République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Centre Universitaire BELHADJ Bouchaib - Aïn Témouchent
Institut des Sciences
Département des Mathématiques et Informatique

Mémoire
Pour l'obtention du Diplôme de Master en Informatique
Option : Réseaux et Ingénierie des Données (RID)

Réalisé par :
Melle.Mahieddine Ilham
Melle.Bouzid Radja

Test d'inconsistance du problème de satisfaction de contraintes basé sur la coloration de la microstructure

Encadrant : Mr.Mohamed Réda Saïdi (M.C.B) à C.U.A.T.
Soutenu le 08/10/2020
Devant le jury composé de :

Président : Mme.Fatima Zahra Belgrana (M.C.B) C.U.A.T.
Examinateurs : Mr.Ali Benzerbadj (M.C.B) C.U.A.T.
Encadrant : Mr.Mohamed Réda Saïdi (M.C.B) C.U.A.T.

Remerciements

Tout d'abord nous remercions ALLAH le tout puissant qui nous a donné la force, la patience, la foi et la volonté d'accomplir ce modeste travail.
Nous tenons à remercier nos parents qui par leurs prières et leurs encouragements, on a pu surmonter tous les obstacles.
Au terme de ce travail nous tenons à remercier vivement notre encadreur Mr.Saïd Mohamed Réda pour sa disponibilité, son guide, ses suggestions, orientations, remarques fructueuses, ses efforts déployés et ses précieux conseils, qu'il trouve ici notre profonde gratitude.
Nous tenons à remercier les membres du jury pour leur présence, pour leur lecture attentive de notre mémoire, ainsi que pour les remarques qu'ils nous adresseront lors de cette soutenance afin d'améliorer notre travail :
Nous tenons à remercier vivement Mme Belgrana de nous avoir fait l'honneur de présider ce jury et M. Ali Benzerbadj d'avoir accepté d'examiner notre travail.
Sans oublier de remercier l'ensemble des enseignants du département mathématiques et informatique du Centre Universitaire Belhadj Bouchaïb d'Aïn Témouchent (CUAT) pour leur 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 cet mémoire :

A nos chers parents pour leurs soutiens indéfectibles et leurs efforts fournis jours et nuits pour notre éducation. Aucune dédicace ne saurait exprimer l'amour, l'estime, le dévouement et le respect que nous avons toujours eu pour eux.
Qu'ALLAH nous protège et que la réussite soit toujours à notre portée pour que nous puissions vous combler de bonheur.
A nos chers sœurs et frères pour leurs encouragements permanent et leur soutien moral. A toutes nos familles, nos amis et nos proches pour leurs soutiens tout au long de notre parcours universitaire.

Résumé

Le problème de satisfaction de contraintes (CSP) binaire consiste à modéliser plusieurs problèmes tels que la coloration de cartes géographiques, la conception d'emplois... Cependant il existe deux grandes familles de méthodes pour tester la consistance (montre que le problème admet au moins une solution). La première famille représente les méthodes complètes qui consiste a faire un parcours totale de l'espace de recherche de solutions. L'avantage de ses méthodes est de prouver l'inconsistance de du problème de satisfaction de contraintes (CSP), cependant leur complexité croît exponentiellement avec la taille du problème. La seconde famille représente les méthodes incomplètes qui font un parcours local de l'espace de recherche de solutions. Ces méthodes ont été utilisées efficacement pour chercher des solutions pour des problèmes de grande taille que les méthodes complètes ne peuvent résoudre.Ces méthodes incomplètes ne prouvent pas l'inconsistance d'une instance du problème de satisfaction de contraintes (CSP).

Parmi les challenges mis en avant par la communauté CP est le challenge de Selman et al en 1997 qui consiste à proposer des méthodes incomplètes efficaces pour montrer l'inconsistance du problème de satisfaction de contraintes (CSP). Notre travail consiste à reprendre et éventuellement continuer la contribution de *Saïdi* et Benhamou faite en 2008 dans laquelle les auteurs introduisent une nouvelle méthode incomplète pour le test d'inconsistance qui s'appuie sur la notion de dominance entre des problèmes de satisfaction de contraintes (CSPs) et la coloration de la microstructure.

Mots-clés : Microstructure de CSP, clique, coloration dans les graphes, dominance entre CSPs (problème de satisfaction de contraintes).

Abstract

The binary Constraint Satisfaction Problem (CSP) consists of modeling several problems such as coloring of geographic maps, job design... However there are two large families : The first family represents the complete methods which consists in making a route total solution search space, however their complexity increases exponentially with the size of the problem. The second family represents the incomplete methods that make a local exploration in the space of search for solutions. These methods have been used effectively to find solutions to large problems that complete methods cannot solve. These incomplete methods do not prove the inconsistency of a Constraint Satisfaction Problem (CSP) instance.

Among the challenges put forward by the CP community of Selman and al in 1997 is to propose efficient incomplete methods to show the inconsistency of Constraint Satisfaction Problem (CSP). Our work consists of taking up and possibly continuing the contribution of *Saïdi* and Benhamou made in 2008 in which the authors introduce a new incomplete method for the inconsistency test which is based on the notion of dominance between CSPs (Constraint Satisfaction Problem) and the staining of the Constraint Satisfaction Problem (CSP) microstructure.

Keywords : Microstructures for CSPs, graph coloring problem, dominance between CSP (Constraint Satisfaction Problem), arc consistency.

الملخص

يعتمد CSP الثنائي على تصميم العديد من المشكلات مثل الخرائط التلوين وتصميم الوظيفة بحيث يوجد نوعان من الأساليب التي يمكن من خلالها اختبار اتساق النوع الأول بالأساليب الكاملة التي تتمثل في القيام بالبحث الكامل للعثور عن حلول من مميزات هذه الأساليب أنها قادرة على إثبات عدم التناسق إلا أن تعقيدها يزداد أضعافًا مضاعفة مع حجم المشكلة. وأما النوع الثاني فهو يعرف بالأساليب الغير المكتملة و بالبحث المطي للعثور عن الحلول هذه الطرق بشكل فعال لإيجاد حلول للمشكلات الكبيرة التي لا تستطيع الطرق الكاملة حلها ولكن غير قادرة على إثبات عدم تناسق.

من بين التحديات التي طرحها مجتمع الإنتاج التحدي الذي واجهه سلمان وآخرون في عام 1997 والذي يتمثل في اقتراح طرق فعالة غير مكتملة لإظهار عدم اتساق CSP . يتمثل عملنا في إعادة و تطوير و استمرار في مساهمة *Saïdi* و Benhamou التي قدمها في عام 2008 حيث قدم المؤلفون طريقة جديدة غير كاملة لاختبار عدم الاتساق والتي تستند إلى فكرة الهيمنة بين CSPs و تلوين البنية الدقيقة.

الكلمات المفتاحية : البنية الدقيقة لـ CSP، النقرات، التلوين في الرسوم البيانية، الهيمنة بين CSPs.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

La programmation par contrainte PPC (en anglais constraint programming) est un paradigme hérité du domaine d'intelligence artificielle. De nombreux problèmes combinatoires sont résolus par l'approche PPC, elle permet de représenter les problèmes d'une manière déclarative en les représentant par un ensemble de contraintes portées sur un ensemble de variables. En effet la modélisation de problèmes combinatoires a été introduite par Montanari en 1974 sous la forme de CSP (en anglais Constraint Satisfaction Problem)[3]..
Le problème de satisfaction de contrainte CSP est un formalisme occupant une grande place au sein des applications de domaine d'intelligence artificielle et de recherche opérationnelle, il existe plusieurs problèmes combinatoires qui peuvent être modélisés efficacement par CSP tel que : le problème de la coloration des graphes, la planification, l'allocation des fréquences, la gestion du trafic aérien et l'allocation des ressources. Le but de résoudre le CSP est d'obtenir une solution qui satisfait toutes les contraintes.
La résolution de CSP repose sur l'utilisation des approches combinatoires qui sont reparties en deux grandes classes :
La première classe définit les méthodes complètes qui consistent en une recherche arborescente pour trouver une solution, son avantage est montrer l'inconsistance de CSP (la preuve de non existence de solution). A cause de la taille de l'espace de recherche, ces méthodes systématiques prennent beaucoup de temps afin d'obtenir une solution, cependant leur complexité croît exponentiellement, elles ne sont donc utiles en pratique que lorsque la taille des problèmes est relativement petite.
La deuxième classe représente les méthodes incomplètes qui font une réparation d'une configuration permettant d'exploiter l'espace de recherche d'une manière non systématique, ces méthodes permettent de résoudre efficacement et d'obtenir la solution pour le problème de grande taille mais elles ne sont pas capable de prouver l'inconsistance d'une instance CSP.

Étant donner que la recherche n'est pas systématique l'obtention de solution n'est pas sûr, autrement dit on ne peut pas savoir si le problème admet une solution ou pas.
En 1997 Selman et Al ont propose un challenge pour prouver l'inconsistance de CSP par les méthodes incomplètes [2], peu de travaux ont été proposés pour la résolution de ce chalenge, on peut citer la méthode de coloration de la microstructure qui a été proposé par Gaur[31], la méthode de dominance et coloration de la microstructure qui a été présente par les auteurs Saïdi et Benhamou en 2008[2].
L'objectif de notre travail est de retrouver les résultats obtenu dans la contribution de Saïdi et benhamou afin de les valider et les améliorer éventuellement, aussi contrairement au auteurs (Saïdi et Benhamou, 2008) qui exportent leurs résultats uniquement sur des problèmes aléatoires, nous comme il est possible d'utiliser cette méthode concrètement pour la résolution d'optimisation (recherche du nombre chromatique des problèmes de coloration des n-reines) qui représente une classe de problème difficile.
L'organisation de notre travail est comme suit :
- nous commençons par une introduction générale sur ce travail.
- Chapitre 2 nous rappelons les notion de graphe et nous représentons le problème de coloration de graphe.
- Chapitre 3 nous présentons les principes fondamentaux de problème de satisfaction de contrainte et ses méthodes de résolution.
- Chapitre 4 nous rappelons le principe de l'optimisation, puis nous passerons à la notion de coloration de la microstructure, ensuite nous représenterons une méthode améliorée qui est basée sur la notion de l'arc de consistance et la coloration de la microstructure dans le CSP binaire quelconque.
- Chapitre 5 nous montrons les résultats de les expérimentations.
- le dernier Chapitre conclut ce travail et donne une vision souhaitée dans le future (utiliser d'autres algorithmes de coloration plus performants ...).

Chapitre 2

Notions de bases sur les graphes

Contents

1 Introduction

La théorie des graphes est une théorie importante dans l'informatique, elle consiste à étudier les graphes et modéliser plusieurs problèmes concrets. Dans ce chapitre nous allons rappeler les différentes notion de graphes, puis nous allons présenter le problème de coloration et ses méthodes de résolution.

2 Notion des graphes

Dans le cadre de la théorie des graphes, nous allons faire un rappel sur les différentes définitions de la notion des graphes.

Définition 1 (Graphes)
Un graphe est un couple $G(S, A)$ qui représente un ensemble de sommets et un ensemble des arrêtes dans lequel les sommets sont reliés entre eux par des arrêtes.

Définition 2 (Graphes non orientés)
Un graphe G est non orienté si tout couple (S,A) composé d'un ensemble de sommets S et d'un ensemble d'arrêtes A.
Une arrête de graphe est une paire de sommets (s, s')[11].

Définition 3 (Graphes orientés)
Un graphe G est orienté si tout couple (S,A) composé d'un ensemble de sommets S et d'un ensemble d 'arcs A.
Un arc de graphe est un couple (s, s') de sommets où s est l'extrémité initiale et s' est l'extrémité terminale [11].

Définition 4 (Graphes complets)
On définit un graphe complet si toute paire de sommets est reliée par un arc ou une arrête [11].

Définition 5 (Graphes partiels)
On dit que $G'(S, A)$ est un graphe partiel de graphe $G(S, A')$ si $A' \subset A$ [12].

Définition 6 (Sous-graphes)
Soit un sous ensemble de sommets $S' \subset S$, le sous graphe de G engendré par S' est le graphe G =(S',$A(S')$ où l'ensemble des sommet est S' et l'ensemble des arrêtes est $A(S') = \{(u, v) \in A$ et $u, v \in S'\}$ [12].

Définition 7(Cliques)
Une clique est un sous ensemble complet de sommets du graphe où chacun des sommets du sous ensemble est relié deux à deux [37].

Définition 8 (Chaînes)
Une Chaîne entre une paire de sommets (u,v) dans un graphe G est une suite des arrêtes, où chaque arrête relie u avec v [12].

Définition 9 (Chemins)
Un chemin dans un graphe G est une suite des arcs, où chaque arc relie une paire de sommets [12].

Définition 10 (Graphes connexes)
Un graphe non orienté G=(S,A) est dit connexe si pour tout couple de sommets (s,s'), il existe une chaine entre s et s'.
Un graphe orienté G=(S,A) est dit connexe si pour tout couple de sommets (s,s'), il existe un chemin de s vers s' et un chemin de s' vers s [11].

Définition 11 (Graphes bipartis)
Un graphe est biparti si l'ensemble de sommets S sont divisés en deux sous ensembles s_1 et s_2 dans lequel toutes les arrêtes relient un sommet dans s_1 à un sommet dans s_2 [11].

Définition 12 (Isomorphismes de graphes)
Deux graphes $G(S, A)$ et $H(S', A')$ sont isomorphismes s'il y a une bijection F telle que : $F : S \to S'$,et (x,y) est une arrête dans G si $(F(x),F(y))$ est une arrête dans H [11].

3 Historique de problème de coloration de graphes

Le problème de coloration de graphes date depuis plus d'un siècle. Parmi les problèmes le plus connu est le problème de quatre couleurs, ce dernier est resté sans solution. Le problème des quatre couleurs permet de colorier n'importe quelle carte géographique en utilisant quatre couleurs au maximum de sorte que deux régions voisines ne porte pas la même couleur. En 1852 un étudiant anglais en cartographie et mathématiques qui s'appelle Francis Guthrie a montré que quatre couleurs suffisantes pour colorier la carte des cantons d'Angleterre, sans affecter la même couleur à deux cantons ayant une frontière commune et Il a essayé de prouver que les quatres couleurs sont suffisent pour colorier n'importe quelle carte géographique de telle sorte que deux pays voisins ne porte pas la même couleur. En 1976 les deux mathématiciens Appel et W. Haken ont montré cette conjecture des quatre couleurs à l'aide d'un programme réalisé qui s'appelle Heesch [10].

4 Le problème de coloration de graphes

Le problème de coloration de graphes consiste à affecter une couleur à chaque sommet de sorte que chaque paire de sommets adjacents ne porte pas la même couleur [10].

Définition 13 (Coloration)

Une coloration C est une application de l'ensemble de sommets S dans l'ensemble des entier compris entres 1 et S [14].

Définition 14 (La k-coloration)

On définit la k-coloration d'un graphe $G = (S, A)$ par une application $c : S \to C$, où C est un ensemble de k couleurs (généralement un ensemble d'entiers, $C = \{1, 2, , , k\}$).
La coloration est dite propre si seulement pour toute arête (u, s) de A (entre les deux sommets u et s) : $C(u) \neq C(s)$.
Un graphe admettant une k-coloration propre est dit k-colorable [10].

Définition 15 (Le nombre chromatique)

Le nombre chromatique de graphe G=(S,A) représente le nombre minimum de couleurs nécessaires pour colorier G, on note $\chi(G)$ [12].

Définition 16 (La coloration des sommets)

La coloration de sommets d'un graphe consiste à assigner des couleurs à tous les sommets de graphe où chaque couple de sommets adjacents ne possède pas la même couleur [10].

Définition 17 (La coloration des arêtes)

La coloration des arêtes d'un graphe consiste à mettre une couleur à chaque arête du graphe de sorte que deux arêtes ayant un sommet commun ne porte pas la même couleur. L'indice chromatique $\chi(G)$ est le nombre minimum de couleurs nécessaires à la coloration d'arêtes [10].

5 Les méthodes de résolution

Montre que un graphe donnée est k-colorable est NP-complet c'est à dire qu'il y a un temps exponentielle pour le pire de cas pour montrer l'inconsistance, alors la recherche de nombre chromatique de graphe est un problème d'optimisation et il est de la classe NP-dure.

5.1 Les méthodes exactes

Les méthodes exactes sont utilisées généralement pour résoudre les problèmes de graphes et de trouver des résultats optimales pour des graphes ayant une petite taille. En 1972 l'algorithme de Randall-Brown a proposé d'effectuer une coloration des sommets du graphe à l'aide d'un algorithme d'énumération implicite de sorte que chaque sommet soit coloré à travers la plus petite couleur possible. Cette conjoncture a été développé par Brélaz en 1979 et Peemöller en 1983, ils ont utilisé une coloration d'une clique maximale pour lancer l'algorithme, ce qui donne une borne inférieure sur le nombre chromatique, en utilisant des éléments de sélection pour la coloration de sommet prochain. Les deux autres chercheurs Mehrotra et Trick transforment la coloration de graphe en un problème d'optimisation linéaire en nombres entiers (OLNE), ils ont développé un algorithme de génération de colonnes qui s'appelle branch and price. Le but est de minimiser le nombre total de stables pour recouvrir tous les sommets de graphe et instancié une couleur déférente à chaque stable. Cette technique de résolution a servi de point de départ pour la plupart des sous-algorithmes exactes [15].

5.2 Les méthodes constructives

Les approches constructives permettent de colorier le graphe un sommet à la fois, en sélectionnant à chaque étape celui qui semble être le meilleur selon un critère bien défini, elles sont rapides et donnent une solution proche à la solution optimale. Brélaz a développé un algorithme simple et efficace qui s'appelle DSATUR, le principe est d 'attribuer la plus petite couleur disponible aux nœuds par ordre décroissant de degré de saturation de nombre de couleurs distincts de ses voisins, en cas d'égalité, prioriser le nœud de degré maximal. L'algorithme s'arrête lorsque tous les sommets sont colorés, dans la même année Leighton a proposé un algorithme Recursive Largest First (RLF) qui permet de construire les classes de couleurs. Généralement l'utilisation des algorithmes constructifs ne donne pas de solution optimale mais on utilise ces algorithmes dans les méta heuristiques [15].

5.3 Les métaheuristiques

Les métaheuristiques consistent à colorier un graphe,attribuent graduellement une couleur à un sommet ou à un ensemble de sommets et essayé d'obtenir un meilleur résultat en manipulant la coloration courante. Cette méthode de résolution est payante car les meilleurs algorithmes actuels pour la coloration de graphe sont indéniablement les métaheuristiques. De plus il existe trois types d'algorithmes : les algorithmes basés sur la recherche locale, les algorithmes génétiques et les algorithmes hybrides [15].

Les Algorithmes de recherche locale

Les algorithmes de recherche locale sont devenus très connus, l'espace de recherche représente toutes les configurations possibles pour un problème donné, l'objectif de ces méthodes est d'approcher aux méthodes exactes en terme de précession et minimiser une certaine fonction objectif. L'un des algorithmes le plus célèbre Tabucol a été proposé par Hertz et Werra en 1987, tente de minimiser le nombre de conflits (fonction objectif) en changeant la couleur d'un sommet intervenant dans un conflit (voisinage) à partir d'une coloration de graphe avec conflits d'arêtes qui porte la même couleur à leurs extrémités. La couleur changée ne peut être affectée au même sommet qu'après un certain nombre d'itérations : cette couleur est dite tabou pour ce sommet [15].

Les algorithmes génétiques

En 1991 Davis a proposé un algorithme purement génétique qui représente une solution par la permutation des sommets et leur attribuent de manière séquentielle la première couleur disponible. Il existe deux algorithmes plus célèbres qui sont introduits par Holland en 1975 et par Gold Berg en 1989 [15].

Les algorithmes hybrides

Les algorithmes hybrides sont une combinaison des algorithmes de recherche locale avec des algorithmes génétiques, généralement les algorithmes hybrides donneront de meilleurs résultats sur les graphes au prix d'une complexité sans cesse accrue. L'une des méthodes les plus connues :
— Une hybridation de l'algorithme génétique et la recherche tabou a été proposé par Galinier and Hao en 1999.

— Une hybridation de l'algorithme de recherche locale et l'algorithme génétique a été proposé par Dorne and Hao en 1998 [15].

6 Domaines d'applications

La coloration d'un graphe est utile dans de nombreux domaines d'applications variés, tels que :
- Coloration des cartes géographiques.
- L'affectation de fréquences dans les réseaux cellulaires.
- Les emplois du temps.
- La gestion de chaines logistiques.
- Le calcul de dérivées, de matrices jacobienne et hessiennes.
- La gestion du trafic aérien.
- Jeux sodoku.
- Jeux de mot croisé.

Exemple :
coloration d'une carte géographique
Nous avons déjà rappelé dans l'historique que la coloration de carte géographique est probablement l'un des premiers problèmes étudiés dans la coloration de graphes, nous présentons la carte géographique de quelque wilayas d'Algérie avec ses 18 wilayas dans la Figure 2.1, le problème consiste d'attribuer une couleur à chaque wilaya de sorte qu'aucune des wilayas voisines ne présente pas la même couleur [16].

Ce problème peut être représenté sous forme d'un graphe de telle sorte :
- les sommets du graphe correspondent aux wilayas .
- les arêtes relient deux sommets qui correspondent à des wilayas ayant une frontière commune.

FIGURE 2.1 – La carte géographique de quelque wilayas de l'Algérie

Nous montrons dans la Figure 2.2 le graphe correspondant à la carte de quelque wilayas d'Algérie.

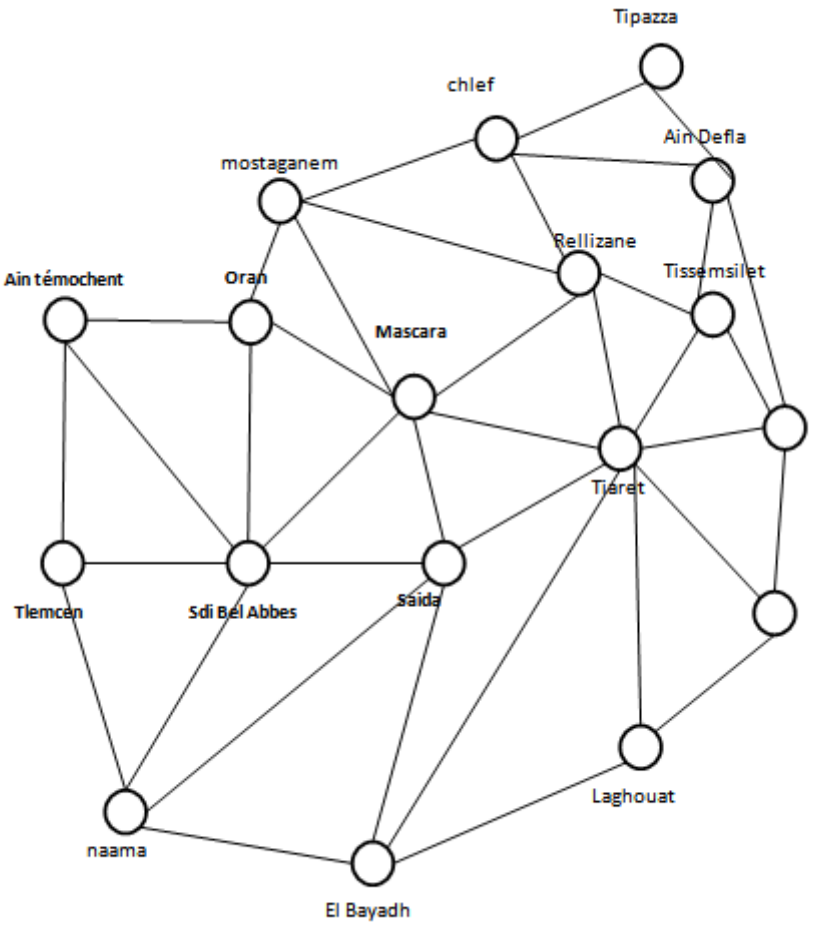


FIGURE 2.2 – Le graphe correspondant à la carte géographique de quelque wilayas de l'Algérie

A partir de la Figure 2.2, nous montrons le coloriage de graphe qui sera illustrée dans la Figure 2.3.

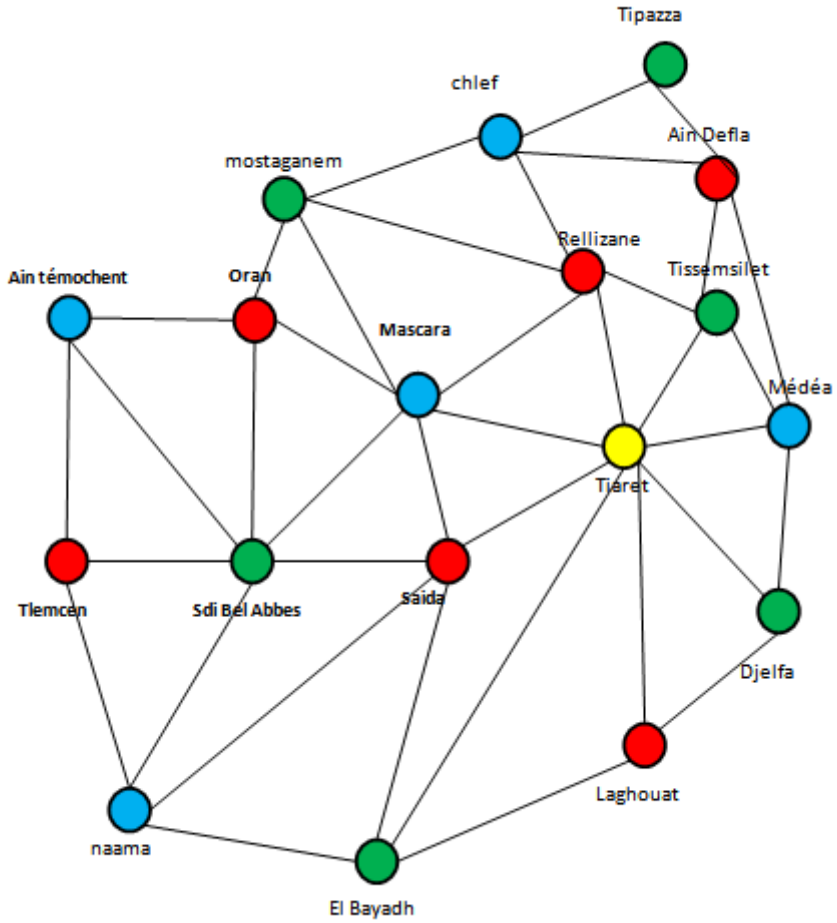


FIGURE 2.3 – Le graphe colorié correspondant à la carte géographique de quelque wilayas de l'Algérie

Nous montrons dans la Figure 2.4 la coloration de carte géographique de quelque wilayas d'Algérie où on a utilisé quatre couleurs possibles qui est un nombre minimum de couleur, donc c'est aussi le nombre chromatique de ce graphe.

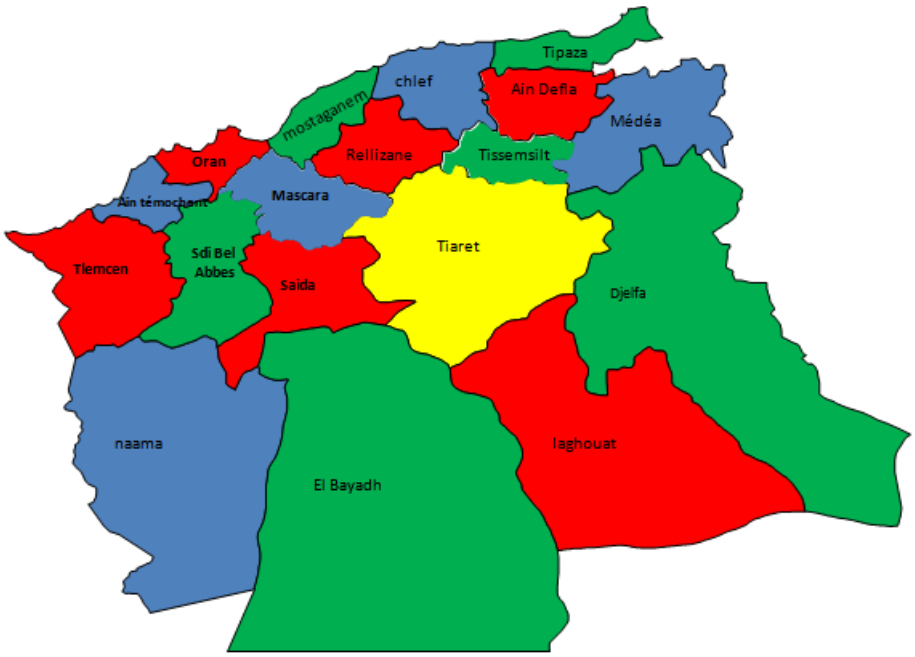


FIGURE 2.4 – La coloration de la carte géographique de quelque wilayas de l'Algérie

7 Conclusion

Dans ce chapitre nous avons cité quelques notions de base de la théorie des graphes et nous avons donné un aperçu sur le problème de coloration de graphes en rappelant toutes ces notions de base et ces différents domaines d'applications, ensuite nous avons donné un exemple sur la coloration d'une carte géographique.
Dans le chapitre qui suit nous allons définir la notion de problème de satisfaction de contraintes, puis nous allons montrer les différentes méthodes de résolution.

Chapitre 3

Problème de Satisfaction de Contraintes (CSP)

Contents

1 Introduction

Dans ce Chapitre, nous rappelons le formalisme CSP introduit par Montanari et les notions de base de CSP, nous nous intéressons au cas de CSP binaire dans lequel nous travaillons. Ensuite nous présentons quelques techniques de résolution.

2 Formalisme CSP

Définition 18 (CSP)
Un problème de satisfaction de contraintes (CSP) P est un quadruplet $P = (X, D, C, R)$, où :
– X est un ensemble fini de n variables X_1, ..., X_n.
– D est un ensemble de n domaines D_1, ..., D_n. Chaque domaine D_i est l'ensemble fini de valeurs pour la variable X_i.
– C est un ensemble fini de m contraintes C_1, ..., C_m. Chaque contrainte C_p est définie par l'ensemble de variables $V\ (C_p) = X_{p1}$, ..., $X_{pnp}\ \subseteq X$.
– R est un ensemble de m relations R_1, ..., R_m. Chaque relation R_p est un sous ensemble du produit cartésien $D_{p1} \times ... \times D_{pn}$. Pour chaque contrainte

C_i est associée à une relation R_i qui représente en fait l'événement des tuples permis (ou interdit) pour la relation [1].

Exemple :

Soit $P = (V; D; C; R)$ un CSP définie par :

— $V = \{v_1; v_2; v_3\}$.

— $D = \{D_1 = \{0, 1, 2\}\,;\ D_2 = \{0, 1, 2\}\,;\ D_3 = \{1, 3\}\}$.

— $C = \{C_1 = (v_1 + v_2 + v_3 = 3)\,;\ C_2 = (v_1 \neq v_2)\,;\ C_3 = (v_3 > 2)\}$.

— $R = \{\ R_{123} = \{(1, 1, 1); (0, 2, 1); (2, 0, 1); (0, 0, 3)\}\,;$
$R_{12} = \{(0, 1); (0, 2); (1, 0); (1, 2); (2, 0); (2, 1)\}\,;\ R_3 = \{(3)\}\ \}$.

Définition 19 (Contrainte)

Une contrainte est définie par une relation logique entre des différentes variables, ces dernières prennent des valeurs dans un domaine qui représente un ensemble fini [4].

Définition 20 (Arité d'une contrainte)

L'arité d'une contrainte représente le nombre de variables sur lesquelles elle porte, on dit que la contrainte est :

- Unaire si son arité est égale à 1.
- Binaire si son arité est égale à 2.
- N-aire si son arité est égale à n [5].

Définition 21 (Contrainte binaire)

Une contrainte binaire est une contrainte qui relie exactement deux variables où on impliquant les deux variables v_i et v_j, on notera C_{ij}. Ainsi R_{ij} est la relation correspondant a la contrainte C_{ij} [2].

Définition 22 (CSP binaire)

Un CSP binaire P est un CSP où toutes ses contraintes sont des contraintes binaires peuvent être représenté par un graphe de contraintes $G(V; E)$ où l'ensemble des sommets V est l'ensemble des variables CSP et chaque arête $(v_i\,;\ v_j) \in E$ connecte les variables v_i et v_j impliquées dans la contrainte $C_{ij} \in C$ [2].

Exemple :

Soit $P = (V\,; D\,; C\,; R)$ un CSP binaire définie par :

— $V = \{v1; v2; v3; v4\}$.

— $D = \{D_1 = \{0, 1\}\,;\ D_2 = \{0, 1\}\,;\ D_3 = \{1\}\,;\ D_4 = \{0, 1, 2\}\}$.

— $C = \{C_{12} = (v_1 \neq v_2); C_{14} = (v_1 \neq v_4); C_{23} = (v_2 = v_3); C_{24} = (v_2 \neq v_4)\}\,;$

— $R = \{R12 = (0, 1), (1, 0); R14 = (0, 1), (0, 2), (1, 0), (1, 2);$
$R23 = (1, 1); R24 = (0, 1), (0, 2), (1, 0), (1, 2)\}$.

La Figure 3.1 donne le graphe de contraintes du CSP P.

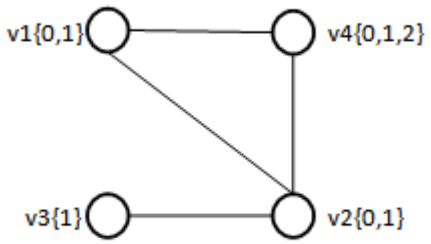


FIGURE 3.1 – Le graphe de contraintes de P

Définition 23 (Graphe de contraintes)
Le graphe de contrainte est défini par des variables qui sont représentées par les sommets et les contraintes par des arrêtes [34].

Définition 24 (La microstructure) La microstructure d'un CSP binaire P est un graphe $M_P(\bigcup_{i=1}^{n}(\{v_i\} \times D_i), \acute{E})$,
où un sommet de ce graphe représente un couple $\langle variable, valeur \rangle$ et où 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 pas de contraintes entre les variables correspondantes [2].
La Figure 3.2 donne le graphe de contraintes du CSP P.

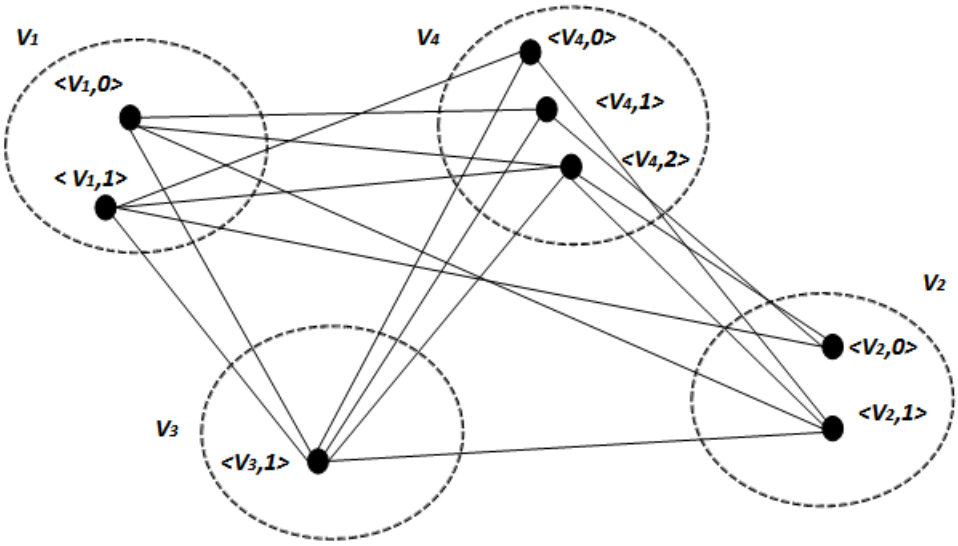


FIGURE 3.2 – La microstructure de P

Définition 25 (Contrainte Alldifferent)
Une contrainte Alldifferent est une contrainte globale (sur toutes les variables de CSP) qui force toutes les variables impliquées par la contrainte à prendre des valeurs différentes [2].

Exemple :
Soit $P = (V; D; C; R)$ un CSP définie par :
— V =$\{v_1; v_2; v_3\}$.
— D =$\{D_1$= $\{0, 1, 2\}$; $D_2 = \{0, 1, 2\}$; $D_3 = \{0, 1, 2\}\}$.
— C =$\{C_1$=$(v_1 \neq v_2 \neq v_3)\}$.
— R =$\{$ R_{123}=$\{(0, 1, 2); (0, 2, 1); (1, 0, 2); (1, 2, 0); (2, 0, 1); (2, 1, 0)\}\}$.
La Figure 3.3 donne le graphe de contraintes du CSP P.

Remarque :
La contrainte Alldifferent est équivalente à un CSP où son graphe de contrainte est complet c'est à dire que chaque couple de variable sont lié par une contrainte binaire de différence. Cette contrainte Alldifferent est très étudiée et utilisée dans PPC.

Exemple :
Soit $P' = (V'; D'; C'; R')$ un CSP binaire équivalent à P définie par :
—$V' = \{v_1; v_2; v_3\}$.
—D'= $\{D_1$= $\{0, 1, 2\}$; $D_2 = \{0, 1, 2\}$; $D_3 = \{0, 1, 2\}\}$.
— C'=$\{C_{12}$=$(v_1 \neq v_2)$; C_{13}=$(v_1 \neq v_3)$;C_{23}=$(v_2 \neq v_3)$ $\}$.

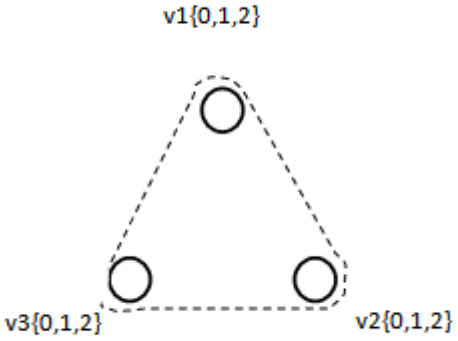


FIGURE 3.3 – La représentation de graphe de contraintes de P

— R'={ R_{12}=$\{(0,1);(0,2);(1,0);(1,2);(2,0);(2,1)\}$.
R_{13}=$\{(0,1);(0,2);(1,0);(1,2);(2,0);(2,1)\}$
R_{23}=$\{(0,1);(0,2);(1,0);(1,2);(2,0);(2,1)\}$ }.
La Figure 3.4 donne le graphe de contraintes CSP P'.

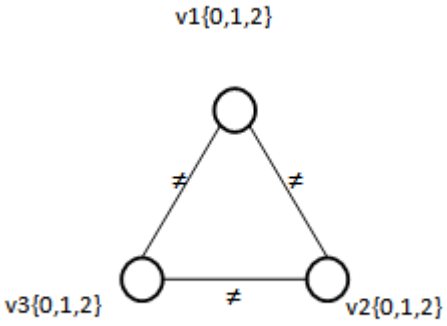


FIGURE 3.4 – La représentation de graphe de contraintes P'

Définition 26 (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 une valeur d_i de son domaine D_i. Une contrainte $C_i \in C$ est *satisfaite* par I si la projection de I sur les variables impliquées dans C_i est un tuple de R_i.
Une instanciation d'un sous ensemble des variables de V d'un CSP $P = (V, D, C, R)$ est appelée *instanciation partielle.*
Une instanciation est totale si elle est définie sur toutes les variables [2].

Définition 27 (Solution)
L'instanciation totale I des variables d'un CSP $P = (V, D, C, R)$ est consistante si elle satisfait toutes les contraintes de C. I est appelée solution du CSP P.
Un CSP est consistant s'il admet au moins une solution autrement il est inconsistant [1].

Définition 28 (Espace de recherche)
L'espace de recherche d'un CSP est le produit cartésien de domaine de n variable, on notera le nombre d'élément par :
Card $\{d_1 \times d_2 \times ...d_n\}$.
Si le domaine de la variable est de même taille d alors : d^n [9].

3 Notion de consistance

3.1 Consistances partielles et méthodes de filtrage

Les techniques de filtrage permettent de supprimer des inconsistants, il existe plusieurs niveaux de consistance telle que la consistance chemin et la k-consistance, la consistance d'arc qui représente la base de la consistance [1].

Définition 29 (Consistance d'arc)
Dans un CSP $P = (V, D, C, R)$, on dit un domaine $D_i \in D$ est 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 $(a, b) \in R_{ij}$. Si tous les domaines sont arc-consistants alors le CSP P est arc-consistant [1].

Définition 30 (Consistance de chemin)
Dans un CSP $P = (V, D, C, R)$, le couple de variables $\{v_i, v_j\}$ vérifie la consistance de chemin si et seulement si $\forall (a, b) \in R_{ij}, \forall v_k \in V, \exists c \in D_k$ telle que $(a, c) \in R_{ik}$ et $(b, c) \in R_{jk}$. Si $\exists v_i, v_j \in V$ le couple $\{v_i, v_j\}$ vérifie la consistance de chemin alors la consistance de chemin est vérifiée par le CSP P [1].

Définition 31 (La k-Consistance)
La k-consistance est une généralisation de la consistance d'arc et la consistance de chemin et d'aprés Freuder un CSP est dit k-consistant si et seulement si toute instanciation de $k-1$ variables sont consistante et qui peuvent être étendue d'une façons consistante à n'importe quelle k^{eme} variable. Un CSP est défini k-consistant s'il est i-consistant pour tout $i \leq k$.
- la 2-consistance correspond à la consistance d'arc.
- La 3-consistance est la consistance de chemin.

Chaque niveau de consistance est une opération de filtrage. La complexité de calcul du filtrage k-consistant est de $O(n^k.d^k)$, où n est le nombre de variables et d la taille du plus grand domaine [1].

4 Les techniques de résolution des CSPs

On définit deux grandes familles de technique de résolution, la première famille qui s'appelle méthodes complètes et la seconde famille sont des méthodes incomplètes [1].

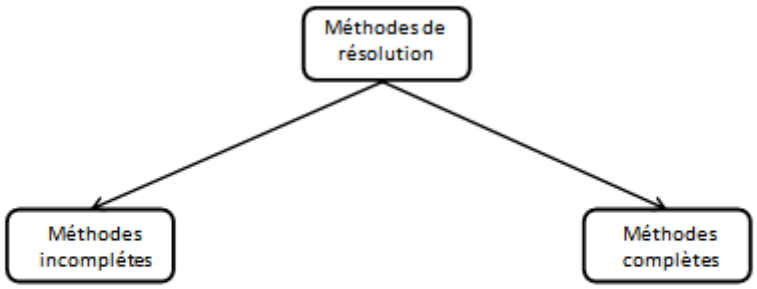


FIGURE 3.5 – Les méthodes de résolution de CSP

4.1 Les méthodes incomplètes

Les méthodes incomplètes (telles que la recherche locale (LS)de [Aarts and Lenstra,1997]) s'appuient sur des heuristiques qui consistent à exploiter des parties spécifiques de l'espace de recherche dans le but d'obtenir une solution. Il existe des différentes approches qui appartiennent à la classe de méthode de la recherche locale tel que :
Le recuit simulé, la recherche tabou et les méthodes basées sur les algorithmes génétiques. Les techniques incomplètes ont un avantage de résoudre des problèmes de grand taille que les méthodes complètes ne peuvent résoudre, mais l'obtention de solution n'est pas garantie [1].

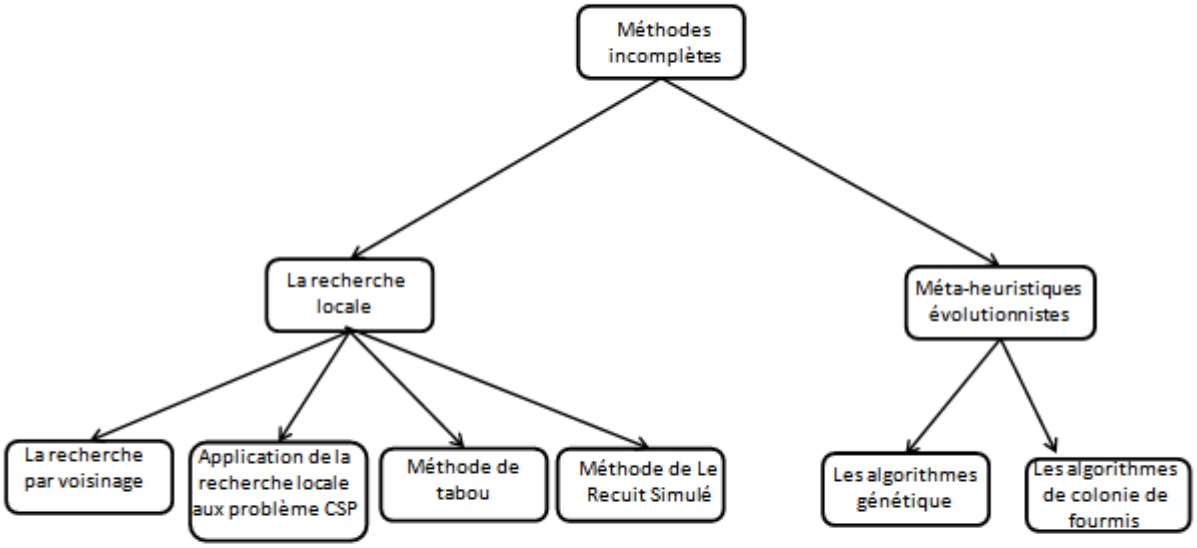


FIGURE 3.6 – Les méthodes incomplètes

4.2 Les méthodes complètes

Les méthodes complètes consistent à parcourir totalement l'espace de recherche de solutions, ces méthodes ont l'avantage de prouver l'inconsistance d'un CSP. Dans cette Section nous allons présenté le principe de différentes méthodes complètes pour résoudre le CSP, on peut citer : la méthode de filtrage, la méthode énumérative, la méthode de décomposition et le principe de chacune d'elles [1].

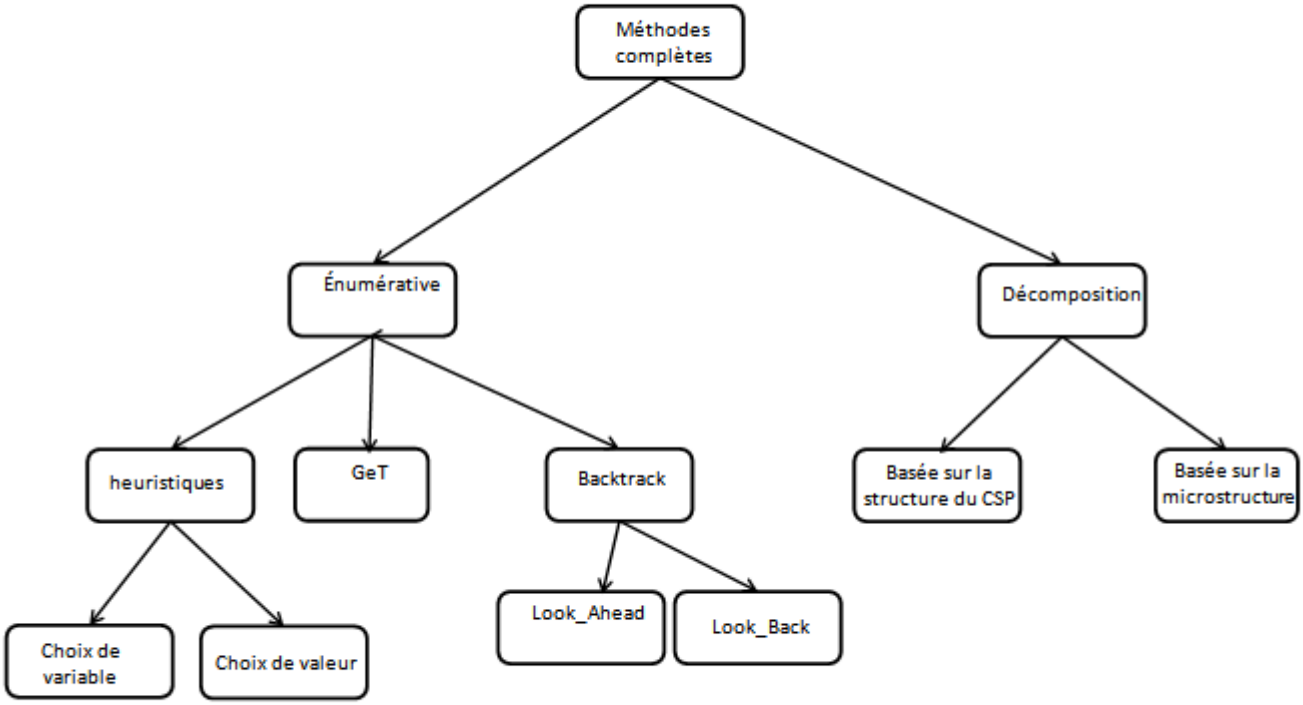


FIGURE 3.7 – Les méthodes complètes

1.Les méthodes énumératives

Les algorithmes de recherche énumératif consistent à exploiter toutes les affectations possibles des variables dans l'espace de recherche[4].

Générer et tester(GeT)

L'approche Générer et tester est une technique qui consiste à générer toutes les configurations possibles, c'est à dire toutes les combinaisons possibles de valeurs des variables et de tester si les contraintes sont vérifiées, jusqu'à obtenir une instanciation qui permet de vérifier toutes les contraintes, qui seront considérées alors comme une solution.
Trouver une solution revient à parcourir potentiellement l'espace de recherche, donc cette méthode est très couteuse puisqu'elle est exponentielle [1].

Exemple : Nous allons illustrer dans la Figure 3.8 un arbre de recherche développé par la méthode GeT correspondant à l'exemple précédent de Figure 3.1.

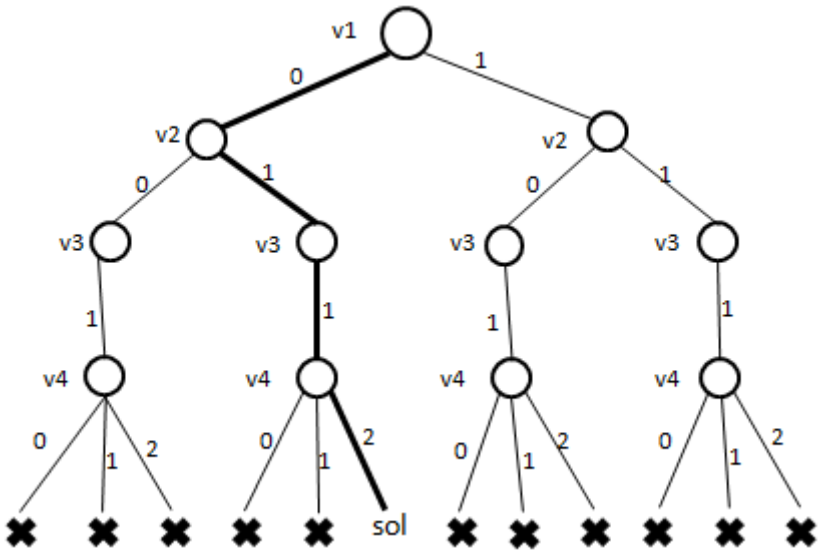


FIGURE 3.8 – L'arbre de recherche de P

Algorithme de Backtrack

L'algorithme de Backtrack est une amélioration de la méthode GeT qui consiste à générer progressivement une solution de sorte que les variables sont affectées une par une et à chaque échec détecté en revenant en arrière.
Elle permet de tester la consistance durant l'instanciation de variable entre cette instanciation courante et la nouvelle instanciation obtenue pour vérifier si l'instanciation est consistance et toutes les contraintes testées sont satisfaites par la nouvelle instanciation.
Donc l'instanciation courante est consistance. Si on a arrive à instancier la dernière variable du problème, l'instanciation courante réalise une solution au problème, sinon on passe à la variable suivante. Si la nouvelle instanciation ne satisfait pas une contrainte, on sélectionne une nouvelle valeur pour instancier la variable courante, s'il n y en a plus alors le Backtrack est réalisé sur l'instanciation de la variable qui précède la variable courante pour essayer une autre valeur. L'algorithme 1 définit cette technique [1].
L'utilisation de l'algorithme de Backtrack pour la résolution d'un CSP est coûteuse car il est exponentielle dans les pires des cas. A cause de cet inconvénient ils ont proposé deux stratégies pour améliorer cet algorithme .

Amélioration du Backtracking

Les améliorations suivent deux stratégies :

Approches du type regarder devant (Look-Ahead)

Les méthodes du type regarder devant (Look-Ahead) consistent à détecter d'avance de futures situations d'échec. Elles diffèrent selon le niveau du

Algorithm 1 algorithme de Backtrack

PROCEDURE Backtrack(I, k) **debut**

Require: $I = \{\langle v_1, d_1\rangle, \langle v_2, d_2\rangle, ..., \langle v_k, d_k\rangle\}$ une instanciation partielle, k un entier

```
1: if (I est consistant) then
2:     if (k = n) {n est le nombre de variables.} then
3:         < I est une solution >
4:     else
5:         for all (d_{k+1} ∈ D_{k+1}) do
6:             I ← I ∪ {⟨v_{k+1}, d_{k+1}⟩}
7:             Backtrack(I, k + 1)
8:         end for
9:     end if
10: end if
```

fin

filtrage appliqué. Le premier niveau représente la procédure FC (ForwardChecking) qui est l'un des algorithmes les plus célèbres (voir Algorithme 2). Il existe d'autres tels que : la procédure Partial Look-Ahead et la procédure Full Look-Ahead ou aussi la procédure Real Full Look-Ahead dont une version célèbre d'implémentation est MAC (Maintaining Arc Consistency). La méthode Forward Checking permet d'utiliser le filtrage qui se fait sur les domaines des variables non encore instanciées en testant le voisinage immédiat de la dernière variable instanciée. Avant de descendre 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 déterminé 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 [1].

Algorithm 2 algorithme FC

PROCEDURE Forward_Checking(D, I, k) **debut**

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

```
1: if (k = n) then
2:     < I est une solution >
3: else
4:     <filtrer les D_i, pour tout k < i de telle sorte que si C_{ki} ∈ C, ∀d_i ∈ D'_i, alors (d_i, d_k) ∈ R_{ki}>
5:     if (∀i, j / D'_i ≠ ∅) then
6:         for all (d_{k+1} ∈ D'_{k+1}) do
7:             I ← I ∪ {⟨v_{k+1}, d_{k+1}⟩}
8:             Forward_Checking(D', I, k + 1)
9:         end for
10:    end if
11: end if
```

fin

Approche du type Regarder en arrière (Look-Back)

Les méthodes de type Regarder en arrière (Look-Back) permettent d'analyser les situations d'échec pour éviter qu'elles se reproduisent. On définit la cause

de l'échec et ensuite on réalise un retour directement vers la variable responsable de l'échec. Parmi ces techniques, nous pouvons citer l'algorithme BJ (Back Jumping) de Gaschnig et plusieurs variantes proposées par Dechter avec le Graph-Based Jumping et Prosser avec le CBJ (Conflic-Directed- Back Jumping) [1].

Heuristiques

Les heuristiques sont des règles qui permettent de définir l'ordre d'instanciation de variable et l'ordre de choix de valeur, elles sont reparties en deux types qui sont :

Heuristiques de choix de variable

Les méthodes énumératives utilisent généralement l'heuristique de l'ordre d'instanciation des variables, le traitement de problème est déterminé selon le choix du meilleur ordre, ce dernier peut être statique [1] ou dynamique [2]. Ces types d' heuristique ont un but de détecter l'échec le plus tôt possible (fail first principal).[4].

Heuristiques de choix de valeur

Les heuristiques de choix de valeur sont limitées par rapport au heuristique du choix de variable. L'heuristique de choix de valeur de type min-conflict est considérée comme meilleure heuristique et qui permet d'instancier la variable courante par une valeur contenant un nombre minimum de valeurs qui ne sont pas compatibles avec les valeurs des variables non instanciées et on détermine le choix de valeur selon l'ordre croissant des nombres de conflits [4].

2.Les méthodes de décomposition

Les méthodes de décomposition consistent à transformer le problème en un ensemble de sous-problème. On peut les répartir en deux classes :

Les méthodes de décomposition basées sur la structure du CSP

Ces méthodes reposent sur les propriétés structurelles du CSP. Elles s'appuient sur le corollaire de Freuder, qui définit que la résolution des CSP acycliques est linéaire. Parmi ces méthodes, nous citons la méthode de l'ensemble coupe-cycle, la méthode du regroupement en arbre et la méthode du regroupement cyclique, qui s'inspire des deux techniques de décompositions précédente [1].

Les méthodes de décomposition basées sur la microstructure

1. L'ordre statique : établit au debut avant de lancer la résolution.
2. L'ordre dynamique : la future variable est choisi au fur et à mesure de la résolution.

Cette méthode repose sur la microstructure d'un CSP, cette dernière permet de diviser un problème en plusieurs sous problèmes, il existe plusieurs méthodes qui utilisent la décomposition telle que la méthode de triangulation de la microstructure qui repose sur la recherche des cliques maximales dans le graphe de la microstructure d'un CSP dans laquelle chaque clique maximale représente une décomposition des domaines.
Ainsi la méthode base sur la recherche d'un sous graphe triangulé induit maximal qui s'appuie sur MTIS dans la microstructure d'un CSP et sur l'exploitation des cliques maximales. Plusieurs méthodes de décomposition s'appuient sur la notion des graphes triangulés.
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 d'autres types d'approches qui sont connues sous le nom méthodes hybrides qui consistent a faire une combinaison entre des approches différentes, tel que : la méthode BTD qui représente la combinaison entre la méthode de décomposition et la méthode de Backtracking et la methode FC-CBJ qui consiste à combiner la méthode ForwardChecking avec la méthode CBJ (Conflic-Directed-BackJumping) [1].

5 Conclusion

Dans ce Chapitre nous avons montré le formalisme de CSP, puis nous avons présenté les différentes méthodes de résolution, chacune de ces méthodes a des avantages et des inconvénients. Donc le choix de la technique dépend du problème à résoudre.
Dans le Chapitre suivant nous allons rappeler quelques notions de base sur les méthodes de preuve d'inconsistance et notre méthode améliore.

Chapitre 4

Les Méthodes de preuve d'inconsistance de CSP

Contents

1 Introduction

Dans ce Chapitre nous allons faire un rappelle sur la notion de la complexité et l'optimisation combinatoire, puis nous allons présenter les différentes méthodes de preuve d'inconsistance.

Définition 32(Complexité algorithmique)
La performance des algorithmes est déterminée par les deux paramètres le temps et l'espace mémoire pour résoudre un problème, ainsi la performance d'un algorithme dépend du temps de calcul nécessaire.
On définit la complexité en temps d'un algorithme la fonction $H(n)$ qui définit le nombre maximum d'opérations élémentaires effectuées afin de pouvoir résoudre un problème de taille n est le nombre de variables décrivant le problème, ainsi on relie chaque opération élémentaire avec une unité de temps [18].

Définition 33(L'algorithme polynômiale et l'algorithme exponentielle)
On appelle un algorithme polynomiale si sa fonction de la complexité $H(n) \in O(p(n))$, P est un polynôme en n, autrement dit s'il existe $k > 0$ telle que $H(n) \in O(n^k)$.
Une fonction est dite non polynomiale (NP) ou exponentielle si la fonction de complexité H n'est pas majorée par un polynôme [18].

2 Les classes de complexité

Dans le problème de satisfaction de contrainte, la difficulté des problèmes appartient à quatre classes : P, NP, NP-Difficile et NP-Complet.
La classe P(Polynomial) : cette classe contient tous les problèmes de décision pouvant être résolus par un algorithme (algorithme polynomial) s'exécutant en temps polynomial (selon la taille de l'entrée) sur une machine de Turing déterministe [7].
La classe NP (Non-déterministe Polynomial) : cette classe contient tous les problèmes de décision pouvant être résolus par un algorithme en temps d'exécution (algorithme) polynomiale sur une machine de Turing non-déterministe [7].
La classe NP-complet : c'est un problème qui appartient à la fois NP et NP-difficile, tous les problèmes qui appartiennent à NP peuvent être réduit en temps polynomial [26].
La classe NP-difficile : c'est un problème plus difficile que NP, il consiste à vérifier s'il existe un problème NP-complet se transformant à ce problème par une réduction de Turing [26].
Nous montrons dans la Figure 4.1 un schéma qui illustre des inclusions des classes de la complexité dans la conjecture $P \neq NP$.

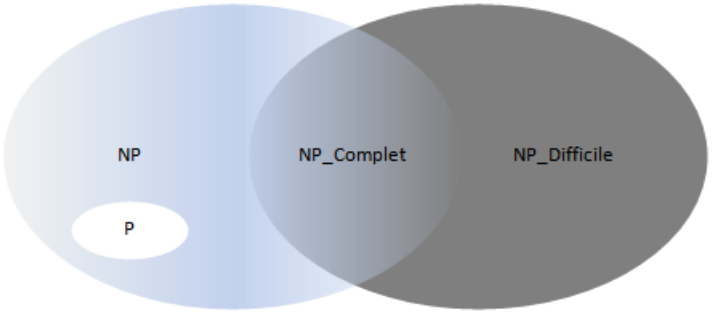


FIGURE 4.1 – La représentation des inclusions des différentes de classes de complexité

3 L'optimisation Combinatoire

3.1 Introduction

L'optimisation combinatoire est une des origines de la théorie mathématique, son principe consiste à chercher une solution optimale (meilleure solution) à travers un ensemble fini. Ainsi elle permet de résoudre des problèmes combinatoires classiques tels que : le problème de sac a dos, le problème de voyageur de commerce, le problème de coloration de graphe, le problème de transport, en utilisant des méthodes combinatoires, en effet l'optimisation combinatoire est utilisée dans plusieurs domaines et elle joue un rôle important dans les problèmes de décision [24].
Dans cette Section nous allons montrer le concept fondamental de l'optimisation combinatoire.

3.2 Le problème d'optimisation combinatoire

On définit le problème d'optimisation comme suit :
- L'espace d'état : c'est un espace de recherche grand et fini, il définit un ensemble de domaine de sorte que des différentes variables prennent des différentes valeurs.
- Les variables de problème : appelées aussi variables de décision possèdent des différents types (réelle, entière, booléenne) et représentent les données qualitatives ou quantitatives.
- Fonction objectif : cette fonction appelée aussi une fonction de coût qu'on cherche à optimiser (soit on maximise ou minimise), elle définit le but à atteindre ainsi elle est liée aux deux problèmes d'optimisations mono-objectif et multi-objectif.
- L'ensemble des contraintes : ils consistent à limiter l'espace de recherche, ils représentent l'ensemble des contraintes égalités et non égalités pour lequel les variables doivent être satisfaites.
Le problème d'optimisation combinatoire est reparti en deux catégories à partir de nombre d'objectif qu'on cherche à maximiser ou minimiser, l'une de ces catégories s'appelle problème d'optimisation mono-objectifs et l'autre s'appelle problème d'optimisation multi-objectif [19].

3.3 Le problème d'optimisation mono-objectif

Le problème d'optimisation mono-objectif est un problème d'optimisation qui possède une seule fonction d'objectif (fonction unique), il permet de chercher à optimiser (maximiser ou minimiser) la fonction objectif par rapport à un ensemble de solution admissible, pour obtenir une meilleure solution qui est représentée à travers un seul objectif (critère) de problème.
La formule mathématique de problème optimisation mono-objectif est définit comme suit :
On représente une seule fonction objectif f.
— L'ensemble S représente l'ensemble des solutions réalisables.
— Soit$x \in S$, min $f(x)$ ou $maxf(x)$, (soit maximiser f ou minimiser).
— On définit le point k est minimum global si :
$k \in S$, $\forall x \in S$, $f(k) \preceq f(x)$.
— On définit le point k est un, maximum global si :
$k \in S$, $\forall x \in S$, $f(k) \succeq f(x)$.
— On note le minimum ou le maximum de la valeur$f(k)$ par $fmin$ ou $fmax$ [24].

3.4 Le problème d'optimisation multi-objectif

Le problème d'optimisation multi-objectif (PMO) est un problème qui permet de maximiser ou minimiser (optimiser) plusieurs objectifs simultanément, il consiste de fournir aux décideurs un ensemble de solutions optimales appelées Front de Pareto.[23]
Le problème d'optimisation multi-objectif peut être définit comme suit :
$\begin{cases} Optimiserf(k)(f_1(k), f_2(k), ..., fp(k)) \text{ et} \\ telquek \in etn \geq 2S. \end{cases}$
La résolution de problème d'optimisation multi-objectif ne fournit pas une seule solution mais plusieurs, ainsi ses principes permettant de trouver la solution par la minimisation d'un ensemble de fonction objectif, de plus on peut transformer un problème de maximisation en un problème de minimisation selon l'équivalence suivante :
Maximiser $f(k) \Longleftrightarrow minimiser - f(k)$ [23].

4 Etat de l'art

L'exploitation des méthodes incomplètes pour la preuve d'inconsistance des CSPs est un défi très intéressant, il a été déjà proposé en 1997 par Selman et Al, mais malheureusement il existe très peu de travaux qui contribuent à ce défi à cause de sa difficulté. L'une des premières méthodes qui a été proposé par Gaur et Al en 1997 pour la preuve d'inconsistance des CSPs binaires est basée sur la coloration de la microstructure. Grâce à cette méthode ils sont arrivés à montrer que si le graphe de microstructure d'un CSP avec n variables peut être coloré avec $(n-1)$ couleurs alors le problème est inconsistant. Pour plus de détail nous vous invitons de lire l'article de Gaur et Al de 1997 [31]. A partir de là, *Saïdi* et Benhamou ont contribué à ce défi avec l'article de la nouvelle méthode de 2008 afin de détecter l'inconsistance des CSPs, cette nouvelle méthode est une amélioration de la méthode de preuve d'inconsistance basé sur la coloration de la microstructure qui a été proposée par Gaur et Al de 1997. Le principe de la nouvelle approche s'appuie sur la notion de la coloration de la microstructure et dominance, en effet les expérimentations faites montrent que les résultats obtenus sont meilleurs (en taux de détection) par rapport aux anciennes méthodes [2].

5 Les méthodes basées sur la coloration

Dans cette Section nous commençons par une définition de l'algorithme de coloration de graphe Trick et nous allons rappeler les notions de bases des méthodes introduites par *Saïdi* et Benhamou en 2008, Gaure et Al en 1997, puis nous présentons notre méthode proposée.

5.1 L'algorithme de coloration de graphe Trick

L'algorithme Trick c'est un algorithme de coloration de graphe qui est parmi les méthodes exactes les plus connu, il donne une borne inférieure sur le nombre chromatique, l'algorithme a été programmait par Michael Trick en 1994, par le droit d'auteur l'utilisation de ce programme est permisse c'est un code facile pour la coloration des graphiques.
L'algorithme Trick a comme entrée un graphe dans un fichier sous la forme suivant :
- La première ligne contient le nombre des sommets et le nombre des contraintes.
- Tout ce qui suit contient les numéros de sommets (de 1 à n) incidents à Contrainte.

Exemple : d'un graphe cycles à quatre sommets C_4.
p edge 4 4
e 1 2
e 2 3
e 3 4
e 4 1
Nous avons représenté dans la Figure 4.2 graphe cycles à quatre sommets C_4 .

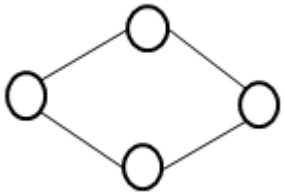


FIGURE 4.2 – Le graphe cycles à quatre sommets C_4

Le résultat de la coloration de graphe par l'algorithme Trick est stocké dans un fichier nommé Solution sous la forme suivant :
- La première ligne représente le nombre de sommets.
- La deuxième ligne représente le nombre de couleurs.
- Les autres lignes représentent la coloration des sommets (de 1 à n) chaque nombre correspond a une couleur .
Exemple : coloration de graphe cycles à quatre sommets C_4 par l'algorithme Trike.
4
2
1
2
1
2
Nous avons représenté dans la Figure 4.3 la coloration de graphe cycles à quatre sommets C_4 par l'algorithme Trike.

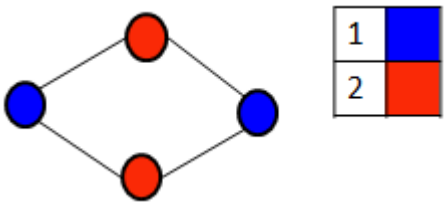


FIGURE 4.3 – La coloration de graphe cycles à quatre sommets C_4 par l'algorithme Trick

5.2 La méthode coloration de la microstructure

Théorème 1 Un CSP binaire composé de n variables possède une solution si et seulement si sa microstructure possède une clique de taille n [31].

Preuve voir [31].

Théorème 2 Si un graphe G de s sommets ($s \geq n$) peut être coloré avec $n-1$ couleurs alors il ne peut avoir une clique de taille n [31].

Preuve voir [31].

Ceci consiste donc de représenter le corollaire suivant, qui définit la principale contribution de Gaur :

Corollaire 1 Si la microstructure d'un CSP binaire à n variables peut-être colorer avec $n-1$ couleurs alors le CSP est inconsistant [31].

Preuve voir [31].

Le principe de l'algorithme de coloration de la microstructure

L'algorithme consiste à traiter des CSPs binaires quelconques pour détecter l'inconsistance par la coloration de la microstructure. Au début l'algorithme commence par la génération de la microstructure du CSP binaire P de n variables, il va la colorie avec un nombre de couleurs et affecter le résultat dans la variable (NB couleur). Dans le cas où le nombre de couleur est inférieur au nombre de variables n, l'algorithme détecte l'inconsistance du CSP, sinon il ne détermine pas si le CSP est consistant ou non [31].
Nous avons représenté dans Figure 4.4 l'organigramme de la méthode de la coloration.

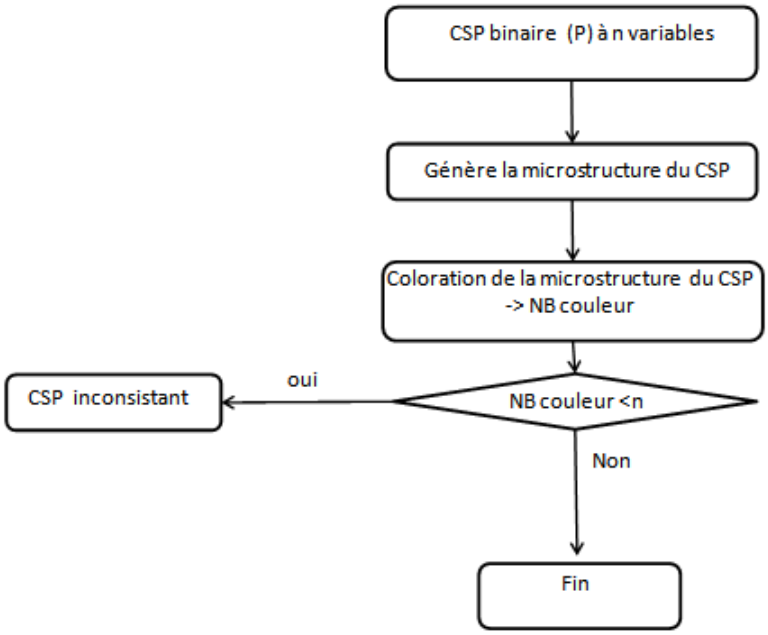


FIGURE 4.4 – L'organigramme de la méthode de Color

5.3 La méthode de Coloration-Dominance entre les CSPs

Définition 34 (Dominance de CSP)
Soit deux CSPs $P(n)$ et $P'(n)$ ayant n variables, s'il existe une application qui couple toute solution de $P(n)$ avec une solution de $P'(n)$, alors on dit que $P(n)$ est dominé par $P'(n)$ [2].

A partir de la définition précédente on définit l'interaction entre la dominance et la consistance de CSP dans la proposition suivante :

Proposition 1 Si un CSP ${P'}_n$ domine le CSP P_n et que ${P'}_n$ est inconsistant alors P_n est inconsistant [1].

Preuve voire l'article de *Saïdi* et Benhamou [2].

Définition 35 (Coloration de microstructure)
Soient le CSP $P(n)$ et sa microstructure $\mathcal{M}_{P_n} = (V \times D, \acute{E})$, une k-coloration de microstructure $\mathcal{M}_{P_n} = (V \times D, \acute{E})$ est une application f_k qui affecte a chaque sommet de $\mathcal{M}_{P_n} = (V \times D, \acute{E})$ une couleur représenté par un entier i tel que $1 \leq i \leq k$ et chaque sommets adjacents n'ayant pas la même couleur [2].

Définition 36 (Coloration de microstructure de CSP Alldiff)
Etant donné un CSP $P_n = (V, D, C, R)$, sa microstructure $\mathcal{M}_{P_n} = (V \times D, \acute{E})$ et $f_k : V \times D \longrightarrow K$ une k-coloration.
On peut toujours associer au CSP P_n et à une k-coloration f_k un CSP que l'on note Alldiff(P_n, f_k).
Le CSP Alldiff(P_n, f_k)= (Vk, Dk, Ck, Rk) associé à P_n et f_k définit par :

- $Vk = \{vk_1, vk_2, ..., vk_n\}$ (à chaque v_i est associé un vk_i)).
- $Dk = \{dk_1, dk_2, ..., dk_n\}$ (on construit chaque Dk_i par : $d_i \in D_i \Rightarrow f_k((v_i, d_i)) \in Dk_i$).
- $Ck = \{Alldifferent(vk_1, vk_2, ..., vk_n)\}$ (il y a une contrainte globale **Alldifferent**).
- $Rk = \{(d_1, d_2, ..., d_n) \in Rk$ si et seulement si $\forall i, j \in [1..n], [1..n] : d_i \neq d_j$ [1].

théorème 3 Étant donnés un CSP P_n et f_k une k-coloration de sa microstructure $\mathcal{M}_{p_n}$. Le CSP P_n est dominé par le CSP associé Alldiff(P_n, f_k) [1].
Preuve voir [1].

Le principe de l'algorithme de Coloration-Dominance

Le principe de l'algorithme consiste à traiter des CSPs binaires quelconques dans le but de détecter les instances inconsistantes par la notion de dominance et la coloration de la microstructure.
Au début on a un CSP binaire P quelconque avec n variables, l'algorithme commence par l'initialisation du nombre de tentative NBT à zéro et fixer le nombre maximum de tentative.
Après la génération de la microstructure, l'algorithme de coloration permet de colorier la microstructure et affecter un nombre de couleurs dans NB couleur et si le nombre de couleurs est inférieur au nombre de variables n alors l'algorithme détecte l'inconsistance du CSP, sinon il va générer Alldiff du CSP et il le résoudra.
La résolution peut donnée comme sorties trois possibilités soit :
- il filtre des valeurs du CSP.
- ne filtre rien.
- détecte l'inconsistance du CSP Alldiff.

Dans le cas où le CSP Alldiff est inconsistant alors le CSP binaire quelconque est inconsistant.
Dans le cas où il filtre des valeurs du CSP, il va mettre le nombre de tentatives à zéro et recommencer l'étape de génération de la microstructure.

Dans le cas où il ne filtre rien, il incrémente le nombre de tentatives et recommencer l'étape de teste (NB couleurs $< n$), dans le cas où le nombre de tentatives dépasse le nombre maximum de tentatives, l'algorithme s'arrête [2]. Nous avons représenté dans la Figure 4.5 un organigramme qui décrit les différentes étapes de la méthode représente.

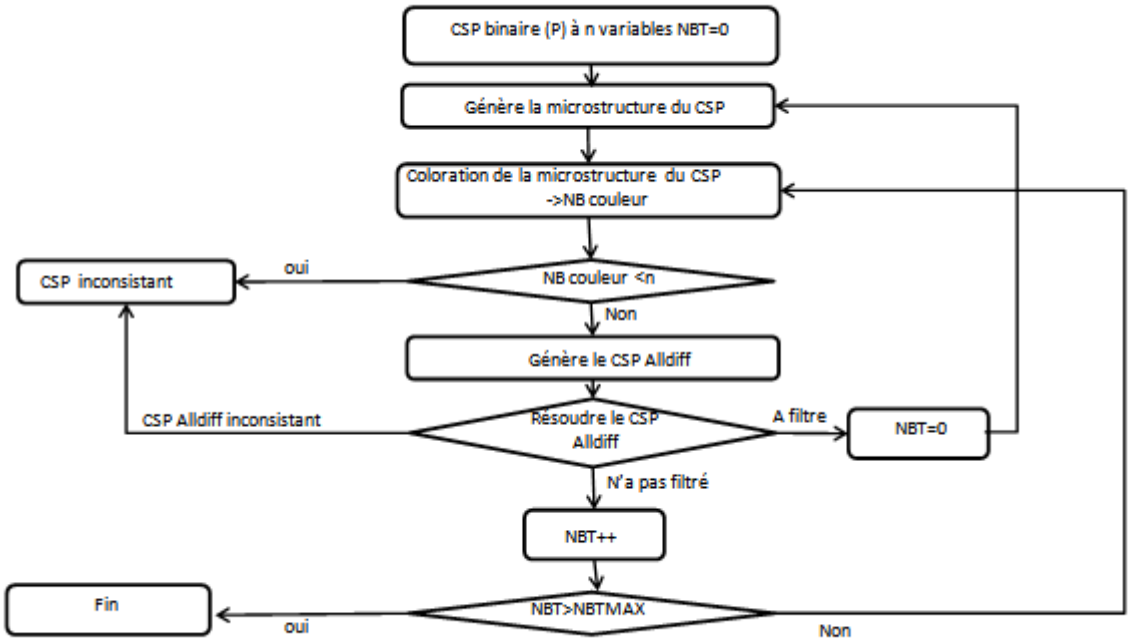


FIGURE 4.5 – L'organigramme de la méthode de Coloration-Dominance

5.4 L'amélioration de la méthode de Coloration-Dominance

Notre contribution consiste à rajouter à la méthode coloration-dominance l'arc consistance. L'algorithme traite un CSP binaire quelconque de n variables et initialise le nombre de tentatives NBT à zéro (le nombre maximum de tentative est fixé d'avance).
Au début il va incrémenter le nombre de tentative NBT et résoudre le CSP par l'arc consistance, sa résolution peut donner comme une sortie trois possibilités soit :
- il filtre des valeurs inconsistantes.
- ne filtre rien.
- il détecte l'inconsistance du CSP.

Dans le cas où le CSP n'est pas arc consistant alors le CSP binaire quelconque est inconsistant et s'il filtre des valeurs inconsistantes il met le nombre de tentative à zéro.
Dans les deux autres cas (s'il filtre des valeurs ou pas) il va générer la microstructure, cette dernière sera coloriée avec un nombre de couleurs (NBcouleur).
Dans le cas où le nombre de couleur est inférieur au nombre de variables n, l'algorithme détecte l'inconsistance du CSP, sinon il va générer le CSP Alldiff, sa résolution peut être donnée comme une sortie trois possibilités soit :
- il filtre des valeurs du CSP.
- ne filtre rien.
- détecte l'inconsistance du CSP Alldiff.

Dans le cas où le CSP Alldiff inconsistant alors le CSP binaire quelconque est inconsistant et s'il filtre des valeurs du CSP, il va initialiser NBT à zéro et filtre le CSP une deuxième fois par l'arc consistance.
Dans les deux autres cas s'il filtre des valeurs ou pas, il va régénérer la microstructure du CSP binaire et refait le même traitement que l'on a cité avant avec un teste d'arrêt, dans le cas où le nombre de tentative dépasse le nombre maximum de tentative l'algorithme s'arrête sans déterminer si le CSP est consistant ou non.
Nous avons expliqué dans la Figure 4.6 les différentes étapes de l'algorithme de la méthode $AC\_Alldiff\_Color$.

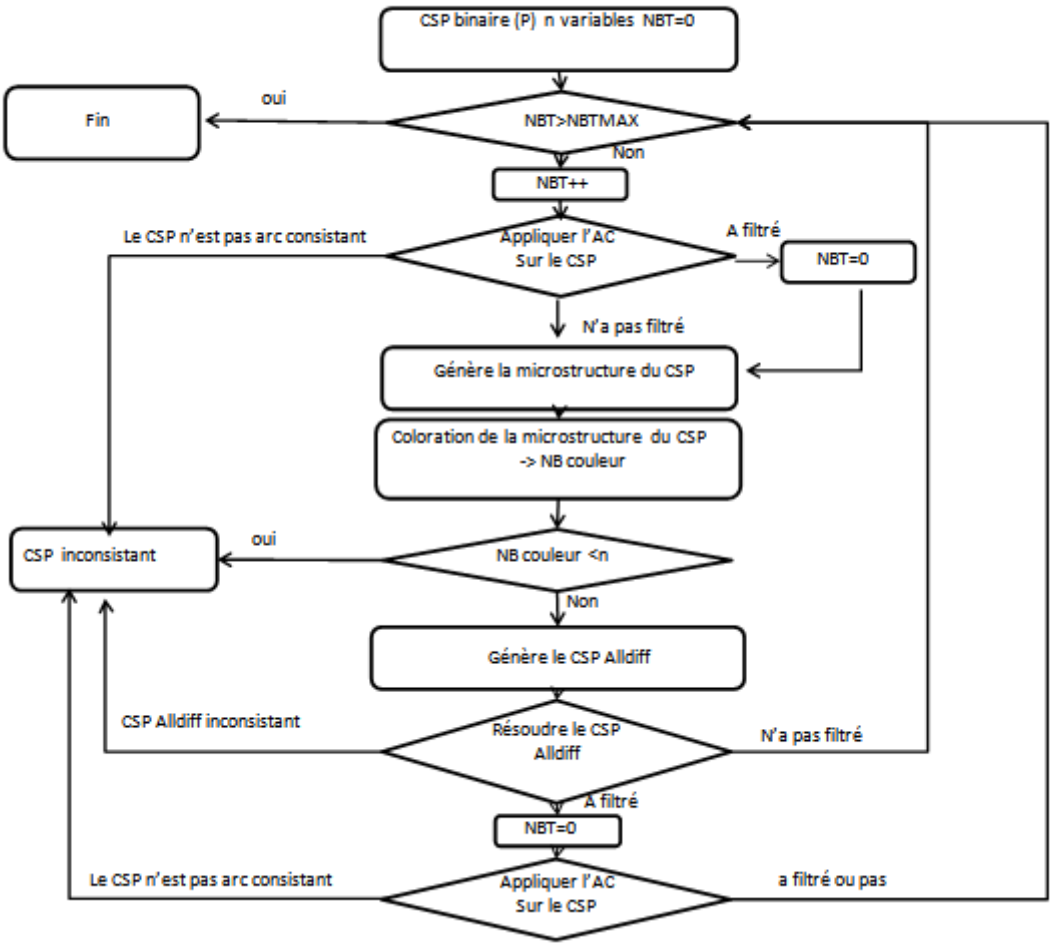


FIGURE 4.6 – L'organigramme de La méthode $AC\_Alldiff\_Color$

6 Conclusion

Dans ce Chapitre, nous avons défini les différentes classes de complexité et quelques définitions de base, ainsi nous avons représenté le concept général de l'optimisation des problèmes combinatoires, puis nous avons montré les différentes méthodes de preuve d'inconsistance afin de décrire le principe de chaque méthode.
Dans le Chapitre suivant nous allons décrire le principe de générateur des CSPs binaires aléatoires pour tester les méthodes proposées pour la preuve d'inconsistance des CSPs, ensuite nous allons évaluer les résultats obtenus par les expérimentations.

Chapitre 5

Expérimentations

Contents

1 Introduction

Dans les chapitres précédents nous avons représenté les concepts fondamentaux de problème de satisfaction de contrainte et la preuve d'inconsistance par les méthodes incomplètes. Dans ce chapitre nous allons utiliser notre amélioration et d'autres méthodes telles que la méthode de Gaur, la méthode de *Saïdi* et Benhamou, afin détecter les instances inconsistantes pour les CSPs binaires aléatoires et quelque instances du problème de coloration des reines.

Définition 37 (Densité)
On définit la densité de graphe d par le nombre des arrêtes dans un graphe, elle est exprimée par le ratio : le nombre de contraintes (le nombre des arrêtes) sur le nombre maximal de contraintes [1].

Définition 38(Dureté)
La dureté des contraintes est exprimée par le ratio : le nombre de couples non permis sur le nombre maximal de couples [1].

2 Les CSPs binaires aléatoires

Ce travail constitue en une étude préliminaire d'une nouvelle approche améliorée et quelque méthodes qu'on a cités, pour l'évaluation de la détection d'inconsistance au sens des CSPs binaires. On a programmé un générateur des CSPs binaires aléatoires.

Définition 39(CSP binaire aléatoire)
Un CSP binaire P est généré aléatoirement est un quadruplet P (n, d, p_1, $p2$) où :
- n est le nombre de variables.
- d le nombre de valeurs de chaque domaine.
- p_1 la probabilité qu'il y ait une contrainte entre deux variables.
- p_2 la probabilité conditionnelle qu'un couple de valeurs soit inconsistante pour une paire de variables, sachant qu'il y a une contrainte entre les couples variables [33].

2.1 Le générateur des CSPs binaires aléatoires

Le générateur de CSP binaire aléatoire nous permet d'avoir un jeu de données pertinent pour l'expérimentation, au début nous fixons certains paramètres :
- Nombre de variables n=20.
- Nombre de valeurs par domaine d=10.
- Densité du graphe de contrainte p_1 =$\{0.5, 0.9\}$.
- Dureté des contraintes : $0.01 \leq p_2 \leq 0.99$ avec un pas de 0.01.

Pour chaque valeur de n, d, p_1 et p_2 on a 100 instance aléatoire, par conséquence le total donné est 19800 CSP binaires aléatoire ($2 \times 99 \times 100$). L'organigramme représente dans la Figure 5.1 les étapes qui permettent de générer l'ensemble de tests des expérimentation (les CSPs binaires aléatoires).

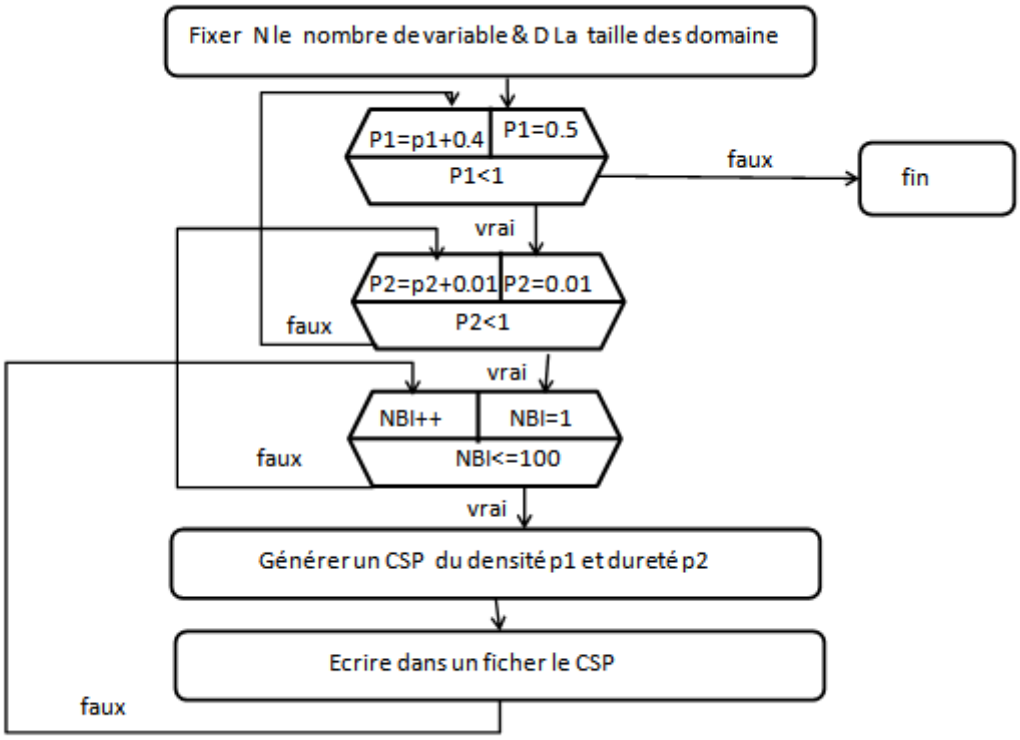


FIGURE 5.1 – L'organigramme de générateur l'ensemble de tests

L'organigramme décrit les étapes pour générer un CSP binaire aléatoire à partir des valeurs donné n, d, p_1, p_2 (le nombre des variables, la taille des domaines, la densité, la dureté).
Au début nous *commençons* par calculer le nombre des contraintes binaires NBC en fonction de la densité $p1$ par la formule : $NBC = p_1 \times (n \times \frac{(n-1)}{2})$ et le nombre des tuples non permis TNP à partir de la dureté p_2 par la formule : $TNP = p_2 \times d^2$.
Nous initialisons la matrice d'adjacence et la matrice de la microstructure à zéro. Ces deux matrices sont carrées, binaires et symétriques.
La matrice d'adjacence est de taille n^2 et la matrice de la microstructure est de taille $(n \times d)^2$.
Nous définissons les valeurs de ces matrice comme suit :
La matrice d'adjacence $[a][b] = 1 \Rightarrow$ il existe une relation entre a et b.
La matrice d'adjacence $[a][b] = 0 \Rightarrow$ il n'existe pas une relation entre a et b.
Nous avons $i \in D_a$, $j \in D_b$.
La matrice de la microstructure $[v_a \times card(d) + i][v_b \times card(d) + j] = 0$
$\Rightarrow (i, j)$ un tuple permis.
La matrice de la microstructure $[v_a \times card(d) + i][v_b \times card(d) + j] = 1$
$\Rightarrow (i, j)$ un tuple non permis.
Nous avons choisi aléatoirement deux variables a, b a partir de la fonction

$rand()$ de telle sorte que la valeur de $a \neq b$ et qu'il n'existe pas une relation entre ces deux variables (v_a, v_b), cette condition est de vérifier si la matrice d'adjacence $[a][b]$=0, nous mettons la matrice d'adjacence $[a][b]$=1, ça signifie qu'on a une contrainte C_{ab} entre ces deux variables $v_a$et v_b, pour ça il faut générer aléatoirement les tuples non permis en fonction de la dureté que nous avons déjà calculé (TNP), les valeurs de ces tuples (i,j) $i \in d_a$, $\in$ j d_b sont générées par la fonction $rand()$ (il faut que les tuples choisis soit déjà permis) et nous mettons la matrice la microstructure
$[v_a \times card(d) + i][v_b \times card(d) + j] = 1$.
Nous avons les valeurs aléatoires à partir de la fonction $rand()$ qui est une fonction connue dans la plupart des langages de programmation pour plus de détails nous vous invitons à voir [36].
Nous avons représenté dans la Figure 5.2 l'organigramme de génération d'un CSP.

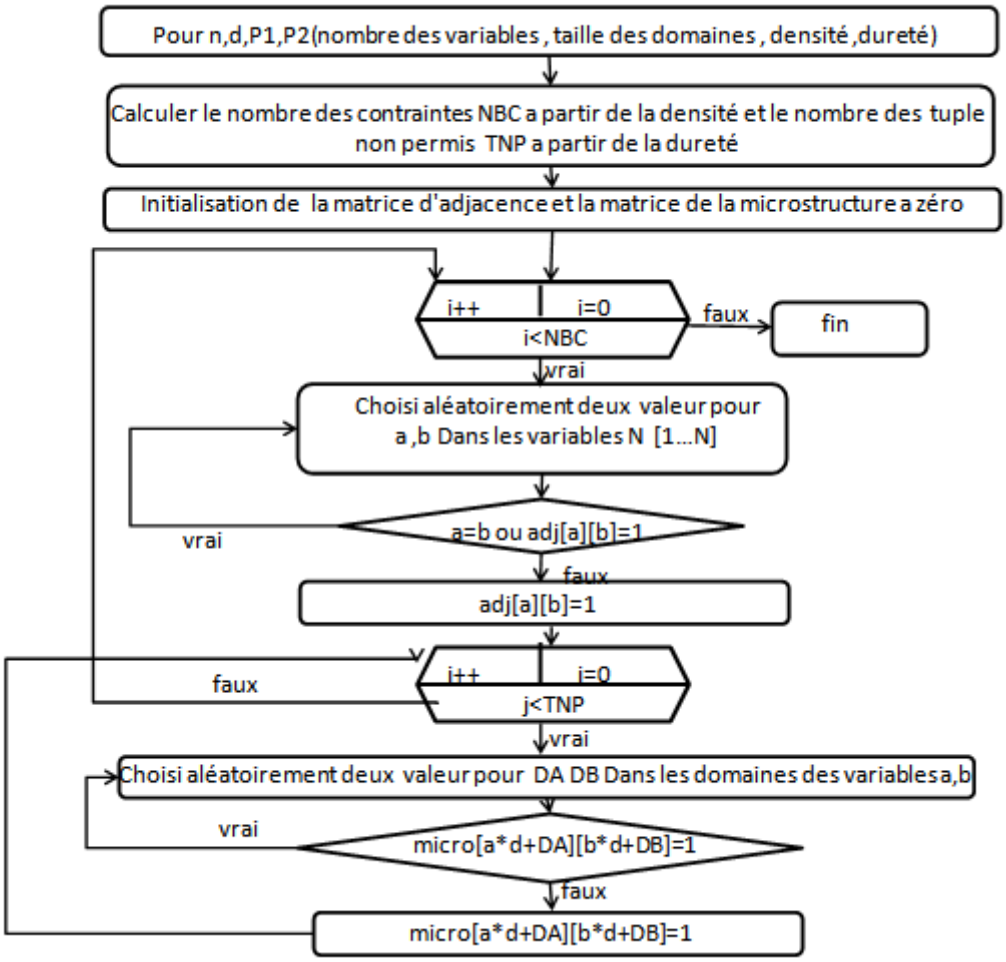


FIGURE 5.2 – L'organigramme de générateur de CSP

Nous avons présenté un exemple de CSP binaire aléatoire.

Exemple :

Soit $P = V; D; C; R)$ un CSP défini par :

— $V = \{v_1; v_2; v_3\}$.

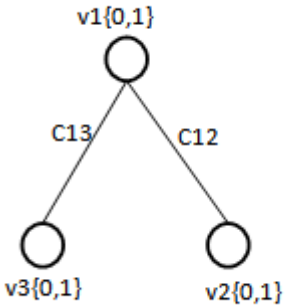
— $D = \{D_1 = D_2 = D_3 = 0, 1\}$.

— C= $\{C_{12}; C_{13}\}$.

— R= $\{R_{12} = (0, 0), (1, 0), (1, 1); R_{13} = (0, 1), (1, 0), (1, 1)\}$(les tuples permis).

Nous avons calculé la dureté et la densité de CSP où : n=3, d=2, p_1=0.66, p_2=0,25.

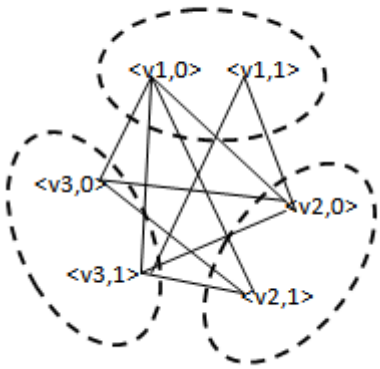
Nous montrons dans la Figure 5.3 le graphe de contrainte avec la matrice d'adjacence de ce CSP.



	v1	v2	v3
v1	0	1	1
v2	1	0	0
v3	1	0	0

FIGURE 5.3 – La représentation du graphe de contrainte avec la matrice d'adjacence

Nous montrons dans la Figure 5.4 la microstructure avec sa matrice de ce CSP.



		v1		v2		v3	
		<v1,0>	<v1,1>	<v2,0>	<v2,1>	<v3,0>	<v3,1>
v1	<v1,0>	0	0	0	0	0	0
	<v1,1>	0	0	1	0	0	1
v2	<v2,0>	0	1	0	0	0	0
	<v2,1>	0	0	0	0	0	0
v3	<v3,0>	0	0	0	0	0	0
	<v3,1>	0	1	0	0	0	0

FIGURE 5.4 – La représentation de la microstructure avec sa matrice

2.2 Les expérimentations sur les CSPs binaires aléatoires

On a choisi de tester les méthodes suivant :
- **AC :** la consistance d'arc (méthode de filtrage).
-**Color :** coloration de la micro structure (méthode incomplète de Gaur et Al en 1997).
- **Color_Alldiff :** Coloration-Dominance entre CSPs (méthode incomplète de *Saïdi* et Benhamou en 2008).
- **Color_AC_Alldif :** le mixage des trois méthodes Color, AC et Alldiff (notre contribution c'est une méthode incomplète).
-**FC :** Forward Checking (méthode complète) qui permet de recenser toutes les instances inconsistantes pour vérifier notre résultat.
Les Figures 5.5 et 5.6 montrent le nombre de problèmes inconsistants détectés pour chaque méthodes sur des problèmes aléatoires pour deux densités (0.5,0.9), nous avons choisi d'utiliser l'algorithme trick de coloration de graphe.
On remarque que les courbes des méthode $Color\_Alldiff$ et $Color\_AC\_Alldiff$ se confondent avec la courbe de la méthode FC, alors ces méthodes détectent toutes les instances inconsistantes.
On remarque que la méthode Color détecte moins d'instance inconsistance que la méthode AC.

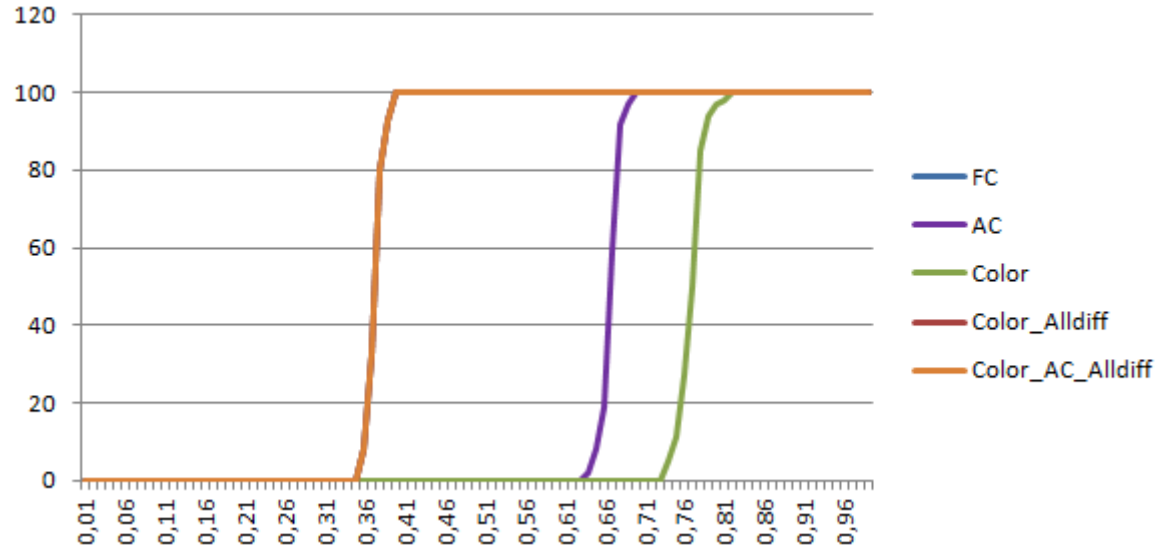


FIGURE 5.5 – Le nombre de problème inconsistants détectés en fonction de dureté avec (n=20, d=10, $density$=0.5)

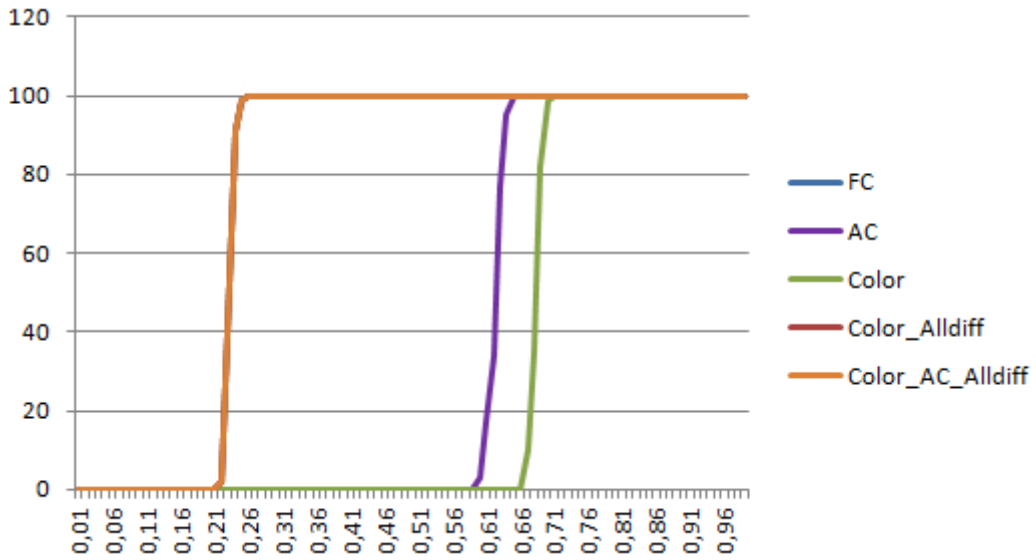


FIGURE 5.6 – Le nombre de problème inconsistants détectés en fonction de dureté avec (n=20, d=10, $density$=0.9)

3 Le problème des reines

Le travail présenté par *Saïdi* et Benhamou en 2008 a été concentré pour traiter que les problèmes aléatoires, dans cette section nous avons traité un autre type de problème qui est le problème de coloration des reines .
Le problème de coloration des reines est un problème difficile qui fait partie des problèmes les plus étudie dans l'intelligence artificielle. Dans cette section nous avons représenté le principe de problème des reines.

3.1 Le principe de problème des reines

Le problème consiste à placer n reines dans un échiquier de taille $n \times n$ dans le cas général mais il existe des cas où l'échiquier et de taille $n \times p$ par exemple le problème de 8 reines avec un échiquier de taille 8×12, de telle sorte qu'il n'y aura pas plus d'une reine par ligne, colonne, diagonale et anti-diagonale. Dans la Figure 5.7 nous représentons un exemple d'une solution du problème des 4 reines avec un échiquier de taille 4×4.

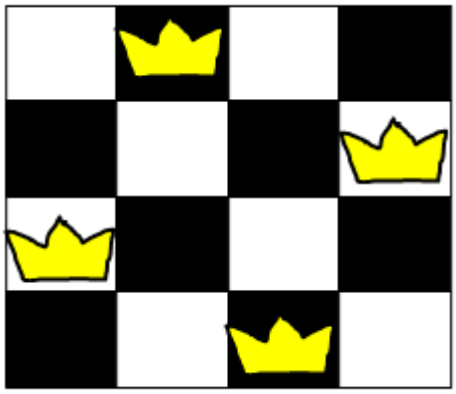


FIGURE 5.7 – La représentation de la solution du problème des 4 reines

Par la suite nous détaillons dans la Figure 5.8 comment présenter un sommet avec ses contraintes de sorte que les sommets contenus dans chaque cercle forment un sous graphe complet. Prenons par exemple le sommet Q_{22} on peut voir qu'il participe à 4 contraintes (représentées en cercle) CL, CC, CD, CAD contrainte sur la ligne, colonne, diagonale et anti-diagonale.

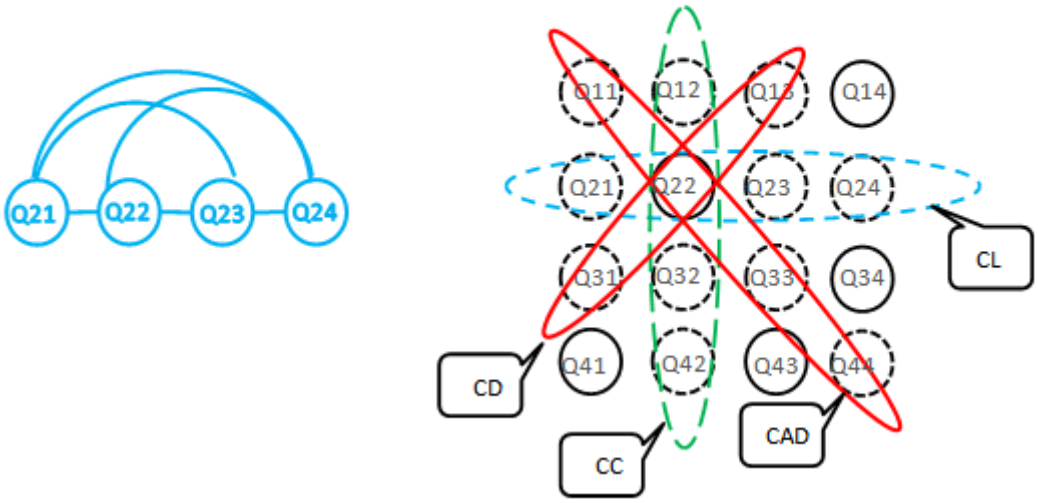


FIGURE 5.8 – La représentation graphique des contraintes du sommet $Q22$

3.2 Le problème de coloration des reines

Le problème de coloration des reines consiste a colorier le graphe de contrainte de problème des n reines dans la Figure 5.9 nous montrons toutes les contraintes du problème des 4 reines, chaque cercle représente une contrainte, ces cercles sont concrètement représentés dans le graphe.

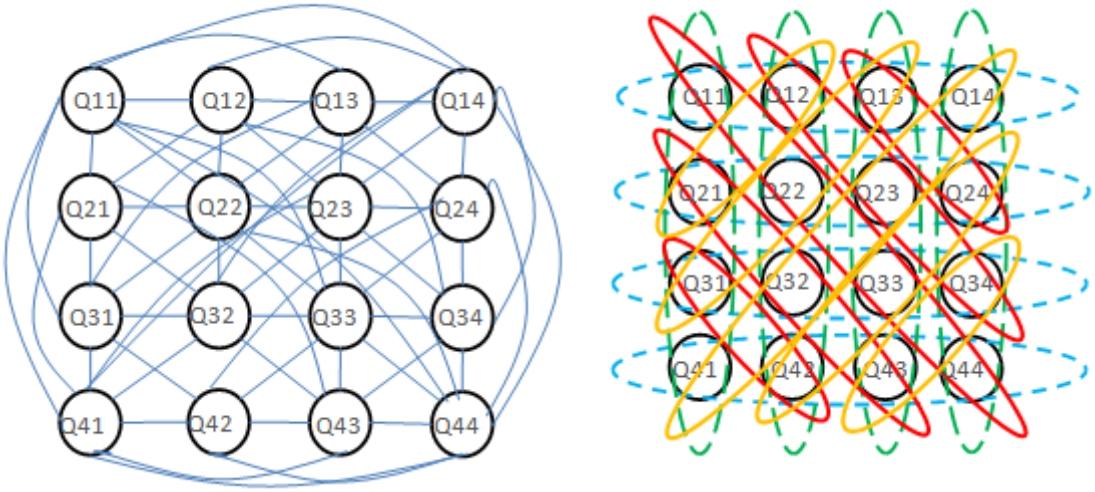


FIGURE 5.9 – La représentation graphique d'un échiquier de taille 4×4 avec les contraintes liées à chaque sommet

Nous commençons à transformer les problèmes des reines à des problèmes de coloration des problèmes des reines (problèmes de coloration du graphe de contraintes des problèmes reines CSP binaires) qui sont des problèmes de désistions .
En utilisant l'algorithme Dimacs2CSP qui a comme entrée un problème des reines sous la forme dimacs, un nombre des couleurs(k) qui permet de formuler le problème de discision coloration du graphe de contrainte de problème des reines avec k couleur.
Dans le cas où le nombre de couleurs est supérieur ou égale au nombre chromatique alors le CSP binaire est consistant, sinon il est inconsistant.
Nous allons montrer dans la Figure 5.10 le résultat de coloration de problème des reines par l'algorithme Trick qui est stocké dans un fichier nommé solution. La matrice représente la coloration des sommets qui est dans le fichier solution.

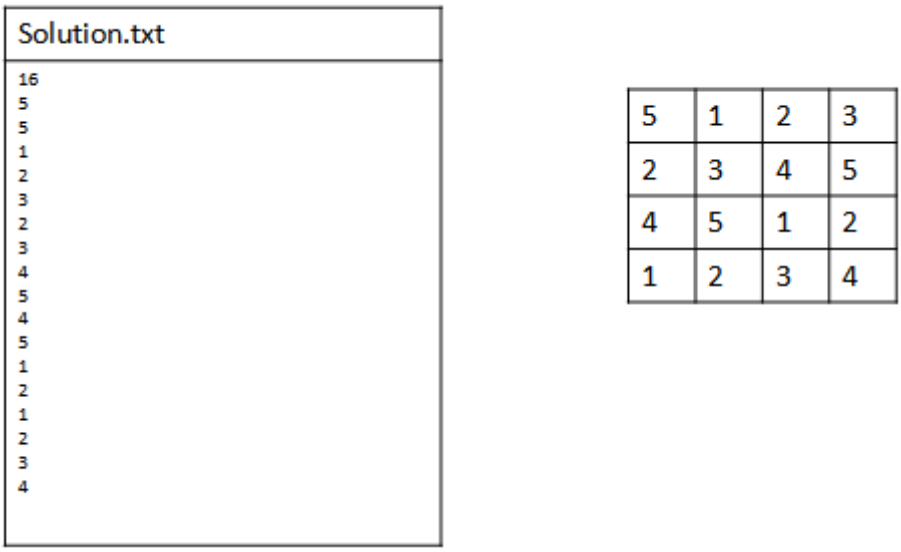


FIGURE 5.10 – La représentation de fichier solution avec la matrice de coloration

Nous allons représenté dans la Figure 5.11 la coloration du graphe de contrainte où chaque nombre représente une couleur .

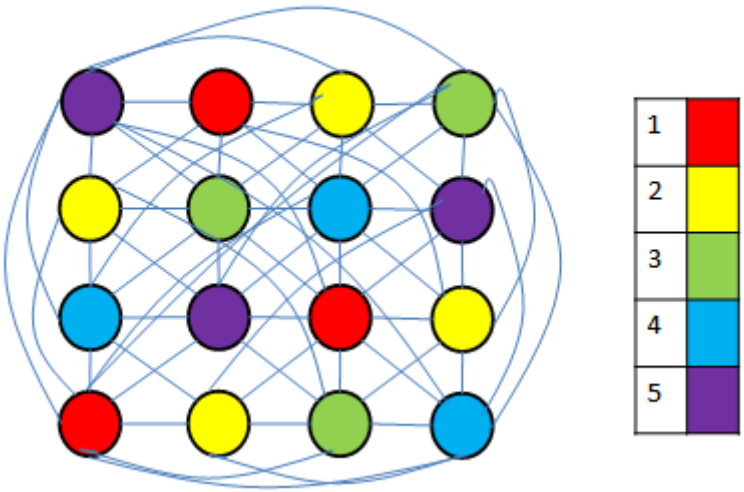


FIGURE 5.11 – La représentation de coloration du graphe de contrainte de problème de quatre reines

Exemple :
Pour vérifier que le résultat (cinq couleurs) qu'on a obtenu par la coloration de graphe de contrainte du problème des 4 reines avec l'algorithme trick soit optimal, il faux résoudre le CSP binaire du problème des 4 reines et colorié par 5 couleurs (qu'il doit être consistant), le CSP binaire du problème des 4 reines est colorié par 4 couleurs (qu'il doit être inconsistant).
On commence par générer les CSP binaires à partir de l'algorithme Dimac2csp qui nous donne comme résultats deux fichiers (*queen*4_4.*col*_5.*csp*)et (*queen*4_4.*col*_4.*csp*). On a choisi la méthode FC pour résoudre les deux CSP, les résultats sont remplis dans un tableau qui est représenté dans la Table5.1.

reines	**FC**
4-4 avec 4 couleurs	inconsistant
4-4 avec 5 couleurs	consistant

TABLE 5.1 – La représentation des résultats de la résolution des CSP binaires par FC

On conclu par les résultats présentés dans la Table5.1 que le nombre de couleur 5 est optimale pour colorier le graphe de contrainte.

3.3 Les expérimentations sur les problèmes de coloration des reines

Nous avons choisi le problème de coloration des reines pour tester les méthodes de l'expérimentation, nous nous intéressons pour le moment d'évaluer la détection des instances inconsistantes (la valeur un représente la détection de l'inconsistance, la valeur zéro signifié qu'il n' a pas détecté l'inconsistance).
Les résultats des problèmes sont remplis dans un tableau de telle sorte que chacun de ces problèmes sont testés sur quatre méthodes :
AC : la consistance d'arc (méthode de filtrage).
Color : coloration de la micro structure (méthode incomplète de Gaur et Al en 1997).
Color_Alldiff : mixage de méthode de la dominance avec la méthode de coloration sur des CSP à contrainte de différence (méthode incomplète de *Saïdi* et Benhamou en 2008).
Color_AC_Alldiff : le mixage des trois méthodes Color, AC et Alldiff (notre contribution c'est une méthode incomplète).
FC (ForwardChecking) : (méthode complète) qui permet de recenser toutes les instances inconsistantes pour vérifier notre résultat.
Nous allons montrer dans la Table 5.2 les résultats de détectés d'inconsistance pour les problèmes de coloration des problèmes des reines.
On conclu que la méthode expérimentale à détecter l'inconsistance des problèmes généré avec $k-1$ coloration (k :représente le nombre chromatique de graphe de contrainte).

Les reines	FC	AC	Color	Color-Alldiff	Color-AC-Alldiff
4-4 avec 4 couleurs	1	0	1	1	1
5-5 avec 4 couleurs	1	0	1	1	1
6-6 avec 6 couleurs	1	0	1	1	1
7-7 avec 7 couleurs	1	0	1	1	1

TABLE 5.2 – La représentation des résultats de problème de coloration des problèmes des reines

4 Conclusion

Dans ce chapitre nous avons testé notre amélioration et d'autres méthodes sur des CSPs binaires aléatoires que nous avons générer par le générateur des CSPs binaires aléatoires.
Les résultats des expérimentations a montré que notre méthode présente des résultats similaires au résultats de la méthode de *Saïdi* et Benhamou de 2008, mais on espérait quelle donne de meilleure résultats dans le cas où les problèmes ont une grande tailles car cela implique que leur microstructure associée est très grande ce que serait un handicap pour la coloration (risque plus de ne pas trouver des colorations pertinentes), alors que grâce à l'ajout de la consistance d'arc il y a toujours une chance qu'elle pouvait filtrer des valeurs soit de montrer l'inconsistance. Donc l'ajout de la consistance d'arc à la méthode de 2008 ne peut être que bénéfique et le coût supplémentaire est relativement faible.

Chapitre 6

Conclusion

De nombreux problèmes peuvent être exprimer avec le formalisme CSP et résolus grâce à des techniques qui peuvent être divisées en deux catégories :
- la premier est basée sur un parcours total de l'espace de recherche (méthodes complètes).
- la deuxième basée sur un parcours local d'espace de recherche (méthodes incomplètes), ça induit que les méthodes complètes peuvent détecter l'inconsistance d'un CSP et que les méthodes incomplètes sont incapables de le prouver.

Dans ce travail nous avons amélioré la méthode de Benhamou et Saïdi de 2008 pour la détection d'inconsistances des CSPs binaires.
Cette amélioration est basée sur un filtrage d'arc consistance, la méthode de coloration de la microstructure et la dominance.
De sorte que les propriété de filtrage par l'arc consistance est comme suit :
- simplification du problème par réduction de l'espace de recherche.
- dans certains cas détection de l'inconsistance.

Nous avons aussi confirmé les résultats de Saïdi et Benhamaou de 2008 et nous avons montre que la méthode de dominance-coloration et notre méthode peuvent aider à la résolution des problèmes importants en IA qui est le problème de la coloration des reines, ce qui implique que ces méthodes peuvent potentiellement avoir des applications pratiques dans plusieurs domaines concrets.
Dans le cas où la taille des instances augmente nous espérons que notre amélioration aura de meilleurs résultats que celle de 2008.
Finalement nous avons montré que notre méthode qui est incomplète est capable de détecter les instances inconsistantes .
Dans le cadre de la réalisation de notre mémoire, nous nous sommes initiés la recherche dans la programmation par contrainte. Nous avons pu apprendre à analyser, déduire et avoir un esprit d'observation et nous a permis d'être plus

motivée et encouragée de continuer ce travail afin d'envisager au futur d'autres types d'algorithmes de preuve d'inconsistance et avoire une machine de calcule plus puissante, il sera intéressant de tester des instances de taille plus importantes.

Liste des Abréviations

AC Arc Consistency

BJ Back Jumping

BTD Backtracking with Tree Decomposition

CAD Contrainte sur la Anti-Diagonale

CBJ Conflic-Directed- Back Jumping

CC Contrainte sur la Colonne

CD Contrainte sur la Diagonale

CL Contrainte sur la Ligne

CP Constraint Programming

CSP Construct Satisfaction Problem

FC-CBJ ForwardChecking awith CBJ

FC ForwardChecking

GeT Générer Et Tester

LS Local Search

MAC Maintaining Arc Consistency

MTIS Maximal Triangulated Induced Subgraph

NBC Nombre des contraintes binaires

NP Non-déterministe Polynomial

OLNE Optimisation Linéaire en Nombres Entiers

PMO Optimisation Multi-Objectif

PPC Programmation Par Contrainte

P Polynomial

RLF Recursive Largest First

TNP Nombre des tuples non permis

Bibliography

[1] Mohamed Réda Saïdi. *Etude de la symétrie et de la dominance dans les réseaux de contraintes au sens CSPs [Constraint satisfaction problems] à domaines finis discrets*. PhD thesis, Aix-Marseille 1, 2007.

[2] Mohamed Saïdi Belaïd Benhamou. Une nouvelle approche pour le test d'inconsistance de csp. *JFPC 2008- Quatrièmes Journées Francophones de Programmation par Contraintes, LINA - Université de Nantes - Ecole des Mines de Nantes Nantes, France. pp.201-208*, Jun 2008.

[3] Bouazza Marouf Khadidja and Boussaid Nouha. Symétries et dominances dans les réseaux de contraintes de différence. Master's thesis, Centre Universitaire Belhadj Bouchaib d'Aïin-Témouchent, 2019.

[4] Mohammed Lalou. Décompositions arborescentes pour résoudre les problèmes de satisfaction de contraintes avec mise en ouvre du parllélisme. Master's thesis, Université de Béjaia, 2009.

[5] Saida Hammoujan. *Raisonnement par Contraintes: Approches de Satisfaction de Contraintes Dynamiques et Distribuées*. PhD thesis, Université Mohammed V, Faculté des sciences-Rabat, 2016.

[6] Achref El Mouelhi, Philippe Jégou, and Cyril Terrioux. Microstructures pour csp d'arité quelconque. *Actes des*, pages 193–202.

[7] Nicolas Paris. *Intégration de techniques CSP pour la résolution du problème WCSP*. PhD thesis, Artois, 2014.

[8] Mihaela Butaru and Zineb Habbas. Problème de satisfaction de contraintes n-aires: une étude expérimentale. Premières Journées Francophones de Programmation par Contraintes, CRIL - CNRS FRE 2499, Jun 2005, Lens, pp.435-438.

[9] Daniel Cosmin Porumbel. *Algorithmes Heuristiques et Techniques d'Apprentissage-Applications au Probleme de Coloration de Graphe*. PhD thesis, 2009.

[10] Kaouther Drira. *Coloration d'arêtes -distance et clustering: études et*

algorithmes auto-stabilisants. PhD thesis, Université Claude Bernard-Lyon I, 2010.

[11] A Bondy. Proposition de sujet de thèse en théorie des graphes. page 706.

[12] Daniela Bronner. Colorations suboptimales. Technical report, 2005.

[13] Benjamin Lévêque. *Coloration de graphes: structures et algorithmes*. PhD thesis, 2007.

[14] Benjamin Weinberg. *Analyse et résolution approchée de problèmes d'optimisation combinatoire: application au problème de coloration de graphe*. PhD thesis, Lille 1, 2004.

[15] Mouhamed Mourchid Adio Adegbindin. *Un algorithme constructif efficace pour le problème de coloration de graphe*. PhD thesis, École polytechnique de Montréal, 2013.

[16] Eglantine Camby. Différents problèmes en théorie des graphes. *Notes de la quatrième BSSM*, 2012.

[17] Tounsi Abdelkader. *Une approche mathématique pour résoudre le problème d'affectation de fréquences en réseau mobile*. PhD thesis, 2016.

[18] Asma Karray. *Contribution à l'ordonnancement d'ateliers agroalimentaires utilisant des méthodes d'optimisation hybrides*. PhD thesis, Ecole centrale de Lille, 2011.

[19] Ilyess Bachiri. Résolution des problèmes d'optimisation combinatoire avec une stratégie de retour-arrière basée sur l'apprentissage par renforcement. 2015.

[20] Hanaa Hachimi. *Hybridations d'algorithmes métaheuristiques en optimisation globale et leurs applications*. PhD thesis, 2013.

[21] Troudi Fatiha. Résolution du problème de l'emploi du temps: proposition d'un algorithme évolutionnaire multiobjectif. *Université Mentouri-Constantine*, 2006.

[22] Samir Mahdi. Optimisation multiobjectif par un nouveau schéma de coopération méta/exacte. 2007.

[23] Karima Mahdi. *L'optimisation multiobjectif et l'informatique quantique*. PhD thesis, Master's thesis, Université Constantine 1, 2006.

[24] Slimane Amira, Bentabet and Benghelima Charafeddine. L'optimisation du déploiement d'un rcsf pour une application de surveillance de sites. Master's thesis, 2019.

[25] Hind Raihani. Adaptation de l'algorithme des colonies d'abeilles pour l'optimisation et le dimensionnement des circuits intégrés analogiques. Master's thesis.

[26] Larachiche Imene and Remadelia Amina. Algorithme branch and bound appliquée au problème de sac à dos. 2018.

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

[28] Lagniez Jean-Marie. *Falsifiabilité propositionnelle et raisonnement par contraintes : modèles et algorithmes*. PhD thesis, faculté Jean Perrin, 2011.

[29] Charlotte Truchet. Contraintes, recherche locale et composition assistée par ordinateur. *Unpublished doctoral dissertation, Université Paris*, 2004.

[30] Asmaa Ghoumari. *Métaheuristiques adaptatives d'optimisation continue basées sur des méthodes d'apprentissage*. PhD thesis, Université Paris-Est, 2018.

[31] Maryam Booshehrian, Torsten Möller, Randall M Peterman, and Tamara Munzner. Vismon: facilitating risk assessment and decision making in fisheries management. Technical report, Tech. Rep. TR 2011-05. School of Computing Science, Simon Fraser University . . . , 2011.

[32] Pierre Berlandier. Filtrage de problemes par consistance de chemin restreinte. *Revue d'intelligence artificielle*, 9(1):225–238, 1995.

[33] Maria-Cristina Riff-Rojas. *Résolution de problemes de satisfaction de contraintes avec des algorithmes évolutionnistes*. PhD thesis, 1997.

[34] David Cohen. *Principles and Practice of Constraint Programming-CP 2010: 16th International Conference, CP 2010, St. Andrews, Scotland, September 6-10, 2010, Proceedings*, volume 6308. Springer, 2010.

[35] Anna Lungarska. *Integrated modelization of land use in France: from the crop choice to the choice of economic sector*. PhD thesis, AgroParisTech, 2015.

[36] Mathieu Nebra. *Apprenez à programmer en C*. OpenClasssrooms, 2015.

[37] Jocelyn Bernard and Hamida Seba. Résolution de problèmes de cliques dans les grands graphes. 2015.