$ un CSP, $a_i \in D_i$ et $b_i \in D_i$ deux valeurs du domaine D_i de la variable courante v_i en cours d'instanciation, $I_0 = \{\langle v_1, a_1\rangle, ..., \langle v_{i-1}, a_{i-1}\rangle\}$ une instanciation partielle des $i-1$ variables instanciées avant v_i telle que l'extension $I = I_0 \bigcup \{\langle v_i, a_i\rangle\} = \{\langle v_1, a_1\rangle, ..., \langle v_{i-1}, a_{i-1}\rangle \langle v_i, a_i\rangle\}$ est inconsistante, T_I est l'arbre d'échec de I et $Var(T_I)$ l'ensemble des variables correspondant aux nœuds de T_I. Si les deux conditions suivantes :*

1. *$b_i \in D_j \Rightarrow a_i \in D_j$, pour tout $v_j \in Var(T_I)$ tel que v_j a une contrainte avec v_i.*
2. *$a_i \in D_j \Leftrightarrow b_j \in D_i$ pour toute autre variable v_j de $Var(T_I)$ qui n'a pas de contraintes avec v_i.*

sont vérifiées, alors l'extension $J = I_0 \bigcup \{\langle v_i, b_i\rangle\} = \{\langle v_1, a_1\rangle, ..., \langle v_{i-1}, a_{i-1}\rangle \langle v_i, b_i\rangle\}$ est inconsistante.

Preuve *La preuve est semblable à celle donnée pour la preuve du théorème 4.*
Voir [Benhamou and Saïdi, 2006]

Quand l'instanciation partielle courante mène à un échec, la dernière propriété deviendera un affaiblissement de la condition de la dominance (théorème 4). Le cas d'instanciations partielles inconsistantes est très important, car il nous permet d'élaguer l'arbre de preuve de la consistance du CSP comme le stipule le corollaire 2.

Certaines dominances ou symétries non capturées par le théorème 4 peuvent l'être grâce à cette condition affaiblie :

Exemple 7 *Considérons le NECSP de la figure 4.5. Si nous prenons l'instanciation partielle $I = \{\langle v_1, 2\rangle, \langle v_2, 1\rangle)$ des variables v_1 et v_2, alors l'arbre d'échec T_I de l'instanciation I est donnée dans la figure 4.6 (partie entourée). On suppose que v_2 est la variable courante, alors dans le domaine de la variable v_2, la valeur 2 domine*[11] *la valeur 3 du domaine de v_2, par application du théorème 5, alors que les deux valeurs ne vérifient pas les conditions du théorème 6. En effet, les valeurs 1 et 3 apparaissent en même temps dans les domaines des variables appartenant à T_I, c'est-à-dire les variables v_1 et v_2. La branche correspondante à l'affectation v_2 à 3 ne sera pas explorée dans l'arbre de la preuve de consistance.*

11. En fait, les deux valeurs 1 et 2 du domaine de la variable v_2 sont symétriques, par application du théorème 5 de l'affaiblissement des conditions de la symétrie. Mais, le théorème 6 détecte en plus la dominance.

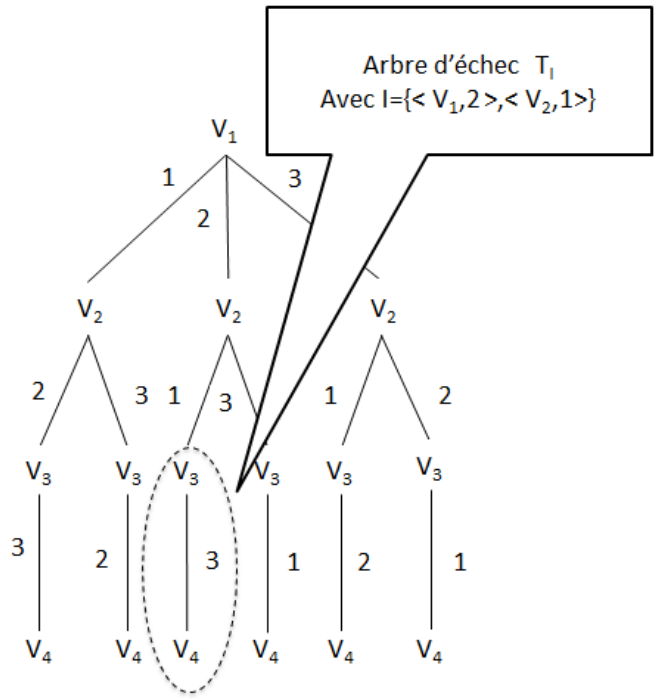


FIGURE 4.6 – Arbre d'échec T_I de l'instanciation partielle $I = \{\langle v_1, 2\rangle, \langle v_2, 1\rangle\}$.

4.4.4 Détection de la Dominance

Maintenant nous abordons le problème de la détection de la dominance. Pour cela, nous donnons l'algorithme 3 qui calcule les valeurs dominées par une valeur a_i d'un domaine D_i donné par application des conditions du théorème 5. Ces valeurs forment une classe de dominance de a_i qui est noté par $Cl(a_i)$.

Algorithme 3 L'algorithme de détection de la dominance dans les NECSPs

PROCEDURE *weak_dominance*($a_i \in D_i$,$Var(T_I)$,**var** Cl(a_i) : classe) ;
Début
Entrée : une valeur $a_i \in D_i$, un ensemble de variables $Var(T_I)$
Sortie : la classe $Cl(a_i)$ des valeurs dominées par a_i.
1: Cl(a_i) $\leftarrow \{a_i\}$
2: **pour tout** ($d_i \in D_i - \{a_i\}$) **faire**
3: **pour tout** (domaine D_k des variables de $Var(T_I)$) **faire**
4: **si** (($c_{ik} \in C$ **and** ($a_i \in D_k \Rightarrow d_i \in D_k$)) **or** ($c_{ik} \notin C$ **and** ($a_i \in D_k \Leftrightarrow d_i \in D_k$))) **alors**
5: Cl(a_i) $\leftarrow$ Cl(a_i)$\cup\{d_i\}$
6: **fin si**
7: **fin pour**
8: **fin pour**
fin

Complexité : Soient n le nombre de variables du NECSP, et d la taille du plus grand domaine. Il est facile de voir que l'algorithme 3 peut exécuter au plus d fois la première boucle et au plus n fois la seconde boucle. Donc, il calcule la classe $Cl(d_i)$ des valeurs dominées par a_i avec une complexité $O(nd)$ dans le pire des cas. En théorie, cet algorithme a la même complexité dans le pire des cas que l'algorithme décrit dans [Benhamou, 2004]. Cependant, cet algorithme détecte un plus grand nombre de symétries, puisque les symétries déjà détectées par l'ancien algorithme ne représentent qu'une partie de celles-ci. De plus, il détecte la dominance.

4.5 Conclusion

Dans ce chapitre, nous avons étudié le principe de la symétrie et de la dominance, puis, nous avons donné les affaiblissements de chacune d'elles. En suite, nous avons parlé sur le principe de symétrie à la dominance, cette nouvelle notion a été publié dans [Benhamou and Saïdi, 2006]. Dans le chapitre suivant, nous allons présenter nos améliorations appliquées sur l'algorithme proposé dans le papier [Benhamou and Saïdi, 2006] pour le rendre plus rapide et plus performant.

Chapitre 5

Exploitation de la symétrie et la dominance dans les NECSPs

Sommaire

5.1 Introduction 42
5.2 Principe de SFC-weak-dom 42
5.3 Améliorations 44
5.3.1 Première amélioration : 44
5.3.2 Deuxième amélioration : 45
5.4 Expérimentations 48
5.4.1 Problèmes aléatoires de coloration des graphes 48
5.4.2 Les benchmarks de Dimacs 53
5.5 Conclusion 56

5.1 Introduction

Il est difficile de trouver un meilleur algorithme pour la coloration de graphe qui possède une complexité et un temps d'exécution raisonnables. Les auteurs dans [Benhamou and Saïdi, 2006] ont proposé de détecter et d'éliminer la symétrie et la dominance dans un problème de contrainte de différence. Les résultats montrent de bonnes performances en général. Le but de cette contribution est d'améliorer les performances de ce travail [Benhamou and Saïdi, 2006].

Au début, nous présentons le principe de l'algorithme Forwoard Checking combiné avec le principe de la symétrie et la dominance proposé par ces auteurs. Ensuite, nous montrons nos améliorations apportées par rapport à cet algorithme et nous finirons par la présentation des expérimentations réalisées.

5.2 Principe de SFC-weak-dom

Benhamou et Saïdi dans [Benhamou and Saïdi, 2006] ont choisi d'implémenter un algorithme **F**orward **C**hecking **S**implifié (notée SFC) dans lequel, les dominances et les symétries sont détectées et éliminées.

Le principe du Forward Checking [Haralik and Elliot, 1980] est basé sur le filtrage des domaines des variables non instanciées en tenant compte des valeurs des variables déjà instanciées. Dans le cas des NECSPs, le filtrage est simplifié. Si la variable courante v_i est instanciée à la valeur d_i, le filtrage consiste à enlever la valeur d_i des domaines des variables futures (variables non instanciées) ayant une contrainte avec v_i.

Grâce à la détection des dominances, les branches de l'arbre de recherche qui sont symétriques sont éliminées, étant donné qu'elles ne participent à aucune solution (Théorème 6). L'algorithme 4 illustre la procédure SFC combinée avec la propriété de la dominance du théorème 6 (notation SFC-weak-dom). Elle permet de tester la consistance d'un NECSP donné. Elle peut être aussi adaptée facilement pour le calcul de toutes les solutions non symétriques/non dominées.

L'algorithme *SFC-weak-dom* a pu résoudre plusieurs instances, certaines connue pour être difficiles. L'heuristique du choix de variable implémentée dans cet algorithme est l'heuristique *domdeg*(domaine/degré). C'est-à-dire que la future variable à choisir parmi les variables non encore instanciées est celle qui minimise le rapport :

$$r = \frac{\mid dom_{v_i} \mid}{degree(v_i)}$$

Algorithme 4 La méthode SFC combinée avec l'algorithme de détection des dominances

PROCEDURE *SFC-weak-dom*$(D, I, i$,**var** VFT : *liste*$)$
Début

Entrée : un ensemble de domaines D,
$I = (\langle v_1, a_1\rangle, ..., \langle v_i, a_i\rangle)$ une instanciation partielle des variables $\{v_1, ..., v_i\}$ où i l'index de la variable courante et VFT l'ensemble des variables $Var(T_I)$ de l'arbre d'échec T_I (au début VFT est vide). empty :boolean
VFT_tmp :liste
VFT_old :liste

1: **si** $(i = n)$ **alors**
2: $\{\langle v_1, a_1\rangle, ..., \langle v_n, a_n\rangle\}$ est une solution, printf(I), stop
3: **sinon**
4: $empty \leftarrow false$
5: $VFT\_tmp \leftarrow VFT$
6: $VFT\_old \leftarrow VFT$
7: **pour tout** ($v_j \in V$, telle que $C_{ij} \in C, v_j \in$ future(v_i)) **faire**
8: **si** (**not**(empty) **and** $a_i \in a_j$) **alors**
9: $D_j \leftarrow D_j - \{a_i\}$
10: **si** (D_j=$\emptyset$) **alors**
11: annuler les effets du filtrage
12: ajouter(v_j, VFT)
13: $empty \leftarrow true$
14: **fin si**
15: **fin si**
16: **fin pour**
17: **si** (**not**(empty)) **alors**
18: $v_{i+1} \leftarrow$ next-variable(v_i)
19: **répéter**
20: prendre $a_{i+1} \in D_{i+1}$
21: $D_{i+1} \leftarrow D_{i+1} - a_{i+1}$
22: $I = I \cup \{\langle v_{i+1}, a_{i+1}\rangle\}$
23: $VFT\_tmp \leftarrow VFT \cup VFT\_tmp$
24: $VFT \leftarrow VFT\_old$
25: SFC-weak-dom$(D, I, i + 1, VFT)$
26: weak_dominance($a_{i+1} \in D_{i+1}, VFT$,Cl(a_{i+1}))
27: $D_{i+1} \leftarrow D_{i+1} - Cl(a_{i+1})$
28: **jusqu'à ce que** $(D_{i+1} = \emptyset)$
29: **fin si**
30: $VFT \leftarrow VFT\_tmp$
31: ajouter(v_i, VFT)
32: **fin si**

fin

5.3 Améliorations

Tout comme *DSATUR*, *SFC-weak-dom* commence par chercher une clique de taille maximale, comme ce problème de recherche de la clique maximale est aussi NP-complet, alors, un algorithme du type glouton est utilisé [Sewell, 1996].

5.3.1 Première amélioration :

Une fois une clique est trouvée, les variables impliquées dans cette clique sont instanciées en premier lieu. Notre première amélioration repose sur le constat que parfois l'heuristique *domdeg* ne choisit pas la variable la plus pertinente.

Exemple 8 *Soit $P = (V, D, C, R)$ un NECSP, et soit v_i et v_j deux variables dans V. Au niveau d'une étape de filtrage quelconque, on a :*

— *La taille du domaine de v_i vaut 1 ($dom = 1$) et le degré de cette variable vaut 2 ($deg = 2$), le rapport :*

$$r_{v_i} = \frac{\mid dom_{v_i} \mid}{degree(v_i)} = \frac{1}{2} = 0.5$$

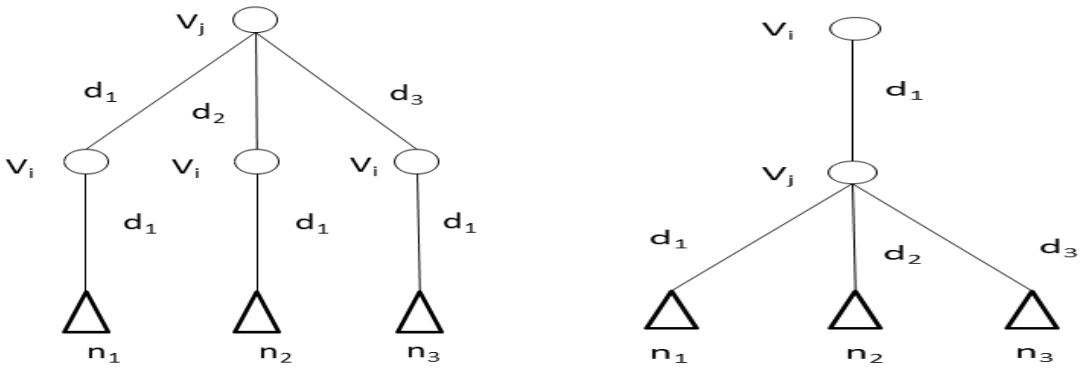
— *La taille du domaine de v_j égal 3 ($dom = 3$) et le degré égal 10 ($deg = 10$) donc, le rapport :*

$$r_{v_j} = \frac{\mid dom_{v_j} \mid}{degree(v_i)} = \frac{3}{10} = 0.3$$

Alors, on a : $r_{v_j} < r_{v_i}$.

$$r_{v_j} = \frac{\mid D_j \mid}{Degree(v_j)} < r_{v_i} = \frac{\mid D_i \mid}{Degree(v_i)}$$

Dans ce cas, malgré que v_i a une seule valeur dans son domaine, elle va être instanciée après la variable v_j.



(a) Favorisant v_j (dom=3). (b) Favorisant v_i (dom=1).

FIGURE 5.1 – Arbres d'affectations de v_i et v_j.

D'après la Figure 5.1, l'arbre (a) nécessite : $3 + 3 + n_1 + n_2 + n_3$ affectations, tandis que l'arbre (b) nécessite : $1 + 3 + n_1 + n_2 + n_3$ affectations. Alors, il est clair que si nous commençons par instancier v_i, nous allons réduire deux affectations.

Favoriser les variables qui ont une valeur par domaine peut booster la résolution, en effet, ils peuvent induire d'autres suppressions de valeurs, donc, d'autres variables peuvent à leur tour être réduit à une valeur par domaine et ainsi provoquer un effet cascade.

5.3.2 Deuxième amélioration :

Lors de la résolution, certaines variables deviennent isolées, c'est-à-dire toutes les variables voisines à celles-ci sont instanciées. Donc, lorsque ces variables isolées aparaissent, il faut ne pas les choisir en premier, car elles n'ont aucune influence sur la consistance localement.

Proposition 3 *Soit P un CSP et I une instanciation partielle, v_i une variable qui devient isolée suite à l'instanciation I. Alors :*
La consistance $P_I - \{v_i\} \Rightarrow$ la consistance de P_I.

Preuve *Tout il faut noter que $P_I - \{v_i\}$ est en fait le CSP local P_I induit par l'instanciation I duquel nous supprimons la variable v_i, donc v_i n'appartient plus aux variables du CSP P_I.*

Montrons alors que : la consistance de $P_I - \{v_i\} \Rightarrow$ la consistance de P_I.

Il y a deux possibilités pour le CSP $P_I - \{v_i\}$, soit il est consistant, soit il ne l'est pas :
Si le CSP $P_I - \{v_i\}$ est consistant cela veut dire qu'il admet au moins une solution. Notons cette solution Sol($P_I - \{v_i\}$). Soit a_i une valeur du domaine de la variable v_i. Or comme v_i est une variable isolée par hypothèse, alors l'affectation $v_i = a_i$ ne violera aucune des contraintes dans lesquels v_i est impliquée car rappelons que toutes les variables voisines à v_i (les variables qui partagent une contrainte avec v_i) sont déjà instanciées (c'est le même principe de l'algorithme FC). On en déduit que Sol($P_I - \{v_i\}$)$\cup\{v_i = a_i\}$ est une solution de P_I. Ce qui prouve que si le CSP $P_I - \{v_i\}$ est consistant alors le CSP P_I l'est aussi.

Si CSP $P_I - \{v_i\}$ est inconsistant alors comme le CSP $P_I - \{v_i\}$ est inclus dans le CSP P_I, puisque par définition le CSP $P_I - \{v_i\}$ est le CSP P_I sans la variable v_i. À cause de cette inclusion nous pouvons affirmer que si $P_I - \{v_i\}$ est inconsistant alors cela impliquera que P_I le serait aussi.

Donc quel que soit la consistance de $P_I - \{v_i\}$ nous aurons toujours :
la consistance de $P_I - \{v_i\} \Rightarrow$ la consistance de P_I. CQFD

Exemple 9 *Si toutes les variables qui ont des contraintes avec v_i ont été instanciées, alors, v_i devient une variable isolée ($deg_{v_i} = 0$).*

Soit $P = (V, D, C, R)$ un CSP définie par :
- $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$;
- $D = \{D_1 = \{a, b, c\}, D_2 = \{a, b, c\}, D_3 = \{a, b, c\}, D_4 = \{a, b, c\}, D_5 = \{c, d\}, D_6 = \{a, d\}, D_7 = \{b, c\}\}$;
- $C = \{C_{12}(v_1 \neq v_2), C_{13}(v_1 \neq v_3), C_{14}(v_1 \neq v_4), C_{23}(v_2 \neq v_3), C_{34}(v_3 \neq v_4), C_{35}(v_3 \neq v_5), C_{36}(v_3 \neq v_6), C_{45}(v_4 \neq v_5)\}$;

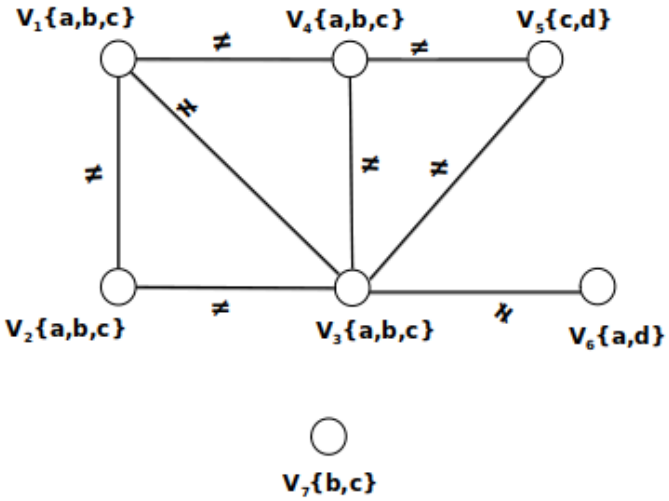


FIGURE 5.2 – Un CSP à contraintes de différence P

— Soit l'instanciation partielle : $I' = \{\langle v_1, a\rangle, \langle v_2, b\rangle, \langle v_3, c\rangle\}$. Donc I' rend v_6 une variable isolée avec le domaine $D_6 = \{a, d\}$. On peut l'instancier avec n'importe quelle valeur de son domaine, car elle joue aucun rôle dans la consistance du CSP (même dans le cas de l'inconsistance).

Nous proposons alors deux heuristiques pour le choix de la variable suivante à instancier.

1. **heur1 :** cette heuristique consiste à choisir les variables qui ont la taille du domaine égal 1. Sinon, elle prend la variable qui a le petit rapport ($r_1 = dom/deg$).

$$r_1 = \frac{\mid D_i \mid}{Degree(v_i)}$$

2. **heur2 :** cette heuristique consiste aussi à choisir les variables qui ont la taille du domaine égal 1 ($dom = 1$). Sinon, elle prend la variable qui a le petit rapport ($r_2 = dom/ddeg$) et de plus, elle ignore les variables qui ont le degré 0 ($deg = 0$).

$$r_2 = \frac{\mid D_i \mid}{Degree\_Dynamique(v_i)}$$

Nous donnons alors deux algorithmes (algorithmes 5 et 6). Dans l'algorithme 4 nous remplaçons à la ligne 18 next-variable() une fois par $heur1$ et une fois par $heur2$, ce que nous donne deux versions nouvelles que nous appelons *SFC-dom-heur1* et *SFC-dom-heur2*.

Algorithme 5 La Recherche de la variable suivante avec heur1

PROCEDURE *heur*1(VNI,r,deg) ;

Entrée : VNI :liste des variables non encore instanciées, deg : l'ensemble de degrés
r :réel avec r=max_réel(valeur trés grande)
L :Liste(liste temporaire)
$vnext$,tmp :Variables locales temporaires

Sortie : $vnext$:la prochaine variable à instancier

Début

```
1: r ← max_réel ;
2: L ← VNI ;
3: tant que (L ≠ ∅) faire
4:    tmp ← tête(L) ;
5:    L = L − {tmp} ;
6:    si (| D_tmp | = 1) alors
7:       VNI = VNI − {tmp} ;
8:       Retourne tmp
9:    sinon
10:      si ( | D_tmp | / deg_tmp < r) alors
11:         r ← |D_tmp| / deg_tmp ;
12:         vnext ← temp ;
13:      fin si
14:   fin si
15: fin tant que
16: VNI = VNI − {vnext} ;
17: Retourne vnext
```

fin

Algorithme 6 La Recherche de la variable suivante avec heur2

PROCEDURE *heur*2(VNI,r,deg) ;

Entrée : VNI :liste des variables non encore instanciées, $ddeg$: l'ensemble de degrés dynamique
L :Liste(liste temporaile)
$vnext$,tmp :Variables locales temporaires r :réel avec r=max_réel(valeur trés grande)

Sortie : $vnext$:la prochaine variable à instancier

Début

```
1: r ← max_réel ;
2: vnext ← val_Bidon ;
3: tant que (L ≠ ∅) faire
4:    tmp ← tête(L) ;
5:    L = L − {tmp} ;
6:    si (| D_tmp | = 1) alors
7:       VNI = VNI − {tmp} ;
8:       Retourner tmp
9:    sinon
10:      si (ddeg_tmp > 0 ET |D_tmp|/ddeg_tmp) alors
11:         r ← |D_tmp| / ddeg_tmp ;
12:         vnext ← temp ;
13:      fin si
14:   fin si
15: fin tant que
16: si (vnext = val_Bidon ) alors
17:    vnext=Premiere_var_avec _deg_zéro(VNI) ;
18: fin si
19: VNI = VNI − {vnext} ;
20: Retourne vnext
```

fin

5.4 Expérimentations

Le problème de coloration des graphes consiste à colorier les sommets d'un graphe sans que deux sommets reliés par une arête ne soient pas coloriés avec une même couleur. Pour cela, nous avons choisi de tester les performances des méthodes *DSATUR*, *SFC-weak-dom*, *SFC-dom-heur1* (heuristique dom/deg) et *SFC-dom-heur2* (heuristique dom/ddeg) sur les problèmes aléatoires et les instances issus de challenge *Dimacs*. Le langage de programmation utilisé est le *C* et le code source est compilé sous *Linux* sur une machine équipé par un processeur intel core i7 RAM 8G. Le temps d'exécution est limité en 7200 secondes.

5.4.1 Problèmes aléatoires de coloration des graphes

Les problèmes aléatoires de coloration de graphe sont générés en fixant les paramètres suivants :

— **n** est le nombre de sommets (les variables).
— **NbCouleur** le nombre de couleurs (les domaines).
— **d** la densité qui est un nombre entre 0 et 1 exprimant le rapport suivant :

$$d = \frac{nombre\ de\ contraintes}{nombre\ totale\ de\ contraintes\ possible}$$

Nous avons fixé n, $NbCouleur$ et d. Dans les Tables 5.1, 5.2 et la Table 5.3, nous donnons les résultats des quatre méthodes mentionnées précédemment pour des problèmes de coloration générés aléatoirement avec $n = 100$ et les densités $d = 0.15$ (faiblement denses), $d = 0.5$ (moyennement denses) et $d = 0.9$ (fortement denses). Chaque table illustre pour chaque méthode le nombre de couleurs (k), le nombre de nœuds moyen *(Nœud)*, le temps moyen d'exécution en minutes ($Temps$).

Pb	-	DSATUR		SFC-weak-dom		SFC-dom-heur1		SFC-dom-heur2	
k	%Taux	Nœud	Temps	Nœud	Temps	Nœud	Temps	Nœud	Temps
5	0	7,8	0,001	0	0	0	0	0	0
6	0	7,8	0,001	0	0	0	0	0	0
7	10	20,1	0,001	14,2	0	15,6	0	15,6	0
8	65	71,6	0,001	69,1	0	70,3	0	70,2	0
9	98	98,3	0,001	98,2	0	98,2	0	98,2	0
10	100	100	0,001	100	0	100	0	100	0
11	100	100	0,001	100	0	100	0	100	0
12	100	100	0,001	100	0	100	0	100	0
13	100	100	0,001	100	0	100	0	100	0
14	100	100	0,001	100	0	100	0	100	0
15	100	100	0,001	100	0	100	0	100	0

TABLE 5.1 – Instances aléatoires de coloration de graphe avec n=100 et d=0.15

Les Figures 5.3 et 5.4 présentent les courbes concernant les résultats des méthodes *DSATUR*, *SFC-weak-dom*, *SFC-dom-heur1* et *SFC-dom-heur2* de la Table 5.1. La Figure 5.3 montre le nombre de nœuds moyen et le temps d'exécution est montré dans la Figure 5.4.

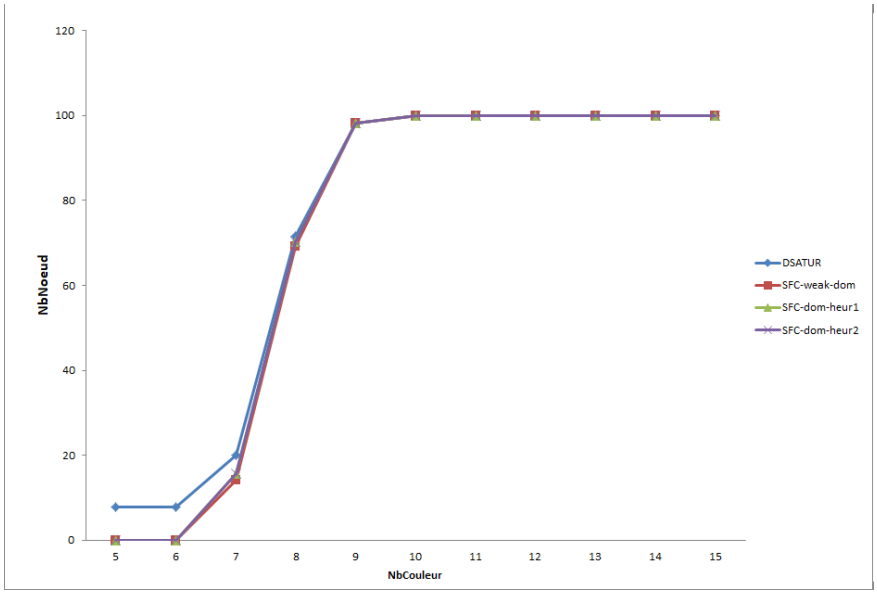


FIGURE 5.3 – Courbe représentant le nombre de noeud moyen d=0.15 n=100

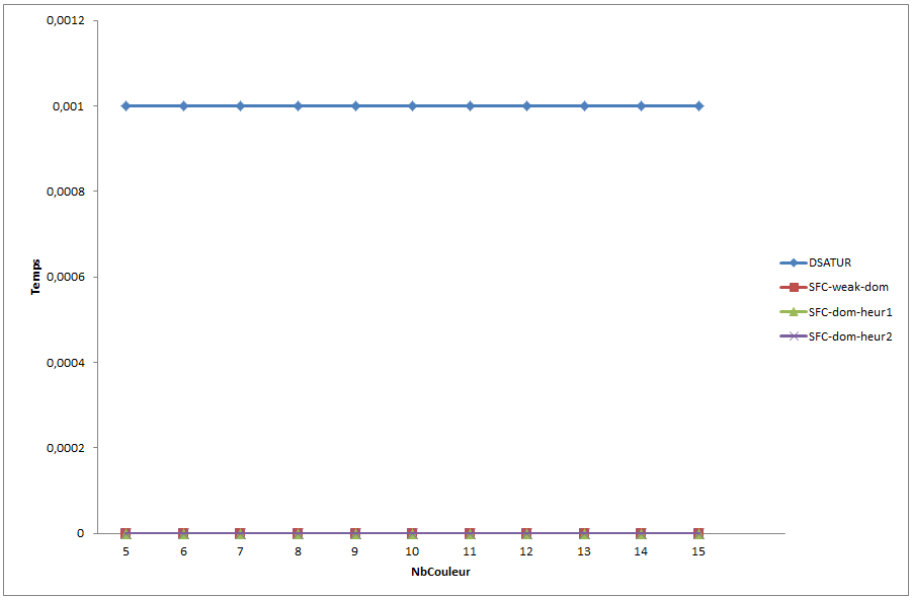


FIGURE 5.4 – Courbe représentant le temps CPU moyen d=0.15 n=100

La Figure 5.3 montre que le nombre de nœuds pour les quatre algorithmes est en croissance, après, il stabilise (NbCouleur=10). Les trois algorithmes *SFC-weak-dom*, *SFC-dom-heur1* et *SFC-dom-heur2* ont donné presque les mêmes résultats.

Dans la Figure 5.4, nous pouvons voir que les quatre algorithmes ont pris le même temps pour résoudre les instances. Malgré que la densité est faible, l'algorithme $DSATUR$ prend plus du temps et de nombre de nœuds que les autres algorithmes.

On a déduit que pour une densité faible 0.5, les algorithmes arrivent à résoudre efficacement toutes les instances et leurs performances sont comparables.

Pb	-	DSATUR		SFC-weak-dom		SFC-dom-heur1		SFC-dom-heur2	
k	%Taux	Nœud	Temps	Nœud	Temps	Nœud	Temps	Nœud	Temps
13	0	25	0,0282	19,6	0	16,5	0	16,5	0
14	0	251,8	0,0282	201,4	0	201	0	201	0
15	0	5836,1	0,0329	4362,9	0	4326,4	0	4289,3	0
16	0	207058,2	0,3008	141778,7	0,188	143367,6	0,1974	142729	0,41
17	44	2259979,8	4,2441	1892577,1	3,8493	1483323,3	2,726	151238	2,9281
18	100	228566,2	0,3619	163484,4	0,2585	140927,6	0,2068	138426	0,46
19	100	7907,4	0,0376	3782,3	0	3159	0	3397,1	0
20	100	251,1	0,0282	127,5	0	124,6	0	147,3	0
21	100	107,8	0,0282	100,6	0	100,5	0	101,1	0
22	100	100,6	0,0282	100	0	100	0	100,1	0

TABLE 5.2 – Instances aléatoires de coloration de graphe avec n=100 et d=0.5

Pour la Table 5.2, les résultats des méthodes *DSATUR*, *SFC-weak-dom*, *SFC-dom-heur1* et *SFC-dom-heur2* sont illustrées dans Les Figures 5.5 et 5.6 (nombre de nœuds moyen et temps d'exécution).

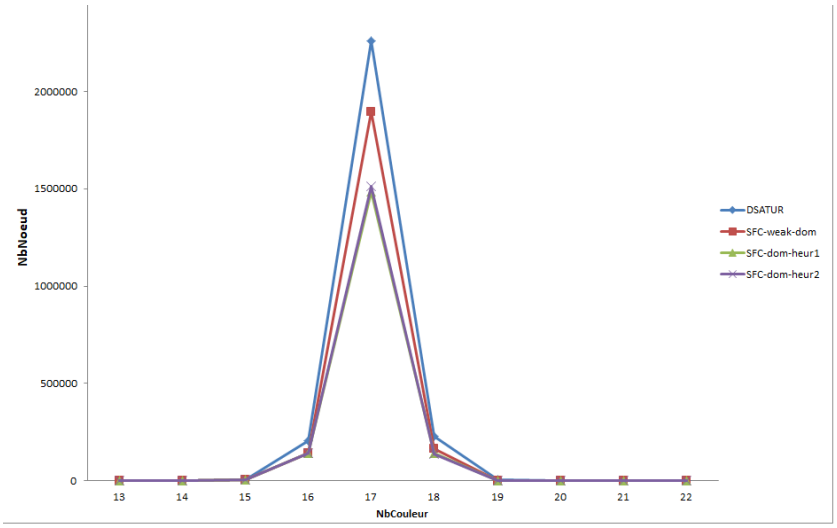


FIGURE 5.5 – Courbe représentant le nombre de nœud moyen d=0.5 n=100

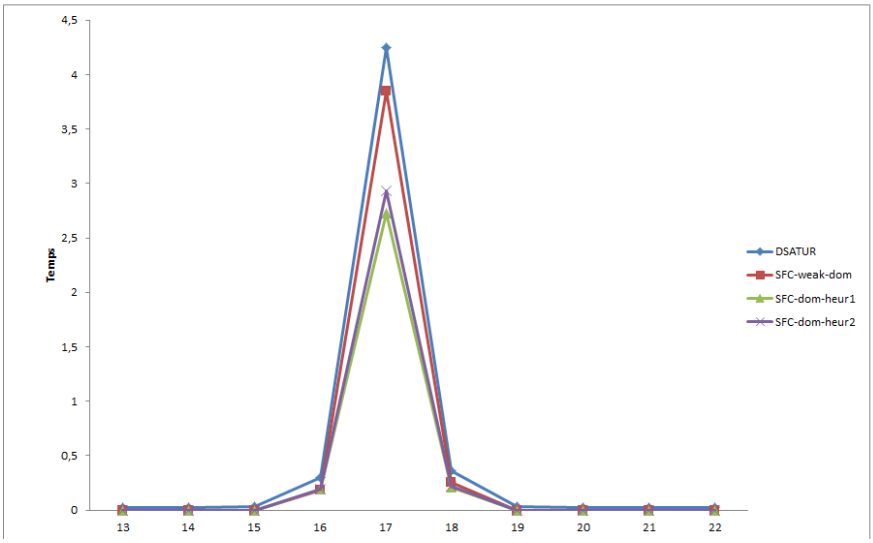


FIGURE 5.6 – Courbe représentant le temps CPU moyen d=0.5 n=100

À partir de la Figure 5.5 et 5.6, nous constatons que *SFC-dom-heur1 et SFC-dom-heur2 sont les meilleurs methodes pour les graphes de densité moyenne(d=0.5) surtout au voisinage du pic de difficulté, c'est-à-dire* $NbCouleur = 17$. *Ces instances sont les plus difficiles à résoudre et les deux méthodes ont les meilleurs performances quant à SFC-weak-dom et DSATUR.*

Pb	-	DSATUR		SFC-weak-dom		SFC-dom-heur1		SFC-dom-heur2	
k	%Taux	Nœud	Temps	Nœud	Temps	Nœud	Temps	Nœud	Temps
31	0	35,4	0,0611	8	0	7,6	0	7,6	0
32	0	56,1	0,0611	30	0	27,1	0	27,1	0
33	0	259,3	0,0611	177,7	0	158,6	0	158,6	0
34	0	2877,8	0,0658	1827,2	0	1726	0	1727,2	0
35	0	44815,6	0,1081	27548,4	0,0423	26298,6	0,0564	25882,6	0,0564
36	0	823269	0,9917	500900,7	1,2173	464302	1,645	454430,1	1,1045
37	13	8508056,3	10,011	4977902,3	12,5913	4506766,8	12,8874	4363171,5	12,22
38	68	22375226,9	27,3728	13656879,9	31,725	11405476,7	31,0153	11004917,2	29,8685
39	98	6035198,1	7,6281	3852947,9	8,7279	3713449,5	9,541	2921351,5	7,7033
40	100	328700,2	0,4089	321118,9	0,6533	308325,2	0,7097	178702	0,3995
41	100	3839	0,0658	8823	0,0047	7024,6	0,0047	4252,7	0
42	100	660,6	0,0611	252,9	0	221,4	0	182,1	0

TABLE 5.3 – Instances aléatoires de coloration de graphe avec n=100 et d=0.9

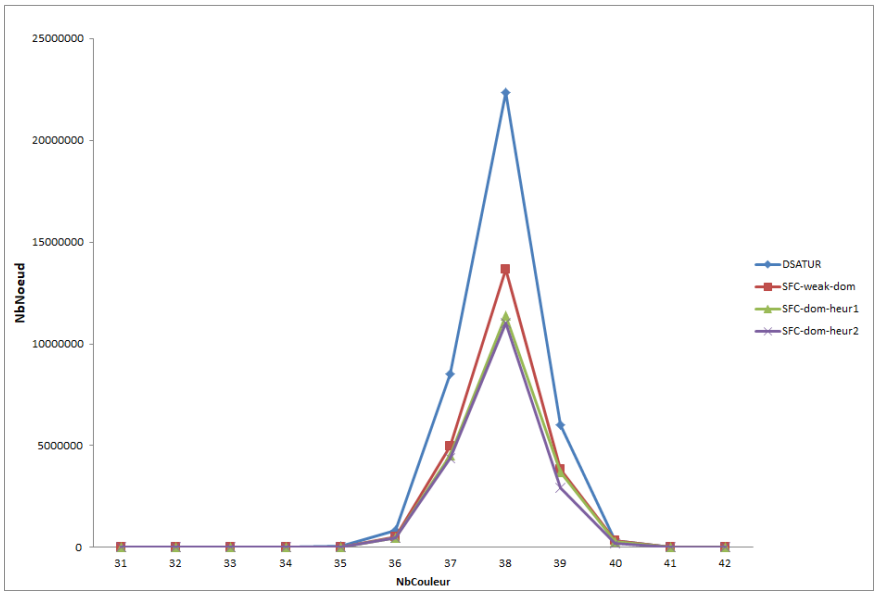


FIGURE 5.7 – Courbe représentant le nombre de nœud moyen d=0.9 n=100

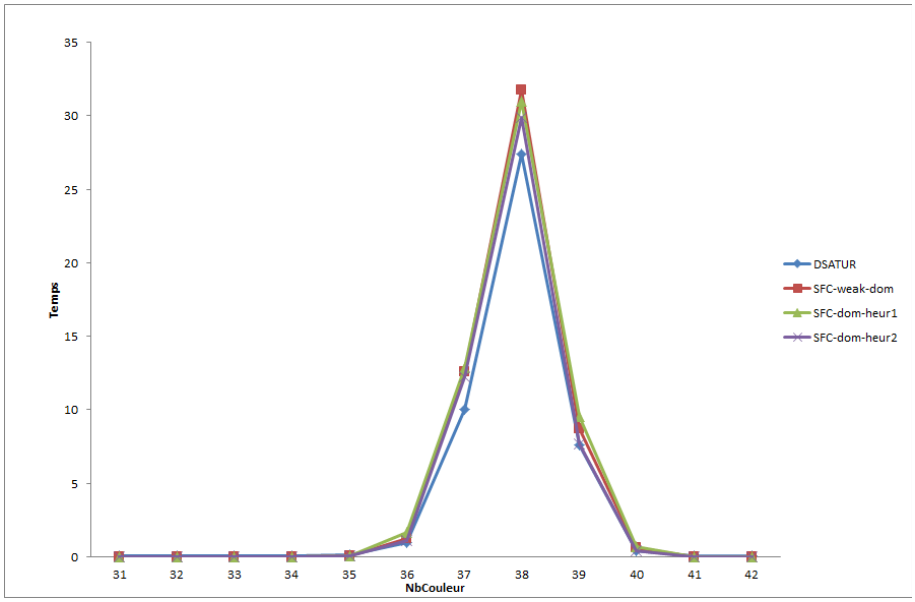


FIGURE 5.8 – Courbe représentant le le temps CPU moyen d=0.9 n=100

D'après les Figures 5.7 et 5.8, nous pouvons voir que DSATUR génère plus de nœuds que les autres, tandis que nos algorithmes proposés génèrent moins de nœuds. Mais, en temps CPU, DSATUR est meilleur que les autres, ce qui implique que ces aléatoires fortement denses contiennent moins de symétries où ces dernières prennent beaucoup de temps pour les détecter et malgré la rapidité de DSATUR, SFC-dom-heur2 a de bonnes performances grâce à son heuristique. La différence de temps entre les deux algorithmes n'ést pas trop grande.

On déduit que les graphes les plus difficiles sont les graphes de densités moyenne à forte. Nos deux approches ont des performances meilleures en moyenne que SFC-weak-dom.

5.4.2 Les benchmarks de Dimacs

Dans cette section, nous avons testé les quatre algorithmes sur un ensemble de graphes de la librairie DIMACS (https ://turing.cs.hbg.psu.edu/txn131/graphcoloring.html).

5.4.2.1 Instances Dimacs

DIMACS est le centre des sciences et de la technologie de la NSF pour les mathématiques discrètes et l'informatique théorique, basé à l'université Rutgers.

L'un des objectifs du challenge DIMACS est de faciliter l'effort des tests et de comparer des algorithmes et des méthodes heuristiques en fournissant un banc d'essai commun d'instances et d'outils d'analyse. Pour faciliter cet effort, un format standard doit être choisi pour les problèmes traités.

— **Fichier d'entrée :** *Un fichier d'entrée contient toutes les informations sur un graphique nécessaires pour définir un problème de clique ou un problème de coloration.*

— **Commentaires :** *Les lignes de commentaires donnent des informations sur le fichier, lisibles par l'homme et ignorées par les programmes. Chaque ligne de commentaire commence par un caractère minuscule c.*

— **Ligne de problème :** *Il y a une ligne de problème par fichier d'entrée. La ligne de problème doit apparaître avant toute ligne de nœud ou d'arc descripteur. La ligne de problème a le format suivant :* **p FORMAT NODES EDGES**.

  — *Le champ de* **FORMAT** *est pour la cohérence avec le challenge précédent, et devrait contenir "edge".*
  — *Le champ* **NODES** *contient une valeur entière spécifiant n, le nombre de nœud dans le graphe.*
  — *Le champ* **EDGES** *contient une valeur entière spécifiant m, le nombre d'arêtes dans le graphique.*

— **Descripteurs d'arêtes :** *Il existe une ligne de description des arêtes pour chaque arête du graphique, chacune a le format suivant : Chaque arête (v, w) apparaît exactement une fois dans le fichier d'entrée et n'est pas répété sous la forme (w, v).*

— **e W V :** *Le caractère minuscule e signifie qu'il s'agit d'une ligne de descripteur d'arête. pour un edge (w, v), les champs W et V spécifient ses extrémités(nombre de sommets et nombre de contraintes).*

Exemple 10 *Soit un exemple de format d'un fichier exemple.col :*
c FILE : exemple.col
p edge 4 4
e 1 2
e 1 3
e 1 4
e 2 3

La Table 5.4 montre les résultats des quatre algorithmes sur certains nombres d'instances de Dimacs. Nous donnons le nombre de nœuds, ainsi que le temps CPU (en seconde) pour chaque algorithme. Nous cherchons pour chaque problème le nombre minimal k (nombre chromatique) pour la coloration de graphe. Aussi, nous prouvons qu'il n'est pas possible de trouver $k-1$ coloration de graphe.

Le symbole "-" signifie que l'algorithme correspondant n'a pas retourné de réponse après deux heures de calcul.

INSTANCES	d	K	DSATUR		SFC-weak-dom		SFC-dom-heur1		SFC-dom-heur2	
instances			Nœud	Temps	Nœud	Temps	Nœud	Temps	Nœud	Temps
1-FullIns_4 (93 593)	0.138	5	-	-	1 368	0,00	1 238	0,00	1 101	0,00
1-FullIns_5 (282 3247)	0.081	6	-	-	748 094 596	1919,92	40 436 167	56,15	25 629 208	38,04
2-FullIns_3 (52 201)	0.151	5	156 663 425	50,7	359	0,00	84	0,00	84	0,00
5-FullIns_4 (1085 11395)	0.019	**9**	-	-	-	-	39 823 712	237,61	39 295 221	253,49
1-Insertions_4 (1085 11395)	0.104	5	-	-	42 358 481	23.76	29 618 501	10.16	1 949 294	0.71
2-Insertions_3 (37 72)	0.108	4	-	-	8 526	0.01	7 894	0.00	7 894	0.00
4-Insertions_3 (114 541)	0.050	4	623 718 510	389.81	1 167 877 441	551.5967	861 998 447	212.53	861 998 447	211.55
abb313GPIA (1557 65390)	0.044	**9**	-	-	-	-	6 605 350	76,92	11 256 792	119,42
ash331GPIA (662 4185)	0.019	4	2 610	0,1	901	0,03	895	0,05	895	0,05
ash608GPIA (1216 7844)	0.010	4	10 244	0,2	1 707	0,12	1 559	0,14	1 559	0,15
ash958GPIA (1916 12506)	0.006	4	-	-	7 167	0,34	16 828	0,45	5 444	0,40

INSTANCES	d	K	DSATUR		SFC-weak-dom		SFC-dom-heur1		SFC-dom-heur2	
instances			Nœud	Temps	Nœud	Temps	Nœud	Temps	Nœud	Temps
le450_5a (450 5714)	0.056	5	-	-	18 460	0,12	16 914	0,08	19 305	0,08
le450_5b (450 5734)	0.056	5	-	-	20 436	0,12	19 294	0,09	40 402	0,14
le450_5c (450 9803)	0.097	5	2 310	0,1	1 697	0,08	1 783	0,09	1 783	0,09
le450_5d (450 9757)	0.096	5	16 080 407	85,1	734	0,07	655	0,08	655	0,08
mug88_1 (88 146)	0.038	4	1 718 443 362	1534.03	995	0,0	995	0,0	407	0,0
mug88_25 (88,146)	0.038	4	576 051 437	359.73	1 631	0,0	1 631	0,0	471	0,0
myciel5 (47 236)	0.218	6	378 311	0,4	21 217	0,02	21 572	0,01	13 297	0,01
myciel6 (95 755)	0.169	7	-	-	29 853 398	44,05	14 210 227	12,59	10 015 341	8,93
qg.order30 (900 26100)	0.064	30	1 680	0.09	1 162	0.09	1 160	0.09	1 160	0.09
qg.order40 (1600 62400)	0.048	40	-	-	10 814 593	80.06	9 710 226	75.78	9 710 226	76.18
queen7_7 (49,952)	0.038	7	6 849	0,0	426	0,0	408	0,0	408	0,0
queen8_8 (64, 728)	0.361	9	1 581 661	2,2	1 353 680	1,66	1 354 613	1,32	1 354 613	1,33
queen9_9 (81, 2112)	0.325	10	1 235 141 962	2550,4	1 171 404 632	2201,89	1 179 158 202	1657,70	1 179 435 134	1706,18
R250.1 (250,867)	0.027	8	243	0,0	250	0,0	250	0,0	250	0,0
R250.1c (250,30227)	0.971	64	1 345 504	36,3	2 085	0,58	543	0,48	543	0,48
R250.5 (250,14849)	0.477	65	-	-	87 776	0,53	23 479	0,21	440 943	1,77
DSJR500.1c (500,121275)	0.972	85	-	-	28 067 141	1347,16	335 614	13,96	335 614	18,56

TABLE 5.4 – Problème de coloration de graphe de *DIMACS*

D'après les résultats de la Table 5.4 SFC-Dom-Heur1 et SFC-Dom-Heur2 sont les plus performants sur l'ensemble des instances testées. Certaines de ces instances telle que 5-FullIns_4, abb313 GPIA et DSJR500.1c sont même réputées pour être très difficiles ou même non résolues jusqu'à ce jour ce qui constitue un résultat remarquable.

Notons que SFC-weak-dom est plus performant que DSATUR mais reste bien sûr moins bien rapide de nos deux méthodes SFC-dom-heur1 et SFC-dom-heur2 qui ont des performances comparables entre-eux.

5.5 Conclusion

Dans ce chapitre, nous avons proposé deux améliorations pour l'algorithme SFC-weak-dom de [Benhamou and Saïdi, 2006]. Nous avons alors comparé nos améliorations à $DSATUR$ ainsi que l'algorithme de base SFC-weak-dom sur des instances de coloration de graphes aléatoire et de $DIMACS$.

Les résultats obtenus confirment les bonnes performances en général sur un grand nombre de ces instances. Nous avons même prouvé l'optimalité de certaines colorations qui sont à ce jour considérées comme des problèmes ouverts.

Chapitre 6

Conclusion

Dû à ses nombreuses applications dans différents domaines, le problème de coloration de graphes est un problème central en théorie de graphes. Ce problème peut être exprimé grâce au formalisme CSP à contraintes de différence (NECSPs).

Dans ce travail, nous avons proposé deux heuristiques afin d'améliorer l'heuristique du choix de variables utilisée dans [Benhamou and Saïdi, 2006] pour la résolution des NECSPs. l'algorithme proposé par les auteurs exploite la dominance et la symétrie afin d'améliorer les performances de la résolution.
Nous avons proposé deux types d'amélioration sur l'heuristique du choix de variables :

- *La première consiste à considérer les variables dont le domaine est réduit à une valeur ;*
- *La deuxième est de retarder la considération des variables isolées dans la résolution étant donnée qu'elles ne peuvent influencer la consistance locale du problème.*

Les résultats obtenus montrent une amélioration notable des performances de l'algorithme de base de [Benhamou and Saïdi, 2006]. D'ailleurs certaines instances de coloration, réputées pour être dures voire non résolues jusqu'à ce jour ont été résolues efficacement grâce à nos améliorations.

Ces résultats concluants nous incitent bien évidement à continuer ce travail. Parmi les points importants que nous souhaitons étudier dans le future, nous pouvons citer :

- *Amélioration des heuristiques en incluant la notion d'apprentissage à partir des échecs rencontrés ;*
- *Combinaison de l'algorithme FC avec des techniques du type Look-Back comme CBJ (Confilict Directed Backjumping)*
- *Amélioration de la propriété de dominance/symétrie afin de détecter et d'éliminer plus de valeurs dominées/symétriques, ce qui augmentera les performances de l'algorithme de résolution.*

Bibliographie

[Aarts and Lenstra, 1997] *E. Aarts and J.K. Lenstra. Local search in combinatorial optimization.* John Wiley and Sons, *1997.*

[Barnier and Brisset, 2004] *Nicolas Barnier and Pascal Brisset. Graph coloring for air traffic flow management.* Annals of operations research, *130(1-4) :163–178, 2004.*

[Benhamou and Saïdi, 2006] *Belaïd Benhamou and Mohamed Saïdi. Étude de la dominance dans les csps à contraintes de différence. In* Deuxièmes Journées Francophones de Programmation par Contraintes (JFPC06), *2006.*

[Benhamou and Sais, 1992] *B. Benhamou and L. Sais. Theoretical study of symmetries in propositional calculus and application.* Eleventh International Conference on Automated Deduction, Saratoga Springs,NY, USA, *1992.*

[Benhamou and Sais, 1994] *B. Benhamou and L. Sais. Tractability through symmetries in propositional calculus.* Journal of Automated Reasoning (JAR),, *12 :89–102, 1994.*

[Benhamou et al., *1994]* *B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language.* in proceedings of STACS'94, Caen France, *pages 71–82, 1994.*

[Benhamou, 1994a] *Belaï Benhamou. Study of symmetry in constraint satisfaction problems. In* PPCP '94 : Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming, *pages 246–254, London, UK, 1994. Springer-Verlag.*

[Benhamou, 1994b] *Belaid Benhamou. Theoretical study of dominance in constraint satisfaction problems. In* AIMSA '94 : Proceedings of the sixth international conference on Artificial intelligence : methodology, systems, applications, *pages 91–97, River Edge, NJ, USA, 1994. World Scientific Publishing Co., Inc.*

[Benhamou, 2004] *Belaid Benhamou. Symmetry in not-equals binary constraint networks. In* Proceedings of the satelite workshop of CP 2004, Symmetry in Constraints (SymCon'04), *pages 2–8, Toronto, september 2004.*

[BENSOUYAD, 2015] *Meriem BENSOUYAD. Université abdelhamid mehri. 2015.*

[Brelaz, 1979] *D Brelaz. New methods to color the vertices of a graph. communication, 1979.*

[Brouwer, 2007] *Andries E Brouwer. Sudoku puzzles and how to solve them.* European Mathematical Society Newsletter, *66 :13–17, 2007.*

[Brown, 1972] *J Randall Brown. Chromatic scheduling and the chromatic number problem.* Management Science, *19(4-part-1) :456–463, 1972.*

[Cohen et al., *2005]* *David A. Cohen, Peter Jeavons, Christopher Jefferson, Karen E. Petrie, and Barbara M. Smith. Symmetry definitions for constraint satisfaction problems. In* CP'05 :

Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, *pages 17–31, 2005.*

[Darga et al.*, 2004] Paul T. Darga, Mark H. Liffiton, Karem A. Sakallah, and Igor L. Markov. Exploiting structure in symmetry detection for cnf. In* DAC '04 : Proceedings of the 41st annual conference on Design automation*, pages 530–534, New York, NY, USA, 2004. ACM.*

[Dechter and Pearl, 1988] Rina Dechter and Judea Pearl. Tree-clustering schemes for constraint-processing. In AAAI*, pages 150–154, 1988.*

[Dechter and Pearl, 1989] Rina Dechter and Judea Pearl. Tree clustering for constraint networks (research note). Artif. Intell.*, 38(3) :353–366, 1989.*

[Dechter, 1990] Rina Dechter. Enhancement schemes for constraint processing : backjumping, learning, and cutset decomposition. Artif. Intell.*, 41(3) :273–312, 1990.*

[Dorne and Hao, 1998] Raphaël Dorne and Jin-Kao Hao. A new genetic local search algorithm for graph coloring. In International Conference on Parallel Problem Solving from Nature*, pages 745–754. Springer, 1998.*

[Freuder, 1982] Eugene C. Freuder. A sufficient condition for backtrack-free search. J. ACM*, 29(1) :24–32, 1982.*

[Freuder, 1991] Eugene C. Freuder. Eliminating interchangeable values in constraint satisfaction problems. In AAAI'91 : Proceedings of the 9th National Conference on Artificial Intelligence (vol. 1)*, pages 227–233, Anaheim, California, United States, 1991. AAAI Press / The MIT Press.*

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

[GAP, 2007] The GAP Group. GAP – Groups, Algorithms, and Programming, Version 4.4.10*, 2007.*

[Gaschnig, 1977] John Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In IJCAI*, page 457, 1977.*

[Glover, 1986] Fred Glover. Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res.*, 13(5) :533–549, 1986.*

[Goldberg, 1989] David E. Goldberg. Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.*

[Hale, 1980] William K Hale. Frequency assignment : Theory and applications. Proceedings of the IEEE*, 68(12) :1497–1514, 1980.*

[Haralik and Elliot, 1980] R. M. Haralik and G. L. Elliot. Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14*, pages 263–313, 1980.*

[Hartmanis, 1982] Juris Hartmanis. Computers and intractability : a guide to the theory of np-completeness (michael r. garey and david s. johnson). Siam Review*, 24(1) :90, 1982.*

[Hertz and de Werra, 1987] Alain Hertz and Dominique de Werra. Using tabu search techniques for graph coloring. Computing*, 39(4) :345–351, 1987.*

[Holland, 1975a] J. H. Holland. Adaptation in natural and artificial systems. The University of Michigan Press.*, 1975.*

[Holland, 1975b] John H Holland. Adaptation in natural and artificial systems ann arbor. The University of Michigan Press*, 1 :975, 1975.*

[Jégou and Terrioux, 2003] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. Artificial Intelligence*, 146 :43–75, 2003.*

[Jégou, 1990] Philippe Jégou. Cyclic-clustering : A compromise between tree-clustering and cycle-cutset method for improving search efficiency. In ECAI*, pages 369–371, 1990.*

[Jégou, 1993] P. Jégou. Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. In In Proceedings AAAI'93*, 1993.*

[Karoui, 2010] Wafa Karoui. Méthodes à divergences pour la résolution de problèmes de satisfaction de contraintes et d'optimisation combinatoire*. PhD thesis, Toulouse, INSA, 2010.*

[Kirkpatrick et al.*, 1983] Scott Kirkpatrick, D. Gelatt Jr., and Mario P. Vecchi. Optimization by simmulated annealing.* Science*, 220(4598) :671–680, 1983.*

[Krishnamurty, 1985] B. Krishnamurty. Short proofs for tricky formulas. Acta informatica*, (22) :253–275, 1985.*

[Lambert, 2006] Tony Lambert. Hybridation de méthodes complètes et incomplètes pour la résolution de CSP*. PhD thesis, Université de Nantes, 2006.*

[Laurent and Hao, 2009] Benoît Laurent and Jin-Kao Hao. Iterated local search for the multiple depot vehicle scheduling problem. Computers & Industrial Engineering*, 57(1) :277–286, 2009.*

[Mackworth, 1977] Alan K. Mackworth. Consistency in networks of relations. Artif. Intell.*, 8(1) :99–118, 1977.*

[McKay, 1981] Brendan McKay. Practical graph isomorphism, 1981.

[Montanari, 1974] Ugo Montanari. Networks of constraints : Fundamental properties and applications to picture processing. Information Science*, 7 :95–132, 1974.*

[Prosser, 1993] Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. Computational Intelligence*, 9 :268–299, 1993.*

[Puget, 1993] Jean-Francois Puget. On the satisfiability of symmetrical constrained satisfaction problems. In ISMIS '93 : Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems*, volume 689, pages 350–361, London, UK, 1993. Springer-Verlag.*

[Rose et al.*, 1976] Donald J. Rose, R. Endre Tarjan, and George S. Lueker. Algorithmic aspects of vertex elimination on graphs.* JOC*, 5(2) :266–283, 1976.*

[Sabar et al.*, 2012] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. A honey-bee mating optimization algorithm for educational timetabling problems.* European Journal of Operational Research*, 216(3) :533–543, 2012.*

[Sabin and Freuder, 1997] Daniel Sabin and Eugene C. Freuder. Understanding and improving the mac algorithm. In Gert Smolka, editor, CP'97 : Proceedings of Third International Conference of Principles and Practice of Constraint Programming*, volume 1330 of* Lecture Notes in Computer Science*, pages 167–181. Springer, 1997.*

[Sadr and Adve, 2012] Sanam Sadr and Raviraj Adve. Hierarchical resource allocation in femtocell networks using graph algorithms. In 2012 IEEE international conference on communications (ICC)*, pages 4416–4420. IEEE, 2012.*

[Sewell, 1996] EC Sewell. An improved algorithm for exact graph coloring. DIMACS series in discrete mathematics and theoretical computer science*, 26 :359–373, 1996.*

[Tarjan and Yannakakis, 1984] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput.*, 13(3) :566–579, 1984.*

[Tsang, 1993] E.P.K. Tsang. Foundations of Constraint Satisfaction. *Academic Press, London and San Diego, ISBN 0-12-701610-4, 1993.*

[Wilson, 2013] Robin Wilson. Four Colors Suffice : How the Map Problem Was Solved-Revised Color Edition*, volume 30. Princeton university press, 2013.*