N° d'ordre : 06/2014-M/INF

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'enseignement supérieur et de la recherche Scientifique
Université des sciences et de la Technologie Houari Boumediène

Faculté de l'Electronique et d'Informatique

MEMOIRE

Présenté pour l'obtention du **diplôme** de **MAGISTER**

En : INFORMATIQUE

Spécialité : Systèmes Intelligents et Ingénierie des Logiciels

Par : Sabrina BOUCHENE

Sujet

Raisonnement par symétrie dans les problèmes combinatoires : application au problème de compétition sportive (Circular Traveling Tournament Problem)

Soutenu publiquement, le 24/12/2014, devant le jury composé de :

M	Meziane AIDER	Professeur	à l'USTHB	Président
Mme	Dalila BOUGHACI	Maître de conférences/A	à l'USTHB	Directrice de mémoire
M	Samir KECHID	Maître de conférences/A	à l'USTHB	Examinateur
M	Amar ISLI	Maître de conférences/B	à l'USTHB	Invité

Résumé

Mots clés : les problèmes d'optimisation combinatoires, raisonnement par symétrie dans la programmation logique, Détection et exploitation de la symétrie dans la programmation par contraintes, compétition sportive, ligue professionnelle, circular traveling tournament problem, instances circulaires.

Le travail de ce projet de magister porte sur l'élimination des symétries pour la résolution efficace de problèmes combinatoires. L'élimination des symétries d'un problème permet lors de la recherche d'une solution à ce problème, d'éviter d'explorer des sous espaces redondants, ce qui permettra d'améliorer l'efficacité de la méthode d'exploration de l'espace de recherche. Le problème de planification de compétition sportive consiste principalement à déterminer la date et le lieu dans lequel chaque match d'un tournoi sera joué. Dans ce projet nous nous intéressons au problème C-TTP (*Circular – Traveling Tournament Problem*). Le TTP est un problème crucial en compétitions sportives dont l'objectif consiste à minimiser la distance totale parcourue par toutes les équipes participant au tournoi. Le C-TTP est un cas particulier de TTP ou les lieux des équipes sont situés sur un cercle. La distance entre les lieux voisins est égale à un. Le C-TTP est un problème très difficile à résoudre en raison de sa structure symétrique. Pour la résolution efficace de ce problème, nous avons choisi d'implémenter une méthode de recherche énumérative dite Branch and Bound à laquelle nous avons intégré une méthode de suppression partielle de symétrie et dynamique dite LDSB. La combinaison des deux méthodes à améliorer significativement le temps d'exécution en supprimant certains schémas de symétries du problème considéré et les résultats ont montré l'intérêt d'exploiter la structure symétrique du problème pour améliorer son exécution.

Remerciements

Je tiens à remercier Monsieur le professeur Meziane AIDER pour l'honneur qu'il me fait en présidant le jury de mon mémoire.

J'adresse mes sincères remerciements à Messieurs les Docteurs Samir KECHID et Amar ISLI, d'avoir en dépit d'une intense activité, bien accepté de participer au jury de soutenance.

Je voudrais également exprimer mes remerciements à Madame le docteur Dalila BOUGHACI, qui, en tant que directrice de mémoire, s'est toujours montrée à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'elle a bien voulu me consacrer et sans qui ce mémoire n'aurait jamais vu le jour.

Je n'oublie pas à remercier tous les membres de ma famille pour leur contribution, leur soutien et leur patience.

Enfin, j'adresse mes plus sincères remerciements à tous mes proches et amies, qui m'ont toujours soutenu et encouragé au cours de la réalisation de ce mémoire.

Dédicaces

À mes parents

À mes frères et à ma sœur et mes belles sœurs et les petits enfants de la famille

À mon oncle Djafar et mon cousin Djamel

À tous ceux qui m'aiment

Table des matières

Table des Figures

Table des Tableaux

Introduction générale

- **Contexte du travail**

L'objet de ce mémoire est de contribuer à la résolution d'un problème parmi les problèmes combinatoires les plus connus.

Mais d'abord à quoi reconnaıt-on de tels problèmes combinatoires ? L'analyse combinatoire est rattachée aux mathématiques et étudie les collections (souvent finies) d'objets qui respectent certaines propriétés. Dénombrer toutes ces collections ou exhiber la meilleure collection parmi toutes celles possibles, sont des questions combinatoires. Euler, par exemple, pose la question suivante en 1779 : en considérant 6 régiments de 6 officiers de rangs distincts, est-il possible de placer les 36 officiers dans un carré de 6 par 6 de telle sorte qu'il n'y ait pas deux officiers de même rang ou de même régiment dans chaque ligne et colonne ? Le problème est simple à exprimer (sa compréhension ne demande aucune connaissance scientifique particulière) comme le célèbre jeu du Sudoku, et sa difficulté vient du trop grand nombre de combinaisons à évaluer pour pouvoir d´déterminer la bonne. Il s'agit de limiter autant que possible cette explosion combinatoire, mais même les problèmes les plus simples sont au-delà des capacités humaines. L'ordinateur est donc un outil indispensable de la résolution. En effet, autant de problématiques qui exigent aujourd'hui de bonnes réponses car elles se posent à des échelles mondiales. La gestion et l'amélioration de ces situations complexes et réelles sont le propre de la Recherche Opérationnelle et de l'intelligence artificielle. Les gens de ce domaine considèrent la réduction de l'explosion combinatoire et l'accélération de la résolution et la minimisation de l'espace de recherche comme un vrai challenge. Donc afin de remédier à ces problèmes-là, plusieurs techniques ont été proposées comme les techniques de filtrages, des paradigmes de programmation comme la programmation par contraintes, les méthodes heuristiques de recherche, les symétries pour certaines classes de problèmes combinatoires …etc. dans ce mémoire nous nous intéressons à l'étude de cette dernière propriété.

La symétrie est une propriété fondamentale qui peut être utilisée pour réduire la complexité de calcul et résoudre efficacement les problèmes combinatoires qui sont

de classe NP-complet. C'est un domaine de recherche prometteur en logique propositionnelle et en programmation par contraintes et qui a connu un succès grandissant ces dernières années. En fait, l'élimination des symétries du problème permet lors de la recherche d'une solution à ce problème, d'éviter d'explorer des sous espaces redondants, ce qui permettra d'améliorer l'efficacité de la méthode d'exploration de l'espace de recherche.

Plusieurs méthodes ont été proposées pour détecter et éliminer les symétries surtout dans le cadre de la programmation par contraintes, en effet, La plupart des applications de la programmation par contraintes sont des problèmes hautement symétriques. La présence de ces symétries dans les problèmes de satisfaction de contraintes augmente de manière significative la difficulté de résolution de ceux-ci. Il n'est donc pas surprenant de remarquer que de nombreuses recherches dans le domaine de la programmation par contraintes se sont focalisées sur la prise en charge des symétries ces dernières années.

Parmi ces travaux, nous citons la reformulation du problème, en effet la modélisation que nous donnons a un problème peut ajouter des symétries a celui-ci et le but est consisté donc a trouver un modèle qui n'admet pas ou admet peu de symétries, nous citons aussi les méthodes dynamiques de suppression de symétries (adaptation d'algorithmes de recherche) comme les approches à contraintes additionnelles (SBDS) et les approches de détection de dominance (SBDD), les approches de suppression de symétries statiques (ajout de contraintes au modèle pour empêcher d'explorer des solutions symétriques) et les approches heuristiques de recherche locale. D'autres méthodes de suppression de symétries structurelles qui étudient la structure combinatoire des problèmes, et d'autres travaux sur l'exploitation des symétries internes aux solutions.

Nous pouvons bien, imaginer le cout gagné en supprimant les symétries de certains problèmes ayant un grand impact sur l'économie et cela sur l'échelle mondial en améliorant la résolution de celle-ci et à titre d'exemple nous citons le problème de planification de compétition sportive qui consiste principalement à déterminer la date et le lieu dans lequel chaque match d'un tournoi sera joué. Dans ce projet nous nous intéressons au problème C-TTP (*Circular – Traveling Tournament Problem*). Le TTP est un problème crucial en compétitions sportives dont l'objectif consiste à minimiser

la distance totale parcourue par toutes les équipes participant au tournoi. Le C-TTP est un cas particulier de TTP ou les lieux des équipes sont situés sur un cercle. La distance entre les lieux voisins est égale à un. Le C-TTP est un problème très difficile à résoudre en raison de sa structure symétrique. L'objet de ce projet de Magister consiste à étudier le problème C-TTP et à proposer par la suite une nouvelle approche basée sur l'élimination des symétries pour la résolution efficace de ce problème.

- **Organisation du mémoire**

Ce mémoire est organisé de la façon suivante. Le chapitre 1 est une introduction aux problèmes combinatoires, une deuxième partie ou nous y présentons le principe de la symétrie et l'exploitation de celle-ci dans différentes domaines de la recherche opérationnelle et de l'intelligence artificielle, dont l'objectif est d'améliorer et optimiser la résolution de certains problèmes combinatoires. Cette partie donc qui représente le noyau de notre étude est divisée en deux chapitres, le chapitre 2 décrit la première apparition de la symétrie dans la programmation logique (nous présentons ici l'exploitation de la symétrie dans les différentes classes de la logique classiques, monotones, …etc.), le chapitre 3 présente le principe de la programmation par contraintes ainsi que les outils nécessaires à la modélisation et la résolution d'un problème. Puis nous donnons un exemple décrivant le principe de la symétrie dans un problème, et nous donnons également les définitions différentes des différents types de symétries ainsi que les bases de la théorie du groupe pour la construction des classes d'équivalences des symétries nous y présentons notamment les différents approches pour l'exploitation de la symétrie dans la programmation par contraintes ou nous distinguons deux axes de recherche dans ce domaine : l'exploitation des symétries entre les solutions du problème et l'exploitation des symétries internes aux solutions du problème. Ici nous présentons les grandes approches de suppression de symétries statiques (tel que les méthodes d'ordre lexicographiques, …etc.) et dynamiques (comme SBDS, SBDD, …etc.), par la suite nous présentons les approches de suppression de symétries dites structurelles qui étudient la structure des problèmes combinatoires, la dernière étude que nous présentons dans ce chapitre c'est l'exploitation des symétries dans les systèmes distribués. Dans ce chapitre nous discutons au fur et aux mesures les avantages et les inconvénients des différentes approches. Le chapitre 4 présente la gestion des tournois sportifs comme une introduction au problème combinatoire qu'on va étudier par la suite. Enfin, dans le

chapitre 5 nous présentons le problème combinatoire qui nous intéresse (Circular traveling tournament probleme) un cas particulier du problème TTP. Puis, nous étudions et détaillons dans un premier temps les symétries du problème, par la suite et après l'étude et l'évaluation des différents approches de traitement des symétries présentés dans l'état de l'art (les premiers chapitres), nous présentons la méthode que nous jugeons la plus adéquate et la plus efficace pour améliorer la résolution de notre problème. Donc nous considérons une méthode générale pour la recherche de la solution optimale du problème à laquelle nous ajouton le traitement des symétries afin d'éviter d'explorer les solutions symétrique et par conséquent optimiser la résolution du problème. Le chapitre 6 présente les résultats obtenus avec notre implémentation.

1 Généralités sur les problèmes combinatoires

1.1 Introduction

On qualifie généralement de « combinatoires » les problèmes dont la résolution se heurte à une explosion du nombre de combinaisons à explorer. C'est le cas par exemple lorsque l'on cherche à concevoir un emploi du temps : s'il y a peu de cours à planifier, le nombre de combinaisons à explorer est faible et le problème sera très rapidement résolu ; cependant, l'ajout de quelques cours seulement peut augmenter considérablement le nombre de combinaisons à explorer de sorte que le temps de résolution devient excessivement long.

Cette notion de problème combinatoire est formellement caractérisée par la théorie de la complexité qui propose une classification des problèmes en fonction de la complexité de leur résolution, et nous introduisons tout d'abord les différentes classes de problèmes combinatoires, nous introduisons ensuite les notions de transition de phase et de paysage de recherche qui permettent de caractériser la difficulté « en pratique » des différentes instances d'un même problème combinatoire.

1.2 Rappels sur les problèmes combinatoires : Notion de théorie de la complexité

1.2.1 Complexité théorique d'un problème combinatoire

On entend ici par « complexité d'un problème » le comportement asymptotique du nombre d'instructions à exécuter pour résoudre les instances de ce problème, cette estimation étant un ordre de grandeur par rapport à la taille de l'instance. Il s'agit là d'une estimation dans le pire des cas dans le sens où la complexité d'un problème est définie en considérant son instance la plus difficile. Les travaux théoriques dans ce domaine ont permis d'identifier différentes classes de problèmes en fonction de la complexité de leur résolution [1]. Il existe en fait un très grand nombre de classes différentes1, et on se limitera ici à une présentation succincte nous permettant de caractériser formellement la notion de problème combinatoire.

- **Problèmes de décision**. Les classes de complexité ont été introduites pour les problèmes de décision, c'est-à-dire les problèmes posant une

question dont la réponse est « oui » ou « non ». Pour ces problèmes, on définit notamment les deux classes P et NP.

- La classe P contient l'ensemble des problèmes polynomiaux, i.e., pouvant être résolus par un algorithme de complexité polynomiale. Cette classe caractérise l'ensemble des problèmes que l'on peut résoudre « efficacement ».
- La classe NP contient l'ensemble des problèmes polynomiaux non déterministes, i.e., pouvant être résolus par un algorithme de complexité polynomiale pour une machine non déterministe (que l'on peut voir comme une machine capable d'exécuter en parallèle un nombre fini d'alternatives). Intuitivement, cela signifie que la résolution des problèmes de NP peut nécessiter l'examen d'un grand nombre (éventuellement exponentiel) de cas, mais que l'examen de chaque cas doit pouvoir être fait en temps polynomial.

Les relations d'inclusion entre les classes P et NP sont à l'origine d'une très célèbre conjecture que l'on peut résumer par «$P \neq NP$ ». En effet, si la classe P est manifestement inclue dans la classe NP, la relation in verse n'a jamais été ni montrée ni infirmée.

Cependant, certains problèmes de NP apparaissent plus difficiles à résoudre dans le sens où l'on ne trouve pas d'algorithme polynomial pour les résoudre avec une machine déterministe. Les problèmes les plus difficiles de NP définissent la classe des problèmes NP-complets : un problème de NP est NP-complet s'il est au moins aussi difficile à résoudre que n'importe quel autre problème de NP, i.e., si n'importe quel autre problème de NP peut être transformé en ce problème par une procédure polynomiale.

Cette équivalence par transformation polynomiale entre problèmes NP-complets implique une propriété forte intéressante : si l'on trouvait un jour un algorithme polynomial pour un de ces problèmes (n'importe lequel) on pourrait en déduire des algorithmes polynomiaux pour tous les autres problèmes, et on pourrait alors conclure que $P = NP$. La question de savoir si un tel algorithme existe a été posée en 1971 par Stephen Cook... et n'a toujours pas reçu de réponse.

Ainsi, les problèmes NP-complets sont des problèmes combinatoires dans le sens où leur résolution implique l'examen d'un nombre exponentiel de cas. Notons que cette classe des problèmes NP-complets contient un très grand nombre de problèmes (dont quelques-uns sont décrits dans la section 4.2). Pour autant, tous les

problèmes combinatoires n'appartiennent pas à cette classe. En effet, pour qu'un problème soit NP-complet,

Il faut qu'il soit dans la classe NP, i.e., que l'examen de chaque cas puisse être réalisé efficacement, par une procédure polynomiale. Si on enlève cette contrainte d'appartenance à la classe NP, on obtient la classe plus générale des problèmes *NP-difficiles*, contenant l'ensemble des problèmes qui sont « au moins aussi difficiles » que n'importe quel problème de NP, sans nécessairement appartenir à NP.

- **Problèmes d'optimisation**. Le but d'un problème d'optimisation est de trouver une solution maximisant (respectivement minimisant) une fonction objective donnée. A chaque problème d'optimisation on peut associer un problème de décision dont le but est de déterminer s'il existe une solution pour laquelle la fonction objectif soit supérieure (respectivement inférieure) ou égale à une valeur donnée. La complexité d'un problème d'optimisation est liée à celle du problème de décision qui lui est associé. En particulier, si le problème de décision est NP-complet, alors le problème d'optimisation est dit *NP-difficile*.

1.2.2 Et en pratique ?

La théorie de la complexité nous dit que si un problème est NP-difficile, alors il est illusoire de chercher un algorithme polynomial pour ce problème (à moins que $P = NP$). Toutefois, ces travaux sont basés sur une évaluation de la complexité dans le pire des cas, car la difficulté d'un problème est définie par rapport à son instance la plus difficile. En pratique, si on sait qu'on ne pourra pas résoudre en temps polynomial *toutes* les instances d'un problème NP-difficile, il apparait que certaines instances sont beaucoup plus faciles que d'autres et peuvent être résolues très rapidement.

Considérons par exemple un problème de conception d'emploi du temps, consistant à affecter des créneaux horaires à des cours en respectant des contraintes d'exclusion —exprimant par exemple le fait que certains cours ne doivent pas avoir lieu en même temps. Il s'agit là d'un problème NP-complet classique se ramenant à un problème de coloriage de graphes. Pour autant, la résolution pratique de ce problème peut s'avérer plus ou moins difficile d'une instance à l'autre, même si l'on considère des instances de même taille, i.e., ayant toutes le même nombre de cours et

de créneaux horaires. Typiquement, les instances ayant très peu de contraintes d'exclusion sont faciles à résoudre car elles admettent beaucoup de solutions ; les instances ayant trop de contraintes pour avoir de solutions sont aussi facilement traitables car on arrive vite à montrer qu'elles n'admettent pas de solution ; les instances intermédiaires —ayant trop de contraintes pour que l'on trouve facilement une solution, mais pas assez pour que l'on montre facilement qu'elles n'ont pas de solution— sont généralement beaucoup plus difficiles à résoudre.

De façon plus générale, la difficulté des instances d'un problème de décision apparait souvent liée à un phénomène de « transition de phases » que nous introduisons dans le paragraphe suivant. Nous introduisons ensuite la notion de « paysage de recherche » qui permet de caractériser la difficulté des instances de problèmes d'optimisation.

- **Notion de transition de phases**. De nombreux problèmes de décision NP- complets peuvent être caractérisés par un paramètre de contrôle de telle sorte que l'espace des instances du problème comporte deux grandes régions : la région sous-contrainte où quasiment toutes les instances admettent un grand nombre de solutions, et la région sur-contrainte où quasiment toutes les instances sont insolubles. Très souvent, la transition entre ces deux régions est brusque dans le sens où une très petite variation du paramètre de contrôle sépare les deux régions. Ce phénomène est appelé « transition de phases » par analogie avec la transition de phases en physique qui désigne un changement des propriétés d'un système (ex., la fusion) provoquée par la variation d'un paramètre extérieur particulier (ex., la température) lorsqu'il atteint une valeur seuil.

Une caractéristique fort intéressante de ce phénomène est que les instances les plus difficiles à résoudre se trouvent dans la zone de transition [2, 3, 4], là où les instances ne sont ni trivialement solubles ni trivialement insolubles. Il s'agit là d'un pic de difficulté dans le sens où une toute petite variation du paramètre de contrôle permet de passer d'instances vraiment très faciles à résoudre à des instances vraiment très difficiles. Notons également que ce pic de difficulté est indépendant du type d'approche de résolution considérée [5, 6].

De nombreux travaux, à la fois théoriques et expérimentaux, se sont intéressés à la localisation de ce pic par rapport au paramètre de contrôle. En particulier, [7] introduit la notion de « degré de contrainte » (« constrainedness ») d'une classe de

problèmes, noté, et montre que lorsque est proche de 1, les instances sont critiques et appartiennent à la région autour de la transition de phase. Ces travaux sont particulièrement utiles pour la communauté de chercheurs s'intéressant à la résolution pratique de problèmes combinatoires. Ils permettent en particulier de se concentrer sur les instances vraiment difficiles, qui sont utilisées en pratique pour valider et comparer les approches de résolution. Ces connaissances peuvent également être utilisées comme des heuristiques pour résoudre plus efficacement les problèmes NP-complets [8].

Notons cependant que ces travaux se basent généralement sur l'hypothèse d'une génération aléatoire et uniforme des instances par rapport au paramètre de contrôle, de telle sorte que les contraintes sont relativement uniformément réparties sur l'ensemble des variables du problème. Cette hypothèse n'est pas toujours réaliste et les problèmes réels sont souvent plus structurés, exhibant des concentrations irrégulières de contraintes.

- **Notion de paysage de recherche**. La notion de transition de phases n'a de sens que pour les problèmes de décision binaires, pour lesquels on peut partitionner les instances en deux sous-ensembles en fonction de leur réponse. Pour les problèmes d'optimisation, on peut caractériser la difficulté d'une instance d'un problème d'optimisation par différents indicateurs.

Considérons par exemple une instance d'un problème d'optimisation définie par le couple (S, f) tel que S est l'ensemble des configurations candidates et $f: S \rightarrow R$ est une fonction associant un coût réel à chaque configuration, l'objectif étant de trouver une configuration s* ϵ S telle que f(s*) soit maximal. Un premier indicateur de la difficulté de cette instance est la densité d'états [9] qui donne la fréquence d'un coût c par rapport au nombre total de configurations : a priori, plus la densité d'un coût est faible, et plus il sera difficile de trouver une configuration avec ce coût.

Ce premier indicateur, indépendant de l'approche de résolution considérée, ne tient pas compte de la façon d'explorer les configurations. Or, on constate en pratique que cet aspect est déterminant : s'il y a une seule configuration optimale, mais que l'on dispose d'heuristiques simples pour se diriger vers elle dans l'espace des configurations, alors l'instance sera probablement plus facilement résolue que s'il y a plusieurs configurations optimales, mais qu'elles sont « cachées ».

Ainsi, on caractérise souvent la difficulté d'une instance en fonction de la topologie de son paysage de recherche [10, 11]. Cette topologie est définie par rapport à une relation de voisinage $v \subseteq S \times S$ qui dépend de l'approche considérée pour la résolution, ou plus précisément de la façon d'explorer l'ensemble des configurations : deux configurations s_i et s_j sont voisines si l'algorithme peut « explorer » s_j en une étape de résolution à partir de s_i. Cette relation de voisinage permet de structurer l'ensemble des configurations en un graphe $G = (S, v)$, que l'on peut représenter comme un « paysage » en définissant l'altitude d'un sommet $s_i \in S$ par la valeur de la fonction objectif $f(s_i)$. La topologie du paysage de recherche d'une instance par rapport à un voisinage donne des indications sur la difficulté de sa résolution par un algorithme explorant les configurations selon ce voisinage. En particulier, un paysage « rugueux », comportant de nombreux optima locaux (des sommets dont tous les voisins ont un coût inférieur), est généralement plus difficile qu'un paysage comportant peu d'optima. Un autre indicateur est la corrélation entre le coût d'un sommet et sa proximité à une solution optimale, l'instance étant généralement d'autant plus facilement résolue que cette corrélation est forte [12, 13].

1.3 Exemples de problèmes combinatoires

De très nombreux problèmes réels sont NP-complets ou NP-difficiles, ex., pour n'en citer que quelques-uns, les problèmes de conception d'emplois du temps, d'équilibrage de chaines de montage, de routage de véhicules, d'affectation de fréquences, de découpe ou d'emballage. Ainsi, [14] décrit près de trois cents problèmes NP-complets ; le « compendium of NP optimization problems[1] » recense plus de deux cents problèmes d'optimisation NP-difficiles, et la bibliothèque de problèmes en recherche opérationnelle « OR library[2] » en recense plus de cent.

On présente ici deux classes de problèmes combinatoires qui nous ont plus particulièrement intéressés, à savoir les problèmes de satisfaction de contraintes et les problèmes d'appariement de graphes.

1.3.1 Problèmes de satisfaction de contraintes

[1] http ://www .nada.kth.se/viggo/problemlist/compendium.html
[2] http ://people.brunel.ac.uk/ mastjjb/jeb/info.html

Les contraintes font partie de notre vie quotidienne, qu'il s'agisse par exemple de faire en emploi du temps, de ranger des pièces de formes diverses dans une boîte rigide, ou encore de planifier le trafic aérien pour que tous les avions puissent décoller et atterrir sans se percuter. La notion de « Problème de Satisfaction de Contraintes » (CSP) désigne l'ensemble de ces problèmes, définis par des contraintes, et consistant à chercher une solution les respectant.

La définition formelle d'un CSP sera présentée dans les chapitres suivants.

De façon générale, résoudre un CSP consiste à affecter des valeurs aux variables de telle sorte que toutes les contraintes soient satisfaites.

Notons que suivant les applications, « résoudre un CSP » peut signifier différentes choses : on peut chercher, par exemple, une solution (n'importe laquelle), ou bien toutes les solutions, ou bien une solution qui optimise une fonction objectif donnée, ou encore une affectation totale qui satisfait le plus grand nombre de contraintes (on parle alors de max-CSP) [15], ou qui minimise une somme pondérée des contraintes violées (on parle alors de CSP valué ou VCSP) [16]. Ces différents types de CSPs ont été généralisés dans le cadre des CSPs basés sur des semi-anneaux [17].

Tous les CSPs ne sont pas des problèmes combinatoires, et leur complexité théorique dépend de la nature des contraintes et des domaines des variables. Dans certains cas le problème peut être polynomial, par exemple lorsque les contraintes sont des équations ou inéquations linéaires et les domaines des variables sont continus.

Dans d'autre cas, le problème peut être indécidable, par exemple lorsque les contraintes sont des fonctions mathématiques quelconques et les domaines des variables sont continus. Bien souvent, le problème est NP complet (lorsqu'il s'agit d'un problème de satisfaction) ou NP-difficile (lorsqu'il s'agit d'un problème d'optimisation). En particulier, les CSPs sur les domaines finis (dont les domaines des variables sont des ensembles finis de valeurs) sont combinatoires dans le cas général.

Le cadre très général des CSPs permet de modéliser de nombreux problèmes réels (voir par exemple la bibliothèque CSPLib [18] qui recense 45 problèmes modélisés sous la forme de CSPs).

1.3.2 Problèmes d'appariement de graphes

Les graphes sont des structures de données utilisées notamment pour modéliser des objets en termes de composants (appelés sommets), et de relations binaires entre composants (appelées arcs). De façon plus formelle, un graphe est défini par un couple $G = (V, E)$ tel que

– V est un ensemble fini de sommets,

– $E \subseteq V \times V$ est un ensemble d'arcs.

Les problèmes d'appariement de graphes consistent à mettre en correspondance les sommets de deux graphes, l'objectif étant généralement de comparer les objets modélisés par les graphes. On introduit ici les quatre problèmes d'appariement les plus connus :

– l'isomorphisme de graphes, qui permet de vérifier que deux graphes sont structurellement identiques ;

– l'isomorphisme de sous-graphes, qui permet de vérifier qu'un graphe est « inclus » dans un autre ;

– le plus grand sous-graphe commun, qui permet d'identifier la plus grande partie commune à deux graphes ;

– la distance d'édition de graphes, qui permet d'identifier le plus petit nombre d'opérations à effectuer pour transformer un graphe en un autre.

1.4 Résolution des problèmes combinatoires : méthodes exactes (approches complètes) et méthodes approchées (approches incomplètes) et les autres approches

De très nombreux problèmes réels devant être résolus quotidiennement sont combinatoires et il s'agit en pratique de trouver des solutions à ces problèmes. La théorie de la complexité nous disant que l'on ne pourra pas —a priori— trouver un algorithme à la fois rapide, correct et complet, on traite ces problèmes en faisant délibérément l'impasse sur un ou plusieurs de ces critères : l'objectif est d'avoir un comportement « acceptable » en pratique, i.e., d'être capable de trouver une solution de qualité suffisante en un temps raisonnable pour les instances à résoudre.

1.4.1 Approches complètes

Pour résoudre un problème combinatoire, on peut explorer l'ensemble des configurations de façon exhaustive et systématique, et tenter de réduire la

combinatoire en utilisant des techniques d'« élagage » —visant à éliminer le plus tôt possible des sous-ensembles de configurations en prouvant qu'ils ne contiennent pas de solution—et des heuristiques de choix —visant à se diriger le plus rapidement possible vers les meilleures configurations.

Ces approches sont complètes dans le sens où elles sont toujours capables de trouver la solution optimale ou, le cas échéant, de prouver l'absence de solution ; en revanche, le temps de résolution dépend de l'efficacité des techniques d'élagage et des heuristiques considérées, et est exponentiel dans le pire des cas.

Un point clé de ces approches réside dans la façon de structurer l'ensemble des configurations, l'objectif étant de pouvoir éliminer des sous-ensembles de configurations de façon globale, sans avoir à examiner les configurations qu'ils contiennent une par une. La structuration la plus utilisée est appelé *structuration en arbre*.

- **Structuration en arbre.** Pour explorer l'ensemble des configurations, on peut le découper en sous-ensembles de plus en plus petits. On construit pour cela un « arbre de recherche » dont la racine correspond à l'ensemble de toutes les configurations à examiner et dont les nœuds correspondent à des sous-ensembles de configurations de plus en plus petits au fur et à mesure que l'on descend dans l'arbre. Plus précisément, les sous-ensembles de configurations associés aux fils d'un nœud constituent une partition de l'ensemble des configurations associées au nœud. Cet arbre est généralement construit selon une stratégie par « séparation et évaluation » (« branch and bound »), en partant de la racine et en itérant sur les opérations suivantes :

  – sélectionner une feuille de l'arbre selon une heuristique de sélection donnée (ex., choisir la dernière feuille créée pour une exploration « en profondeur d'abord », ou choisir la feuille la plus « prometteuse » pour une exploration « du meilleur d'abord ») ;

  – séparer (partitionner) l'ensemble des configurations associé à une feuille sélectionnée en plusieurs sous-ensembles (ex., partitionner l'ensemble des configurations en autant de sous-ensembles qu'il y a de valeurs possibles pour une variable) ;

– évaluer le sous-ensemble de configurations associé à un noeud, afin de déterminer s'il peut contenir une solution (auquel cas le nœud correspondant devra être développé ultérieurement), ou s'il ne contient assurément aucune solution (auquel cas le nœud correspondant peut être « élagué »).

La fonction d'évaluation est déterminante pour la viabilité de l'approche en pratique : cette fonction doit pouvoir déterminer efficacement (i.e., plus rapidement qu'en énumérant toutes les configurations de l'ensemble) qu'un ensemble de configurations ne peut pas contenir de solution.

Pour les problèmes d'optimisation, la fonction d'évaluation retourne une estimation du coût de la meilleure solution de l'ensemble de sorte que si cette estimation est moins bonne que la meilleure solution trouvée jusqu'ici, alors on peut couper le nœud correspondant. Cette estimation du coût est généralement obtenue en relâchant certaines contraintes du problème de sorte que le problème « affaibli » ne soit plus combinatoire et puisse être évalué en temps polynomial. Par exemple, pour résoudre un problème linéaire en nombres entiers (consistant à trouver les valeurs entières à affecter à un ensemble de variables de décision optimisant une fonction objectif linéaire et satisfaisant un certain nombre de contraintes linéaires) selon une stratégie par « séparation et évaluation », on estime généralement le coût d'un nœud de l'arbre de recherche en relâchant les contraintes d'intégralité sur les variables, le problème sur les réels pouvant être résolu en temps faiblement polynomial [19].

Pour les problèmes de décision, la fonction d'évaluation doit être capable de détecter le fait qu'un ensemble ne contient pas de configuration solution. Cette évaluation est souvent réalisée en « filtrant » l'ensemble des configurations, i.e., en éliminant des parties dont on infère qu'elles ne peuvent pas contenir de solution. Ainsi, une stratégie classique pour résoudre des problèmes de satisfaction de contraintes consiste à exploiter (« propager ») les contraintes du problème pour éliminer du domaine des variables les valeurs violant les contraintes, filtrant ainsi l'ensemble des configurations candidates. Le résultat de cette étape de filtrage vérifie généralement une certaine consistance partielle, ex., la consistance de noeud, d'arc ou de chemin [20]. L'intégration de ces techniques de propagation de contraintes ou les paires de valeurs permises à l'intérieur d'une exploration arborescente est à la base de

nombreux « solveurs » de contraintes qui, intégrés dans des langages de programmation, constituent le paradigme de la « programmation par contraintes » [21, 22].

1.4.2 Approches incomplètes

Les approches complètes permettent de résoudre en pratique de nombreux problèmes combinatoires, comme en témoigne le succès industriel de la programmation par contraintes. Cependant, les techniques d'élagage ne permettent pas toujours de réduire suffisamment la combinatoire, et certaines instances de problèmes ne peuvent être résolues en un temps acceptable par ces approches exhaustives.

Une alternative consiste alors à explorer l'ensemble des configurations de façon opportuniste, en utilisant des heuristiques pour se guider vers les zones de l'espace des configurations qui semblent plus prometteuses. Ces approches sont incomplètes dans le sens où elles n'explorent délibérément qu'une partie de l'espace des configurations. Par conséquent, elles peuvent ne pas trouver la solution optimale dans certains cas, et encore moins prouver l'optimalité des solutions trouvées ; en contrepartie, le temps de résolution est généralement faiblement polynomial.

On distingue essentiellement deux types d'approches heuristiques :

– Les approches par voisinage explorent l'espace des configurations en le structurant en termes de voisinage ; ces approches sont qualifiées de « instance-based » dans [23] dans le sens où les nouvelles configurations sont créées à partir de configurations existantes.

– Les approches constructives génèrent de nouvelles configurations de façon incrémentale en ajoutant itérativement des composants de solution aux configurations en cours de construction ; ces approches sont qualifiées de « model-based » dans [23] dans le sens où elles utilisent un modèle, généralement basé sur un mécanisme stochastique adaptatif, pour construire les configurations.

- **Intensification versus diversification de la recherche**. Pour ces différentes approches heuristiques, explorant l'espace des configurations de façon

opportuniste, un point critique dans la mise au point de l'algorithme consiste à trouver un compromis entre deux tendances duales :

– il s'agit d'une part d'intensifier l'effort de recherche vers les zones les plus « prometteuses » de l'espace des configurations, i.e., aux alentours des meilleures configurations trouvées ;

– il s'agit par ailleurs de diversifier l'effort de recherche de façon à être capable de découvrir de nouvelles zones contenant (potentiellement) de meilleures solutions.

La façon d'intensifier/diversifier l'effort de recherche dépend de l'approche considérée et se fait en modifiant certains paramètres de l'algorithme. Pour les approches par voisinage, l'intensification de la recherche se fait en favorisant l'exploration des « meilleurs » voisins. Par exemple, pour les algorithmes génétiques, on augmente la probabilité de croisement des meilleures configurations de la population et/ou on diminue la probabilité de mutation ; pour la recherche locale taboue, on diminue la longueur de la liste taboue ; pour le recuit simulé, on diminue la température pour diminuer la probabilité de choisir des « mauvais » voisins. Pour les approches par construction, l'intensification de la recherche se fait en augmentant la probabilité de choisir les « meilleurs » composants de solution. Par exemple, pour les algorithmes gloutons aléatoires, on augmente la probabilité de choisir les meilleurs composants de solution en diminuant le nombre k ou le ratio ; pour la méta-heuristique ACO, on augmente la probabilité de choisir les composants de solution ayant appartenu aux meilleurs configurations trouvées en augmentant l'influence de la phéromone.

En général, plus on intensifie la recherche d'un algorithme en l'incitant à explorer les configurations proches des meilleures configurations trouvées, et plus il « converge » rapidement, trouvant de meilleures solutions plus tôt. Cependant, si l'on intensifie trop la recherche, l'algorithme risque fort de « stagner » autour d'optima locaux, concentrant son effort de recherche sur une petite zone autour d'une assez bonne configuration, sans plus être capable de découvrir de nouvelles configurations. Notons ici que le bon équilibre entre intensification et diversification dépend clairement du temps de calcul dont on dispose pour résoudre le problème : plus ce temps est petit et plus on a intérêt à favoriser l'intensification pour converger rapidement, quitte à converger vers des configurations de moins bonne qualité.

Le bon équilibre entre intensification et diversification dépend également de l'instance à résoudre, ou plus particulièrement de la topologie de son paysage de recherche. En particulier, plus la corrélation entre la qualité d'une configuration et sa distance à la solution optimale est forte et plus on a intérêt à intensifier la recherche. Différentes approches ont proposé d'adapter dynamiquement le paramétrage au cours de la résolution, en fonction de l'instance à résoudre. Par exemple, l'approche taboue réactive de [24] ajuste automatiquement la longueur de la liste taboue, tandis que l'approche locale de [25] adapte automatiquement le nombre de voisins considérés à chaque mouvement.

1.4.3 Et les autres approches

Cette présentation des approches pour la résolution pratique de problèmes combinatoires n'est certainement pas exhaustive, cette classe de problèmes ayant stimulé l'imagination d'un très grand nombre de chercheurs. En particulier, la « classification » proposée —distinguant approches complètes et incomplètes, structuration en arbre et en treillis, exploration par voisinage et par construction— n'est clairement pas exclusive, et de nombreuses approches hybrides ont été proposées, combinant les principes de plusieurs approches pour donner naissance à de nouveaux algorithmes. On peut par exemple combiner approches complètes et recherche locale [26, 27, 28], algorithmes génétiques et recherche locale [29], approches constructives et recherche locale [30].

Une alternative aux approches complètes et incomplètes pour la résolution de problèmes d'optimisation combinatoire, consiste à résoudre le problème en s'autorisant délibérément une marge d'erreur. En particulier, les algorithmes de « approximation » [31] sont des algorithmes de complexité polynomiale qui calculent une configuration dont la qualité par rapport à la qualité de la configuration optimale est bornée par p. Les problèmes d'optimisation NP-difficiles ne sont pas tous équivalents en termes « d'approximabilité » : certains comme le problème du « bin-packing » peuvent être approximés avec un facteur quelconque, la complexité en temps de l'algorithme d'approximation augmentant lorsque le facteur d'erreur diminue ; d'autres comme le problème de la clique maximum ne peuvent pas être approximés avec un facteur constant, ni même polynomial (à moins que $P = NP$).

1.5 Conclusion

Dans ce chapitre, nous avons fait un rappel sur les problèmes combinatoires ainsi sur les notions de théorie de complexité. Nous avons montré aussi divers méthodes de résolution de problèmes combinatoires tels que les méthodes exactes et les méthodes approchées. Malgré la diversité des méthodes de résolution, il est toujours difficile de résoudre un très grand nombre de problèmes combinatoires comme les problèmes SAT de grandes instances et les classes de problèmes de satisfaction de contraintes. En effet, certains problèmes possèdent une propriété qui peut être exploité afin d'améliorer la résolution de ces problème, c'est la propriété de symétrie. L'objectif est donc de savoir exploiter la symétrie présente dans ces problèmes et de la supprimer de leur espace de recherche pour accélérer la résolution de ces classes de problèmes. Dans la suite nous allons étudier les travaux faits dans ce domaine de recherche.

2 La symétrie dans la programmation logique

2.1 Introduction

L'utilisation des symétries dans les langages de représentation de connaissances ou dans les logiques a reconnu un grand succès. En effet, différentes modes de représentation de connaissances et différentes méthodes de résolution existent, plusieurs qualités sont souhaitées par ces langages de représentation, parmi eux on cite l'efficacité des méthodes de déduction associées en termes de sureté et de rapidité, cette dernière peut être assurée et améliorée en introduisant une propriété fondamentale qui a été largement utilisée dans les langages mathématiques, il s'agit de la propriété de symétrie. D'abord, le principe de symétrie a été introduit par Krishnamurty en logique propositionnelle [32]. Il a montré que ce principe apporte une amélioration considérable pour la résolution. En effet l'utilisation des symétries évite de calculer dans une preuve formelle les formules intermédiaires qui sont déductibles par application de symétries sur des formules déjà produites. Par la suite, le problème de détection des symétries et leurs utilisation systématique dans les méthodes de démonstration ont été abordés par B.Benhamou [33], cette propriété a été étendue par la suite dans différentes classes de la logiques, tel que les logiques classiques et les logiques non classiques. Dans ce qui suit nous allons montrer les différentes extensions et illustrer comment les méthodes d'inférence et de déduction sont améliorées.

2.2 Symétrie dans la logique classique

2.2.1 Définitions de symétrie et rappels en logique propositionnelle

2.2.1.1 La logique propositionnelle

On donne ici une courte description de la logique propositionnelle. Soit V un ensemble de *variables propositionnelles*. Les variables propositionnelles seront distinguées des *littéraux* qui sont des variables propositionnelles avec une valeur d'affection 1 ou 0 qui signifie respectivement *Vrai* ou *Faux*. Pour une variable propositionnelle l, il y a deux littéraux : le littéral positif l et le littéral négatif $\neg l$.

Une clause est une disjonction de littéraux $\{ l_1, l_2, \ldots, l_n\}$ tel qu'aucun littéral n'apparait plus d'une fois, ni un littéral et sa négation en même temps. Cette clause

est désignée par $l_1 \vee l_2 \vee \ldots \vee l_n$. Une formule F est mise sous *forme normale conjonctive* (CNF) si et seulement si c'est une conjonction de clauses.

Une interprétation d'une CNF F est une correspondance I définie de l'ensemble des variables de F dans l'ensemble {*Vrai; Faux*}. Nous pouvons considérer I comme l'ensemble des littéraux interprétés. La valeur d'une clause $l_1 \vee l_2 \vee \ldots \vee ln$ dans I est à *Vrai*, si la valeur *Vrai* est affectée à au moins un de ses littéraux dans I ; sinon, elle est à *Faux*. Par convention, nous définissons la valeur de la *clause vide* ($n = 0$) à *Faux*.

La valeur I [F] est *V rai* si la valeur de chaque clause de F est à *V rai*, *Faux* sinon. Une formule CNF F est *satisfiable* s'il existe une interprétation I qui affecte la valeur *V rai* à F, sinon elle est *insatisfiable*. Dans le premier cas, I est appelé *modèle* de F. Une formule G est une *conséquence logique* d'une formule F si F implique G et est notée $F \models G$. Si une interprétation I est définie seulement sur un sous-ensemble de variables de F, alors elle est appelée *interprétation partielle*. Elle est appelée un *no-good* si I [F] = *Faux*.

2.2.2 Les permutations

Soit l'ensemble $\Omega = \{1, 2, \ldots, N\}$ pour un entier donné N, où chaque entier représente une variable propositionnelle. Une permutation de Ω est une application bijective σ de Ω à Ω qu'on représente usuellement comme un produit de cycles de permutations. On d´dénote par *Perm*(Ω) l'ensemble des toutes les permutations de Ω et o la composition de permutations de *Perm*(Ω). La paire (*Perm*(Ω), o) forme le groupe de permutation de Ω. En effet, o est close, associative, l'inverse d'une permutation est une permutation et la permutation identité est l'´élément neutre.

Une paire (T, o) forme un sous-groupe de (S, o) si et seulement si T est un sous-ensemble de S et forme muni de l'opération o un groupe.

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

$$\omega^{Perm(\Omega)} = \{ \omega^{\sigma} : \sigma \in Perm(\Omega)\}.$$

Un ensemble de générateurs du groupe *Perm*(Ω) est un sous-ensemble *Gen* de *Perm*(Ω) tel que chaque élément de *Perm*(Ω) peut être écrit comme composition des

éléments de *Gen*. Nous écrivons *Perm*(Ω) = < *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*.

2.2.3 Symétrie dans la logique propositionnelle

Tout d'abord, on donne la définition de symétrie sémantique dans la logique propositionnelle :

Définition 1 (Symétrie sémantique) *Soient F une formule propositionnelle sous la forme CNF et L_F son ensemble complet de littéraux. Une symétrie sémantique de F est une permutation σ définie sur L_F telle que F |= σ*(F) *et σ*(*F*) |= *F*.

En d'autres termes, une symétrie sémantique d'une formule est une permutation de littéraux qui conserve l'ensemble des modèles de la formule. Elle conserve également l'ensemble des no-goods (contre modèles). Maintenant, nous rappelons la définition d'une symétrie plus restreinte, qui est la symétrie syntaxique [34, 35] et qui peut être calculée de manière efficace.

Définition 2 (Symétrie syntaxique) *Soient F une formule propositionnelle sous la forme CNF et L_F son ensemble complet de littéraux. Une symétrie syntaxique de F est une permutation σ définie sur L_F telle que les conditions suivantes sont vérifiées :*

1. $\forall\, l \in L_F, \sigma(\neg l) = \neg\sigma(l)$,
2. $\sigma(F) = F$

En d'autres mots, une symétrie syntaxique d'une formule est une permutation de littéraux qui laisse invariant la formule. Si on dénote par *Perm*(L_F) le groupe de permutations de L_F et par *Sym*(L_F) $\subset$ *Perm*(L_F) le sous ensemble des permutations de *LF* qui sont les symétries syntaxiques de *F*, alors *Sym*(L_F) est trivialement un sous-groupe de *Perm*(L_F).

2. La symétrie dans la programmation logique

Une symétrie syntaxique d'une formule est une permutation de variables qui laisse invariant la formule. Il est trivial de voir que chaque symétrie syntaxique est une symétrie sémantique mais, en général, l'inverse n'est pas vérifié.

Définition 3 *Deux littéraux l et l' d'une formule F sont symétriques, s'il existe une symétrie σ de F telle que $\sigma(l) = l'$.*

Définition 4 *Soit F une formule, l'orbite d'un littéral $l \in L_F$ sur lequel le groupe de symétries Sym (L_F) opère est $l^{Sym(LF)} = \{\sigma(l) : \sigma \in Sym(L_F)\}$.*

Proposition 1 Tous les littéraux d'une orbite l sont symétriques deux à deux.

Si I est un modèle de F et σ une symétrie, nous pouvons avoir un autre modèle de F en appliquant σ sur les variables qui apparaissent dans I. Autrement dit, si I est un modèle de F alors $\sigma(I)$ est un modèle de F. Une symétrie σ transforme chaque modèle en un autre modèle et chaque no-good en un autre no-good.

Dans la proposition suivante, on assume que σ est une symétrie de l'ensemble des clauses F.

Proposition 2 Soit l un littéral, σ une symétrie telle que $l' = \sigma(l)$ et $I' = \sigma(I)$. Si I est telle que $I[l] =$ Vrai, alors I' est telle que $I'[l'] =$ Vrai.

On déduit la proposition suivante :

Proposition 3 Si un littéral l a la valeur vrai dans un modèle de F, alors σ (l) doit avoir la valeur vrai dans un modèle de F.

Théorème 1 *Soient l et l' deux littéraux de F qui sont dans la même orbite en ce qui concerne le groupe de symétries Sym (L_F), alors l est vrai dans un modèle de F si et seulement si l' est vrai dans un modèle de F.*

Corolaire 1 *Soit l un littéral de F, si l n'est pas vrai dans aucun modèle de F, alors chaque littéral l' ∈ orbit* ^{LF} *n'est pas vrai dans aucun modèle de F.*

Proposition 4 Si *L* est la logique propositionnelle, *A* un ensemble de formules de *L*, *B* une formule de L et σ une symétrie syntaxique de A, alors la règle de la symétrie peut être définie comme suit :

$$\frac{A \vdash B}{A \vdash \sigma(B)}$$

2.2.4 Etude de cas particulier de la logique classique :

2.2.4.1 Le problème SAT (problème de satisfiabilité de formules CNF)

On présente dans cette partie une méthode pour l'exploitation des symétries dans un problème central de la théorie de la complexité connu par le problème SAT (satisfiabilité de formules) qui est un problème de décision de classe NP-complet. Ce travail [36] consiste à appliquer une méthode pour détecter et éliminer les symétries locales et cela se fait dynamiquement durant la recherche qui apparaît à chaque nœud de l'arbre de recherche. En effet, cette contribution représente un challenge vu que quasiment tous les travaux sur l'exploitation des symétries traitent uniquement le cas des symétries globales, c'est-à-dire, les symétries initiales du problème (le problème à la racine de l'arbre de recherche) qui sont détectés et éliminés. Alors la méthode proposée consiste à réduire l'instance partielle SAT, non encore résolue, correspondante à chaque nœud de l'arbre de recherche, à un graphe dont le groupe d'automorphisme est équivalent au groupe de symétries de l'instance partielle SAT. Cette méthode a été implémentée en utilisant l'outil Saucy pour le calcul du groupe d'automorphismes plus l'utilisation d'une technique de coupure de symétries dans un solveur SAT. Cette approche à donner de bons résultats et en la comparant avec les méthodes exploitant les symétries globales, elle leurs sont complémentaires si on combine les deux techniques et elle à améliorer l'exploitation des symétries globales dans la résolution de nombreux problèmes difficiles.

2. La symétrie dans la programmation logique

Avant de donner une aperçue de cette technique de détection et de suppression de symétrie on définit d'abord les problèmes SAT.

2.2.4.2 Présentation du problème SAT [37]

Définition 5 (Problème SAT).

Soit F *une formule CNF. Le problème SAT est un problème de décision qui consiste à déterminer si* F *admet ou non un modèle.* En d'autres termes, il est décrit par la question suivante : « Est-ce que la formule CNF est satisfiable ? ». Remarquons que cela n'implique pas de trouver un modèle de la formule, mais qu'il faut en prouver l'existence. Parfois, la formule propositionnelle F est appelée une *instance* SAT.

Exemple 1.3.1.

Donnée : *Soit* F *une formule CNF définie comme suit :*

$$F = (a \vee b \vee c) \wedge (\neg a \vee b) \wedge (\neg b \vee c) \wedge (\neg c \vee a)$$

Remarquons qu'on peut définir cette formule aussi comme suit :

$$F = \{(a \vee b \vee c), (\neg a \vee b), (\neg b \vee c), (\neg c \vee a)\}$$

Question *Est-ce que la formule F est satisfaisable ?*

Réponse *Pour cet exemple, la réponse est oui : l'instanciation I : a = vrai, b = vrai, c = vrai satisfait la formule F.*

La formule F admet un seul modèle m = {a, b, c}. Prenons la formule F′ = F $\wedge$ ($\neg a \vee \neg b \vee \neg c$). *Comme F admet un seul modèle m qui ne satisfait pas la clause* ($\neg a \vee \neg b \vee \neg c$), *alors F′ est insatisfaisable.*

Le problème SAT est un des problèmes qui a été très étudié en informatique, du fait de son importance aussi bien théorique que pratique.

En fait, le problème SAT est le premier problème prouvé comme étant NP-complet (Non déterministe Polynomial) par Cook en 1971 [38]. Par sa simplicité, la forme CNF est généralement utilisée dans les solveurs SAT modernes. En effet, de nombreux problèmes s'expriment naturellement comme une conjonction de clauses.

2.2.4.3 Détection et élimination de la symétrie locale

Pour la détection des symétries syntaxiques, un outil appelé Saucy a été utilisé pour le calcul des automorphismes de graphes. En effet, il existe des travaux [39, 40, 41] qui ont montré que chaque formule CNF F peut être représentée par un graphe GF qui peut être construit de la façon suivante :

Chaque variable booléenne est représentée par deux sommets (sommet littéral) dans *GF* : le littéral positif et son négatif. Ces deux sommets sont connectés par une arête dans le graphe *GF*.

– chaque clause non binaire est représentée par un sommet (sommet clause). Une arête connecte ce sommet à chaque sommet représentant un littéral de la clause.

– Chaque clause binaire est représentée par une arête connectant les deux sommets représentant les deux littéraux de la clause. Les sommets correspondants aux clauses binaires ne sont pas ajoutés.

Le groupe de symétrie syntaxique de F est identique au groupe d'automorphisme de sa représentation graphique GF, Saucy à le rôle de détecter le groupe de symétries syntaxiques de F à partir de GF en retournant l'ensemble des générateurs Gen du groupe de symétries duquel on peut déduire chaque symétrie. Avec cet outil on a la possibilité de colorier les sommets, deux sommets coloriés avec la même couleur signifie qu'ils peuvent être permutés, ou deux couleurs sont utilisés, une pour le coloriage des clauses et l'autre pour le coloriage des littéraux, cela permet de prévenir la génération de symétries entre clauses et littéraux.

Par exemple, soit la formule F= {$a \vee b \vee c, \neg a \vee b, \neg b \vee c, \neg c \vee a, \neg a \vee \neg b \vee \neg c$}, et la figure montre le graphe GF correspondant à *F* :

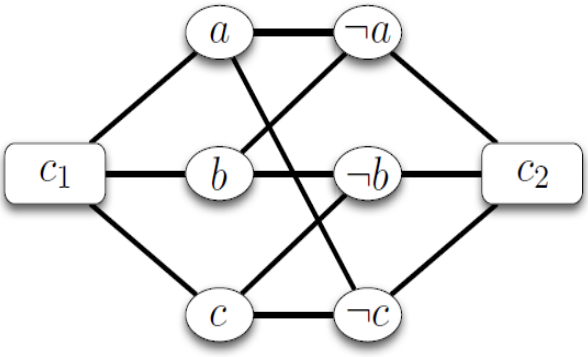


Figure 2-1 Le graphe GF correspondant à F

- **Détection dynamique des symétries :**

Pour la détection dynamique de la symétrie, l'idée consiste à simplifier une formule F en une formule FI en effectuant un assignement partiel I qui s'arrête au niveau du littéral courant l en cours d'assignement et qui correspond au nœud courant n_I de l'arbre de recherche. Donc il consiste de maintenir dynamiquement le graphe GFI de la sous formule FI correspondant au sous problème local définit au nœud courant n_I, après coloriage du graphe GFI, Saucy déduit l'ensemble des générateurs Gen du sous-groupe de symétries existants entre les littéraux de LFI à partir duquel l'orbite du littéral l est calculé afin d'être utilisé dans les coupures de symétries.

- **Elimination des symétries locales:**

Pour l'élimination de la symétrie qui consiste à élaguer des espaces de recherche des méthodes de résolution le corolaire suivant est utilisé :

Corolaire 1 *Soit l un littéral de F, si l n'est pas vrai dans aucun modèle de F, alors chaque littéral l'* $\in$ *orbit* ^{LF} *n'est pas vrai dans aucun modèle de F.*

On comprend de ce corolaire qu'il faut assigner la valeur faux à chaque littéral de l'orbite d'un littéral courant l (défini à un nœud n_I de l'arbre de recherche) si l'assignation de la valeur vrai à l mène à un échec, cela implique la suppression des sous espaces correspondant à l'assignement de la valeur alternative vrai à ces littéraux qui correspondent la suppression des symétries.

Cette méthode a été comparée avec d'autres méthodes tel que la méthode pour sélection et élimination des symétries globales, la méthode sans élimination de symétries et une dernière méthode combinant les deux approches de suppression des symétries locales et globales, la comparaison a été faite par rapport au temps nécessaire pour le calcul et l'exploitation des symétries globales et locales et aussi par rapport au nombre de nœuds traités, les quatre méthodes ont été implémentés sur différentes instances SAT, les résultats obtenus ont montré que l'approche de suppression de symétries globales est meilleure pour quelques instances avec la capacité de l'approche de suppression locale de les résoudre aussi d'autres instances plus dur sont résolus efficacement avec l'approche de suppression locale donc on conclut que l'élimination de la symétrie locale est plus avantageuse que celle de

l'élimination des symétries globales et c'est l'approche combinant les deux techniques qui offre les meilleurs résultats, donc Les résultats expérimentaux confirment que l''élimination des symétries locales est d'un apport non négligeable pour la résolution des problèmes SAT et améliore les m´méthodes n'exploitant que l'élimination des symétries globales sur certains problèmes, et qu'elle peut être complémentaire à l'élimination des symétries globales si on les combine.

Un autre travail [42] qui est une extension de ce travail qui consiste à l'amélioration de l'apprentissage des clauses par symétrie dans les solveurs SAT : en effet, des solveurs de type CDCL ont la capacité de résoudre de manière efficace des problèmes industriels de très grande taille, ils utilisent donc la notion d'apprentissage, alors l'idée était de combiner l'exploitation de la symétrie avec la notion d'apprentissage de clauses afin d'améliorer l'apprentissage dans des solveurs de type CDCL, le principe c'est que l'apprentissage par symétrie permet en plus de l'ajout de la clause assertive dans un nœud de l'arbre de recherche, d'ajouter toutes les clauses assertives symétriques, ces clauses assertives symétriques permettent aux solveurs SAT d'éviter d'explorer des sous espaces iso-morphiques. Alors l'apprentissage par symétrie a été implémenté dans des MiniSat et a été expérimenté sur différents problèmes.

2.3 Symétrie dans la logique non classique (logique non monotone)

En Intelligence Artificielle, on a l'habitude de manipuler des informations incomplètes qui nécessitent d'inclure l'incertitude dans le raisonnement sur la connaissance avec exceptions et la non-monotonie. Plusieurs logiques non classiques sont mises en place à cet effet, mais, la symétrie dans ces logiques n'a pas encore été étudiée contrairement à leur utilisation dans la logique classique. Alors dans cette partie on va présenter les premiers travaux de Benhamou et al [43] qui s'intéresse à étendre la notion de la symétrie à des logiques non classiques telles que les logiques préférentielles, X-logiques et les logiques des défauts, et donne des nouvelles règles d'inférence par symétrie pour les X-logiques et les logiques des défauts.

2.3.1 Symétrie dans la logique préférentielle

2. La symétrie dans la programmation logique

Dans cette section, Benhamou étend la notion de la symétrie dans les logiques préférentielles, il donne d'abord un aperçu de la logique préférentielle en donnons un ensemble de définitions et de propositions décrivant les notions de : relation préférentielle, modèle minimal d'une formule, …etc. par la suite il donne un ensemble de définitions de la symétrie dans cette logique.

2.3.1.1 Préliminaires

Benhamou définit d'abord la problématique de la nécessité de la révision du raisonnement et des conclusions dans le cas ou des exceptions sont rajoutées à un système de connaissances (formules), et il mit l'accent sur l'importance de considérer une classe qui regroupe les individus ou les instances qui sont symétriques par rapport à une proposition donnée. Donc au départ, ça consiste de partir d'une approche préférentielle, telle qu'il a été initié par Bossu- Siegel [44, 45], repris aussi par Shoam [46] et Besnard-Siegel [47], puis par Kraus, Lehmann et Magidor dans [48]. Toutes ces approches sont construites sur une logique classique (calcul propositionnel, calcul des prédicats, logique modale), où la sémantique de l'inférence a été donnée par "une formule *A* implique une formule *B* si chaque modèle de *A* est un modèle de *B*". Cependant, une approche préférentielle, dans sa forme la plus générale, dit "*A* implique *B* si tous les modèles préférés de *A* sont des modèles de *B*". Les modèles préférés de A sont des modèles qui ont des propriétés utiles pour la gestion des exceptions. Ce concept de préférence peut être défini par une relation de pré ordre (une relation transitive et réflexive) sur les interprétations. Les modèles préférés étant les modèles minimaux pour cette relation. Par exemple, étant donné deux interprétations *I* et *J* et une information (propriété) pertinente (utile) ω, alors la relation de pré ordre peut être définie par : $I < J$ si et seulement si tout individu vérifiant la propriété ω en J vérifie la propriété ω dans *I*.

Définition 6 *Soient L une logique classique et F l'ensemble des formules de L. Si A est un sous-ensemble de formules (ou une formule) de F, alors* $\bar{A}$ *est l'ensemble des formules logiquement impliquées par A. L'ensemble des formules A est déductivement clos si* $A = \bar{A}$.

Définition 7 *Une relation préférentielle $<$ est une relation de pré ordre (transitive et réflexive) sur les interprétations. Par ailleurs, si la relation $<$ est antisymétrique, alors $<$ devient un ordre. En plus $I < J$ si l'ensemble des exceptions de I est inclus dans l'ensemble des exceptions de J.*

Définition 8 *Si A est un ensemble de formules, un modèle minimal M de A est une interprétation qui satisfait A (ie $M \vdash A$) et qui est minimale par rapport à la relation $<$ définie sur l'ensemble des modèles de A. Autrement dit, si M'est un modèle de A tel que $M' < M$, alors $M < M'$ (ou de manière équivalente, $M' = M$ si $<$ est antisymétrique).*

Définition 9 *De façon classique, si $\prec$ est une relation préférentielle, on définit l'inférence logique de modèles préférentiels $\vdash_{\prec}$ comme suit : $A \vdash_{\prec} B$ si et seulement si chaque modèle minimal de A est un modèle de B.*

Dans [49] la proposition suivante est fournie :

Proposition 5 Si un langage *L* a un ensemble fini de variables, alors chacune de ses formules consistantes *F* a au moins un modèle minimal, et pour chaque modèle de *M* de *F*, il existe un modèle minimal *M*' tel que $M' < M$.

En fin de cette partie, Benhamou explique à travers un exemple que dans la logique classique il n'est pas possible de déduire certains connaissances à partir d'un système de formules contenant des exceptions, cela revient aux différents modèles du système initial de formules ou la connaissances qu'in veut déduire prend des valeurs différents dans les différents modèles, la solution est donc d'utiliser l'approche de modèles préférentiels et donc de proposer une préférence sur les modèles, une fois l'ordre de préférence est établi sur les modèles, le modèle minimal est choisi et la valeur de la connaissance est déduite dans ce modèle, il explique par la suite l'idée de pouvoir déduire toutes les connaissances symétriques à la connaissance déduite par rapport à cette relation de préférence.

2.3.1.2 Symétrie

2. La symétrie dans la programmation logique

Benhamou étend la définition de la symétrie sémantique à la logique de modèles préférentiels et montre comment les littéraux peuvent être symétriques dans cette logique non classique, mais pas symétrique dans une logique classique.

Définition 10 *(Symétrie préférentielle sémantique) Si* $\vdash <$ *est une inférence de modèles préférentiels, A est un ensemble de formules et σ une permutation définie sur les littéraux de A, alors σ est une symétrie de A, si et seulement si A et σ(A) ont le même ensemble de modèles minimaux.*

Définition 11 *Deux littéraux l et l' sont symétriques dans A si et seulement si il existe une symétrie préférentielle sémantique σ de A tel que σ (*l*) = l'.*

Alors, Benhamou montre à travers un exemple que des littéraux peuvent être symétriques dans la logique préférentielle, ces même littéraux peuvent ne pas être des symétries dans la logique classique, si par exemple on prend d'une manière générale une formule A avec trois littéraux : l_1, l_2 et l_3 et deux individus : ind1 et ind2 tel que :

$$A= \{ l_1(ind_1), l_1(ind_2), l_2(ind_1), l_3(ind_1), l_2(ind_2), l_3(ind_2)\}$$

On peut remarquer facilement que les individus sont symétriques dans les deux logiques: classique et préférentielle, cela dans le sens que chaque littéral, ou ind_1 apparait, est symétrique au littéral ou l'on remplace ind_2 par ind_1, si le littéral 2 est considéré comme une information pertinente sur laquelle la préférence et la permutation $\sigma = (l_1(ind_1), l_1(ind_2))(l_2(ind_1), l_2(ind_2))(l_3(ind_1), l_3(ind_2))$ sont fondées, alors il est facile de voire que σ est une symétrie préférentielle sémantique de la formule A, cela revient au modèle minimal $M= \{ l_1(ind_1), l_1(ind_2), \neg l_2(ind_1), \neg l_2(ind_2), l_3(ind_1), l_3(ind_2)\}$ qui est conservée par σ, donc tous les littéraux sont symétriques deux à deux. Maintenant dans le cas où on ajoute l'information que « ind_2 ne vérifie pas le littéral l_2 », alors les littéraux restent toujours symétriques deux à deux dans la logique préférentielle ce qui n'est pas vrai dans la logique classique. L'explication à cela est que dans la logique préférentielle, toujours le modèle préférentiel minimal qui est utilisé et qui conserve les symétries par contre, dans le cas de la logique classique, plusieurs modèles peuvent être utilisés dont certains modèles d'entre eux ne vérifie pas la symétrie.

Du fait que la définition de la symétrie syntaxique dans les logiques de modèles préférentiels semble ne pas être triviale, Benhamou donne la définition de la symétrie syntaxique dans une autre logique non classique ou il choisit la X-logique [49] comme un cadre de référence, cette dernière à l'avantage de posséder des propriétés syntaxiques très importantes.

2.3.2 Symétrie dans la X-logique

Dans cette partie, Benhamou donne une définition de la X-logique et étudie par la suite la symétrie syntaxique dans cette logique.

2.3.2.1 Préliminaires

Définition 12 (X-logique) *Soit X un ensemble de formules de la logique propositionnelle L (X n'est pas nécessairement déductivement clos). La relation d'inférence non monotones* $\vdash_X$ *est définie par* $A \vdash_X B$ *si et seulement si* $(A \cup B) \cap X \subseteq \bar{A} \cap X$.

En d'autres termes, $A \vdash_X B$ si tout théorème de (A U B), ce qui est dans X, est aussi un théorème de ($\bar{A}$). Cela veut dire, en ajoutant la connaissance B à A, l'ensemble des théorèmes qui sont en X n'augmente pas. Comme l'inférence logique classique $\vdash$ est monotone, alors nous avons également $\bar{A} \cap X \subseteq (A \cup B) \cap X$, et par conséquent, il est possible de définir l'inférence dans la X-logique par $A \vdash_X B$ si et seulement si $(A \cup B) \cap X = \bar{A} \cap X$.

Pour le cas particulier où X = F (l'ensemble de toutes les formules possibles de la logique L), l'inférence $\vdash_X$ est identique à l'inférence classique $\vdash$. En revanche, si X est vide, alors toute formule B peut être déduite par A.

L'ensemble X peut être considéré comme un potentiomètre qui règle l'inférence. Intuitivement, si une formule A encode une information (des connaissances, ou de certaines croyances..), X peut être considérée comme l'ensemble des informations "pertinentes". L'ensemble A implique un ensemble d'information B, pour l'inférence dans la X-logique, si l'ajout de B à A ne produit pas de formules plus pertinentes que celles produites par A seul.

2.3.2.2 Symétrie

Benhamou étend la définition de la symétrie syntaxique au cadre de la X-logique et donne une règle étendue de la symétrie qui peut être utilisée pour faire des courtes démonstrations en utilisant des formules symétriques dans ce cadre.

Définition 13 *Soient A un ensemble de formules de la logique propositionnelle, X le sous-ensemble de formules pertinentes sur lequel l'inférence* $\vdash_X$ *de la X-logique est construite et σ une permutation de littéraux. La permutation* σ *est une symétrie syntaxique de A dans la X-logique considérée, si les conditions suivantes sont vérifiées :*

1. σ (*A*) = *A*,
2. σ(X) = X

Par la suite, Benhamou étend la règle de la symétrie de Krishnamurthy au cadre de la X-logique.

Proposition 6 Soient *A* et *B* deux formules ou deux ensembles de formules de la logique classique L et σ une symétrie syntaxique de A dans la X-logique considérée. Nous avons la règle suivante :

$$\frac{A \vdash_X B}{A \vdash_X \sigma(B)}$$

La dernière notion étudiée par Benhamou est la notion de symétrie dans la logique des défauts qui est une logique non monotone introduite par Reiter [50].

2.3.3 Symétrie dans la logique des défauts

Dans cette partie on présente la symétrie dans la logique des défauts, après présentation de quelques notions préliminaires de cette logique.

2.3.3.1 Préliminaires

2. La symétrie dans la programmation logique

Définition 14 *Une théorie des défauts T est une paire <D, W> où W est un ensemble de formules logiques de premier ordre, appelé la théorie de base, qui formalisent les faits qui sont connus avec certitude. D est un ensemble de règles de défaut, chacune étant de la forme :*

$$\frac{\textit{Prérequis} : \textit{Justification}_1, \ldots, \textit{Justification}_n}{\textit{Conclusion}}$$

Intuitivement, cela signifie que par défaut, si le *Prérequis* est vrai, et chaque $Justification_i$ pour tout $i \in \{1, \ldots, n\}$ est consistante avec nos croyances actuelles, alors on est amenés à croire que *Conclusion* est vraie, et on l'infère (on l'ajoute à la théorie). Pour définir formellement le sens d'une règle de défaut, on a besoin d'introduire avant cela, la notion importante d'extensions dans une logique des défauts.

Une extension d'une théorie des défauts $T = <D, W>$ est un ensemble de formules déductivement clos E contenant W et qui vérifie : si $\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D$ est un défaut tel que $\alpha \in E$ et $\forall i \in \{1, \ldots, n\}$, $\neg\beta_i$ n'appartient pas à E, alors $\gamma \in E$.

Formellement :

Définition 15 *Si Th(E) est l'ensemble des conséquences logiques de l'ensemble des formules E, alors E est une extension de la théorie des défauts T = <D, W> si et seulement si* $E = U_{i=0,\ldots,\infty}$ *Ei et les conditions suivantes sont vérifiées* : $E_0 = W$, et $\forall i \geq 0$, $E_{i+1} = Th(E_i) \cup \{\gamma :$
$\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma} \in D, \alpha \in E_i, ¥ j \in \{1, \ldots, n\}, \neg\beta_j$ *n'appartient pas à E*$\}$.

Une règle de défaut $\frac{\alpha : \beta_1, \ldots, \beta_n}{\gamma}$ peut être appliquée à une théorie donnée des défauts $T = <D, W>$ Si son prérequis α est dans W, et la négation de chacun de ses justifications $\neg\beta_j$ n'est pas dans l'extension E.

On retire quelques remarques :

- Lorsque tous les défauts de la théorie sont normaux (i.e. sous la forme $\alpha: \beta$), la condition de la justification $\frac{\gamma}{\beta}$ $\neg\beta \in E$ est simplifiée en $\neg\beta \in E_i$.

On obtient alors une méthode plus simple et constructive pour construire l'extension E.

- Les règles de défaut peuvent être appliquées dans un ordre différent et cela peut conduire à différentes extensions *E* pour la même théorie *T*.
- Si un défaut contient des formules avec des variables libres, il est considéré comme représentant de l'ensemble de tous les défauts obtenus en donnant une valeur à toutes ces variables.

La notion la plus importante dans une logique des défauts est le calcul des extensions d'une théorie des défauts. Selon cet ensemble d'extensions, les chercheurs ont défini des sémantiques différentes pour la logique des défauts. L'implication d'une formule depuis d'une théorie des défauts $T = <D, W>$ peut être défini en différentes façons :

Définition 16 *étant donné une théorie des défauts* $T = <D, W>$ *et l'ensemble de tous ses extensions* E_T.

– ***Approche sceptique*** : une formule *f* est impliqué de la théorie des défauts *T* si elle est impliquée de tous ses extensions, c'est-à-dire $T \vdash_S f$ si et seulement si $\forall E \in E_T, E \vdash f$.

– ***Approche crédule*** : une formule *f* est impliqué de la théorie des défauts *T* si elle est impliquée au moins d'une de ses extensions, c'est-à-dire $T \vdash_C f$ si et seulement si $\exists E \in E_T : E \vdash f$.

– ***Approche semi-crédule*** : une formule *f* est impliqué de la théorie des défauts *T* si elle est impliquée au moins d'une de ses extensions et tous ces extensions n'implique pas sa négation, c'est-à-dire $T \vdash_{SC} f$ si et seulement si $\ni E \in E_T : E \nvdash f$ et $\forall E \in E_T, E \vdash \neg f$.

2.3.3.2 Symétrie

2. La symétrie dans la programmation logique

Benhamou introduit les notions de symétrie syntaxique et de symétrie sémantique dans la logique des défauts et montre l'amélioration de l'implication par la propriété de symétrie.

Définition 17 (Symétrie sémantique) *étant donné une théorie des défauts T = <D, W>, L_T est l'ensemble de ses littéraux et E_T l'ensemble de toutes ses extensions. Une symétrie sémantique σ est une permutation de littéraux définie sur L_T telle que $E_T = E_\sigma(T)$.*

En d'autres termes, une symétrie sémantique d'une théorie des défauts *T* est une permutation de variables qui laisse invariant l'ensemble de ses extensions. Il résulte de là que chaque extension $E_i \in E_T$ est transformé par la symétrie σ à une autre extension $Ej = \sigma(E_i)$. Ces extensions sont ce que nous appelons extensions symétriques. Il est alors possible d'obtenir une famille des extensions symétriques, sans duplication des efforts, si on a que E_i est une extension et on a le groupe de symétrie de la théorie T. Malheureusement, le calcul de la symétrie sémantique est coûteux au niveau de temps, car il a besoin de calculer toutes les extensions.

Définition 18 (Symétrie syntaxique) *étant donné une théorie des défauts T = <D, W>. Une symétrie syntaxique est une permutation σ définie sur l'ensemble L_T de littéraux de T, qui laisse la théorie T invariant. C'est-à-dire σ(T) = T, plus précisément, les conditions suivantes sont vérifiées : σ(D) = D et σ(W) = W.*

Benhamou présente dans le théorème suivant la relation entre la symétrie syntaxique et la symétrie sémantique d'une logique des défauts.

Théorème 2 *étant donné une théorie des défauts T. Si σ est une symétrie syntaxique de T, alors σ est une symétrie sémantique de T.*

Ce théorème est prouvé par le fait que σ est une symétrie syntaxique de *T*, alors σ est une symétrie sémantique de *T*. cela montre que la symétrie sémantique contient la symétrie syntaxique.

2. La symétrie dans la programmation logique

Benhamou introduit la nouvelle règle d'inférence par symétrie dans les logiques des défauts. Par exemple, le cas de l'inférence sceptique $\vdash_S$:

Proposition 7 Soient *T* une théorie des défauts, *f* une formule et σ une symétrie syntaxique de *T*, alors on a la règle suivante :

$$\frac{T \vdash_S f}{T \vdash_S \sigma(f)}$$

Cette règle est également valable pour les crédules ($\vdash_C$) et les demi-crédules ($\vdash_{SC}$) inférences.

La proposition suivante est une proposition importante qui utilise la symétrie pour le calcul de l'ensemble des extensions.

Proposition 8 étant donné une théorie des défauts $T = <D, W>$, un sous-ensemble de formules *E* et une symétrie syntaxique σ de *T*, alors *E* est une extension de *T* si et seulement si σ(*E*) est une extension de *T*.

Cette proposition est prouvée par le théorème précédent qui montre que la symétrie syntaxique est une symétrie sémantique et donc elle conserve l'ensemble E_T des extensions de *T*.

Proposition 9 étant donné une théorie des défauts $T = <D, W>$, un sous-ensemble D_1 ς *D* et une symétrie syntaxique σ, alors il existe une extension E^{D1} de *T* obtenue par l'application des défauts de D_1, si et seulement si, il existe une extension $E^{\sigma(D1)}$ de *T* obtenue par l'application des défauts de $\sigma(D_1)$.

Une autre proposition montre que si l'application d'un défaut conduit à une extension de la théorie considérée, alors l'application de chacun de ses défauts symétriques conduit aussi à une extension de la même théorie.

Proposition 10 Si $T = <D, W>$ est une théorie des défauts et σ un de ses symétries syntaxiques. Il existe une extension de *T* où le défaut *d* est appliqué, si et seulement si, il existe une extension où le défaut symétrique σ(*d*) est appliqué.

Le corollaire suivant est déduit à partir de la proposition précédente :

Corollaire 1 *étant donné une théorie des défauts T = <D, W>, un défaut d ϵ D et une symétrie syntaxique σ de T, alors il n'existe pas une extension de T où le défaut d est appliqué, si et seulement si, il n'existe pas une extension où le défaut symétrique σ(d) est appliqué.*

2.4 Conclusion

Ce chapitre a été divisé en trois grandes parties, la première c'est une aperçue de la symétrie dans les logiques classiques, un cas a été étudié c'est la logique propositionnelle, dans la deuxième partie on a vu comment la notion de la symétrie a été étendue aux formalismes logiques non classiques. Donc on a vu la notion de la symétrie sémantique dans les logiques préférentielles et comment certaine information peut être déduite à l'aide de cette notion, alors que ces déductions ne peuvent pas être faites dans une logique classique, le seul inconvénient de cette approche et qu'elle n'exploite pas la symétrie syntaxique. L'autre point qu'on a vu est l'extension de la définition de la symétrie syntaxique dans le cadre de la X-logique où une nouvelle règle de déduction a été introduite, aussi dans cette logique la symétrie sémantique n'est pas encore prise en compte. Par la suite, on a vu la définition de la symétrie sémantique et syntaxique dans le cadre le plus général de la logique des défauts. On a vu comment la symétrie peut être utilisée pour améliorer la recherche des extensions et on a vu la nouvelle règle d'inférence par symétrie introduite qui peut être utilisée pour faire de courtes démonstrations. De manière générale à travers cette étude on a vu la rentabilité du raisonnement par symétrie pour ces logiques et comment ces dernières gèrent certaines symétries qui n'existent pas dans des logiques classiques.

3 La symétrie dans la programmation par contraintes

3.1 Introduction

L'étude de la symétrie dans les problèmes de contraintes a toujours été importante, mais ces dernières années elle est devenue un domaine de recherche majeur dans son propre droit. Un problème clé dans la programmation par contraintes est reconnu depuis longtemps: la recherche peut revisiter des états équivalents un très grand nombre de fois. En principe, ce problème a été résolu, avec un certain nombre de techniques différentes. Cependant, la recherche demeure très active pour deux raisons principales. Tout d'abord, il y'a beaucoup de difficultés dans l'application pratique des techniques utilisées pour l'exclusion de la symétrie, et de surmonter celles-ci demeurent des problèmes de recherche importants. Deuxièmement, les succès obtenus dans ce domaine jusqu'à présent ont encouragé les chercheurs à trouver de nouvelles façons d'exploiter la symétrie. Ce chapitre est devisé en trois parties, on commence par définir la programmation par contraintes et nous donnons quelques notions et préliminaires nécessaires, par la suite nous expliquons les symétries dans la programmation par contraintes et nous donnons les différentes définitions pour la symétrie et enfin nous expliquons en détails les différentes techniques pour l'élimination des symétries dans la programmation par contrainte.

3.2 La programmation par contraintes

3.2.1 Définition

La programmation par contraintes (PPC) fournit un cadre et des méthodes pour la résolution de problèmes de la recherche opérationnelle. Les problèmes de satisfaction de contraintes (CSP) sont exprimés en termes de variables (les inconnues du problème) et de contraintes qui sont des relations entre ces variables. La résolution d'un CSP consiste à affecter une valeur aux différentes variables de telle manière que les contraintes soient respectées. L'intérêt de la PPC est l'indépendance entre le modèle et la recherche de solution : La modélisation d'un problème ne présuppose aucune m´méthode de résolution de celui-ci.

3.2.2 Les contraintes

Les contraintes sont les relations qui lient entre elles les variables du problème en restreignant les combinaisons de valeurs qu'elles peuvent prendre simultanément. Comme pour les problèmes de programmation mathématique, ces contraintes sont :

- ✓ *déclaratives* : elles ne décrivent pas de procédure chargée de les faire respecter ;
- ✓ *Non-directionnelles* : on ne distingue pas de variables d'entrées et de variables de sortie ;
- ✓ *additives* : toutes les contraintes d'un CSP doivent être satisfaites simultanément ; la propriété à vérifier est donc implicitement la conjonction de toutes les contraintes.

3.2.3 Les domaines

La généricité de l'expression des contraintes permet d'avoir le même formalisme pour des problèmes portant sur des variables ayant des domaines de types variés. En effet, toutes les structures mathématiques dotées d'une théorie équationnelle d´décidable peuvent intervenir dans l'expression d'un CSP : booléens, entiers, réels, ensembles,...

Pour la prise en compte des symétries, nous nous intéresserons plus particulièrement aux CSP sur les domaines finis.

3.2.4 Les problèmes de satisfaction de contraintes

Le cadre de la PPC est très générique et fait intervenir des domaines quelconques ainsi que des relations quelconques entre les variables. Cependant, pour la prise en compte des symétries, nous limiterons notre étude au cas des domaines finis. Nous donnons ici le formalisme des CSP sur les domaines finis, ainsi que des méthodes constructives de résolution basiques.

3.2.4.1 Définition

Les définitions présentées ici sont les d´définitions classiques du formalisme des CSP sur les domaines finis.

Définition 1 (CSP sur les domaines finis) : Un CSP sur les domaines finis est défini par un quadruplet P = (X, V, D, C), où :

- $X = \{x_1, \ldots, x_n\}$ est un ensemble fini de variables.
- V est un ensemble fini de valeurs.
- $D = \{D_1, \ldots, D_n\}$ est un ensemble fini des domaines de cardinal fini. Le domaine

Di de la variable xi est l'ensemble des valeurs qu'elle peut prendre. En outre, $\forall i \in \{1, 2, \ldots, n\}$, $D_i \subset V$.

- $C = \{c_1, \ldots, c_m\}$ est l'ensemble des contraintes. Chaque contrainte $ci = (X_i, R_i)$ est définie par l'ensemble Xi _ X des variables sur lesquelles elle porte et par la relation Ri définie en extension par l'ensemble des valeurs que peuvent prendre simultanément les variables de X_i :
- $Xi = \{x_{i1}, \ldots, x_{iki}\}$. Le cardinal $k_i = |X_i|$ est appelé arité de la contrainte.
- $Ri \subset D_{i1} \times \cdots \times D_{iki}$ est un sous-ensemble du produit cartésien des domaines de chaque variable de X_i.

Exemple 1 (Un premier problème) : Soient trois variables $x \in \{1, 2\}$, $y \in \{2, 3, 4\}$, $z \in \{1, 2, 3\}$. On cherche à instancier ces variables de telle manière qu'elles soient différentes deux à deux. Ce problème se modélise par le CSP suivant :

$$
\mathbf{P} \begin{cases} \mathbf{X} = \{x, y, z\} \\ \mathbf{V} = \{1, 2, 3, 4\} \\ \mathbf{D} = \{\{1, 2\}, \{2, 3, 4\}, \{1, 2, 3\}\} \\ \mathbf{C} = \begin{cases} (\{x, y\}, \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4)\}), \\ (\{x, z\}, \{(1, 2), (1, 3), (2, 1), (2, 3)\}), \\ (\{y, z\}, \{(2, 1), (2, 3), (3, 1), (3, 2), (4, 1), (4, 2), (4, 3)\}) \end{cases} \end{cases}
$$

Contraintes en intention : La définition 1 utilise une expression dite en extension des contraintes, c'est-à-dire la liste exhaustive des combinaisons de valeurs

autorisées. Mais la structure de nombreux problèmes applicatifs utilise des contraintes exprimées à l'aide de :

– formules analytiques ;

– équations mathématiques : $(x+y \leq 4)$;

– équations avec des opérateurs d'ensembles ;

– opérateurs logiques : $(x \leq 1 \Rightarrow y = 2)$;

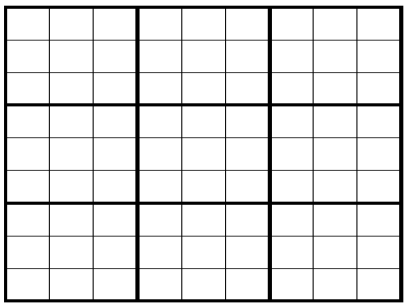
– spécifications de plus haut niveau (e.g. « toutes ces variables sont différentes », que nous noterons alldiff).

Ces contraintes sont exprimées en intention. Ceci n'affecte pas la définition 1 puisqu'on peut exprimer en extension toute contrainte en intention. Dans la notation d'une contrainte en intention, on pourra omettre l'ensemble de variables sur lesquelles elle porte car celles-ci apparaissent dans l'expression de la contrainte.

L'exemple suivant illustre cette notion par la modélisation du problème de Sudoku.

Problème 1 (Sudoku) : Le problème du Sudoku de taille n consiste à remplir une grille de taille $n^2 \times n^2$ séparée en n^2 blocs de taille $n \times n$ avec des nombres de 1 à n^2, de telle manière que dans chaque ligne, chaque colonne et chaque bloc, les n^2 éléments sont différents deux à deux.

La figure 4.1(a) montre une grille vide de Sudoku de taille 3.



(a) Grille de Sudoku de taille 3

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n^2} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n^2} \\ \cdots & & & \\ \cdots & & & \\ \cdots & & & \\ x_{n^2,1} & x_{n^2,2} & \cdots & x_{n^2,n^2} \end{pmatrix}$$

$\forall i, j \in \{1, 2, \ldots, n^2\}, x_{i,j} \in [|1, n^2|]$

(b) Variables du modèle

Figure 3-1 Variables de décision pour la modélisation du Sudoku par un CSP

Associons comme indiqué sur la figure 1.1(b) une variable à chaque case de la grille. Le CSP qui traduit ce choix est le suivant :

$$
\mathbf{P_1}\begin{cases}
\mathbf{X} = \{x_{i,j}\ ,\ 1 \le i \le n^2,\ 1 \le j \le n^2\} \\
\mathbf{V} = \{1, 2, \ldots, n^2\} \\
\mathbf{D} = \{D_{i,j} = [|1, n^2|]\ ,\ 1 \le i \le n^2\ ,\ 1 \le j \le n^2\ \} \\
\mathbf{C} = \begin{cases}
\mathbf{U}_{i=1}^{n2}\ \{x_{i,k} \neq x_{i,l}\ ,\ 1 \le k < l \le n^2\ \} & \text{lignes} \\
\mathbf{U} \\
\mathbf{U}_{j=1}^{n2}\ \{x_{k,j} \neq x_{l,j}\ ,\ 1 \le k < l \le n^2\} & \text{colonnes} \\
\mathbf{U} \\
\mathbf{U}_{k=1}^{n2}\ \{x_{i,j} \neq x_{i',j'}\ ,\ (i,j) \neq (i',j'),\ (i,i') \in J_k^2\ \} & \text{blocs}
\end{cases}
\end{cases}
$$

avec

$$I_k = [|n[k/n] + 1, n[k/n] + n$$
$$J_k = [|n((k-1) \bmod n) + 1,\ n((k-1) \bmod n) + n|]$$

Ce modèle possède n4 variables, ayant chacune un domaine de taille n2. L'espace de recherche pour ce modèle a donc une taille de $(n^2)n^4$.

L'utilisation de contraintes globales alldiff permet une expression plus naturelle du problème, qui se modélise alors par le CSP suivant :

$$
\mathbf{P_1}\begin{cases}
\mathbf{X} = \{x_{i,j}\ ,\ 1 \le i \le n^2,\ 1 \le j \le n^2\} \\
\mathbf{V} = \{1, 2, \ldots, n^2\} \\
\mathbf{D} = \{D_{i,j} = [|1, n^2|]\ ,\ 1 \le i \le n^2\ ,\ 1 \le j \le n^2\ \} \\
\mathbf{C} = \begin{cases}
\mathbf{U}_{i=1}^{n2}\ \{x_{i,k} \neq x_{i,l}\ ,\ 1 \le k < l \le n^2\ \} & \text{lignes} \\
\mathbf{U} \\
\mathbf{U}_{j=1}^{n2}\ \{x_{k,j} \neq x_{l,j}\ ,\ 1 \le k < l \le n^2\} & \text{colonnes} \\
\mathbf{U} \\
\mathbf{U}_{k=1}^{n2}\ \{x_{i,j} \neq x_{i',j'}\ ,\ (i,j) \neq (i',j'),\ (i,i') \in J_k^2\ \} & \text{blocs}
\end{cases}
\end{cases}
$$

avec

$$I_k = [|n[k/n] + 1, n[k/n] + n$$
$$J_k = [|n((k-1) \bmod n) + 1,\ n((k-1) \bmod n) + n|]$$

La contrainte alldiff(x_1, . . ., x_n) spécifie que les variables x_1, . . ., x_n doivent être instanciées à des valeurs différentes.

- ***Satisfaction de contraintes*** : Pour s'abstraire des types de contraintes utilisées lors de la définition d'un CSP, on associe à chaque contrainte une fonction de

satisfaction qui indique si la contrainte est vérifiée pour une instanciation donnée.

Définition 2 (Fonction de satisfaction) : Soit P = (X, V,D , C) un CSP. On associe à chaque contrainte $c_i = (X_i,R_i) \epsilon C$ une fonction booléenne de satisfaction définie par :

$\tilde{c}_i : D_{i1} \times \cdots \times D_{iki}$ --> {vrai, faux}

$(v_{i1}, \ldots, v_{iki})$ --> $\tilde{c}_i (v_{i1}, \ldots, v_{iki}) =$ {vrai si $(v_{i1}, \ldots, v_{iki}) \epsilon R_i$
faux sinon.

Définition 3 (Instanciation) : Soit P = (X, V, D, C) un CSP.

– On appelle littéral une affectation de la forme xi = j avec xi 2 X, j 2 Di.

– Une instanciation totale est un ensemble de littéraux, un pour chaque variable de P.

– Une instanciation partielle est un sous-ensemble d'une instanciation totale.

Pour une instanciation partielle I, on notera XI l'ensemble des variables sur lesquelles elle porte.

On utilisera la notation $I(x_i) = v_i$ pour indiquer que la variable x_i est affectée à la valeur v_i par l'instanciation I. On notera également $\tilde{c}_i(I)$ l'application de la fonction de satisfaction d'une contrainte c_i à la restriction de I sur les variables de X_i si $X_i \varsigma X_I$.

Définition 4 (Satisfaction de contrainte) : Soit P = (X, V, D, C) un CSP et soit $c_i = (X_i,R_i) \epsilon C$ une contrainte. On dit qu'une instanciation partielle I telle que $X_i \varsigma X_I$:

- satisfait la contrainte c_i si et seulement si $\tilde{c}_i(I)$ = vrai, on notera I |= ci ;
- viole la contrainte c_i si et seulement si $\tilde{c}_i(I)$ = faux.

Pour qu'une instanciation partielle puisse mener à une solution, il faut que toutes les

Contraintes concernées par cette instanciation soient satisfaites ; on parle d'instanciation consistante :

Définition 5 (Instanciation consistante) : Soit P = (X, V, D, C) un CSP, une instanciation

I est consistante si et seulement si :

$$\forall\ c_i = (X_i,R_i) \epsilon C \text{ telle que } X_i \subseteq X_I,\ I \models c_i$$

Définition 6 (Solution) : Soit P = (X, V, D, C) un CSP. Une solution S de P est une instanciation totale et consistante. On notera S |= P et on appellera SP l'ensemble des solutions de P.

- ***Types de CSP :***

Définition 7 (CSP binaire) : un CSP binaire est un CSP dont les contraintes ne portent au plus que sur deux variables.

Définition 8 (CSP n-aires) : un CSP n-aire est un CSP d'arité supérieure à deux. Un CSP n-aire est nécessairement Np-complet et on peut le réduire en un CSP binaire.

Définition 9 (Binarisation d'un CSP) : La binarisation passe par la création de nouvelles variables, chacune < encapsulant > une contrainte d'arité supérieure à 2. Par exemple, dans un CSP à 3 variables toutes définies sur le domaine D = {0, 1, 2}, la contrainte X1 + X2 = X3 peut être encapsulée dans une nouvelle variable U représentant le triplet (X_1, X_2, X_3) avec pour domaine {(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 2), (0, 2, 2), (2, 0, 2)}. De nouvelles contraintes, binaires cette fois, de type element, lient la nouvelle variable avec chacune des variables du triplet : la contrainte liant U et X_2 s'écrit element(X_2,U,2) (X_2 est le second élément de U) ou, plus formellement, $C_{(U,2)}$ = {U,X_2} et $R_{(U,2)}$ = {((0, 0, 0), 0), ((0, 1, 1), 1), ((1, 0, 1), 0), ((1, 1, 2), 1), ((0, 2, 2), 2), ((2, 0, 2), 0)}.

- ***Contraintes n-aires :***

Définition 10 : malgré que les contraintes n-aires ou les contraintes globales présentent un aspect important de la programmation par contraintes, il n'existe pas une définition claire de ce qui est une contrainte globale. Une contrainte globale est une contrainte sur une séquence de variables.

- ***Exemples de contraintes n-aires :***

Un exemple célèbre de contrainte globale est la contrainte **all-different**, pour spécifier qu'un ensemble de variables doivent prendre des valeurs toutes différentes. D'autres ensemble de contraintes globales appliquées largement comme : la contrainte globale **cumulative** [51] s'emploie pour modéliser les contraintes de ressources intervenant, Elle indique que, µa chaque instant, la quantité d'une ressource requise par l'ensemble des activités en cours d'exécution n'excède pas la quantité disponible de la ressource. Formellement, la contrainte globale cumulative :

$$([S_1, p_1, r_1], \ldots, [S_n, p_n, r_n], R, T)$$

Est satisfaite si les inégalités suivantes sont vérifiées :

$$\Sigma_{i:Si\leq t<Si+pi}\ r_i \leq R, \forall\ t \in \{0, \ldots., T\},$$
$$S_i + p_i\ \leq T\ \forall\ i \in \{1, \ldots., n\},$$

Ou R désigne la capacité de la ressource et, pour chaque activité $i \in \{1, \ldots, n\}$, S_i, p_i et r_i désignent respectivement sa date de début (variable), sa durée et sa consommation sur la ressource.

Aussi la contrainte globale de cardinalité (**gcc**) sur un ensemble de variables et de valeurs avec le nombre de variables instanciés en valeurs doit être compris entre une limité supérieure et une limite inférieure donnée. Ces deux types de contraintes surviennent fréquemment dans l'ordonnancement, séquencement et planification des applications.

Pour plus de détails Beldiceanu et Demassey [52] proposent une classification complète des contraintes globales.

- ***Puissance des contraintes globales*** **:** l'utilisation des contraintes globales à deux avantages :

✓ elles facilitent la tâche de modélisation d'une application en CSP.

✓ Elles sont généralement utilisées avec des algorithmes de propagation de contraintes et filtrage qui en tirant la sémantique de la contrainte globale, ces algorithmes deviennent plus efficaces. Ceci sera plus détaillé dans la section propagation de contraintes dans la suite du chapitre.

Définition 11 (Graphe de contraintes) : Le graphe de contraintes est un graphe simple où les sommets représentent les variables et les arêtes représentent les contraintes. Dans le graphe de contraintes d'un CSP binaire il y a une arête entre les sommets représentant les variables x et y si et seulement s'il existe une contrainte c telle que $var(c) = \{x, y\}$. Le concept de graphe de contraintes peut être défini pour des CSP quelconques (non binaires). Les contraintes ne sont alors pas représentées par des arêtes mais par des hyper-arêtes qui relient des sous-ensembles de variables impliquées dans une même contrainte. Dans ce cas, on a un hypergraphe de contraintes.

Définition 12 (Problème d'optimisation sous contraintes) : Un problème d'optimisation sous contraintes (COP) est un CSP augmenté d'une fonction objectif f. Cette fonction est souvent modélisée par une variable dont le domaine est défini par les bornes supérieures et inférieures de f.

Exemple 2 : le problème des *n* reines : Le but de ce problème, inspiré du jeu d'échec, est de placer n reines sur un échiquier de dimension $n \times n$ de manière à ce

qu'aucune ne soit en prise. Deux reines sont en prises si elles sont sur la même ligne, la même colonne ou la même diagonale. Un modèle classique sans contraintes globales est défini dans la figure 4.2. En observant que deux reines ne peuvent pas être placées sur la même colonne, on peut imposer que la reine *i* soit sur la colonne *i*. Ainsi, la variable *li* de domaine {1, . . ., *n*} représente la ligne où est placée la reine dans la colonne *i*. Les contraintes (2.1) imposent que les reines soient sur des lignes différentes alors que les contraintes (2.2) et (2.3) imposent que deux reines soient placées sur des diagonales différentes.

$$l_i \neq l_j \qquad 1 \leq i < j \leq n \qquad (2.1)$$

$$l_i \neq l_j + (j - i) \qquad 1 \leq i < j \leq n \qquad (2.2)$$

$$l_i \neq l_j - (j - i) \qquad 1 \leq i < j \leq n \qquad (2.3)$$

La figure 2.2 montre plusieurs affectations partielles pour le problème des 4 reines dans lesquelles le placement d'une reine sur l'échiquier est représenté par un cercle dans la case correspondante. Les reines sont ordonnées de gauche à droite et les positions de haut en bas.

On peut observer en figure 2.2 que contrairement aux affectations totales $\{l_1 \leftarrow 2, l_2 \leftarrow 4, l_3 \leftarrow 1, l_4 \leftarrow 3\}$ et $\{l_1 \leftarrow 3, l_2 \leftarrow 1, l_3 \leftarrow 4, l_4 \leftarrow 2$, l'affectation partielle $\{l_1 \leftarrow 1, l_2 \leftarrow 3, l_3 \leftarrow 2\}$ n'est pas consistante car elle viole une des contraintes (2.2).

Si l'on définit un COP à partir du problème des *n* reines en définissant la fonction objectif $min(l_1)$, alors le problème admet une unique solution optimale : $\{l_1 \leftarrow 2, l_2 \leftarrow$ 4,

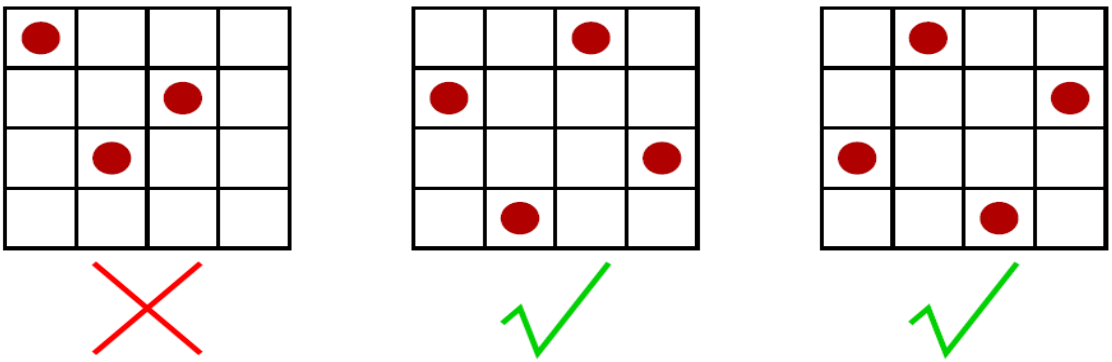


Figure 3-2 le problème des 4 reines

3.2.4.2 La résolution d'un problème de satisfaction de contraintes

3. La symétrie dans la programmation par contraintes

La principale technique algorithmique pour résoudre les problèmes de satisfaction de contraintes est la recherche. Un algorithme de recherche pour résoudre un CSP peut être complet ou incomplet. Les algorithmes complets, ou systématiques, donnent la garantie qu'une solution sera trouvée si elle existe, et peuvent être utilisés pour montrer qu'un CSP n'a pas de solution et aussi pour trouver une solution optimale. Les algorithmes incomplets ou non systématiques, ne peuvent pas être utilisés pour montrer qu'un CSP n'a pas de solution ou pour trouver une solution optimale. Cependant, ces algorithmes sont souvent efficaces pour trouver une solution si elle existe et peuvent être utilisés pour trouver une approximation de la solution optimale.

3.2.4.2.1 Algorithmes simples de recherche (recherche systématique)

Les méthodes de recherche de solutions pour un CSP P = (X, V, D, C) consistent en une exploration de l'espace de recherche. Pour effectuer cette exploration, on structure la recherche sous la forme d'un arbre dont chaque noeud correspond à une instanciation partielle, et ses fils à l'extension de cette instanciation avec une nouvelle variable, chaque fils correspondant à une valeur dans le domaine de cette variable.

- **Générer et tester (Recherche exhaustive) :**

L'algorithme generate-and-test (GT) énumère les affectations totales et vérifie leur consistance, c'est-à-dire qu'aucune contrainte n'est violée. En général, les affectations sont énumérées grâce à une recherche arborescente qui instancie itérativement les variables. Cet algorithme ne réveille les contraintes que lorsqu'une affectation est totale et Chaque vérification de contrainte s'effectue en temps polynomial, L'algorithme GT est très couteux en temps de calcul, puisque il considère par contre un nombre exponentiel d'affectations (parcourir la totalité de l'espace de recherche) le rendant impraticable même pour des problèmes de petite taille.
L'algorithme GT maintient l'ensemble des variables non instanciées N et l'instanciation partielle I qui porte sur les autres variables (X − N). Au départ, aucune variable n'est instanciée donc la fonction de l'algorithme 1 est appliquée à l'ensemble des variables du CSP et à l'ensemble vide : générer et tester(X, Ø). Ensuite, à chaque

appel récursif, on choisit une nouvelle variable x de N et on complète I avec une valeur de son domaine. Lorsque I est totale (i.e. toutes les variables sont instanciées, N = Ø), on vérifie si elle vérifie toutes les contraintes. Si c'est le cas, on renvoie la solution, sinon on continue d'explorer l'arbre.

- **Backtrack searching (retour arrière) :**

Le problème de l'algorithme GT est que le test de satisfaction des contraintes n'est effectué qu'aux feuilles (i.e. instanciations totales) de l'arbre. La méthode d'extension d'un nœud dans l'arbre de recherche s'appelle une stratégie de branchement. L'algorithme appelé « backtrack » (BT) vérifie la consistance à tous les noeuds de l'arbre. D`es qu'une variable x est instanciée, on vérifie la satisfaction des contraintes c = (X_c,R_c) telles que x ϵ Xc. Si une instanciation partielle est inconsistante avec une contrainte c, alors toutes ses extensions seront également inconsistantes avec c. Dans ce cas, on peut élaguer le sous arbre correspondant en effectuant un backtrack (retour- arrière) jusqu'`a la première instanciation qui offre une autre branche non encore explorée.

On modifie l'algorithme GT pour obtenir l'algorithme 2, de manière à tester la consistance à chaque noeud. Afin de ne pas tester plusieurs fois des contraintes dont on sait qu'elles sont consistantes à chaque nouvelle valeur essayée, on se contente de vérifier celles qui concernent la variable x, c'est-à-dire celles qui vérifient :

$$c \in C \wedge x \in X_c \wedge X_c \varsigma X_I,$$

La figure 1.2 montre le parcours de l'arbre de recherche de solution pour le CSP de l'exemple 1.

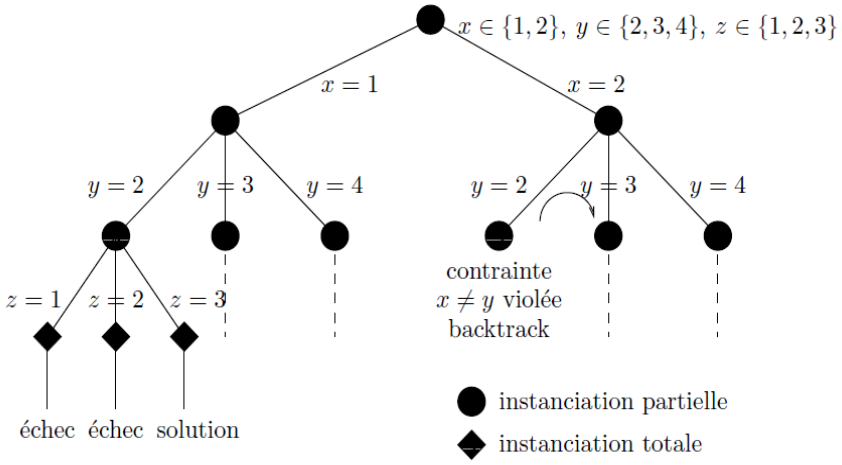


Figure 3-3 Parcours d'un arbre de recherche par backtrack

Dans cette technique le nombre d'affectations considéré est ainsi considérablement réduit, mais en revanche, la consistance des contraintes est vérifiée à chaque affectation partielle.
La découverte redondante d'inconsistance locale due à la perte d'information sur l'inconsistance d'affectation partielle dégrade les performances de l'algorithme backtrack.

Depuis les premières utilisations des algorithmes de Backtrack en informatique [53, 54], de nombreuses techniques pour améliorer l'efficacité d'un algorithme de recherche de Backtrack ont été proposées et évaluées. Les techniques les plus importants sont la propagation de contraintes, l'enregistrement de nogood, les heuristiques pour l'ordre des variables et des valeurs et les stratégies de redémarrage. La meilleure combinaison de ces techniques entraîner des algorithmes de Backtrack robustes qui peuvent résoudre les grandes instances combinatoires et difficiles qui sont d'une importance pratique.

3.2.4.2.2 Filtrage et propagation de contraintes

Le filtrage et la propagation des contraintes [55] permettent d'améliorer les capacités de résolution des CSP lors de la construction d'un arbre de recherche. Une fonction associée à chaque contrainte réalise le filtrage des domaines, c'est-à-dire qu'elle supprime les valeurs inconsistantes des domaines de ces variables. La propagation calcule un point fixe global qui est atteint lorsque les techniques de consistance locale ne peuvent plus réaliser aucune inférence.

Nous rappelons les principales techniques de consistance. Une contrainte est dite arc consistante si pour chaque valeur de chaque variable x, il existe une affectation des autres variables telle que la contrainte soit satisfaite. Néanmoins, il est insuffisant d'effectuer une révision unique par contrainte pour atteindre le point fixe global. Par conséquent, l'algorithme AC-1 révise toutes les contraintes jusqu'à ce qu'aucun domaine n'ait changé. L'algorithme AC-3 utilise une file de contraintes à réviser dans laquelle il ajoute les contraintes portant sur une variable dont le domaine a changé. On pourra se référer à [56] pour plus de détails à propos de ces améliorations.

Ainsi, à chaque noeud de l'arbre de recherche, c'est-à-dire à chaque instanciation d'une variable, l'algorithme de recherche applique des techniques de

consistance locale, propre à chaque contrainte, qui retire des valeurs inconsistantes des domaines. À leur tour, les modifications des domaines peuvent inférer de nouvelles inconsistances. Ce mécanisme, appelé propagation, continue jusqu'à ce qu'un point fixe global soit atteint. L'algorithme fait alors un nouveau choix de variable et de valeur et teste la nouvelle instanciation. Si durant le filtrage d'une contrainte, le domaine d'une variable devient vide, alors l'instanciation courante est inconsistante. Le noeud est fermé et l'algorithme passe au noeud suivant s'il existe. Cette opération permet d'agir sur les domaines de toutes les variables d'un CSP durant la construction des affectations partielles réduisant ainsi le nombre de noeuds composant l'arbre.

3.2.4.2.3 Algorithmes de recherche avancés

Une technique prospective simple pour anticiper les effets d'une instanciation est nommée ***forward checking***. On vérifie que les variables non instanciées peuvent chacune prendre une valeur consistante lorsque l'affectation partielle courante est étendue par l'instanciation d'une nouvelle variable. Les techniques de ***look-ahead*** sont plus lentes, mais procurent un meilleur filtrage basé sur l'arc-consistance. L'arc-consistance est appliquée sur toutes les variables non instanciées pour chaque affectation étendue. Ainsi, les techniques de forward checking considèrent une seule variable non instanciée à la fois pour la consistance alors que celles de look-ahead considèrent une paire de variables non instanciées.

D'un autre côté, les techniques rétrospectives remettent en cause l'affectation partielle lorsqu'une inconsistance est détectée lors de la propagation. Le ***backtrack chronologique*** consiste simplement à remettre en cause le dernier choix. Cet algorithme est sensible au phénomène de *thrashing* où des inconsistances dues à un nombre restreint d'instanciations sont redécouvertes de manière redondante lors de l'exploration de l'arbre de recherche

Le conflit directed backjumping ou ***intelligent backtracking*** [57] consiste à calculer une explication à un backtrack puis à remettre en cause le choix le plus récent de cette explication. Chaque échec sur le choix d'une valeur est expliqué par les précédents choix qui entrent en conflit. Si toutes les valeurs d'un domaine ont été testées sans succès, l'explication de cet échec est l'union des explications des valeurs du domaine.

Le dynamic backtracking [58] utilise un mécanisme similaire mais sans remettre en cause les choix indépendants de l'explication.

Le nogood recording [59] consiste à mémoriser les causes des conflits.

En effet, Demassey [60] montre dans sa thèse que pour améliorer la recherche d'une solution, il est nécessaire d'étudier les combinaisons possibles entre les stratégies et méthodes de branchements utilisées afin de trouver le meilleur compromis entre le nombre de nœuds du graphe de recherche et le temps de résolution passé à chaque nœud.

3.3 La symétrie dans la programmation par contraintes

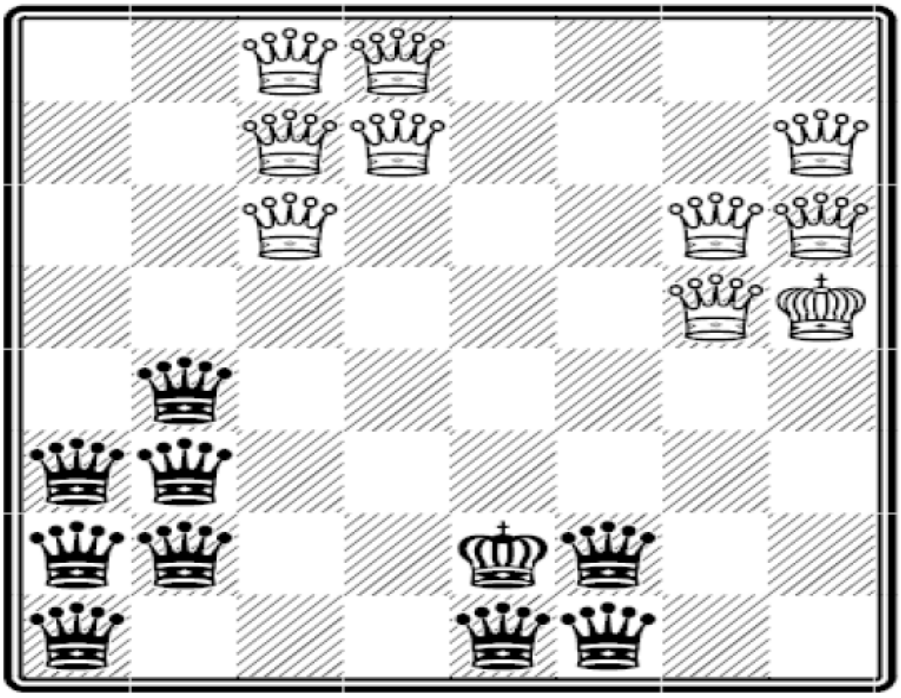


Figure 3-4 La solution au puzzle qui consiste à trouver une position d'échec contenant neufs reines et un roi de chaque couleur, avec la règle selon laquelle aucune pièce n'est sur la même ligne (ligne, colonne ou diagonale) avec la reine de la couleur opposée. En considérons la symétrie, la solution est unique.

Avant de donner les différentes définitions des différents types de symétries, nous commençons à expliquer la notion de la symétrie à travers un exemple. Alors pour illustrer ce que nous entendons par symétrie, nous considérons le puzzle d'échecs montré dans la figure 4.4. La solution à ce puzzle est unique en considérons la symétrie [61]. Mais qu'est-ce que nous entendons par symétrie dans ce contexte? En effet, la symétrie veut dire une opération qui modifie les positions des pièces, mais dont l'état final satisfait toutes les contraintes si et seulement si l'état

initial l'est. Étant donné une solution, qui, par définition, satisfait toutes les contraintes, nous pouvons trouver une nouvelle solution en appliquant une symétrie à la première solution que nous trouvons. Par exemple, en tenons compte de la solution de l'énigme sur la figure, on peut permuter les couleurs de chaque pièce, de sorte que les reines noires apparaissent là où les reines blanches sont, et vice versa. De même, on peut faire pivoter l'échiquier par un multiple de 90 degrés pour donner une solution nouvelle. Enfin, on peut tenir compte de l'échiquier autour de l'axe horizontal, l'axe vertical et les deux axes diagonaux. Étant donné que ces symétries peuvent être combinés, il y'a 16 symétries disponibles, y compris l'opération identité qui consiste à tout laisser où il est.

Pourquoi la symétrie est importante? La raison principale est que nous pouvons exploiter la symétrie pour réduire la quantité de recherche nécessaire pour résoudre le problème. C'est un avantage énorme. Par exemple, supposons que nous cherchons une solution pour notre puzzle d'échecs, et la première tâche est de placer une reine blanche dans le coin supérieur gauche. En fait, la décision de recherche ne consiste pas vraiment à essayer de placer une reine blanche dans le coin supérieur gauche, mais la décision consiste à essayer toutes les solutions possibles avec une reine de n'importe quelle couleur dans n'importe quel coin de l'échiquier. Puisqu'il y'a 16 symétries, nous avons la possibilité de réduire la recherche par un facteur de 16. Une deuxième raison de l'importance de la symétrie est que de nombreux problèmes de contraintes admettent des symétries dans elles. En effet, la modélisation d'un problème par un CSP peut introduire des symétries. Par exemple, si nous modélisons l'échiquier ci-dessus avec une variable pour chaque reine allant de 1 à 64 exprimant sa mise en place, il y aurait 2 $(9!)^2$ versions symétriques de chaque solution, car chaque ensemble de reines peuvent être permutées et la permutation des deux ensembles.

3.3.1 La symétrie et la théorie du groupe

L'étude de la symétrie en mathématiques est appelée la théorie des groupes. Ce dernier est essentiel afin de comprendre le rôle de la symétrie dans la programmation par contraintes, pour cela nous donnons une brève aperçue des concepts clés dont nous avons besoin. Cette section devrait être prise seulement comme une légère introduction à ce qui est l'un des plus grands domaines de recherche en

mathématiques. Heureusement pour la plupart des programmeurs de contraintes, un peu de connaissance de la théorie des groupes est en fait assez suffisant pour comprendre l'essentiel du travail accompli à ce jour. Donc nous allons vous expliquer la théorie des groupes par permutations, puisque chaque groupe peut être exprimé comme un ensemble de permutations. Et nous allons présenter le lien qui existe entre une symétrie comme étant un élément d'un groupe et une symétrie comme étant une action. Nous allons mettre l'action sur la notion « action du groupe », car cela va exprimer comment les symétries transforment les états de recherche, et c'est notre principal intérêt.

Exemple 10.1. Symétries échiquier

Considérons un échiquier 3 × 3. Nous marquons les neuf cases avec les chiffres de 1 à 9. Ces chiffres représentent les points qui seront déplacés par les symétries. Il y'a huit symétries naturelles pour cet échiquier y inclut la symétrie identité, qui consiste à laisser tout les points à leurs places. La symétrie identité est montrée en haut à gauche de la figure 4.5. Ensuite, on peut faire pivoter l'échiquier de 90, 180 et 270 degrés dans le sens horaire. Les emplacements des points résultant sont présentés dans le reste de la rangée supérieure de la figure 4.5. Enfin, il existe des réflexions dans l'axe vertical, dans l'axe horizontal, et dans les deux axes diagonaux principaux, et ceux-ci sont présentés dans la rangée inférieure.

1	2	3	7	4	1	9	8	7	3	6	9
4	5	6	8	5	2	6	5	4	2	5	8
7	8	9	9	6	3	3	2	1	1	4	7
	id			*r*90			*r*180			*r*270	

3	2	1	7	8	9	1	4	7	9	6	3
6	5	4	4	5	6	2	5	8	8	5	2
7	8	9	1	2	3	3	6	9	7	4	1
	x			*v*			*d1*			*d2*	

Figure 3-5 Les 8 symétries de l'échiquier 3 * 3

Pour voir le lien entre les symétries et les permutations, chaque symétrie définit une permutation de l'ensemble des points, la permutation est une correspondance 1 à

1 de l'ensemble vers lui-même. Deux différentes formes permettent d'écrire les permutations :

La forme de Cauchy **:** avec deux rangées de nombres, la première rangée représente l'ensemble des éléments dans l'ordre croissant sur laquelle la permutation est réalisée, la deuxième rangée indique les chiffres résultats du déplacement des éléments de la rangée supérieure. Par exemple dans la symétrie de la rotation par 90 dégrées (r90 de la figure 4.5) on remarque que le point 1 est remplacé par 7, 7 est remplacé par 9, le 9 par 3 et le 3 par 1, cela forme un cycle (1 7 9 3), un autre cycle (2 4 8 0) et un autre cycle qui contient (5). La représentation des permutations par les cycles est appelée la forme cyclique.

$$id: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} \qquad r90: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

$$r180: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} \qquad r270: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

$$x: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} \qquad y: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

$$d1: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} \qquad d2: \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

Figure 3-6 Permutations représentants les symétries de l'échiquier écrites avec la forme de Cauchy

- ***Forme cyclique*** **:** la symétrie r90 au-dessus contient les cycles (1 3 9 7), (2 4 6 8), (5), la permutation déplace chaque point vers l'élément successeur du cycle auquel il appartient à l'exception du dernier élément qui est déplacé vers le premier et le point qui n'appartient à aucun cycle est déplacé vers lui-même.

Les symétries par la forme cyclique de l'échiquier 3*3 sont montrées dans la figure 10.4.

id: ( ) $r90$: (1 3 9 7) (2 4 6 8)

$r180$: (1 9) (2 8) (3 7) (4 8) $r270$: (1 7 9 3) (2 4 8 6)

x: (1 3) (4 6)(7 9) y: (1 7)(2 8) (3 9)

$d1$: $(2\ 4)\ (3\ 7)\ (6\ 8)$ $d2$: $(1\ 9)\ (2\ 6)\ (4\ 8)$

Figure 3-7 Permutations représentant les symétries de l'échiquier écrites en forme cyclique

Les deux formes d'écriture des permutations montrent facilement comment la permutation agit une sur un point. En général, si p est un point et g est une permutation, alors nous écrivant p^g qui représente le point vers lequel est déplacé p par le moyen de g. Par exemple, $1^{r90} = 7$, et $1^{r270} = 3$. Un autre exemple pour les ensembles : $\{1, 3, 8\}^{r90} = \{1^{r90}, 3^{r90}, 8^{r90}\} = \{7, 1, 6\} = \{1, 6, 7\}$.

Il y'a certains facteurs importants au sujet des permutations qui assurent le lien entre eux et les groupes. Nous allons expliquer ceux-ci, puis nous donnons la définition fondamentale d'un groupe.

Il est simple de composer deux permutations, étant donnée les permutations *f* et *g*, la composition de ses permutations est notée *f* o *g*. Le résultat de *f* o *g* est calculé en prenant, pour chaque point, le résultat du déplacement du point par la permutation *f* et ensuite par la permutation *g*. donc plus formellement, la composition *f* o *g* de deux permutations *f* et *g*, définies sur un ensemble de points, est définie par : $\forall\ i \in \Omega,\ i^{f\,o\,g} = (i^f)^g$.

Il est important de noter l'ordre de l'action, c'est à dire que nous appliquons *f* ensuite *g* et nous écrivons *f* o *g*. Puisque les permutations f et g sont des correspondances un à un, alors leurs composition fournit une autre permutation.

Exemple : Composition de Permutations

$$r90 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \end{pmatrix}$$

$$x = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 2 & 1 & 6 & 5 & 4 & 9 & 8 & 7 \end{pmatrix}$$

$$r90ox = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 6 & 3 & 8 & 5 & 2 & 7 & 4 & 1 \end{pmatrix} = d2$$

Nous avons déjà décrit l'existence de la permutation identité, que nous appelons id.

Cela peut être défini comme l'ensemble de cycles vides pour tout ensemble de points. Pour toute permutation *f* il y'a une permutation inverse, telle que *f* o f^{-1} = id. Ceci est

facile à calculer: dans la forme cyclique il suffit d'inverser l'ordre de chaque cycle, et sous la forme de Cauchy nous échangeons les deux lignes et puis réorganiser les colonnes de sorte que la première ligne soit en ordre numérique.

Exemple Inverse d'une permutation

$$r90 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \end{pmatrix}$$

$$r90^{-1} = \begin{pmatrix} 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} \quad changer\ les\ lignes$$

$$= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 6 & 9 & 2 & 5 & 8 & 1 & 4 & 7 \end{pmatrix} \quad réorganiser\ les\ colonnes = 270$$

Enfin, la composition de permutations est associative. Autrement dit, *f* o (*g* o *h*) = (*f* o g*)* o *h*. c'est à dire que *g* o *h* donne, pour chaque point, le même résultat que l'application de *g* ensuite de *h*. Ainsi, par exemple, $7^{(f \circ g) \circ h)}$ est le résultat de l'application de *f* à 7, g au résultat, et h au résultat de ce dernier : mais c'est exactement la même chose que $7^{f\ o\ (g\ o\ h),}$ qui est fourni également en appliquant f, à 7, g au résultat, et h au résultat de ce dernier. Nous allons maintenant présenter les axiomes définissant un groupe et les différentes notions de la théorie de groupes, illustrées de quelques exemples.

Définition 7 (Groupe) : Un groupe est un couple (G, *), ou G est un ensemble et * une loi interne sur G (i.e. une application de G × G dans G), qui a de plus les propriétés :

- ✓ Fermeture : ∀ x ϵ G, ∀ y ϵ G, x*y ϵ G
- ✓ Élément neutre (identité) : ∋ e ϵ G, ∀ x ϵ G, x * e = e * x = x
- ✓ Symétrique (élément inverse): ∀ x ϵ G, ∋ x^{-1} ϵ G, x * x^{-1} = x^{-1} * x = e
- ✓ Associativité : ∀ (x, y, z) ϵ G, x * (y * z) = (x * y) z

Définition 10.6. (Ordre d'un groupe) : L'ordre d'un groupe G est le nombre d'éléments dans l'ensemble G. Il est noté | G |.

Exemple: L'ensemble des symétries de l'échiquier de l'exemple précédent {id, x, y, d1, d2, r90, r180, r270} forment un groupe d'ordre 8.

Pour les permutations la composition représente la loi interne sur G. Nous avons : $r90^{-1}$ = r270, $r270^{-1}$ = r90, et tous les autres éléments *g* sont auto-inverse, i.e. *g* = g^{-1}. Un élément du groupe *g* opère par l'opérateur de composition pour permuter

les valeurs des autres éléments du groupe. Autrement dit, *f* o *g* donne un autre élément du groupe.

Nous avons montré qu'il existe une permutation identité, que chaque élément permutation à un élément inverse et que la composition des permutations est associative. La dernière condition est la fermeture, nous avons vu que la composition de deux permutations fournit une autre permutation. Nous avons aussi, pour l'ensemble de permutations pour former le groupe, la composition de toutes deux permutations du groupe doit être dans le groupe, et cela dépend de l'ensemble de permutations qu'on choisit. Il existe une manière qui permet de vérifier la propriété de fermeture qui consiste à prendre un ensemble de permutations et de générer toutes les permutations possibles qui résultent de la composition arbitraire de ces derniers.

Définition 10.8. (Les générateurs d'un groupe) Soit S un ensemble d'éléments qui peuvent être composés par l'opération du groupe (par exemple, la composition de permutation). On dit que l'ensemble S engendre G si tout élément de G peut s'écrire comme un produit d'éléments de S et chaque produit d'une séquence d'éléments de S est dans G. L'ensemble S est appelé l'ensemble de générateurs de G et on note G = < S >.

Exemple 10,9. Générateurs de symétries de l'échiquier : Les symétries sont générés par {R90, d1} puisque:

$$id = r90\ o\ r90\ o\ r90\ o\ r90 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

$$r90 = r90 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 4 & 1 & 8 & 5 & 2 & 9 & 6 & 3 \end{pmatrix}$$

$$r180 = r90\ o\ r90 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{pmatrix}$$

$$r270 = r90\ o\ r90\ o\ r90 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 6 & 9 & 2 & 5 & 8 & 1 & 4 & 7 \end{pmatrix}$$

$$d1 = d1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 4 & 7 & 2 & 5 & 8 & 3 & 6 & 9 \end{pmatrix}$$

$$y = d1\ o\ r90 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 7 & 8 & 9 & 4 & 5 & 6 & 1 & 2 & \end{pmatrix}$$

$$d2 = r90\ o\ r90\ o\ d1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 9 & 6 & 3 & 8 & 5 & 2 & 7 & 4 & 1 \end{pmatrix}$$

$$x = r90\ o\ d1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 2 & 1 & 6 & 5 & 4 & 9 & 8 & 7 \end{pmatrix}$$

L'avantage de travailler avec les générateurs de groupe se présente comme suit : étant donné |G | éléments d'un groupe, il y'a toujours un générateur de groupe de log2 taille plus petite.

Définition 8 (Sous-groupe) : Soient G un groupe et H ⊆ G, H ≠ Ø ; H est un sous-groupe de G si :

- ✓ H est stable par *;
- ✓ e ϵ H ;
- ✓ x ϵ H => x^{-1} ϵ H.

Si H est un sous-groupe de G, alors H est un groupe. Deux exemples simples et universels de sous-groupes sont : G qui est toujours un sous-groupe de G et {id}.

Exemple 10.11. Sous-groupe de symétries de l'échiquier : L'ensemble {id, r90, r180, r270} forment un sous-groupe d'ordre 4. Comme on peut le voir dans l'exemple 10.1, ce sous-groupe peut être généré par l'élément r90.

Définition 11 (Chaine de sous-groupes) : Soit G un groupe. Une chaine de sous-groupes de G est une suite de la forme :

$$G = G^{(1)} \geq G^{(2)} \geq \cdot \cdot \cdot \geq G^{(k)} \geq G^{(k+1)} = \{id\}$$

Ou $G^1 \geq G^2$ signifie que G^2 est un sous-groupe de G^1.

Définition 10.15. Stabilisateur : Le stabilisant d'un point est l'ensemble des éléments qui fixent ou stabilisent le point. Il indique quels éléments peuvent être appliqués au point sans déplacer la valeur du point.

Soit G un groupe de permutations agissant sur un point β. Le stabilisateur de β en G est définie par: $G_\beta = \{g \in G \mid \beta^g = \beta\}$. Le stabilisateur G_β est un sous-groupe du groupe G.

Exemple 10.16. Stabilisateurs des symétries de l'échiquier : D'après la figure 10.2 représentant les symétries de l'échiquier, pour tout point nous pouvons identifier le stabilisateur. Par exemple, le stabilisateur du point 1 est G_1 = {id, d1} vu que ces

éléments mappent le point 1 vers lui-même. Le stabilisateur du point 5 est l'ensemble du groupe G, avec G = G_5, puisque aucun des symétries ne déplace le point 5.

Définition 10 (Groupe symétrique) : Soit E un ensemble, on note S(E) l'ensemble des bijections de E dans E. (S(E), o), ou o est la loi de composition dans E, est un groupe appelé groupe symétrique de E.

On notera S_n l'ensemble des bijections (permutations) de {1, 2, . . ., n} dans {1, 2, . . ., n}.

On a la propriété suivante : card(S_n) = n!

On notera I_n = {1, 2, . . ., n}, et pour i ϵ I_n, σ ϵ S_n on notera i^{σ} l'image de i par la permutation σ. Les permutations seront écrites en notation cyclique : σ = (1, 3, 2) signifie 1^{σ}= 3, 3^{σ} = 2, 2^{σ} = 1. L'identité est notée ().

Exemple 2 :

– S3 = {(), (1, 2), (1, 3), (2, 3), (1, 2, 3), (1, 3, 2)} est un groupe symétrique de taille 3.

– Soit n ϵ N, on définit : $S^{(1)} = S_n$ et ∀ i ϵ [2, n], $S^{(i)} = \{s \epsilon S^{(i-1)}, (i-1)^s = i-1\}$.

Alors la suite ($S^{(i)}$) iϵ [1, n] constitue une chaine de sous-groupe.

Définition 12 (Orbite d'une permutation) : Soit σ ϵ S_n. La relation binaire R définie sur S_n par : ∀ x, y ϵ S_n, x R y ⇔ ∃ k ϵ N, y = $\sigma^k(x)$ est une relation d'équivalence sur S_n. La classe d´équivalence de x pour R est appelée orbite de x.

Exemple 10.14. Les orbites des points sur l'échiquier :

En regardant le diagramme de symétries de l'échiquier donné dans l'Exemple 4.4, on remarque qu'on peut calculer les orbites de tout point. Par exemple, l'orbite de 1 est {1, 3, 7, 9}, car 1^{id} = 1, 1^{r90} = 7, 1^{r180} = 9, 1^{r270} = 3, et tous les autres éléments du groupe mappent le point 1 à l'un de ces points.

3.3.2 Définitions de la symétrie

Les symétries sur les CSP sont définies comme des bijections de l'ensemble des solutions vers lui-même. La symétrie a été définie aussi par Brown, Finkelstein et Purdom [62] par : « La symétrie est une permutation qui laisse invariant l'ensemble de solutions d'un problème ». Plus formellement une symétrie γ d'un *CSP P* est une bijection de *P* sur lui-même qui préserve ses solutions, i.e. pour toute instanciation *I*, $I \epsilon S_P \Leftrightarrow I^{\gamma} \epsilon S_P$.

3.3.3 Types de symétries

Il existe plusieurs types de symétries sur les CSP. Les définitions de symétrie diffèrent à la fois sur :

- quel aspect de la CSP les symétries agissent (les variables, les valeurs ou paires valeur-variable).
- en ce que les symétries conservent (les contraintes ou l'ensemble de solutions).

Dans cette section, nous donnons un bref aperçu des définitions contenues dans la littérature

3.3.3.1 Symétrie de variables/ valeurs/ variable-valeur

De nombreuses définitions définirent des formes restreintes de symétrie qui affectent uniquement les variables ou uniquement les valeurs ou qui affectent à la fois les variables et les valeurs.

3.3.3.1.1 Symétrie sur valeurs

La symétrie de valeurs est basée sur la notion d'interchangeabilité indiquée dans la définition 4.18 par Freuder [63], qui représente une forme des symétries de solutions qui n'opère que sur les valeurs.

3.3.3.1.1.1 Interchangeabilité de valeurs

Définition 4.18. Pour une variable v, deux valeurs a, b, sont entièrement interchangeables ssi chaque solution au CSP contenant l'assignement <v, a> reste une solution lorsque a est remplacé par b, et vice-versa.

Comme le décrit Freuder, en général, l'identification de l'interchangeabilité entière des valeurs demande de trouver toutes les solutions du CSP. Il définit des formes locales d'interchangeabilité qui peuvent être identifiés en examinant le problème. La définition 4.19 décrit l'interchangeabilité de voisinage, qui est une forme de symétrie de contraintes.

3.3.3.1.1.1.1 Interchangeabilité de valeurs au voisinage

Définition 4.19. Pour une variable v, deux valeurs a et b, sont interchangeables au voisinage ssi pour chaque contrainte C sur la variable v, l'ensemble des paires

variable-valeurs qui satisfirent la contrainte avec la paire <v, a> est le même que l'ensemble des paires variables- valeurs qui satisfont les contraintes avec la paire <v, b>.

Benhamou [64] étend légèrement l'idée d'interchangeabilité de valeur et fait la distinction entre la symétrie sémantique et syntaxique dans les CSP, qui correspondent respectivement à la symétrie de solution et la symétrie du problème. Il définit la symétrie sémantique comme suit : deux valeurs ai et bi sont symétriques pour tous les solutions si: chaque solution contenant la valeur ai peut-être mappé à une solution contenant la valeur bi, et vice versa. Si deux valeurs sont symétriques pour toutes les solutions alors elles sont aussi symétriques pour la satisfiabilité. Identifier les symétries sémantiques nécessite la résolution du CSP pour trouver toutes les solutions, par la suite les examiner. Benhamou définit aussi la symétrie syntaxique pour dire que la permutation ne change aucune relation de contrainte, définie comme un ensemble de tuples. La notion de valeurs interchangeables a été et est encore largement utilisé et étudié.

Nous finirons cette aperçue des symétries de valeurs par la définition formelle présentée par [65, 66] : Une symétrie sur les valeurs est une symétrie γ telle qu'il existe une permutation θ des valeurs telle que $(x_i = j)^{\gamma} = (x_i = j^{\theta})$. On notera alors γ par θ.

3.3.3.1.2 Symétrie sur variables

Dans les CSPs, la permutation des variables dans une contrainte définie intentionnellement cause en général la modification des contraintes, par exemple, la contrainte x + y = z est différente de la contrainte x + z = y.

Puget [67] définit la notion de contrainte symétrique, c'est à dire une contrainte qui n'est pas affecté par l'ordre des variables. Par exemple, la contrainte $\neq$ Est symétrique. Puget définit la symétrie d'un CSP comme une permutation des variables qui mappe l'ensemble des contraintes en un ensemble symétrique équivalent: toute contrainte est soit inchangée par la permutation soit elle représente une instance d'une contrainte symétrique et est mappée en une contrainte qui porte sur le même ensemble de variables. Ils ont définit la symétrie sur les variables comme une symétrie γ telle qu'il existe une permutation σ des variables telle que $(x_i = j)^{\gamma} = (xi^{\sigma} = j)$. On notera alors γ par σ.

3.3.3.1.2.1 Permutation intentionnelle

Roy et Pachet [68] définissent la notion de permutabilité intensionnelle comme suit : deux variables sont permutables intentionnellement ssi ils ont le même domaine et toutes les contraintes affectant l'une des deux variables doivent affecter les deux variables à la fois, et les deux variables doivent être interchangeables dans ces contraintes. La contrainte doit être définie intentionnellement, c'est à dire en termes d'une formule.

3.3.3.1.3 Symétrie sur les pairs variables-valeurs

Meseguer et Torras [69] donnent une définition des symétries qui agissent sur les variables et sur les valeurs d'un CSP. Leurs définitions permettent les symétries de variables (qui permutent seulement les variables) et les symétries de valeurs (qui permutent seulement les valeurs) comme cas particuliers. Cependant, cela ne convient pas à tous les CSPs. C'est-à-dire qu'ils existent des types de symétries, qui ne peuvent pas être exprimées comme une composition de symétries sur les variables et/ou sur les valeurs. Meseguer et Torras utilisent à titre d'exemple les symétries des n reines, dans la formulation CSP la plus utilisée là où une seule pièce est placée par ligne, les variables correspondent aux lignes de l'échiquier et les valeurs correspondent aux colonnes. Ils montrent que la réflexion de 180° est une symétrie du CSP selon leur définition, mais quatre symétries ne sont pas: la réflexion dans les diagonales, la rotation de 90° et de 270°.

Exemple : Illustrons les notions introduites ci-dessus par quelques exemples :

(Symétries et Sudoku) Le problème du Sudoku possède de nombreuses symétries sur les variables et sur les valeurs. Toute permutation sur les valeurs {1, 2, . . ., n2} est une symétrie du problème (une permutation sur les valeurs n'affecte pas la propriété que dans chaque colonne, ligne ou bloc, on conserve le fait que tous les membres sont différents deux à deux), ce qui donne n2! symétries. Ces symétries sont bien sûr des symétries sur les valeurs.

Le Sudoku présente également de nombreuses symétries sur les variables. Pour une meilleure compréhension, introduisons les deux appellations suivantes. Une pile consiste en un arrangement vertical de trois blocs (9×3 cases) et une bande consiste en

un arrangement horizontal de trois blocs (3 × 9 cases). Le Sudoku possède les symétries suivantes :

– permutations des trois piles ;

– permutations des trois bandes ;

– permutations des trois colonnes dans une pile ;

– permutations des trois lignes dans une bande ;

– rotations d'angles 90°, 180° et 270° de la grille ;

– réflexions par rapport aux diagonales de la grille.

L'outil GAP (voir [70]) permet de calculer la taille du groupe généré par ces symétries.

Pour le problème classique du Sudoku de taille 3, il y a 3 359 232 symétries sur les variables et 9 ! = 362 880 symétries sur les valeurs.

3.3.3.2 Symétrie de solutions/ contraintes

Il existe deux grands types de définition: celles qui définissent la symétrie comme une propriété de l'ensemble des solutions (symétrie sémantique) et celles qui définissent la symétrie comme une propriété qui peut être indiquée dans l'énoncé du problème (symétrie syntaxique), sans le résoudre. Celles-ci seront appelées symétrie de solution et symétrie de problème.

La détection des symétries sémantiques est intraitable [71, 72]: pour trouver le groupe complet de symétrie, nous avons besoin de toutes les solutions du CSP, pour cela la définition de la symétrie comme étant la préservation de l'ensemble des solutions n'offre pas une voie pratique pour identifier la symétrie dans les CSP. D'autre part, le groupe de symétrie de solution est bien définie, tandis que les CSP équivalents qui ne diffèrent que légèrement dans la façon dont les contraintes sont exprimées, peuvent avoir des symétries de problème différentes. Il peut être possible d'écrire les contraintes d'un CSP de telle sorte que la symétrie du problème modélisé n'apparait pas.

3.3.3.2.1 Symétrie de solution

Une symétrie de solution est une permutation de l'ensemble des paires de < variable, valeur > qui préserve l'ensemble des solutions.

3.3.3.2.2 Symétrie de problème (de contraintes)

Une symétrie de problème est une permutation de l'ensemble des paires de < variable, valeur > qui préserve l'ensemble des contraintes.

Les deux types de symétrie (symétrie de problème et symétrie de solution) permettent les symétries sur variables et les symétries sur valeurs comme des cas particuliers.

Pour compléter la définition de 4.21, une interprétation appropriée de ce que signifie ' pour préserver les ensembles de contraintes est nécessaire'. Toute contrainte ci avec une portée Vi ⊆ V peut être définie par un ensemble de paires < variable, valeur > satisfaisant la contrainte. La symétrie qui agit sur un ensemble paires < variable, valeur > peut être appliqué à l'ensemble des paires < variable, valeur > satisfaisant la contrainte, ce qui donne un nouveau ensemble de paires < variables, valeurs >. Les paires < variable, valeur > résultants ne sont pas tous liés à la même série de variables que la contrainte initiale. Cependant, si les résultats de l'application de la symétrie à tous les paires < variable, valeur >, définissant toutes les contraintes sont les mêmes paires < variable, la valeur >, on peut dire dans ce cas-là que les contraintes sont inchangées par l'action de la symétrie. Donc cette définition ne nécessite pas que la symétrie laisser chaque contrainte inchangée individuellement, mais plutôt l'ensemble de toutes les contraintes.

Pour les deux types de symétries on a les propriétés suivantes :

- Les symétries de contrainte (respectivement de solution) forment le groupe de symétrie de contrainte (respectivement de solution).
- Les symétries partitionnent l'ensemble des assignements partielles d'un CSP en des orbites.
- Les membres de l'orbite de solutions sont des solutions symétriquement équivalentes et forment une classe d'équivalence.

3.4 Techniques de suppression de symétries entre solutions dans les CSPs

Les Techniques de suppression de symétrie peuvent être classées en quatre catégories: reformulation du problème, en ajoutant des contraintes de symétries avant la recherche, la modification des algorithmes de recherche, ou en utilisant des méthodes heuristiques. Dans ce qui suit, on va décrire chacune des méthodes.

3.4.1 La reformulation du problème

La modélisation à un effet important sur l'efficacité de résolution d'un problème. Une reformulation appropriée d'un modèle peut transformer un problème impossible dans la pratique à un autre problème plus faisable. La modélisation et la reformulation sont également importantes pour la suppression des symétries. Différents modèles d'un même problème peuvent avoir des symétries différentes, une formulation peut avoir des symétries qui sont plus faciles à traiter que les symétries des autres formulations. Dans les cas extrêmes, une formulation peut ne pas avoir de symétries. Dans d'autres cas, la quantité de symétrie peut être considérablement réduite d'un modèle à un autre. En outre, une fois qu'un problème est reformulé, les symétries restantes peuvent encore être traitées avant ou pendant la recherche, or que d'autres méthodes pour la suppression des symétries peuvent conduire à de grandes difficultés en les combinant les uns avec les autres. Ainsi, la reformulation d'un problème peut être critique dans le traitement des symétries.

Exemple1 : Pour un premier exemple nous citons le fameux problème "social golfeurs", Dans ce problème, 32 golfeurs veulent jouer en 8 groupes de 4 par semaine, de sorte que deux golfeurs jouent dans le même groupe au plus une fois, pour autant de semaines que possible, le cas difficile est de 10 semaines. Nous pouvons construire un modèle avec 32! 10! $8!^{10}$ $4!^{80}$ symétries: Nous pouvons permuter les 32 joueurs, on peut permuter les 10 semaines, au sein de chaque semaine, nous pouvons permuter les groupes (séparément) et nous pouvons permuter les quatre joueurs (séparément) au sein de chaque groupe. Dans ce modèle, il y'a plus de 10^{198} versions symétriques de chaque solution différente. En reformulant le problème, Smith réduit ce nombre à 32! 10! [73]. nous avons une variable indiquant pour chaque paire de joueurs la semaine où ils jouent ensemble (ou une variable supplémentaire qui indique si ils ne se rencontrent jamais): il ne reste que les symétries de la semaine et les symétries des joueurs. Il reste encore un très grand nombre de symétries, mais ils sont d'une forme plus simple.

Une technique importante est l'utilisation de variables ensemblistes où nous avons un certain nombre de variables qu'on ne peut pas les distinguer. Variables ensemblistes sont disponibles dans la plupart des solveurs de contraintes, et nous permettent d'exprimer les contraintes avec les opérateurs d'intersection, union, etc. Dans le problème de golfeurs, par exemple, on peut coder les groupes jouant au sein de chaque semaine par huit variables ensemblistes. Chacune doit être de taille 4 et

leurs intersection deux à deux doit avoir l'ensemble vide. Le fait qu'il n'existe aucun ordre implicite entre les éléments des variables définis, nous avons perdu les 4! Symétries dans chaque groupe, ce qui réduit le total des symétries par un facteur de 24^{80} dans le problème. En outre, dans le problème des golfeurs nous pouvons représenter la contrainte que deux joueurs ne peuvent pas jouer ensembles plus d'une fois, en imposant que l'intersection de deux groupes dans les différentes semaines est égale à 1 ou 0.

Deux inconvénients pour l'utilisation des variables, ensemblistes :

- Le solveur admet différentes représentations des variables ensemblistes qui produisent différentes comportements dans la propagation [74], donc si le solveur choisit la représentation la moins convenable pour les contraintes cela va influencer sur l'efficacité de la recherche.
- La possibilité d'avoir des contraintes sur les éléments d'un ensemble, ces contraintes convient le mieux à des variables entières, en plus, on a la difficulté de la canalisation entre les variables d'ensembles et les variables entières qui conduisent la propagation dans la recherche à un échec.

Pour ces deux raisons, les techniques de suppression de symétries évitent l'utilisation de variables ensemblistes dans plusieurs domaines, malgré ça, les variables ensemblistes représentent une technique de modélisation très importante et constituent un domaine d'étude très important.

Un autre exemple de reformulation illustre encore les améliorations spectaculaires qui peuvent être atteintes. Le problème d'intervalle de séries qui consiste à trouver une permutation des n entiers de 0 à n-1 de sorte que les différences entre les nombres adjacents sont également une permutation des nombres de 1 à n - 1. Il y'a quatre symétries évidentes du problème: l'identité, inversant la série, la négation de chaque élément, la soustraction de n - 1, et de faire les deux. Gent et al [75] écrivent sur la reformulation du problème qui repose sur l'observation qu'on peut cycler une solution au problème autours d'un pivot pour générer une autre solution. L'emplacement de ce pivot est fonction des affectations faite. A titre d'exemple, voici deux solutions pour n = 11. Les différences sont écrites sous les numéros:

0 10 1 9 2 8 3 7 4 6 5	3 7 4 6 5 0 10 1 9 2 8
10 9 8 7 6 5 4 3 2 1	4 3 2 1 5 10 9 8 7 6

La différence entre le premier nombre à gauche (0) et le dernier nombre (5) est 5. Cela signifie qu'on peut diviser la séquence entre le 8 et 3, avec perte de la différence 5. On peut rejoindre le reste de la séquence sur le début, parce que le 5-0 remplacera 8 - 3. Cela donne exactement la solution représentée sur la droite. Dans ce cas, le pivot se situe entre les valeurs 8 et 3. La différence entre le premier et le dernier terme doit toujours dupliquer la différence dans la séquence, donc cette opération peut être appliquée à n'importe quelle solution. Pour cette raison, on va ne pas simplement reformuler le modèle à contrainte, mais on va passer à résoudre un problème différent, dont les solutions conduisent aux solutions du problème original. Dans le problème reformulé, on trouve une permutation de la séquence 0, 1, . . . n - 1, mais on inclut la différence entre le premier et le dernier numéro, ce qui donne n différences au lieu de n - 1. La séquence doit obéir à deux contraintes: que la permutation commence à 0, n - 1, 1; et que les n différences entre les numéros consécutifs contiennent tous 1,. . . n-1 avec une différence qui apparaît exactement deux fois. [76] montrent que (pour n> 4) chaque solution donne 8 solutions distinctes au problème des séries tout-intervalle, mais la reformulation n'a aucune symétrie. La recherche dans ce modèle est cinquante fois plus rapide que d'autres techniques concurrentes.

On peut aussi prendre l'avantage des différents points de vue au problème, par exemple, supposons on a n variables, chacune à les même n valeurs possibles, et que les variables doivent prendre des valeurs différentes, on peut voir ce problème de deux points : on peut chercher les valeurs pour chaque variable ou chercher les variables pour chaque valeur, alors s'il existe une symétrie, donc par le premier point de vue la symétrie par valeur peut être inter- changée, avec la symétrie de variable, et vice ver sa du deuxième point de vue, par exemple, Roney-Dugal[77] utilise cette idée, il utilise les techniques efficaces pour la suppression des symétries de valeurs dans la suppression des symétries de variables, Flener aussi travaille avec cette idée dans les modèles des matrices [78].

Puisque cette approche de reformulation est entièrement spécification du problème alors il n'existe pas des principes généraux sur la façon de la faire et donc en raison de sa dépendance aux problèmes, cette approche ne peut pas être largement utilisée dans des applications réelles.

3.4.2 Les méthodes statiques de suppression de symétries (Ajout de contraintes au modèle)

Sans doute, la méthode de suppression de symétrie qui a été la plus utilisée dans le passé consiste à ajouter des contraintes au modèle de base. Dans ce contexte, le terme "brisure de symétrie" est tout à fait approprié. On passe d'un problème avec beaucoup de symétrie à un nouveau problème avec nombre de symétries considérablement réduit – dans l'idéal on passe à un problème sans symétries. Les contraintes qu'on ajoute pour atteindre cela sont appelées "contraintes de suppression de symétries". Souvent, il est facile de penser à des contraintes qui cassent tout ou une grande partie de symétries. Par exemple, supposons qu'on a 100 variables dans un tableau X qui sont indiscernables (afin qu'ils puissent être librement permutés). Il est simple et correcte, d'insister pour que les variables soient en ordre non décroissante, X [1] ≤ X [2] . . . ≤ X [100]. Si on sait encore que toutes les variables doivent être différentes, on peut rendre cette ordonnance strictement croissante: X [1] <X [2] . . . <X [100]. S'il arrive que les variables prennent les valeurs 1... 100, alors une propagation de contraintes simple en déduira que X [1] = 1, X [2] = 2, . . ., X [100] = 100. Si le programmeur le remarque à l'avance, alors dans ce cas on peut reformuler le problème afin de remplacer chaque variable X [i] avec la valeur i au long de notre programme. Il existe de nombreux exemples où les programmeurs de contraintes ont ajouté des contraintes plus complexes pour briser les symétries. Un exemple typique de la littérature est d'ajouter des contraintes pour briser la symétrie dans le problème de conception de modèle [79]. Ceci est bien s'il est correctement fait, mais peut perdre des solutions dans le cas contraire. Des méthodes standard ont été développées qui peuvent rendre le processus plus facile et plus susceptibles d'être correct, dans les situations où ils s'appliquent. Même lorsque ceux-ci ne sont pas directement applicables, une connaissance d'eux servira aux programmeurs de contraintes, comme ils doivent simplifier le calcul des contraintes correctes qui peuvent être ajoutés manuellement.

3.4.2.1 La méthode Lex-Leader

3.4.2.1.1 Principe

Puget [67] a montré que chaque fois qu'un CSP a des symétries, il est possible de trouver une forme réduite du CSP avec les symétries éliminés, en ajoutant des

contraintes au problème initial. Puget a trouvé une telle réduction pour trois problèmes de contraintes simples, et a montré que ce CSP réduit pourrait être résolus plus efficacement que dans sa forme originale. Suite à cela, l'avancement majeur a été de montrer une méthode pour générer cet ensemble de contraintes. Crawford, Ginsberg, Luks et Roy décrivent une technique, appelée "lex-leader" pour la construction de contraintes d'ordre pour la suppression de symétries pour les symétries de variables [80]. Dans des travaux suivants, Aloul et al ont également montré comment les contraintes lex-leader de la suppression de symétrie peuvent être exprimées de manière plus efficace [81].

En effet, l'idée de lex-leader est une idée qui est simple. Pour chaque classe d'équivalence de solutions en vertu du groupe de symétries, on prédéfini une des solutions d'être la solution canonique. On peut atteindre cet objectif en ajoutant des contraintes avant le début de la recherche qui sont satisfaites par les solutions canoniques, et non par les autres solutions.

La technique exige d'abord le choix d'un ordre sur les variables statiques. De cela, on induit un ordre sur les affectations complètes. L'ordre des affectations complètes est simple. Les tuples représentent les valeurs affectées aux variables, énumérées dans l'ordre défini par l'ordre statique. Comme la méthode est définie pour les symétries sur variables, toute permutation g convertit ce tuple dans une autre tuple, et le moins lexicographique de ces derniers qui sera choisi. Cette méthode est en pratique simple à mettre en œuvre. Chaque permutation dans le groupe fournie une contrainte $\leq_{Lex}$. Ainsi, l'ensemble des contraintes définies par la méthode de lex-leader est :

$$\forall\, g \in G,\ V \leq \mathrm{Lex} V^{g} \qquad (10,1)$$

Où V est le vecteur des variables du CSP, et $\leq_{Lex}$ est la d'ordre relation lexicographique. L'ordre lexicographique est exactement comme il est connu dans l'informatique, par exemple,

$AD \leq_{Lex} BC$ ssi soit $A < B$ ou bien $A = B$ et $D \leq C$.

L'exemple suivant illustre la méthode. Considérons une matrice 3 × 2 montée sur la figure 10.5, dans un contexte où les lignes et les colonnes peuvent être permutées librement. Les symétries forment le groupe S3 × S2, avec l'ordre 3! 2! = 12. On choisit les variables par ordre alphabétique, de sorte que le vecteur des variables du problème soit ABCDEF. Les 12 symétries conduisent aux 12 contraintes lex-leader indiquées dans la figure 10.5, y compris la symétrie vide de l'identité.

A	B	C
D	E	F

→

F	E	D
C	B	A

1. ABCDEF $\preceq_{lex}$ ABCDEF
2. ABCDEF $\preceq_{lex}$ ACBDFE
3. ABCDEF $\preceq_{lex}$ BACEDF
4. ABCDEF $\preceq_{lex}$ CBAFED
5. ABCDEF $\preceq_{lex}$ BCAEFD
6. ABCDEF $\preceq_{lex}$ CABFDE
7. ABCDEF $\preceq_{lex}$ DEFABC
8. ABCDEF $\preceq_{lex}$ DFEACB
9. ABCDEF $\preceq_{lex}$ EDFBAC
10. ABCDEF $\preceq_{lex}$ FEDCBA
11. ABCDEF $\preceq_{lex}$ EFDBCA
12. ABCDEF $\preceq_{lex}$ FDECAB

Figure 3-8 matrice 3*2 contenant 6 variables, le résultat de la permutation de deux lignes et les colonnes, donnant la permutation d'ABCDEF à FEDCBA. Les 12 contraintes lex-leader sont également représentées. Chaque contrainte correspond à une permutation des variables, et comme la méthode est définie pour les symétries de variables : la transformation de matrice illustrée donne la contrainte 10.

Une question pratique importante avec les contraintes lex-leader, c'est qu'ils ne respectent pas les heuristiques d'ordre de variables et de valeurs utilisées dans la recherche. Autrement dit, il se pourrait bien que la solution la plus à gauche dans l'arbre de recherche, qui, autrement, serait trouvé en premier, ne soit pas canonique et est donc rejetée, ce qui conduit à la recherche accrue. Ceci est en contraste avec d'autres techniques comme SBDS et SBDD, qui respecte l'heuristique. Des exemples ont été signalés où une "mauvaise" heuristique peut conduire à une augmentation spectaculaire dans l'exécution [82]. Ce problème est inhérent à la méthode, mais dans de nombreux cas, il est facile de travailler sur une "bonne" heuristique. En particulier, si le même ordre de variable statique est utilisé dans la recherche comme il a été utilisé pour construire l'ordre lex-leader, et les valeurs sont testés ou essayés à partir de du plus petit au plus grand, ce conflit ne devrait pas se produire. Cependant, ceci limite la puissance du programmeur de contraintes pour utiliser l'ordre dynamique des heuristiques pour les variables.

Les problèmes résolus moins facilement avec les contraintes lex-leader c'est dans le cas ou il s'agit de nombreux groupes qui contiennent un nombre exponentiel de symétries. Lex-leader nécessite une contrainte pour chaque élément du groupe.

Dans le cas d'une matrice à m lignes et n colonnes, c'est m! n!, Qui est peu pratique en général. Par conséquent, il existe de nombreux cas où lex-leader est applicable mais pas pratique. Cependant, lex-leader reste d'importance la plus elevée, car il est très utilisé pour dériver de nouvelles méthodes de suppression de symétries.

La méthode lex-leader est définie uniquement pour les symétries de variables: c'est à dire ceux qui permutent les variables mais laisser les valeurs inchangées. Pour des problèmes ayant un nombre petit de valeurs, tel que SAT, qui peut avoir seulement deux valeurs [80]. Il est possible d'ajouter une nouvelle variable pour représenter la négation de chaque variable, ainsi les symétries sur valeurs peuvent être transformées en symétries de variables. Malheureusement, si on a d valeurs, on aura besoin de d! versions de chaque variable pour appliquer cette idée. Par conséquent, une généralisation appropriée de la lex-leader pour gérer les symétries de valeur serait utile, même si elle sera restreinte à certains cas spéciaux.

3.4.2.1.2 Simplification des contraintes Lex-Leader

Les contraintes Lex-leader peuvent être simplifiées pour supprimer les redondances [80, 83]. Suivant Frisch et Harvey [84], on peut illustrer cela en utilisant l'exemple de la figure 10.5. La première idée est d'observer chaque contrainte individuellement. Par exemple, considérons la contrainte 2, ABCDEF $\leq_{lex}$ ACBDFE. On peut supprimer la première et la quatrième variable de chaque tuple, depuis A = A et D = D, donne BCEF $\leq_{lex}$ CBFE. Mais si B <C, la contrainte est satisfaite quels que soient les autres valeurs, et par ailleurs on a B = C pour satisfaire la contrainte. En d'autres termes, si les variables secondaires dans les tuples sont pertinentes, alors elles doivent être égales. En appliquant ce raisonnement partout, on obtient les contraintes indiquées sur la figure 10.6.On peut aller plus loin, en traitant les contraintes comme un ensemble et non pas individuellement. Par exemple, $\leq_{lex}$ est transitif, donc les contraintes 2 et 3 impliquent la contrainte 4. Un exemple plus complexe est la contrainte 11. Les derniers éléments de chaque tuple sont E et C. Mais si elles sont pertinents, on a A = E et B = F = D = C. Mais la contrainte 3 implique A$\leq$ B, d'où E $\leq$ C, de sorte que les derniers éléments du tuple ne sont pas pertinents et peuvent être supprimés. Ce raisonnement conduit à un ensemble de 8 contraintes indiquées sur la figure 10.7.

Malheureusement, l'approche décrite ici ne contourne pas le problème fondamental du nombre exponentiel de symétries. En général, le nombre de symétries, même dans l'ensemble réduit sera toujours exponentielle [83]. Cependant, l'approche permet d'illustrer comment l'ensemble des contraintes peut être simplifiée et cela conduit à avoir de bon résultats.

1.	*true*	7.	ABC $\preceq_{lex}$ DEF
2.	BE $\preceq_{lex}$ CF	8.	ABC $\preceq_{lex}$ DFE
3.	AD $\preceq_{lex}$ BE	9.	ABC $\preceq_{lex}$ EDF
4.	AD $\preceq_{lex}$ CF	10.	ABC $\preceq_{lex}$ FED
5.	ABDE $\preceq_{lex}$ BCEF	11.	ABCDE $\preceq_{lex}$ EFDBC
6.	ABDE $\preceq_{lex}$ CAFD	12.	ABCDE $\preceq_{lex}$ FDECA

Figure 3-9 The lex-leader constraints for the 3×2 matrix reduced on an individual basis.

2.	BE $\preceq_{lex}$ CF	9.	ABC $\preceq_{lex}$ EDF
3.	AD $\preceq_{lex}$ BE	10.	ABC $\preceq_{lex}$ FED
7.	ABC $\preceq_{lex}$ DEF	11.	ABCD $\preceq_{lex}$ EFDB
8.	ABC $\preceq_{lex}$ DFE	12.	ABC $\preceq_{lex}$ FDE

Figure 3-10 The lex-leader constraints for the 3 × 2 matrix reduced as a set.

3.4.2.1.3 La méthode lex-leader et la contrainte all-different

La contrainte «a*ll-different* » se produit très souvent dans la programmation par contraintes et beaucoup plus dans des problèmes avec symétrie. Puget a montré [85] que dans le cas des problèmes ayant une symétrie de variable (le seul cas où la méthode 'lex-leader' est définie) sur un ensemble de n variables contrainiées par une contrainte a*ll-different*, la symétrie peut être rompu complètement par seulement (n – 1) contraintes binaires. Ce résultat s'applique à n'importe quel groupe G qui agit sur l'ensemble des variables. Ce résultat remarquable ne pouvait guère être amélioré, cependant, il est relativement simple.

Prenons à titre d'exemple le problème « Graceful Graph » qui contient des symétries de variables sur un ensemble de variables toutes différentes avec un groupe symétrique non assez simple tel que Sn.

Exemple : 'Graceful Graph'

On dit qu'un graphe avec m arêtes est gracieux (graceful) s'il existe un étiquetage f de ses sommets tels que:

- Pour chaque sommet i on a : $0 \leq f(i) \leq m$,
- L'ensemble des valeurs f(i) sont toutes différentes,
- L'ensemble des valeurs abs (f(i), f(j)) pour chaque arrete (i, j) sont toutes différentes.

Une simple translation en un CSP existe lorsqu'il existe une variable vi correspondante a chaque sommet vi, voir [86]. En effet, ce problème a deux types de symétries : Les symétries de variables qui sont induites par l'automorphisme du graphe et une symétrie de valeur qui mappe v a m-v, dans ce qui suit cette dernière va être ignorée et seulement les symétries de variables qui sont considérées. Petrie et Smith ont examiné diverses formes de méthodes de suppression de symétrie statiques et dynamiques dans les graphes gracieux [87], en utilisant ces techniques, ils ont trouvé des cas de graphes gracieux qui n'étaient pas connus auparavant . A titre d'exemple, considérons le graphe K3×P2 représenté dans la figure 4.11. Le groupe permet toute des 3!permutations de K3, aussi que la même permutation est appliquée sur les deux copies de K3 dans le même temps, ainsi que la permutation des deux triangles. En tout le problème contient 12 symétries. En fait, le groupe est isomorphe à celui de la matrice de la figure 4.9 de sorte que les contraintes sont les qui ont été présentés dans l'exemple précédant (cas de la matrice).

En utilisant le fait que les variables sont soumises à une contrainte « all-different », on peut réduire considérablement le nombre de contraintes de brisure de symétrie. Par exemple, considérons la contrainte de suppression de symétrie :

$$ABCDEF \leq_{lex} ACBDFE$$

Comme A = A est trivialement vrai, et comme B = C ne peut être vrai à cause de la contrainte all-different, cette contrainte peut être simplifiée a : B < C.

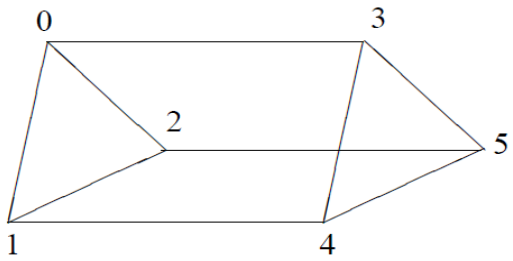


Figure 3-11 Le graphe K3 ×P2.

En général, cette simplification est vrai et peut etre formaliée comme suit : soit une permutation g, soit s(g) le plus petit i tel que ig i, et t(g) est égal à (s(g))g.

Lemme : [88]

Etant donné un CSP où les variables V sont soumises à une contrainte all-different, et G le groupe symétrique de variables pour ce CSP, alors toutes les symétries de variables peuvent être supprimées en ajoutant les contraintes suivantes:

$\forall \sigma \in G, vs(\sigma)<vt(\sigma)$

Etant donné deux permutations g et h tels que s (g) = s (h) et t (g) = t (h), alors les contraintes de suppression de symétries correspondant sont identiques. Par conséquent, il suffit d'indiquer seulement une contrainte de suppression de symétrie pour chaque paire i, j tels qu'il existe une permutation g avec i = s (g) et j = t (h). L'ensemble de ces paires peut être calculée en utilisant l'algorithme *Schreier Sims* [89]. Dans cet exemple, ces contraintes sont:

A < B, A < C, A < D, A < E, A < F, B < C

Notez que ces contraintes sont redondantes. La contrainte A < C est entraîné par la première et la dernière contrainte. Cette observation peut être utilisée pour réduire le nombre de contraintes en tenant compte de la transitivité des contraintes <. L'algorithme *Schreier Sims* nous donne une chaîne stabilisatrice et un ensemble de cosets représentatifs Ui, tel que défini à la section 10.1.2. Puget utilise ça pour prouver:

Théorème : [88]

Etant donné un CSP à n variables V, de sorte qu'il existe une contrainte tout différent sur celles-ci des variables, alors toutes les symétries de variables peuvent être brisés par au plus n -1 contraintes binaires.

Pour notre exemple, nous obtenons exactement 5 contraintes: notez ceci est n - 1 dans ce cas.

A < B, A < D, A < E, A < F, B < C

Bien que cela soit seulement une réduction d'une seule contrainte, qui est tout simplement à cause de la petite taille de l'exemple. En général, Puget a réduit le nombre de symétries nécessaires à partir d'un éventuellement n! à aussi peu que n-1, pour le cas se produisant couramment de symétries de variables, en présence d'une contrainte tout différent. Ainsi que sa valeur dans son propre droit, ce qui montre la puissance de la combinaison des contraintes de rupture de symétrie avec les contraintes d'un problème, et cela reste un domaine mûr pour l'exploitation.

3.4.2.1.4 Sous ensemble de contraintes Lex

Des travaux ont montré qu'un nombre polynomial de contraintes peut conduire à un raisonnement équivalent à l'ensemble complet de contraintes lex-leader. Malheureusement, un tel sous-ensemble n'est pas toujours disponible. Plusieurs chercheurs ont proposé des moyens d'affirmer qu'un nombre polynomial de contraintes sans conserver la suppression complète de symétries. Puisque l'ensemble complet de contraintes lex-leader laisse exactement une solution dans chaque classe d'équivalence, l'utilisation d'un sous-ensemble de contraintes doit laisser au moins une solution dans chaque classe, mais il peut laisser plus qu'une. Ainsi, les contraintes de suppression de symétrie ne garantissent pas la suppression de toutes les symétries. En général, l'objectif est de parvenir à un compromis acceptable, avec un nombre considérablement réduit des contraintes induisant à une recherche plus efficace.

Aloul et al ont montré, en SAT, que les résultats réussis peuvent être obtenus à partir de très petits sous-ensembles de contraintes lex-leader, sur des exemples tels que les problèmes de routage FPGA [90].

3. La symétrie dans la programmation par contraintes

Shlyakhter a montré que un bon (bien que généralement incomplète) sous-ensemble de contraintes lex-leader peut être trouvée pour des graphes acycliques, permutations, relations et fonctions [91]. Outre les contributions individuelles, ce ci établit la méthodologie d'utilisation de contraintes lex-leader comme un moyen de trouver des ensembles incomplets de contraintes de suppression de symétries, mais des sous-ensembles qui sont efficaces dans la pratique. Un cas particulier la méthode lex-leader à donner de très bon résultats c'est le modèle de matrices pour la supression des symétries de lignes et symétries de colonnes, pour cela d'autre variantes du lex-leader ont été proposés tel que les contraintes doublex et snaklex, et d'autres travaux autour de la suppression combinée c'est à dire à la fois la suppression des symétries de lignes et symétries de colonnes. On peut déduire que cette méthode à donner de bons résultats surtout dans des modèles particuliers comme les modèles de matrices.

3.4.2.1.5 Suppression partielle des symétries et les règles de réduction dans la méthode lex-leader

Il existe deux approches principales pour faire face à la complexité des contraintes lex-leader. La première approche est la suppression partielle de symétrie, c'est à dire, on construit les contraintes lex-leader pour un sous-ensemble de symétries [92].

La seconde approche consiste en des règles de réduction qui simplifient un ensemble de contraintes lex-leader tout en préservant leur équivalence. On présente les deux règles de réduction les plus communs. Pour l'introduction des règles de réduction, proposons la contrainte lex-leader suivante :

$$[x_1, x_2, \ldots, x_n] \leq_{\text{lex}} [x_{\sigma(1)}, x_{\sigma(2)}, \ldots, x_{\sigma(n)}] .$$

On peut simplifier cette contrainte Lex Leader comme suit :

- Si σ(i) = i, alors xi peut être supprimée des deux parties de la contraintes lex-leader.
- Pour les indices a et b pour a $<$ *b* si σ swap les indices a et b, la contrainte va avoir d'habord xa dans la partie gauche, couplé avec xb dans la partie droite, et par la suite, xb dans la partie gauche et xa dans la partie droite. Les arguments suivants montrent qu'on doit supprimer la deuxième paire. Si on a xi $<$ xσ(i) pour i tel que 1 ≤ i $<$b, en particulier on doit avoir xa $<$ xb. Et la

deuxième pair n'a pas d'effet car l'affectation sur la partie gauche est lexicographiquement moins que l'affectation sur la partie droite. Si xi = xσ(i) pour 1 ≤ i <b donc en particulier pour xa = xb et encore la seconde paire n'a pas d'effet.

- Frish et hervey ont montré que la contrainte résultante est équivalant a la contrainte originale [93]. En plus, les règles de réduction pour les contraintes Lex-Leader sont connues [93, 94].

Il faut rappeler qu'il est toujours possible de combiner la suppression partielle des symétries avec les règles de réduction.

Réductions Valides

Théorème : *Il existe un nombre infini de DSP de P correspondant et (sub) de groupe du groupe de symétrie P de la variable, telle que la nombre de contraintes de lexleader est (cn) pour tous les ordonnancements possibles de variables, où c est une constante supérieure à 1 et n est le nombre de variables dans n.*

Théorème : Un CSP P est satisfiable si et seulement si toute reduction valide de P est satisfiable.

3.4.2.2 Les symétries des contraintes de suppression de symétries

Une façon de construire de nouvelles contraintes de suppression de symétries est d'utiliser la symétrie elle-même [***95***]. Toute symétrie agissant sur un ensemble de contraintes de suppression de symétrie va elle-même supprimer la symétrie dans le problème. Cela a induit à la définition de l'action de la symétrie sur un ensemble de contraintes de suppression de symétrie. La symétrie a été définie comme agissant sur les affectations, mappant des solutions à d'autres solutions. Cette definition peut etre soulevée aux contraintes. Par exemple, l'action d'une symétrie de variable à une contrainte modifie le champ d'application de la contrainte (c'est-à-dire, les variables sur lesquelles agit la contrainte).

Théorème 1 [***95***]: Étant donné un ensemble de symétries Σ de C, si S est un son (complet) jeu de symétrie briser les contraintes pour Σ alors σ (S) pour tout σ ∈ Σ est également un son (complet) ensemble de la brisure de symétrie des contraintes pour Σ.

Différentes symétries choisir des solutions différentes dans chaque classe de symétrie. En fait, si nous avons une solution particulière dans esprit, nous pouvons le récupérer à l'aide d'une symétrie appropriée d'un ensemble brisure de symétrie des contraintes. Laissez sol (C) l'ensemble des des solutions d'un ensemble de contraintes C.

Théorème 2 [***95***]: Étant donné un ensemble de symétries Σ d'un ensemble de contraintes C, un ensemble de sons S de sym-métrie rupture contraintes, et une solution A de C, alors il existe une symétrie σ ∈ Σ tel que A ∈ sol (C ∪ σ (S)).

L'application de la symétrie à un ensemble de brisure de symétrie con-traintes ne change pas les symétries qui sont élimi-vire. On dit qu'un ensemble de contraintes S rompt la symétrie σ d'un problème C si et seulement si il existe une solution A de C ∪ S tels que σ (A) n'est pas une solution de C ∪ S, et complètement pauses ssi de symétrie pour chaque solution A de C ∪ S, σ (A) n'est pas une solution de C ∪ S. Étant donné un groupe de symétrie Σ, on dit que un ensemble de contraintes supprime (complètement) Σ ssi il bat tous les symétries non-identité dans Σ.

Théorème 3 [***95***]: Étant donné un problème C et un groupe de symétrie Σ, si S (complètement) pauses Σ alors σ (S) (totale) rompt Σ pour tout Σ σ ∈ de.

Ces idées ont été utilisées comme base théorique pour une méthode de suppression de symétrie hybride qui essaie d'atteindre le meilleur des deux mondes, affichant les contraintes de suppression de symétrie statique pendant la recherche de façon dynamique en fonction de la direction de l'heuristique de branchement [***95***]:

3.4.2.3 Suppression des symétries des (lignes/ colonnes) (Modèle des matrices)/ Symétries des valeurs

Comme la suppression de symétrie semble insoluble en général, une direction majeure de recherche est d'identifier des cas particuliers qui se produisent dans la pratique, où le groupe de symétrie est plus facile à supprimer. On trouve dans la littérature deux de ces cas: la symétrie des lignes/ colonnes, et la symétrie de valeurs.

3.4.2.3.1 Suppression des symétries des lignes et des colonnes

Une matrice de variables de décision à une symétrie de ligne si et seulement si, étant donné une solution, toute permutation des lignes donne aussi une solution. De même, la matrice a une symétrie de colonne si et seulement si, étant donné une solution, toute permutation de colonnes est aussi une solution. Prenons l'exemple du

problème de l'EFPA, le modèle a à la fois des symétries des lignes et des symétries de colonnes.

Exemple : le problème 'The Equidistant Frequency Permutation Array (EFPA)' [96] C'est un problème difficile dans la théorie du codage. L'objectif est de trouver un ensemble de v mots de code, chacun de longueur qλ de telle sorte que chaque mot λ contient une copie des symboles 1 à q, et chaque paire de mots de code a une distance de Hamming d. Par exemple, pour v = 4, λ = 2, q = 3, d = 4, une solution est :

$$\begin{matrix} 0 & 2 & 1 & 2 & 0 & 1 \\ 0 & 2 & 2 & 1 & 1 & 0 \\ 0 & 1 & 0 & 2 & 1 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \end{matrix}$$

Ce problème à des applications dans la théorie de la communication, et est liée à d'autres problèmes combinatoires comme 'finding orthogonal Latin squares'. On peut modéliser ce problème par un v par qλ tableau de variables de décision avec les domaines 1 à q.

Ce modèle du problème EFPA présente une symétrie de variables puisque on peut permuter les variables dans n'importe quelles deux lignes, ou dans n'importe quelles deux colonnes et on obtient toujours une solution. Le modèle a également une symétrie de valeur puisque on peut inter-changer les valeurs dans toute solution.

3.4.2.3.1.1 La méthode DOUBLE LEX

Les symétries des lignes et de colonnes se produit dans de nombreux modèles avec des matrices de variables de décision [97, 98, 99]. On peut supprimer les symétries des lignes et de colonnes en utilisant la méthode Lex-Leader. Malgres que cela n'est pas pratique en général. En effet, une matrice n × m a à n! m! symétries de lignes et de colonnes, et chaque symétrie exigerait une contrainte d'ordre lexicographique séparée. Pour supprimer toutes les symétries des lignes, on peut poster un certain nombre linéaire de contraintes d'ordre lexicographique sur les lignes. De même, pour supprimer toutes les symétries de colonnes, on peut écrire un nombre linéaire de contraintes d'ordre lexicographique sur les colonnes. Dans le cas des symétries de lignes et de colonnes a la fois, on peut utiliser les contraintes DOUBLELEX qui ordonne lexicographiquement les lignes et les colonnes [99]. En

fait, les contraintes DOUBLELEX peuvent être dérivés d'un sous-ensemble des contraintes introduit par la méthode Lex-Leader [97]. En conséquence, DOUBLELEX ne supprime pas toutes les symétries des lignes et des colonnes. Néanmoins, c'est une technique souvent très efficace en pratique.

Reprenons la solution précèdent de l'exemple EFPA : si on ordonne les lignes de la solution lexicographiquement, on peut obtenir une solution dont les lignes et les colonnes sont ordonnées lexicographiquement :

$$\begin{matrix} 0 & 2 & 1 & 2 & 0 & 1 \\ 0 & 2 & 2 & 1 & 1 & 0 \\ 0 & 1 & 0 & 2 & 1 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \end{matrix} \quad \begin{matrix} \text{order} \\ \Rightarrow \\ \text{rows} \end{matrix} \quad \begin{matrix} 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 0 & 2 & 1 & 2 \\ 0 & 2 & 1 & 2 & 0 & 1 \\ 0 & 2 & 2 & 1 & 1 & 0 \end{matrix}$$

D'une manière similaire, si on ordonne les colonnes de la meme solution lexicographiquement on obtient une solution différente dont lequel les lignes et les colonnes sont ordonnées à la fois lexicographiquement. :

$$\begin{matrix} 0 & 2 & 1 & 2 & 0 & 1 \\ 0 & 2 & 2 & 1 & 1 & 0 \\ 0 & 1 & 0 & 2 & 1 & 2 \\ 0 & 0 & 1 & 1 & 2 & 2 \end{matrix} \quad \begin{matrix} \text{order} \\ \Rightarrow \\ \text{cols} \end{matrix} \quad \begin{matrix} 0 & 0 & 1 & 1 & 2 & 2 \\ 0 & 1 & 0 & 2 & 1 & 2 \\ 0 & 1 & 2 & 0 & 2 & 1 \\ 0 & 2 & 2 & 1 & 1 & 0 \end{matrix}$$

Les trois solutions sont dans la même classe de symétries de lignes et de colonnes. Ces deux nouvelles solutions satisfont la contrainte DOUBLELEX. Par conséquent, DOUBLELEX peut laisser plusieurs solutions dans chaque classe de symétrie donc cette méthode de suppression de symétrie n'est pas complète. En fait, DOUBLELEX est une méthode très incomplète pour la rupture de la symétrie. Elle peut laisser n! solutions symétriques dans un modèle matriciel $2n \times 2n$. En outre, elle fournit seulement une traitabilité partielle. S'il est polynomial de vérifier une contrainte DOUBLELEX étant donnée une affectation complète, il ne est pas traitable à propager complètement lors d'une affectation partielle. Autrement dit, supprimer toutes les symétries de valeurs est NP-difficile.

Théorème 5 [100] : la propagation de la contrainte DOUBLELEX est NP-difficile.

3.4.2.3.1.2 Les contraintes Snake-lex

Une méthode alternative intéressante pour casser les symétries des lignes et de colonnes utilise un ordre lexicographique qui linéarise la matrice d'une manière comme un serpent au lieu de ranger les lignes. Cette méthode ajoute la première ligne à l'inverse de la deuxième ligne, et le résultat est ajouté à la troisième ligne, puis l'inverse de la quatrième ligne, et ainsi de suite. La suppression des symétries des lignes et de colonnes avec cet ordre n'est très applicable, comme il est avec l'ordre lexicographique le plus classique où les matrices sont ordonnées par rangée des lignes.

Théorème 6 [101] : Décider si une affectation est la plus petite dans l'ordre Snake-lex dans le cas des permutations des lignes et des colonnes est NP-difficile.

La contrainte SNAKELEX, qui est dérivée de la méthode Lex-Leader utilisant un tel ordre de snake-lex, a été définit comme une alternative prometteuse à DOUBLELEX [102]. Pour supprimer les symétries de colonnes, SNAKELEX assure que la première colonne est lexicographiquement inférieure ou égale à la deuxième et la troisième colonne a la fois, qua l'inverse de la deuxième colonne est lexicographiquement inférieure ou égale à l'inverse des troisième et quatrième colonnes à la fois, et ainsi de suite jusqu'à ce que l'avant dernière colonne comparée à la dernière colonne. Pour la suppression des symétries des lignes, Snakelex assure que chaque paire de lignes voisines : $X_{1,\,i}, \ldots, X_{n,\,i}$ et $X_{1,\,i+1}, \ldots, X_{n,\,i+1}$ satisfont l'ordre lexicographique :

$$< X1, i, X2, i+1, X3, i, X4, i+1, \ldots > \leq \text{lex} < \text{X1, i} + 1, \text{X2, i, X3, i} + 1, \text{X4, i}, \ldots >$$

DOUBLELEX supprime le sous-ensemble des symétries de lignes et colonnes qui échangent les lignes et les colonnes voisines. SNAKELEX, par comparaison, supprime un sur-ensemble strict de ces symétries. Cette méthode est un peu plus couteuse dans la propagation, mais ce coût plus élevé est souvent utile lors de la suppression de la recherche. Comme la méthode DOUBLELEX, SNAKELEX est aussi une méthode de suppression de symétrie incomplète qui peut laisser un nombre exponentiel de solutions symétriques [100].

3.4.2.3.2 Suppression des symétries de valeurs

Les Problèmes avec symétrie des lignes/ colonne contiennent aussi souvent des symétries de valeur. Par exemple, le problème de l'EFPA a la symétrie de ligne/ colonne et la symétrie de valeur.

3.4.2.3.2.1 La méthode DOUBLE LEX

Il faut noter d'abord que l'interaction entre le problème et les contraintes DOUBLELEX peut dans certaines circonstances casser toutes les symétries de valeurs. Par exemple, dans le modèle du problème de l'EFPA, toutes les symétries valeur sont déjà éliminées [96].

3.4.2.3.2.2 La méthode de Puget

En général, les symétries de valeur peuvent rester présentes après la suppression des symétries des lignes et des colonnes. Comment ces symétries de valeur peuvent etre eliminées par la méthode de Puget? Puget a donné une méthode générale pour casser un certain nombre de symétries de valeur en temps polynomial [103]. Compte tenu d'un problème de surjection dans lequel toutes les valeurs se produisent au moins une fois, il introduit des variables Zj pour représenter l'index de la première occurrence de chaque valeur:

$$X_i = j \Rightarrow Z_j \leq i$$
$$Z_j = i \Rightarrow X_i = j$$

Les symétries de valeurs sur le Xi sont transformées en symétrie de variable sur les Zj. Cette symétrie de variable est particulièrement facile à casser comme la variable Zj prend des valeurs toutes différentes. Nous avons simplement besoin de poster contraintes d'ordre appropriées sur Zj.

Considérons, par exemple, la symétrie d'inversion qui mappe 1 à m, 2 à m - 1, etc. la méthode de Puget supprime cette symétrie avec l'unique contrainte d'ordre: Z1 <Zm. Malheureusement la méthode de Puget pour casser les symétries de valeur n'est pas compatible en général avec la suppression symétries des lignes/ colonnes en utilisant ROWWISE LEXLEADER. Cela corrige le théorème 6 et le Corollaire 7 dans [103], qui prétendons que, à condition d'utiliser le même ordre des variables dans chaque méthode, il est compatible pour poster descontraintes lex-leader pour supprimer la symétrie de variable et les contraintes de Puget pour supprimer la symétrie de valeur. Il n'y a aucun ordre des variables dans la méthode de Puget qui est compatible avec la suppression des symétries de lignes/ colonnes en utilisant la méthode lex-leader (ou toute autre méthode comme DOUBLELEX sur cette base).

3.4.2.3.2.3 La précédence de valeurs

On termine cette section par un cas particulier mais commun où la suppression de symétrie de variable et de valeur ne tombe pas en conflit. Lorsque le partitionnement des valeurs en ensembles interchangeables, la méthode de Puget est équivalent à supprimer les symétries en appliquant la précédence de valeur [104, 105]. Soient deux valeurs interchangeables i et j avec i <j, une contrainte de PRECEDENCE de valeur assure que si i se produit alors la première occurrence de i est avant celle de j. Il est correcte de supprimer les symétries des lignes/ colonnes avec ROWWISE LEXLEADER et la symétrie de valeur avec PECEDENCE lorsque la précédence de valeur considère les variables soit dans l'ordre ligne par ligne ou dans l'ordre colonne par colonne. Il s'agit d'une simple conséquence du théorème 1 dans [104]. Il s'ensuit qu'il est aussi correct d'utiliser PECEDENCE pour supprimer les symétries de valeur lors de l'utilisation des contraintes comme DOUBLELEX dérivée de la méthode lex-leader.

3.4.2.4 L'ordre Gray-code

Une autre méthode alternative intéressante pour supprimer les symétries qui essaye d'éliminer toutes les solutions symétriques sauf la solution symétrique la plus petite mais dans un autre ordre que l'ordre lexicographique. Par exemple, on a vu certaines avantages de l'utilisation de l'ordre Gray-code [101]. L'ordre Gray-code est un ordre total sur les affectations utilisées dans les codes de correction d'erreurs. Le code Gray 4- bits ordonne les affectations comme suit:

$$0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100$$
$$1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000$$

Un tel ordre peut prendre différentes solutions dans chaque classe de symétrie. Cela peut réduire le conflit entre la suppression de symétrie, les contraintes du problème et l'heuristique de branchement. L'ordre Gray-code possède certaines propriétés qui peuvent le rendre utile pour supprimer la symétrie. Par exemple, les affectations voisins dans l'ordre diffèrent seulement à une position, et en retournant un bit renverse l'ordre des bits suivants. En plus, il n'y a pas renommage des variables et des valeurs qui mappent l'ordre des solutions dans l'ordre Gray-Code en celui de

l'ordre lexicographique. Néanmoins, casser les symétries des lignes et des colonnes par cet ordre sur les solutions reste intraitable.

Théorème 7 [101] : Décider si une affectation est plus petite dans l'ordre Gray-code lors des permutations des lignes et des colonnes est NP-difficile.

3.4.2.5 Combinaison des symétries dans les méthodes statiques

Une autre question importante est de savoir comment la combinaison de symétries est traitée. En effet, on peut toujours prendre un groupe de symétrie dont la suppression complète des symétries est traitable (symétries des lignes), le combiner avec un autre groupe de symétrie dont la suppression complète est aussi traitable (symétries de colonnes), et se retrouver avec un groupe de symétrie (symétries de lignes et de colonnes) dont la suppression et donc en général nous manquons de bonnes méthodes pour la suppression de toutes les solutions symétriques.

Une autre combinaison difficile, c'est lorsqu'il s'agit d'une symétrie de variable et d'une symétrie de valeur à la fois. Comment supprimer les deux types de symétrie simultanément? On peut utiliser la méthode Lex-Leader, mais ce n'est pas très pratique lorsqu'on a beaucoup de symétries de variable et de valeur. Malheureusement, la méthode de Puget pour supprimer les symétries valeurs sur les variables Zj n'est pas compatible en général avec la suppression des symétries de variables via la Méthode Lex-Leader sur les variables Xi. Cela corrige les théorèmes 6 et 7 dans le Corollaire [106] qui prétendent que les deux méthodes sont compatibles si nous utilisons le même ordre de variables dans chaque méthode.

Thèorème [100] :

Il existe des problèmes sur lesquels poster les contraintes lex-leader pour supprimer les symétries de variables et l'application de la méthode de Puget pour supprimer les symétries de valeurs supprime toutes les solutions dans une classe de symétrie indépendamment des ordres sur les variables utilisées par les deux méthodes.

Il y'a, cependant, un cas commun et important et où les deux méthodes de suppression de symétries de variables et de suppression de symétries de valeurs sont compatibles. Si les valeurs sont partitionnées en des ensembles interchangeables, alors il est toujours cohérent et consistant pour poster les contraintes de suppression de

symétries en utilisant les deux méthodes fournies [107, 108;109]. C'est encore un cas où la traçabilité est limitée comme l'élagage ou la suppression de toutes les valeurs symétriques est NP-difficile.

3.4.2.6 Les effets négatives des techniques de modélisation dans sur les performances de recherche

Les Techniques statiques d'ajout de contraintes pour la suppression des symétries (modélisation) n'améliorent pas forcément les performances de recherche et elles dépendent fortement de l'algorithme de résolution utilisé qui lui-même a des comportements différents vis-à-vis de la topologie de l'espace de recherche.

En effet, la littérature montre que le grand nombre de travaux sur les techniques de suppression de symétries statiques ont été fait et étudié autour de la recherche avec retour arrière (backtrack), il est évident aussi que l'application de cette méthode pour les méthodes de recherche exhaustives ou énumératives donne de bons résultats en terme de temps d'exécution et d'étapes de recherche d'une solution, cependant, pour les autres algorithmes de recherche, la technique de modélisation peut avoir un effet négatif sur la performance de résolution du problème, et on peut observer dans certains cas qu'elle donne des anomalies comme celles du à l'interaction entre ces contraintes et les heuristiques ordonnées. Ce type d'anomalie peut être aussi observé dans des cas de combinaison de techniques de recherche qui représentent des bonnes techniques mais la combinaison produit des résultats inattendues et surprenons.

Pour les méthodes de recherche locale, la suppression de symétries peut avoir elle aussi des effets négatives sur les performances de recherches mais elle ne produit pas d'anomalies. Cela peut être expliqué par, la résolution d'un problème avec peu de solutions par une méthode de recherche locale sera intuitivement très lourde et donc les techniques d'ajout de contraintes pour suppression de symétries qui éliminent certaines solutions ne sont pas adéquates. Exceptionnellement, il a été trouvé que l'ajout de contraintes pour suppression de symétries peut améliorer les performances de recherche locale comme pour (rondom k SAT, 3-coloring et les CSP(s) binaires.).

La question maintenant qui doit être posée, est comment la suppression de symétries accélère la recherche '*backtrack'* ?autrement dit, si la suppression de solutions ralenti la recherche locale alors pourquoi pas la recherche avec retour

arrière ? la réponse à cette question est que la distance totale à une solution dépend du partitionnement des solutions, alors si les solutions apparaissent en regroupement (clusters) la distance totale à une solution sera plus grande que si les solutions sont distribuées uniformément et que les temps moyen pris par la méthode *Backtrack* pour la recherche de solutions augmente avec la distance moyenne à la solution. On conclut donc que le regroupement par clusters ralenti la recherche *backtrack*, les techniques de suppression de symétries remède à ce problème, ou elles réduisent la taille des clusters, par conséquent, la suppression de certains régions de l'espace de recherche réduit la distance entre clusters et accélère la recherche *Backtrack,* contrairement à la méthode de recherche locale qui sa performance dépend plus du nombre de solutions et non pas de leurs distributions, à partir de cette explication et ou observation on peut tirer un ensemble de conclusion, la première est que l'ajout de contraintes pour suppression de symétries ralenti la recherche locale et a des effets négative sur les performances de cette recherche. On conclut aussi que la recherche locale et la recherche backtrack nécessitent des approches de modélisation différentes. Et finalement la conclusion qui nous intéresse le plus c'est que pour certains modèles avec symétries, il est préférables de les résoudre avec une méthode de recherche comme la recherche locale et sans ajout de contraintes pour la suppression de symétries que d'ajouter des contraintes complexes et couteuses, de telle manière la tâche de modélisation sera facile, le temps d'exécution est réduit est moins de consommation d'espace mémoire, de plus, les avantages de ces modèles comme (peu de variables et de contraintes, des contraintes de basse arrité et un cout de raisonnement ou propagation efficace) et qui ne sont pas adéquats pour une recherche avec suppression de symétries ne seront pas rejetés (elles sont préservés).

Ces conclusions ne s'appliquent que pour les techniques d'ajout de contraintes au modèle pour la suppression des symétries et non pas aux autres approches qui suppriment la symétrie dynamiquement pendant la recherche. Ces dernières ont été créés afin d'éviter les effets que nous avons discutés avant. Cependant l'approche statique semble plus populaire car il est évident qu'il est plus facile de modifier un modèle que de modifier ou d'adapter un algorithme de recherche. Il faut noter que ces conclusions ne concernent pas les contraintes unaires pour la suppression des symétries, qui simplement fixe la valeur de la variable, semble probablement aide la recherche locale.

3.4.3 Les méthodes dynamiques de suppression de symétries (Adaptation d'algorithmes de recherche)

Les méthodes dynamiques de suppression des symétries sont celles qui opèrent pour supprimer la symétrie au cours du processus de recherche. SBDD et SBDS sont deux de ces méthodes décrites dans cette section. Dans ces deux méthodes la symétrie agit sur les paires variable / valeur. L'élimination de la symétrie par heuristique est incluse dans cette catégorie, car bien que ces heuristiques ordonnant les variables et les valeurs soient entièrement définies avant que la recherche commence, ils sont utilisés lors de la recherche. On va décrire ces méthodes dans les sections suivantes.

3.4.3.1 Suppression des symétries pendant la recherche de solution (SBDS)

Les arbres de recherche excluant la symétrie ont été mis en place par Backofen et Will [110, 111]. Gand et Smith [112] décrivent plus en détail la mise en œuvre de cette technique, en utilisant le nom "Suppression des Symétries pendant la recherche", mais c'est ce dernier nom et son acronyme «SBDS» qui semble avoir été gardé comme nom général pour cette méthode. C'est peut-être un peu malheureux car il y'a beaucoup d'autres façons de supprimer la symétrie lors de la recherche, notamment la méthode SBDD discutée à la section suivante.

L'idée de base de SBDS consiste à ajouter des contraintes à un problème de sorte que, après retour en arrière d'une décision de recherche, les contraintes SBDS s'assure qu'aucune symétrie équivalente de cette décision n'est explorée. C'est une technique dynamique, puisque on ne peut pas ajouter des contraintes jusqu'à la connaissance de la décision de recherche courante. En générale, SBDS peut s'appliquer sur n'importe quel type de décision recherche. Cependant, Pour simplicité on va considérer dans la suite sue les décisions de recherche sont de la forme var = val.

On va d'abord illustrer par un exemple le type de contraintes ajoutées par SBDS. Un arbre de recherche pour le problème des huit reines où les contraintes de symétries ajoutées par SBDS sont indiquées par ♦ peuvent sont présentées sur la figure 10.10.

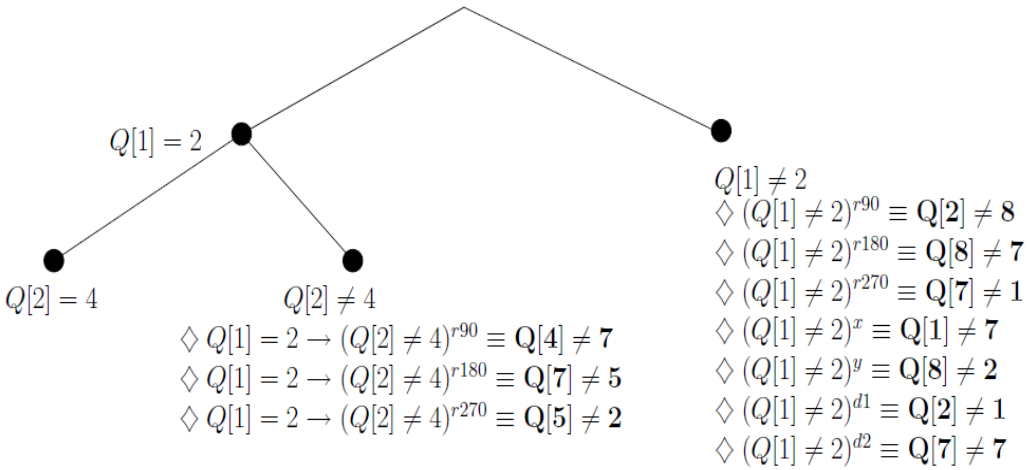


Figure 3-12 Example of SBDS on a search tree with the 8-queens problem.

Description de la méthode SBDS sur le problème symétrique des 8 reines :

- En commençant à partir de la racine, la première décision de recherche dans l'arbre de recherche sur la figure 10.10 est Q [1] = 2, c'est à dire la reine en ligne 1 va en position 2. SBDS n'ajoute pas de contraintes à la décision positive Q [1] = 2. Si on revient à la racine par un backtrack, on peut affirmer que Q [1] ≠ 2. A partir de là, on essayera jamais d'explorer tout état qui contient une équivalence symétrique à Q [1] = 2. On peut atteindre ça en ajoutant des versions symétriques de Q [1] ≠ 2, c'est-à-dire en ajoutant (Q [1] ≠ 2)^{g} pour chaque g dans le groupe. Comme résultat on obtient la branche droite de la figure 10.10.
- Au niveau suivant de la recherche, sur le côté gauche de l'arbre de recherche, la prochaine décision de recherche est Q [2] = 4. De même, on n'ajoute pas de contraintes à cette décision positive. Mais si on revient en arrière à partir de cette decision, on affirme Q [2] = 4. Il ne serait plus correct d'interdire chaque version symétrique de cette décision, parce qu'il se pourrait que la décision initiale de recherche Q [1] = 2 à supprimer déjà une partie des symétries. Dans cet exemple, Q [1] = 2 supprime Q [1] = 7, Q [7] = 7, Q [2] = 1 et Q [8] = 2 puisque ces carrés sont sur la même ligne, colonne, diagonale, et en diagonale, respectivement. Cela signifie qu'aucune solution contenant Q [1] = 2 peut éventuellement avoir les symétries x, y, d1 ou d2. Il n'est pas nécessaire donc

d'ajouter des contraintes pour ces symétries en dessous de Q [1] = 2, et elles sont donc supprimées de la figure Figure 10.10.

- Toujours en considérant le même noeud, le cas le plus complexe est celui des symétries r90, r180 et r270. À ce stade de la recherche, on ne sait pas si ces symétries s'appliquent ou non. Ils peuvent tenir dans certains états futurs et non dans d'autres. On ne peut pas exclure, par exemple, $(Q\ [2] = 4)^{r90}$ pour tous les états futurs, comme on risque de perdre des solutions dans les états où r90 ne tient pas, mais également on peut faire une recherche redondante si on l'exclut pas dans les états où r90 ne détiennent pas. SBDS résout ce dilemme en ajoutant des contraintes qui excluent conditionnellement $(Q\ [2] = 4)^{r90}$. Cette contrainte conditionnelle stipule que si le r90 n'est pas supprimé (c'est à dire $Q\ [2] \neq 8$), alors on obtient $(Q\ [2] \neq 4)^{r90}$, c'est-à-dire $Q\ [4] \neq 7$.

De manière générale, les contraintes que SBDS ajoute sont décrites comme suit : Considérons un nœud à la recherche où l'assignement partiel A doit être étendu par la décision var = val. Pour toute symétrie g du problème, on peut ajouter la contrainte:

$$A \text{ et } A^g \text{ et } var \neq val \Rightarrow (var \neq val)^g$$

Cette contrainte est équivalente à:

$$(A \Rightarrow var \neq val) \Rightarrow (A \Rightarrow var \neq val)^g$$

En effet, cette équation est vraie avant le début de la recherche, car elle est valable pour tout A, var, val et g. De ce point de vue, on peut voir que SBDS met des choix heuristiques pour une variété infinie de telles contraintes à ajouter. Si l'on ajoute la contrainte au point de la recherche là où on fait un backtrack à partir du choix de var = val, alors on sait que A est vrai et on sait aussi que var = val, conduisant à la plus simple mais dans un contexte équivalent à la forme:

$$Ag \Rightarrow (var \neq val)^g$$

On peut expliquer cette simple forme en faisant remarquer qu'il faut s'assurer que seules les symétries non supprimées qui sont traitées, il est donc vérifié que A^g est toujours valable. Ensuite, pour s'assurer que les sous arbres symétriquement équivalents de l'arborescence actuelle ne seront pas explorés, la $(var \neq Val)^g$ est placée.

Backofen et Will ont prouvé que cette méthode est bonne et efficace dont le fait que l'espace de recherche non-symétrique sera complètement exploré [111], c'est à dire y'aura pas de solution qui peut être ratée ou perdue complètement. Backofen et will ont également montré que, tant que toutes les symétries sont correctement traitées, toutes les symétries seront éliminés, c'est à dire il n'y aura pas deux solutions équivalentes retournées par SBDS.

Ces implémentations exigent toujours aux programmeurs de contraintes une fonction distincte pour implémenter l'action de chaque symétrie g. Si un problème a un grand nombre de symétries donc ça va être un grand travail pour l'utilisateur pour les identifier et de les mettre en œuvre à la main. SBDS est utilisé avec succès, en dépit de cette difficulté, avec les problèmes contenant quelques milliers de symétries [113]. Une caractéristique de SBDS est qu'il supprime seulement les symétries qui ne sont pas déjà supprimées dans l'affectation partielle courante, ce qui évite de placer des contraintes inutiles. Une symétrie est supprimée lorsque l'équivalent symétrique à l'affectation partielle courante n'est pas compatible avec les contraintes du problème.

Puisque A^g implique tout l'ensemble des valeurs définies jusqu'à présent, Donc vérifier que la symétrie n'est pas supprimée peut être coûteux. Cependant, si A est étendue à la prochaine affectation partielle A1 alors A_1 = A + (var = val) (où var = val est la décision suivante sur l'arbre de recherche). Alors $A^g_1 = A^g + (var = val)^g$.

Un problème avec SBDS est que lorsque le nombre de symétries est grand, un grand nombre de fonctions de symétrie doivent être décrites. Dans le pire des cas, il pourrait être trop nombreux pour être compilé avec succès. Cette difficulté peut être résolue en choisissant un sous-ensemble des fonctions de symétrie à utiliser avec SBDS. McDonald et Smith [114] ont explorer cette idée de «suppression partielle de symétrie» avec des sous-ensembles aléatoires de symétries, et aussi donnent un algorithme pour choisir un sous-ensemble des fonctions de symétrie qui devrait, heuristiquement, supprimer une grande quantité de symétries. Malheureusement, il n'est possible d'utiliser la méthode en son intégralité avec tous les problèmes, mais seulement avec des petits problèmes avec de petits groupes de symétrie.

SBDS a quelques avantages majeurs par rapport à l'ajout de contraintes pour suppression des symétries avant la recherche. Tout d'abord, les symétries ne sont pas nécessairement des symétries de variables, contrairement la méthode lex-leader.

Deuxièmement, la solution trouvée dans chaque classe de solutions équivalentes est toujours la plus à gauche dans l'arbre de recherche étant traversé. Par conséquent, contrairement à la lex-leader, cela signifie que les heuristiques d'ordonnoncement de variables ou de valeurs peuvent être utilisées sans modification ni de SBDS ni de l'ensemble des fonctions de symétries utilisées. Alors on dit que SBDS «sont compatibles» avec les heuristiques utilisées par la recherche.

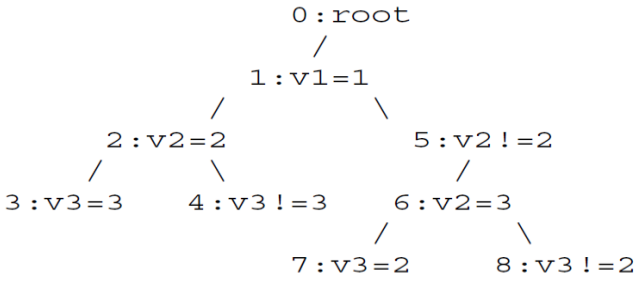
3.4.3.2 Suppression des symétries par la détection de dominance (SBDD)

La méthode de détection des symétries par Dominance (SBDD) a été développée indépendamment par Focacci & Milano [115], et par Fahle, Schamberger & Sellmann [116]. Le titre de SBDD provient de la dernière de ces articles, et a été adopté par la communauté CP comme un nom standard du procédé. Un algorithme similaire a été proposé par Brown, Finkelstein et Purdom en 1988 [62]. En fait, ce document décrit une version du groupe de calcul théorique de cet algorithme qui est maintenant connu de la communauté CP comme GAP-SBDD.

SBDD fonctionne en effectuant une vérification à chaque nœud de l'arbre de recherche pour voir si le noeud à explorer est équivalent symétriquement à celui déjà exploré, si c'est le ca, alors la branche va être supprimée. L'idée est simple, mais on a le problème qu'on doit sauvegarder tout l'arbre exploré dont la taille peut être en exponentielle. Une seule idée clé transforme SBDD en une méthode efficace en espace. Cette idée consiste à sauvegarder uniquement les nœuds des sous arbres au niveau des racines pleinement explorées. On ne vérifie pas si un nœud est équivalent en totalité à l'un de nos nœuds stockés. Au lieu de cela, on détermine si un nœud est équivalent à n'importe quel nœud qui est une extension de l'un des noeuds stockés, c'est-à-dire un noeud qui est dans un sous arbre complètement explorée. Puisque la recherche a fait retour arrière, alors soit avoir visité le nœud équivalent avant, ou déduite pour une autre raison qu'il n'est pas nécessaire de le visiter: dans les deux cas il n'est pas nécessaire de visiter l'équivalent symétrique.

Comme SBDS, SBDD est basé sur le branchement binaire entre l'affectation de varaible et la suppression d'une valeur du domaine de la variable. Un exemple montre le fonctionnement de SBDD dans la pratique. Soit un problème à trois variables v1, v2 et v3 soumises à une contrainte 'All-différent'. Le domaine de toutes les variables

est {1, 2, 3, 4}, et toutes les valeurs peuvent être permutées. Il y'a 24 solutions au problème. La Figure 10.11 montre une partie de l'arbre de recherche qui sera explorée



par une procédure de recherche en profondeur d'abord pour énumérer toutes les solutions, en supposant seulement la procédure de propagation 'All-différent'.

node	decisions	v_1	v_2	v_3	solution
0	-	1,2,3,4	1,2,3,4	1,2,3,4	
1	$v_1 = 1$	1	2,3,4	2,3,4	
2	$v_1 = 1,\ v_2 = 2$	1	2	3,4	
3	$v_1 = 1,\ v_2 = 2,\ v_3 = 3$	1	2	3	yes
4	$v_1 = 1,\ v_2 = 2,\ v_3 \neq 3$	1	2	4	yes
5	$v_1 = 1,\ v2 \neq 2$	1	3,4	2,3,4	
6	$v_1 = 1,\ v2 \neq 2,\ v_2 = 3$	1	3	2,4	
7	$v_1 = 1,\ v2 \neq 2,\ v_2 = 3,\ v_3 = 2$	1	3	2	yes
8	$v_1 = 1,\ v2 \neq 2,\ v_2 = 3,\ v_3 \neq 2$	1	3	4	yes

Table 1 Un arbre de recherche partiel pour illustrer SBDD qui montre les états de recherches correspondant a chaque noeud.

Les décisions sont sélectionnées dans un ordre lexicographique. Les nœuds sont représentés par n: δ où n désigne le nœud c'est le nième nœud à être élargi par la procédure de recherche et δ est la décision ou la négation d'une décision pour l'arc entre le nœud n et ses parents. La figure donne aussi pour chaque nœud n l'ensemble des décisions prises sur le chemin à partir du noeud racine à n, ainsi que les domaines des variables correspondant à son état. Quatre solutions ont été obtenues dans la recherche illustrée. Toutefois, la solution trouvée au noeud 7 est symétrique avec celle qui se trouve au nœud 3. Ces solutions sont {v1 = 1, v2 = 3, v3 = 2}, {V1 = 1, v2 = 2, v3 = 3} La première solution peut être mappée dans à la seconde par la permutation des symétries des variables v2 et v3. Toute permutation de variables est une symétrie du problème.

SBDD est basé sur la notion de 'No-goods' qui représentent les racines des sous- arbres maximales qui sont complètement traversés par une recherche en profondeur d'abord avant n. En Figure 4.11, le nœud 3 est un no-good w.r.t. pour le

noeud 4, le noeud 2 est un no-good pour le noeud 5 et tous les nœuds dans sa sous-arborescence.

Définition 4.25. (No-good). Le nœud ν est un no-good w.r.t. n s'il existe un ancêtre na de n s.t. ν est le fils gauche de na et ν n'est pas un ancêtre de n.

Pour chaque no-good, SBDD sauvegarde l'information à comparer avec l'état courant. On utilise l'ensemble des décisions en matière d'étiquetage du chemin depuis la racine de l'arbre à un no-good [117]. On écrit δ (n) pour cela. La colonne appelée «décisions» dans le tableau de la figure 10.11 donne l'information de décision correspondante à chaque noeud. On utilise également les informations d'état au niveau du nœud recherché. Plus précisément, on écrit Δ (S) pour un ensemble de paires vi = ai pour tous vi des variables dont les domaines sont réduits à un singleton. Dans le noeud 8 dans notre exemple, les décisions prises à partir du nœud racine sont δ (8) = {v1 = 1, v2 = 3}, alors Δ (8) = {v1 = 1, v2 = 3, v3 = 4}.

Définition 4.26. (Dominance) : On dit qu'un nœud n est dominée s'il existe un no-good ν *w.r.t.* par rapport à n et une symétrie g s.t. $(\delta(\nu))^g \subseteq \Delta(n)$. On dit que ν domine n.

SBDD est conceptuellement simple, il ne génère jamais les nœuds fils dominés, et il exclut les solutions dominées. Par conséquent, un nœud n est une feuille ssi est soit une solution, un échec ou un noeud dominé. Dans notre exemple, le no-good 2 domine le noeud 7. On a δ (2) = {v1 = 1, v2 = 2}. Utilisons la symétrie g qui permute les variables v2 et v3, On obtient $(\delta(2))^g$ = {v1 = 1, v3 = 2} qui est un sous-ensemble de Δ (7) = {v1 = 1, v3 = 2, v2 = 3}.

La définition originale de la dominance dans [116] est un peu plus différente et basée sur l'inclusion d'état. Un nœud n est dominé s'il existe un no-good ν pour n et une symétrie g tels que les domaines des variables dans ν^g contient les domaines des variables de n. cela présente l'inconvénient qu'il faut d'espace pour stocker les no-goods. En outre, Étant donné qu'on souhaite établir qu'un ensemble de ce noeud est un sur-ensemble du no-good passé, on souhaite que l'ensemble à ce noeud soit grand, et l'ensemble au no-good soit petit: ceci va le rendre aussi facile que possible afin de passer le test de dominance. D'autre part, on veut aussi la vérification de domination soit aussi facile que possible pour la mettre en œuvre et la plus rapide que possible

pour s'exécuter. On pourrait donc préférer d'utiliser une définition de la domination qui échoue aussi rapidement que possible, ce qui peut être un argument en faveur de la domination d'inclusion d'état.

Jusqu'à présent, on a montré juste la façon dont le contrôle de domination est effectué. L'algorithme de SBDD nécessite une fonction spécifique au problème Φ: (ν, n) → {false, true} qui donne vrai si le no-good précédent ν domine l'affectation partielle courante n. Pour des problèmes avec de petits groupes de symétrie, la fourniture de Φ peut présenter une plus grande charge pour le programmateur que les quelques fonctions de symétrie requises par SBDS. Pour des plus grands groupes, SBDD a l'énorme avantage qu'il a besoin d'espace très limité. Cependant, cela ne résout pas les problèmes de complexité en temps. Pour chaque paire ν et n, la recherche de g revient à résoudre un problème graphique d'isomorphisme, qui est connu pour être NP-complet.

Il semble y avoir trois techniques générales pour la mise en œuvre des contrôles de dominance. La première, le programmeur peut implémenter la vérification de la domination pour une classe particulière de problèmes, par exemple le cas de problème des 'golfeurs sociaux' [116]. La nécessité ici est un problème majeur avec SBDD. Codant une fonction qui va reconnaître si un nœud dans l'arbre de recherche est symétriquement dominé par un autre peut être difficile, et n'est pas généralisé entre des problèmes différents avec différents types de symétrie. Cependant, si une telle fonction peut être trouvée alors SBDD devient une méthode très efficace pour éliminer de grandes quantités de symétrie.

Sellmann et Van Hentenryck ont créé une fonction de détection de domination plus générale [118]. Cela peut conduire à des solutions très efficaces accordées à une application particulière, mais cette approche repose sur la compétence des programmateurs et impose un fardeau considérable sur eux. Deuxièmement, étant donné qu'il s'agit d'un problème NP-complet, on peut construire l'encodage d'une contrainte pour le problème de domination [117]. Ceci est particulièrement intéressant car il constitue à l'utilisation de la programmation par contraintes pour la théorie des groupes. Toutefois, il est encore nécessaire de construire un problème particulier de contraintes pour chaque nouvelle classe de problèmes à résoudre. La troisième approche consiste à utiliser la théorie des groupes directement [62, 119].

3. La symétrie dans la programmation par contraintes

Un raffinement important de SBDD est que, parfois, on peut avoir une notice pour le contrôle de dominance pendant la propagation [115, 116]. Supposons que la vérification de dominante peut signaler pour certain paire variable-valeur, si elle est définie dans le nœud courant, conduire vers le nœud courant déjà dominé.

Par la suite, on peut retirer cette valeur du domaine de la variable et de propager sur le résultat. Ceci peut effectuer une propagation extrêmement utile. Cependant, on doit prendre soin que les avantages ne compensent pas les coûts d'exécution des calculs nécessaires. Un autre point à noter est qu'il n'est pas nécessaire d'effectuer la vérification de dominance à chaque nœud de l'arbre de recherche. Tant que la vérification est réalisée à chaque nœud feuille donc seulement les solutions isomorphes seront retournées. De décider où appliquer la vérification est un compromis entre le coût des contrôles et l'optimisation de recherche qui en résulte.

Comme SBDS, SBDD peut être démontré comme une méthode de brisure de symétrie correcte et complète à condition que la vérification de domination soit implémentée correctement. C'est, exactement une solution de chaque classe d'équivalence est retournée. Comme SBDS, la solution trouvée est la plus à gauche dans l'arbre de recherche par rapport aux heuristiques ordonnant variables et valeurs. SBDD respecte donc l'ordre de variable de l'heuristique, et l'ordre de variable dynamique peut être utilisé sans aucune modification à SBDD [116].

Harvey [120] explique la liaison entre SBDS et SBDD. La différence entre le deux algorithmes est l'endroit où la brisure de symétrie aura lieu. SBDS place des contraintes de cesser les nœuds symétriquement équivalents à ceux déjà explorés dans la recherche, qui ne sont jamais atteints. D'autre part, SBDD élimine les nœuds ayant atteints et qu'il trouve qu'ils sont symétriques par rapport à une autre partie déjà explorée dans la recherche. En fait, on peut remarquer que la différence entre SBDD et SBDS que comme simplement une implémentation. Comme l'ensemble des solutions acceptables est le même dans chaque cas, une implémentation de SBDS est, en un sens, une implémentation de SBDD, et vice versa. Cependant, il y'a d'énormes différences pratiques. SBDD peut surpasser SBDS sur de nombreux problèmes, car il n'affiche pas les contraintes, donc n'a pas la charge d'attente d'un grand nombre de contraintes pour la brisure de symétrie à être propagées. SBDD peut être utilisé avec

succès dans des problèmes qui ont trop de symétries pour SBDS pour être une technique appropriée.

3.4.3.3 SBDS, SBDD et la théorie du groupe

Les deux SBDS et SBDD peuvent correctement et complètement supprimer toutes les symétries d'un problème. Cependant, ils ont aussi besoin de toutes ces symétries, ou une fonction de domination de symétrie, à être spécifiés par l'utilisateur. Depuis, en général, cela est impossible, GAP-SBDS [121] et GAP-SBDD [122] utilisent le système de la théorie du calcul de groupes de GAP [123] pour composer et traiter automatiquement les symétries. En conséquence, les utilisateurs ne doivent fournir qu'un ensemble de générateurs pour le groupe de symétrie, au lieu de toutes les symétries. En outre, GAP évite l'affichage des contraintes vides et multiples, et contribue au traitement des symétries supprimées. Toutefois, supprimer chaque symétrie seule peut introduire des frais généraux très importants à chaque noeud de recherche. Si ces débordements ne sont pas compensés par des réductions importantes, de grands ralentissements peuvent se produire.

3.4.3.4 La méthode GE-Trees

L'idée de construire GE-trees est l'idée la plus récente pour joindre la suite des méthodes de suppression de symétries dynamiques [77]. (GE signifie "Group Equivalence" », mais l'abréviation est utilisée universellement.) Il diffère des autres dans cette classe, comme un moyen beaucoup plus pour considérer les méthodes de suppression de symétrie, que d'une méthode de suppression de symétrie elle meme. Un GE-trees est défini comme un arbre dans lequel on ne trouve pas deux noeuds symétriquement équivalents, et dans lequel, pour chaque solution au problème, un noeud symétriquement équivalent doit être dans l'arbre. Les GE-trees sont définis de manière analogue aux recherches arborescentes en général: en particulier, les algorithmes sont libres pour s'arrêter avant qu'ils construisent un arbre complet, par exemple après avoir trouvé une première solution. GE-trees sont destinés à être considéré comme un paradigme conceptuel, utiliser pour classer et comparer les techniques de suppression de symétries. Toute méthode de construction GE-trees va (par définition) supprimer toutes les symétries du problème comme la recherche de solutions procèdent. SBDS et SBDD peuvent tous deux être considérés comme des méthodes pour construire GE-trees. Une analyse des propriétés de GE-trees (construit

par différentes méthodes telles que SBDS et SBDD, appliqués aux mêmes instances) peut permettre l'amélioration et l'extension des techniques existantes et même le développement de nouvelles méthodes.

Considérons l'exemple du coloriage du graphe où chaque couleur est indiscernable. De nombreux programmeurs ont réalisé que le premier nœud peut être coloré arbitrairement. Pour le deuxième nœud, il suffit de considérer ce qui lui donne de la même couleur ou une autre un arbitraire. Le troisième nœud ne doit être prendre que les couleurs attribuées aux deux premiers noeuds par un choix une arbitraire. Le processus se poursuit jusqu'à l'utilisation de toutes les couleurs. Cette intuition peut généralisée et formalisé. Roney-Dougal et al. Ont utilisé le paradigme GE-trees pour créer un algorithme polynomial pour supprimer arbitrairement les symétries de valeur. Cela peut être considéré comme une généralisation a base de la théorie du groupe d'un algorithme présenté dans [124]. C'est un algorithme spécial qui crée de nouveaux nœuds qui sont garantis pour être uniques dans l'arbre. En général, cela est difficile, mais Roney-Dougal et al. Ont montré que la nature particulière des groupes de symétries de valeur peut être utilisée pour construire GE-trees très efficaces [77]. Ils ont fait une comparaison expérimentale entre cette méthode et GAP-SBDD pour les problèmes qui ont seulement des symétries de valeur. GE-trees a été la meilleure méthode dans tous les cas.

3.4.3.5 La méthode de suppression partielle de symétries Shortcut SBDS

Shortcut SBDS est présenté par Gent et Smith [125] par le problème de coloration de graphe. L'idée est de réduire les débordements en utilisant uniquement des symétries actifs (ie, ceux pour qui σ (A) = A), depuis lors, les contraintes peuvent être affichés sans condition (à savoir, nous pouvons envoyer des messages: ┐σ (x = d) plutôt que σ (A)): ┐σ (x = d)). Notez que la méthode est aussi correcte que SBDS et SBDD mais, en général, on sacrifie l'intégralité d'accroître l'efficacité. Malheureusement, elle exige un effort au nom du programmeur pour obtenir la règle de raccourci réduit. En outre, le raisonnement qui mène à une règle simple pour ce problème ne transfère pas à tous les problèmes.

La méthode Shortcut SBDS est un exemple de suppression de symétrie partielle [125], qui observe que les méthodes de suppression de symétrie comme

SBDS peuvent être parfois plus efficace en considérant uniquement un sous-ensemble des symétries d'un problème. Ce sera le cas à chaque fois que le surcoût introduit par le traitement des symétries et l'affichage des contraintes ne sont pas compensés par les réductions associées à la recherche spatiale. McDonald et Smith [125] ont montré expérimentalement sur les tuiles étrangères et les golfeurs sociaux repères que l'utilisation d'un sous-ensemble des symétries des problèmes avec SBDS est bénéfique. Ils ont également montré que la taille du sous ensemble choisi n'est pas le seul facteur important, car le choix des symétries qui sont inclus affecte également les performances, avec des symétries qui suppriment la recherche à proximité de la racine étant la plus efficace. Bien que les auteurs donnent un algorithme pour choisir le meilleur sous-ensemble pour une donnée problème, il a deux inconvénients: il peut prendre un certain temps à courir pour seulement un petit avantage et, comme les méthodes de suppression de symétrie statiques, il faut une stratégie de recherche fixe.

3.4.3.6 La méthode STAB

Une autre méthode de suppression de symétrie partielle est la méthode STAB [126], une méthode de suppression dynamique qui, comme la méthode de Shortcut SBDS, supprime seulement les symétries actives (l'ensemble stabilisateur). La principale différence réside dans le type des contraintes supplémentaires de suppression de symétrie: STAB ajoute des contraintes lexicographiques, plutôt que les contraintes conditionnelles ajoutées par la méthode SBDS. Pour ce faire, STAB a besoin de calculer un isomorphisme de graphique à chaque noeud de recherche.

3.4.3.7 La méthode de suppression de symétrie par l'optimisation non stationnaire

La suppression de symétrie par la méthode d'optimisation non stationnaire [127] fonctionne par la recherche de contraintes lexicographiques violées, plutôt que de les appliquer. Donc à chaque nœud de recherche, elle fait une recherche locale pour une symétrie dont la contrainte lexicographique correspondant qui a été violée a été postée. La méthode est partielle (parce que le temps consacré à la recherche à chaque noeud est limité) et elle peut être combinée avec des contraintes lexicographiques statiques.

3.4.3.8 La méthode de suppression partielle de symétries LDSB

LDSB [128], est une méthode de suppression de symétrie partielle dynamique qui peut être vu soit comme une extension de la méthode Shortcut SBDS mais qui ne se base pas sur la composition des symétries, ou comme une restriction de la méthode

de STAB qui ne nécessite pas le calcul d'un isomorphisme de graphique à chaque noeud. Ceci est réalisé en se concentrant sur quatre types communs de symétries qui peuvent être cassées, en utilisant des algorithmes très efficaces. Il y'a plusieurs points importants à noter. Tout d'abord, pour trois des quatre types de symétries, les algorithmes peuvent être prouvés pour être complet, malgré la simplicité des algorithmes. Deuxièmement, LDSB ne compose pas vraiment les symétries, mais les applique plutôt à plusieurs reprises pour les littéraux. Ceci est important pour l'efficacité: lors de la composition des symétries peut conduire à de grands ensembles de symétries de façon exponentielle, LDSB calcule au plus quadratique (|X| * D|) du nombre de littéraux dans (X; D; C) pour obtenir le même résultat. En troisième lieu, tout ceci est réalisé en utilisant uniquement des calculs très simples sur la base des résultats de la théorie de groupe.

3.4.3.9 Les méthodes heuristiques de recherches

Une autre approche pour la suppression des symétries dans les CSP est d'utiliser les symétries pour guider la recherche. Meseguer et Torras [69] proposent des heuristiques pour ordonner les variables qui sélectionnent la variable menant à un sous-espace de recherche avec le plus grand nombre d'états distincts. Ils proposent aussi une procédure pour la supression des symétries de valeurs pendant recherche basée sur l'enregistrement de nogood.

3.4.4 Les méthodes structurelles de suppression de symétries statiques et dynamiques

Beaucoup de problèmes de satisfaction de contraintes (CSP) présentent naturellement des symétries. La suppression des symétries peut considérablement améliorer les performances [129, 130, 131, 132]. Comme on a déjà vu dans les sections précédentes, une contribution importante dans ce domaine a été faite consistant au développement de différents schémas de suppression de symétries comme les méthodes de suppression de symétries pendant la recherche (SBDS), et les méthodes de suppression de symétries par la détection de dominance (SBDD), pour ces approches, des travaux ont été fait pour la recherche de classes de symétries pour les CSPs qui sont réalisables et qui admettent des algorithmes de détection de dominance en temps polynomial, par exemple, les premiers travaux ont montré que la suppression de symétries pour plusieurs classes de symétries de valeurs nécessite un

temps et un espace constant Par la suite, ces résultats ont été généralisés pour la suppression de toutes les symétries de valeurs. Cependant, en général, ces approches dynamiques de suppression des symétries peuvent nécessiter des ressources exponentielles pour supprimer toutes les symétries avec interchangeabilité de valeurs et de variables en même temps. En effet, certains algorithmes peuvent exiger un espace exponentielle pour stocker tous les nogoods générés par les symétries, tandis que d'autres peuvent prendre un temps exponentiel afin de savoir si une affectation partielle est symétrique à l'un des nogoods existants. En conséquence, les applications pratiques placent souvent des limites et des restrictions sur le nombre de nogoods pouvant être stockées et/ ou des restrictions sur le choix des symétries à supprimer. une autre approche importante qui consiste à éliminer les symétries statique en ajoutant des contraintes avant que la recherche commence. Malheureusement, en général, un certain nombre super-exponentiel de contraintes peut être nécessaire pour supprimer toutes les symétries. Par exemple le schéma lex-leader ajoute une contrainte par symétrie, mais le nombre de symétries est souvent super-exponentielle (une matrix m × n avec interchangeabilité entière des lignes et des colonnes à m! · n! symétries). Par conséquent, les applications pratiques généralement n'ajoutent que certaines de ces contraintes de suppression de symétrie. Par la suite, d'autres travaux ont revisité les CSP avec symétrie de valeur et de variables par morceaux simultanément, où un algorithme de détection de la domination à temps polynômial a été donné et le nom de «suppression de symétrie structurelle» a été inventé. Et cela à donner l'ouverture a d'autres travaux qui étudient cette fois ci la suppression de symétries d'un point de vue différent. Ou l'objectif de ces travaux consiste à identifier des classes de CSP pour lesquels la suppression de symétrie est réalisable. I.e., en un temps et un espace polynomial, en utilisant des procédures de recherche dédiées, qui exploite la structure du problème.

3.4.4.1 Définitions et préliminaires

On trouve ici une autre définition de CSP non standard mais qui simplifie les preuves et d'autres définitions considérablement. L'idée de base consiste à l'abstraction de l'ensemble des contraintes du CSP par une fonction booléenne qui retourne vrai si toutes ces contraintes sont satisfaites (la structure des contraintes n'est pas prise en considération), les solutions sont alors représentées comme des fonctions partant des variables vers les valeurs possibles.

3. La symétrie dans la programmation par contraintes

Définition 1 : (CSP, Assignement, Solution) Un CSP est un triplé <V, D, C>, ou :

V représente l'ensemble des variables,

D représente l'ensemble des valeurs possibles pour ces variables,

C (V-> D) -> Bool représente une contrainte qui spécifie qu'elle affectations de valeurs aux variables qui représentent des solutions.

Une affectation pour un CSP P = <V, D, C> est une fonction α:V -> D.

Une solution au CSP P = <V, D, C> est une affectation σ pour P tel que C (σ) = Vrai.

L'ensemble de toutes les solutions au CSP P est noté par Sol (P).

Définition 2 : (Affectation partielle, Scope, Image) : Une affectation partielle pour un CSP P = <V, D, C> est une fonction α : W→ D, ou W $\underline{C}$ V. W représente le scope de α noté par scope (α). L'image de α noté par image (α) est l'ensemble {α(v) | v ϵ scope (α)}. Pour chaque valeur d ϵ image (α), on utilise α^{-1}(d) pour désigner l'ensemble {v | v ϵ scope (α) et α (v) = d}. on désigne par ε l'affectation partielle vide.

Il faut noter que chaque affectation ou solution a un CSP P = <V, D, C> est une affectation partielle pour P, avec scope V. on désigne une affectation partielle α par une conjonction d'équations et donc on peut la voir comme une contrainte : vi1 = α (vi_1) et … et vi_k = α (vi_k) et scope (α) = {vi1, …, vik}.

Exemple : L'affectation partielle v1 = 1 et v2 = 2 et v3 = 3 représente la fonction qui a le scope {v_1, v_2, v_3} et qui affecte la valeur i a v_i.

Définition 3 : (compléter une affectation partielle). compléter une affectation partielle α pour un CSP P = <V, D, C> consiste a faire l'extension θ de α avec scope (θ) = V. l'ensemble de toutes les terminaisons (complétions) de α pour P est noté par Comp(α, P). il faut noter également que l'ensemble de toutes les terminaisons d'une solution σ est un singleton {σ}.

Définition 4 : (Nogood) : Un « Nogood » pour un CSP P est une affectation partielle α pour P qui ne peut etre étendue en une solution, soit Comp (α, P) ∩ Sol (P) = ϕ.

L'idée derrière le substantif < nogood > veut qu'aucune affectation partielle ne doit étendre tout nogood précédemment identifié.

Définition 6 : (Violation d'un nogood). Une affectation partielle θ d'un CSP P viole un nogood α de P si θ est une extension de α.

A noter que toute extension d'un nogood est elle-même un nogood:

Proposition 1 :

Autres définitions pour la suppression de symétries de valeurs et de variables:

On montre dans ce qui suit qu'il existe des algorithmes de suppression de symétrie structurelles efficaces pour les CSPs ou les deux ensembles de valeurs et de variables peuvent être partitionnées en des sous-ensembles, tel que, au sein de chaque sous ensemble, toutes les variables (respectivement les valeurs) sont interchangeables. Ces problèmes sont appelées les CSPs avec interchangeabilité de morceaux.

Définition 7 : (Bijection par morceau) Soit S = U Pi tel que les ensembles Pi sont disjoint, i.e., Pi ∩ Pj ≠ 0 implique i = j. Alors, on écrit S= ∑i Pi et on appelle ∑i Pi une partition de S. Une bijection b : S → S est une bijection par morceau sur ∑i Pi si et seulement si b (Pi) = Pi, ou b (Pi) = {b(e) | e ϵ Pi}.

Définition 8 : (Interchangeabilité par morceau/ Interchangeabilité complète du CSP). Un CSP P = ($\sum_{k=1}^{m} V_k$, $\sum_{\ell=1}^{n} D_l$, C) est un CSP avec interchangeabilité par morceau si et seulement si, pour chaque solution σ ϵ Sol (P), chaque bijection par morceau a sur $\sum_k V_k$, et chaque bijection par morceau b sur $\sum_\ell D_\ell$, on a b ○ σ ○ a ϵ Sol (P).Si la seule bijection sur $\sum_k V_k$ (ou $\sum_\ell D_\ell$) est l'identité, alors, on dit que le CSP a une interchangeabilité de valeurs (ou interchangeabilité de variables) par morceau. Si m = 1 (ou n = 1), alors le CSP est entièrement avec valeurs interchangeables (respectivement variables interchangeables). Si m = n = 1, alors le CSP est entièrement interchangeable.

Dans ce qui suit on va voir comment toutes les symmétries sont supprimés dans les CSPs avec interchangeabilité par morceau par le moyen de SBDD [9, 15]. Rappelons que SBDD est une technique pour supprimer toutes les symétries pendant la recherche. L'idée est comme suit : A chaque point de choix au cours de la recherche, on vérifie si le sous-arbre routé au nœud courant mappe, sous l'application de symétrie, en un autre sous-arbre qui a été entièrement exploré auparavant. Si oui, alors le nœud actuel n'a pas besoin d'etre exploré plus et peut être supprimé.

Différentes façons de contrôler et de limiter le nombre de sous-arbres préalablement explorés qui doit être revérifiée ont été développées dans [165, 166]. Avec ces résultats, la procédure de base de tout code SBDD qui détermine son efficacité est l'algorithme de détection de dominante qui vérifie si une (ensemble) affectation partielle donnée est dominée par une autre. Formellement, la dominance d'une affectation est définie par :

Définition 9 : (Dominance d'une affectation). Soit $P = (\sum_k V_k, \sum_\ell D_\ell, C)$ un CSP avec interchangeabilité par morceau. Une affectation α domine une autre affectation β si et seulement si il existe des bijections par morceaux a sur $\sum_k V_k$ et b sur $\sum_\ell D_\ell$ tel que pour chaque v ϵ scope (α) on a β (a (v)) = b (α(v)).

Etant donné deux assignements α et β pour un CSP avec interchangeabilité par morceaux, on appelle le problème de détermination de la dominance de β par α le problème de détection de dominance. Par conséquent, si on peut résoudre le problème de détection de dominance de manière efficace, alors on peut aussi supprimer les symétries efficacement.

L'idée clé pour traiter le problème de la détection de dominance pour les CSP s avec interchangeabilité par morceaux consiste en l'introduction d'abstractions structurelles: En effet, de manière générale, pour modéliser un CSP, on a besoin de nommer de façon unique chaque valeur et chaque variable. Lorsque certaines variables et certaines valeurs sont effectivement interchangeables, une telle désignation n'est évidemment pas naturelle. Pour remédier à cela il faut voire chaque variable et chaque valeur en tant que membre d'une classe de symétrie. Au début, ces classes correspondent directement aux ensembles V_k et D_ℓ. Afin de vérifier quels objets CSP sont toujours interchangeables, on a besoin d'introduire des sous-classes des classes de symétrie d'origine. On va voir qu'on peut détecter les symétries en désignant chacun de ces sous-classes avec une signature appropriée qui est définie par l'ensemble des symétries initiales et les affectations faites. On verra également que l'abstraction à partir du modèle CSP à la structure actuelle du problème est faite grâce à ces signatures.

3.4.4.2 Les signatures

3. La symétrie dans la programmation par contraintes

Considérons l'exemple suivant :

Exemple : Soit les variables V = {v_1, …, v_8} et les domaines de valeurs D = {d_1, …, d_8}. Supposons que les quatre premières et les quatre dernières variables sont interchangeables : V_1 = {v_1, …, v_4} et V_2 = {v_5, …, v_8}. Supposons que les trois premières et les trois dernières valeurs sont interchangeables : D_1 = {d_1, …, d_3} et D_2 = {d_4, …d_6}. Considérons les deux affectations partielles suivantes : (Voire figure . 1a) : α_1 = ($v_1 = d_1$ & $v_2 = d_1$ & $v_3 = d_2$ & $v_6 = d_5$ & $v_7 = d_1$ & $v_8 = d_2$) et α_2 = ($v_1 = d_6$ & $v_2 = d_1$ & $v_3 = d_2$ & $v_4 = d_2$ & $v_5 = d_1$ & $v_6 = d_6$ & $v_7 = d_2$ & $v_8 = d_2$). En examinons α1 on trouve que :

1. Il existe une valeur (nommée d_1) dans D_1 qui est prise par deux variables dans V_1 et une variable dans V_2.
2. Il existe une valeur (nommée d_2) dans D_1 qui est prise par une variable dans V_1 et une variable dans V_2.
3. Il existe une valeur (nommée d_5) dans D_2 qui est prise par une variable dans V_2.

Par ailleurs, en examinant α_2 on trouve :

a. Il existe une valeur (nommée d_2) dans D_1 qui est prise par deux variables dans V_1 et deux variables dans V_2.
b. Il existe une valeur (nommée d_1) dans D_1 qui est prise par une variable dans V_1 et une variable dans V_2.
c. Il existe une valeur (nommée d_6) dans D_2 qui est prise par une variable dans V_1 et une variable dans V_2.

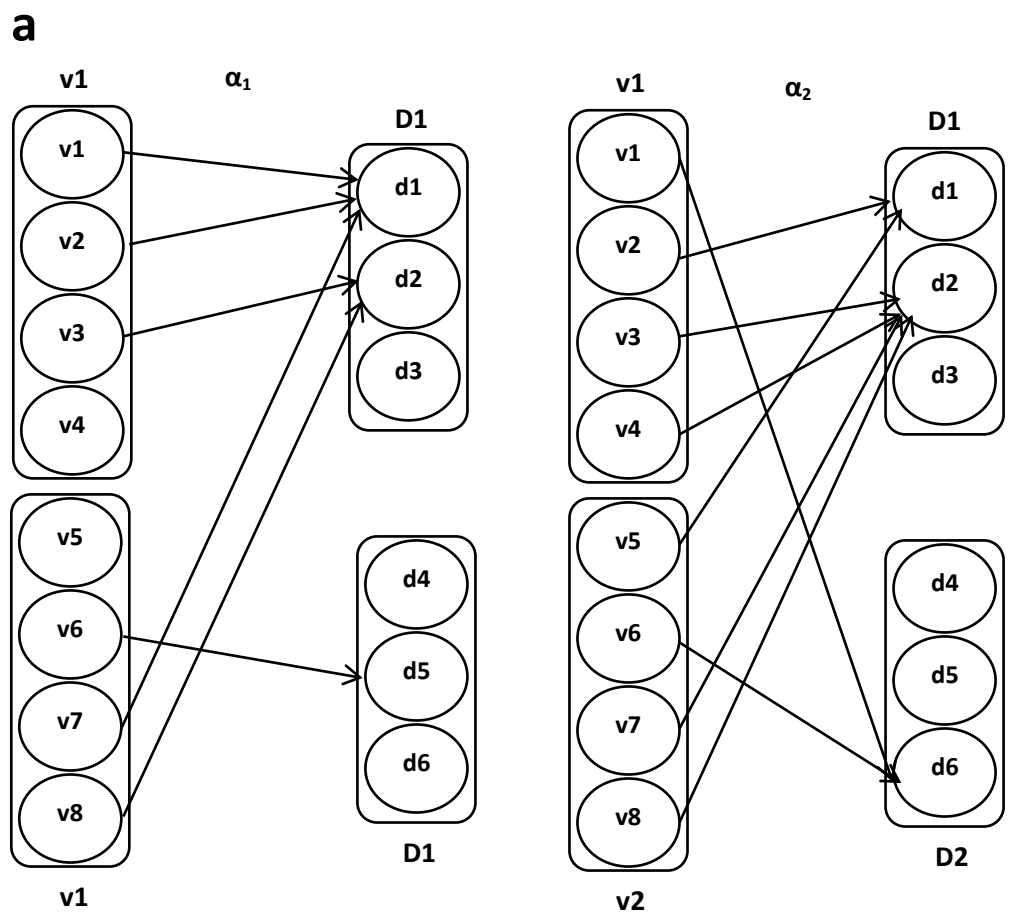


Figure 3-13 Les signatures.

1-a: ($d_1 \rightarrow d_2$, $\{v_1, v_2\} \rightarrow \{v_3, v_4\}$, $\{v_7\} \rightarrow \{v_7, v_8\}$),

2-b: ($d_2 \rightarrow d_1$, $\{v_3\} \rightarrow \{v_2\}$, $\{v_8\} \rightarrow \{v_5\}$),

3-c: ($d_5 \rightarrow d_6$, $\{v_6\} \rightarrow \{v_6\}$),

On constate que α_2 est structurellement une affectation partielle étendue de α_1, ou, en d'autres termes, que α_1 domine α_2 (Voir Fig. 4.13).

Ce qui a été fait dans cet exemple c'est l'abstraction a partir d'un modèle donné et des noms de variables et de valeurs (arbitraires) à la structure actuelle du problème. Alors, au lieu de parler de variables et de valeurs spécifiques, on considère des membres de classes. Spécifiquement, pour chaque affectation partielle, on affecte implicitement chaque valeur une signature qui capture le nombre de membres pris de chaque classe de symétrie de variable. Par exemple, dans α_1, la valeur d_1 a la signature ($2 * V_1$, $1 * V_2$) donc la signature de d_1 sous α_1 est $sig_{\alpha 1}(d_1) = (2, 1)$. Sous α_2, en d'autres termes, la signature de d_2 est $sig_{\alpha 2}(d_2) = (2, 2)$. Par conséquent, on peut voir d_2 dans α_2 plus spécial que d_1 dans α_1, ou bien que, d_1 dans α_1 domine d_2 dans α_2. Dans cette terminologie, d_1 dans α_2 a la signature $sig_{\alpha 2}(d_1) = (1, 1)$ et par conséquent il domine d_1 dans α_1. Noter que $sig_{\alpha 2}(d_6)$ est aussi égale a (1, 1), cependant d_6 dans α_2 ne domine pas d_1 dans α_1 puisque $d_6 \in D_2$ et $d_1 \in D_1$. En général :

Définition 10 (Domination d'une valeur) Une valeur d dans une affectation partielle α domine une valeur e dans une affectation partielle β si et seulement si d et e appartiennent a la même classe de symétrie de valeurs et $sig_\alpha(d) \leq sig_\beta(e)$.[3]

On dit aussi qu'une valeur d dans une affectation partielle α est équivalente structurellement a une valeur e dans une affectation partielle β si et seulement si d et e appartiennent a la même classe de symétrie de valeurs et $sig_\alpha(d) = sig_\beta(e)$.

La section suivante montre comment ces notions de dominance et d'équivalence structurelle peuvent être exploitées pour obtenir un algorithme à temps polynomial qui résout le problème de détection de dominance dans les CSPs avec interchangeabilité par morceaux.

[3] La relation ≤ sur les vecteurs est définie comme un opérateur de comparaison qui suit l'ordre de dominance « dominance ordering », qui est différent de l'ordre lexicographique « lexicographic ordering ».

3.4.4.3 La détection de dominance en utilisant les signatures

Le lemme suivant montre comment les abstractions de signatures peuvent aider à détecter les relations de dominance entre les affectations partielles.

Lemme 1 une affectation partielle α domine une autre affectation partielle β dans un CSP avec interchangeabilité par morceau si et seulement si il existe des bijections par morceaux b sur $D = \sum_{\ell} D_{\ell}$ tel que quel que soit d ϵ D, d dans α domine b (d) dans β.

Par conséquent, on a α domine β si et seulement si il existe un matching parfait dans un graphe bipartie ou les arcs sont définis par les relations de signature sur les valeurs (Voir Fig. 1b). Soit des D' l'ensemble des dupliations de valeurs dans D obtenues par ($D' := \{d' \mid d \epsilon D\}$).

Définition 11 (Le Graphe de détection de dominance) Soient deux affectations partielles α et β, le graphe de detection de dominance DDG (α, β) est (D U D′, E), ou E := {(d, e′)| d dans α domine e dans β}représente l'ensemble des arcs.

Théorème 1 Le problème de détection de dominance entre deux affectations partielles α et β pour un CSP avec interchangeabilité par morceaux a une complexité de $O(M + m^2 + mn)$, ou $M = O(m^{2.5})$ est le temps nécessaire pour déterminer si il existe un matching parfait dans DDG (α, β), avec m est le nombre de valeurs et n est le nombre de variables. Par conséquent tous les sous arbres symétriques causés par les symétries de valeurs et les symétries de variables d'un CSP avec interchangeabilité par morceau peuvent être éliminés avec « overhead » temps polynomial à chaque nœud exploré.

Cela est le premier résultat montrant que toutes les symétries d'un CSP avec interchangeabilité par morceaux peuvent être supprimés en un temps polynomial, dans le sens qu'il y'a pas de sous arbres symétriques dans l'arbre de recherche restant.

On peut remarquer aussi que chaque graphe biparti peut être aussi vu comme un graph de détection de dominance d'un CSP et des affectations (α et β) qui peuvent être déterminées en temps linéaire a la taille du graphe donné. Par conséquent, un matching biparti parfait peut exister si et seulement si α domine β, qui rend le problème de détection de dominance au moins assez difficile que la matching bipartie.

Donc le temps pris par la détection de dominance est égal a T, ou T ϵ Ω (M) ∩ O (M + m^2 + mn).

- ***Filtrage à base de symétrie***

D'après le théorème 1, tous les sous arbres symétriques causés par les symétries d'un CSP avec interchangeabilité par morceau peuvent être éliminés en temps polynomial à chaque nœud exploré en utilisant la suppression de symétries par l'approche de détection de dominance (SBDD) [133, 134]. Ce qui implique ici la vérification a chaque point de choix s'il est dominé par un autre déjà exploré. Pour cela un algorithme a été développé évitant cet inconvénient.

En effet, cet algorithme est basé sur l'idée de l'utilisation de la détection de dominance pour le filtrage beaucoup plus que pour la suppression.

Avec SBDD, il existe une distinction naturelle entre deux types de filtrages qui sont appliquées : la première consiste à vérifier qu'aucun des nœuds nouveau créés ne sont symétriques a un nœud qui a été exploré complètement avant le nœud courant sur lequel on est branché. On appliquant les contraintes de branchement unaires, ceci peut être atteint par 'division' les domaines de variables. L'autre type de filtrage consiste à la création d'enfants qui ne sont pas symétriques entre eux. Les deux types doivent être adressés pour atteindre un arbre de recherche (libre des symétries) (correspondant à GE-trees). Les deux types de filtrages peuvent être distingués par : filtrage à base de 'Symmetric-ancestor' et filtrage à base de 'Symmetric-sibling'.

- ***Filtrage à base de ' Symmetric-ancestor '***

L'objectif de ce type de filtrage est de réduire les domaines tel que l'instanciation d'une variable avec une de ses valeurs de domaine ne va pas induire a la création de nœud de recherche qui est symétrique a un qui a été exploré précédemment.

Définition 12 (Ancestor-Symmetry Resistant) Soit un arbre DFS T, on dit qu'un point de choix β est 'Ancestor-Symmetry Resistant' si et seulement si pour tous les nœuds α explorés précédemment complètement α ϵ T (α est appelé l'ancestor de β) et pour toutes les variables v et les valeurs d ϵ dom (v) on a α ne domine pas β & (v = d).

Supposant que actuellement on est dans le point de choix β et que α est un nœud ancestor qui ne domine pas β. Alors, instancier encore une variable v ϵ V_k pour certain k par mettre v → e ϵ D_ℓ pour certain ℓ va changer seulement la signature de e de sig_β (e) à sig_β (e) + e_k. On met β' := β & (v = e). Donc, G_1 := DDG (α, β) et G_2 := DDG (α, β'). Le deuxième graphe biparti doit contenir des arcs additionnels qui doivent tous aller à e' dans la partition a droite. Si G_2 contient un m-matching, ce matching doit contenir exactement une de ces arcs additionnels. Par conséquent si α domine β' alors G_1 doit contenir un (m-1)-matching. Dans ce cas il faut trouver une solution pour rendre β ancestor smmetry resistant avec respect à α.

Exemple 4 Le filtrage à base de Symmetric-ancestor est illustré dans Fig. 2.on considère un problème ou le partitionnement de variable a trois parties et toutes les trois valeurs sont symétriques. Les lignes en gras indiquent le graph de matching qui représente un matching presque parfait. Les lignes en discontinue indiquent les arcs critiques qui leurs addition au graphe rend le matching parfait. L'algorithme énumère les arcs critiques afin de trouver les affectations critiques qui doivent mener à une vérification de dominance réussie. Dans cet exemple, chaque affectation de la valeur v3 à n'importe quelle variable non affectée dans la partie variable 1 ou 3 doit mener à une vérification de dominance réussie. Par conséquent, la valeur v3 doit être supprimée des domaines de toutes ces variables.

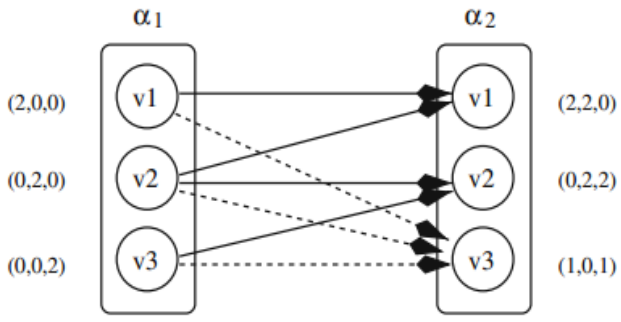


Figure 3-14 Le filtrage à base d'ancestor : à gauche et à droite du graphe biparti on trouve les signatures des valeurs symétriques v1, v2 et v3 sous deux affectations α1 et α2 pour un problème avec partitionnement de variables égal à trois parties. Les lignes solides indiquent les relations de dominance entre valeur. Les lignes en discontinue indiquent les arcs critiques.

Pour résumer, à partir du théorème 1, le temps d'exécution nécessaire pour l'algorithme de matching de valeurs initial est limité à O ($m^{2.5}$ + mn). Par conséquent, l'algorithme de filtrage complet s'exécute en un temps égal à O ($m^{2.5}$ + mn). Et donc,

puisque dans SBDD au plus n (m – 1) nœuds ancestor nécessite d'être considérés, on peut prouver le théorème suivant :

Théorème 2 Pour un CSP avec interchangeabilité par morceau, on peut atteindre la resistance ancestor-symmetrie pour un nœud de recherche donné dans un temps O ($nm^{3.5} + n^2m^2$).

- ***Filtrage à base de ' Symmetric-sibling '***

Pour atteindre une prévention de symétrie complète, on a aussi besoin de garantir que les nouvelles siblings crées ne sont pas symétriques ' each other '. Par conséquent, après le choix de la prochaine variable a affecté, mais avant de brancher sur cette variable, une étape de filtrage en plus est nécessaire (c'est plus d'une étape de suppression implicite), lorsque on choisit une valeur représentative à sortir de chaque classe d'équivalence tel que, lorsqu'elle est affectée à une variable donnée, cela peut mener à la création de points de choix symétriques. En effet, lorsqu'un sibling domine un autre, les deux doivent êtres équivalents structurellement (voire definition10). On peut éviter à produire des sibling symétriques par choisir exactement une valeur représentative à partir de ceux qui sont équivalents structurellement. La complexité de cet algorithme de filtrage est dominée par le filtrage à base de symmetric-ancestor.

L'utilisation des concepts de filtrage à base de sibling ou à base de symétrique ancestor a compléter le développement d'un algorithme de suppression de symétries efficace pour les CSPs avec interchangeabilité par morceau qui s'exécute en un temps polynomial.

3.4.4.4 Algorithmes rapides pour la suppression des symétries de valeurs

On présente ici le cas particulier des CSP interchangeables sans symétrie de variables, appélée CSP avec interchangeabilité de valeur par morceaux.

- **CSP avec interchangeabilité de valeurs complète. []**

Lorsque toutes les valeurs sont interchangeables et pas de symétrie de variable est présente, on parle d'un CSP pleinement de valeur interchangeables.

Definition 13 : (CSP avec interchangeabilité de valeurs complète.) Un CSP P =(V, D, C) est un CSP avec interchangeabilité de valeurs complète si pour chaque solution σ ϵ sol(P) et chaque bijection b a travers D, on a b o σ ϵ sol(P).

Théorème 3 : Soit α un nogood pour un CSP P avec interchangeabilité de valeurs complète P= =(V, D, C) et soit b : D →D une bijection. Alors b o α est un nogood pour P.

Definition 14 : (fermeture d'un nogood). Soit α un nogood pour un CSP P avec interchangeabilité de valeurs complète P= (V, D, C). la fermeture α pour P représentée par closure (α, P) est l'ensemble {b o α| b est une bijection sur D}.

Abstract nogoods

Definition15 (Abstract Nogood). Soit α un nogood pour un CSP P avec interchangeabilité de valeurs complète P= (V, D, C). soit image (α) = {d1, …, dk} et soit $v_{ri} \in \alpha^{-1}(d_i)$, pour 1<i<k. l'abstract nogood de α Anogood(α, P) est l'ensemble de toutes les fonctions ϒ : scope (α) → D satisfait la condition :

$$\forall i \in 1 \dots k : allequal\ (\gamma(vj) | vj \in \alpha^{-1}\ (di)) \ \&\ alldiff\ (\gamma(vr1), \dots, \gamma(vrk))$$

Ou allequal(a1, …, an) est satisfaite si toutes les a_i représentent la même valeur, et alldiff(a1, …an) si toutes les valeurs sont différentes.

3.4.4.5 Discussion

Dans [133] Flener et al, ont étudier théoriquement plusieurs classes de CSP pour lequel la suppression des symétries est réalisable, dans le sens ou tous les sous arbres symétriques causés par les symétries des CSPs dans ces classes peuvent être supprimés dans un temps polynomial c'est-à-dire avec un overhead constant à chaque nœud exploré ces classes de CSP qui engendrent plusieurs problèmes pratiques contient plusieurs formes d'interactions de valeurs ou de variables et permet à la suppression des symétries d'être effectuée avec un overhead polynomial (qui est souvent constant) par rapport au temps d'exécution et à l'espace à chaque nœud exploré, et cela en utilisant des procédure de recherches dédiés. Malheureusement la suppression de symétries par ces schémas de détection de dominance a parfois des

limites, vu qu'il existe certaines classes qui ont été identifiés et pour lequel la détection de dominance n'est pas traitable.

Table 1 Traitabilité de la suppression de symétries et la détection de dominance

		Interchangeabilité de variables			
	None	Complète	Partielle	Wreath	
Interchangeabilité de valeurs					
None		P (Thm 1)	P (Thm 1)	P (Thm 10)	scalar CSP
		P (Thm 1)	P (Thm 1)	P (Thm 10)	set- CSP
Complète	P (Thm 4)	P (Thm 1)	P (Thm 1)	NP (Thm 11)	scalar CSP
	P (Thm 6)	NP (Cor. 1)	NP (Cor. 1)	NP (Thm 11)	set- CSP
Partielle	P (Thm 5)	P (Thm 1)	P (Thm 1)	NP (Thm 11)	scalar CSP
	P (Thm 9)	NP (Cor. 1)	NP (Cor. 1)	NP (Thm 11)	set- CSP
Wreath	P (Thm 8)	P [13, Thm4]	P [13, Thm4]	NP (Thm 11)	scalar CSP
	P (Thm 9)	NP [13, Cor. 1]	NP [13, Cor. 1]	NP (Thm 11)	set- CSP

Table 2 Traitabilité de la suppression de symétries et la détection de dominance [133].

Le tableau 2. Résume les résultats obtenus par [133].

Dans [134] on trouve une contrepartie statique structurelle de la méthode dynamique structurelle pour la suppression des CSPs avec interchangeabilité de morceaux, le concept de signature a été exploité pour concevoir l'ensemble de contraintes de suppression de symétrie qui supprime toutes les symétries considérés.

Il y'a beaucoup de directions pour la recherche future. Un intérêt particulier est l'étude des classes de CSPs traitables présentant des symétries de variables où l'ensemble de variables a une structure plus complexe que les structures étudiés dans [133]. En particulier, lorsque l'ensemble variable est obtenue par un produit cartésien sur certains ensembles d'indices (modèle de matrice). Il existe de nombreuses formes intéressantes d'interchangeabilité par morceaux / complète, dans les modèles de matrice (lignes, colonnes, ...). Pour beaucoup de ces formes d'interchangeabilité de variables, y compris leurs compositions avec différentes formes de interchangeabilité de valeur, les résultats de la traçabilité pour supprimer les symétries sont toujours portés disparus donc trouver des procédures de recherche efficaces est un vrai

challenge. En outre, les mauvais résultats de la faisabilité appellent pour l'identification de cas particuliers où la suppression de symétrie est traitable.

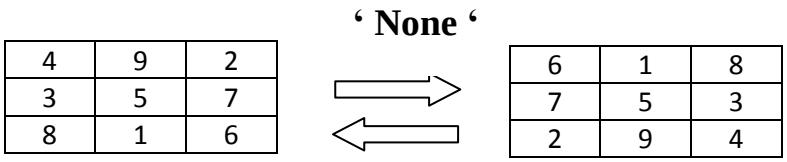
3.5 Techniques de suppression de symétries internes aux solutions dans les CSPs

3.5.1 Principe

Supprimer les symétries de l'espace de recherche est souvent critique pour la résolution de grandes instances des problèmes, cela à induit a la définition d'un nouveau concept de symétries plus faciles à exploiter, il s'agit des symétries internes aux solutions [**135**] (la propriété de symétrie est relative a la solution).

En effet, les symétries peuvent être trouves dans les solutions individuels du problème de satisfaction de contraintes, On dit qu'une solution A contient une symétrie interne σ (ou par équivalence σ est une symétrie interne à cette solution) si et seulement si $\sigma(A) = A$.

Exemple : Considérant le problème '' ce problème contient une symétrie interne, pour montrer sa, considérons la symétrie de solution σ_{inv} qui inverse les labels, en mappant k en n^2+1-k. cette symétrie de solution mappe '' en une solution différente mais symétrique. Cependant, si on applique maintenant la symétrie σ_{180} qui fait la rotation du carré avec 180°, on revient a la solution originale :



Si on considere la symétrie $\sigma_{inv} \circ \sigma_{180}$ qui est la composition des deux symétries : σ_{inv} et σ_{180}, comme cette symétrie mappe le probleme '' en lui-même, alors la solution '' contient la symétrie interne $\sigma_{inv} \circ \sigma_{180}$.

Une différence significative entre la symétrie interne et la symétrie de solution est que la symétrie de solution est une propriété relative à chaque solution, tandis que la symétrie interne est une propriété de juste la solution donnée.

3. La symétrie dans la programmation par contraintes

Exemple : considérons le problème ' M*agic Square'* suivant :

1	4	13	16
14	15	2	3
8	5	12	9
11	10	7	6

(1)

La symétrie $\sigma_{inv} \circ \sigma 180$ ne représente pas une symétrie interne à cette solution :

1	4	13	16
14	15	2	3
8	5	12	9
11	10	7	6

$\sigma_{inv} \circ \sigma_{180}$ ⟺

11	10	7	6
8	5	12	9
14	15	2	3
1	4	13	16

(2)

Par contre, la symétrie $\sigma_{inv} \circ \sigma_{180}$ est une symétrie interne qu'on peut trouver dans la solution suivante :

1	8	12	13
14	11	7	2
15	10	6	3
4	5	9	16

A partir de ça on peut conclure que $\sigma_{inv} \circ \sigma_{180}$ est une symétrie interne contenue dans seulement quelques solutions du probleme '*Magic Square*'. En fait, 48 sorties des 880 'carrés magiques' distinctes d'ordre 4 contiennent cette symétrie interne. D'autre part, $\sigma_{inv} \circ \sigma_{180}$ *représente* une symétrie de solution des problèmes 'carrés magiques' de toute taille.

Une solution qui contienne une symétrie interne peut être souvent décrite par un sous ensemble d'affectations et un ou plusieurs symétries actant sur ce sous-ensemble qui génère un ensemble complet d'affectations. Etant donné un ensemble de symétries $\sum$ on note $\sum^*$ pour la clôture de $\sum$. Tel que, $\sum^0 = \sum$, $\sum^i = \{\sigma 1 \circ \sigma_2 / \sigma_1 \in \sum, \sigma_2 \in \sum^{i-1}\}$, $\sum^* = U_i \sum^i$. étant donné une solution A, on dit qu'un sous ensemble B de A et l'ensemble des symétries $\sum$ génèrent la solution complète A ssi $A = B \cup \sum^*(B)$. Dans ce cas, on décrit également A comme une solution contenant les symétries internes $\sum$.

Exemple : considérons encore la solution 2 qui contient la symétrie interne $\sigma_{inv} \circ \sigma_{180}$ la moitié de ce carré magique et $\sigma_{inv} \circ \sigma_{180}$ génèrent la solution complète :

1	8	12	13
14	11	7	2
-	-	-	-
-	-	-	-

$\sigma_{inv} \circ \sigma_{180}$ ⟺

-	-	-	-
-	-	-	-
15	10	6	3
4	5	9	16

En fait, la solution (2) peut être générée a partir juste du premier quadrant et des deux symétries : $\sigma_{inv} \circ \sigma_{180}$ et une symétrie τ qui construit une rotation de 180° du premier quadrant au deuxième quadrant, décrémentant les carrées sur la diagonale principale et incrémenter ceux sur l'autre diagonale (la même symétrie construit le troisième quadrant a partir du quatrième). Plus précisément, τ fait les applications suivantes:

a	c	-	-
b	d	-	-
-	-	-	-
-	-	-	-

τ ⟹

-	-	d+1	c-1
-	-	b-1	a+1
-	-	-	-
-	-	-	-

Cet exemple fait allusion à la façon dont on peut exploiter les symétries internes au sein de la solution. La recherche est donc limitée à un sous-ensemble de variables de décision qui génère l'ensemble complet d'affectations et construit le reste de la solution en utilisant les symétries générées.

Dans [135], trois points important ont été discutés, dans un premier lieu un ensemble de propriétés théoriques aux symétries internes et un ensemble de propositions ont été identifiées, l'utilisation de ces propriétés aide a la recherche de solutions, le deuxième point discute la relation entre les symétries de solutions et les symétries internes et enfin une étude a été présentée discutant la compatibilité entre les techniques de suppression de symétries de solutions et l'exploitation des symétries internes et la possibilité d'exploiter les deux types de symétries à la fois.

3.5.2 Les propriétés théoriques des symétries internes

Le groupe théorique des symétries internes : comme les symétries de solutions, les symétries internes aux solutions forment elles aussi un groupe. Une solution A contient l'ensemble des symétries internes $\sum$ (ou de façon équivalente, $\sum$ représente l'ensemble des symétries internes de la solution) ssi A contient une symétrie σ pour chaque $\sigma \in \sum$.

Proposition : dans une solution A, l'ensemble des symétries internes Σ forme un groupe sous l'opérateur de composition.

Preuve : Il est évident que la symétrie identité est une symétrie interne. Il est évident aussi que les symétries internes sont fermées sous la composition. Finalement, quel que soit la symétrie $\sigma \in \sum$. Comme $\sigma(A)=A$, $\sigma^{-1}(\sigma(A))= \sigma^{-1}(A)$. Par conséquent, $A= \sigma^{-1}(A)$. Ainsi, l'inverse de σ est une symétrie interne.

3.5.3 La relation entre les symétries de solutions et les symétries internes

En général, il n'existe aucune relation entre les symétries de solutions d'un problème et les symétries internes au sein d'une solution de ce problème. Il existe des symétries de solutions d'un problème qui ne sont pas des symétries internes au sein de toute solution de ce problème, et vice versa.... Par exemple, le problème $Z_1 \# Z_2$ a une symétrie de solution qui swappe Z_1 et Z_2, mais pour $Z_1 \# Z_2$ il n'existe pas de solution contenant cette symétrie interne. D'autre par, la solution $Z_1=Z_2=0$ pour $Z_1<=Z_2$ contient la symétrie interne qui swappe Z_1 et Z_2, mais cette dernière ne représente pas une symétrie de solution pour $Z_1<=Z_2$ (comme $Z_1=0$, $Z_2=1$ est une solution et sa symétrie ne représente pas une solution). Cependant, Lorsque toutes les solutions d'un problème contiennent la même symétrie interne, cette dernière est forcément une symétrie de solution au problème lui-même.

Proposition : Si toutes les solutions du problème contiennent la symétrie interne alors cette dernière est une symétrie de solution.

Preuve : Etant donné une solution A, comme toutes les solutions du problème contiennent la symétrie interne σ alors $\sigma(A)=A$. Ainsi, σ permute la solution A vers elle-même et donc $\sigma(A)$ est aussi une solution au problème.

Par modus pollens, si σ n'est pas une symétrie de solution d'un problème donné, alors il existe au moins une solution qui ne contienne pas la symétrie interne σ.

Une autre proposition toujours défini dans [135] qui dit que la symétrie d'une solution contient le conjugué de toute symétrie interne contenue dans la solution originale.

Proposition 3 : si une solution A contient une symétrie σ *et* τ une autre symétrie alors $\tau(A)$ contient la symétrie interne $\tau \circ \sigma \circ \tau^{-1}$.

Preuve : Soit l'application de $\tau\circ\ \sigma\circ\tau^{-1}$ sur $\tau(A)$, $\tau_{(}\sigma_{(}\tau^{-1}$ $(\tau(A)))) = \tau_{(}\sigma$ $(A))$, et comme A contient la symétrie interne σ alors $\sigma(A)=A$. ainsi, $\tau(\sigma\ (A)) = \tau\ (A)$. Par conséquent, $\tau\circ\ \sigma\circ\tau^{-1}$ permute mappe la solution symétrique $\tau(A)$ vers elle-même.

Alors, une propriété importante qui en résulte, dans des cas spéciaux, ou les symétries commutent entre elles, la symétrie d'une solution contient les mêmes symétries internes comme le problème d'origine.

Proposition 4. Si une solution A contient la symétrie interne σ et τ est une symétrie qui commute avec σ alors la solution τ (A) contient aussi la symétrie interne σ.

Preuve : de la proposition 3, τ (A) contient la symétrie interne $\tau\circ\ \sigma\circ\tau^{-1}$ et $\tau\circ\sigma\circ\tau^{-1} = \tau\circ\tau^{-1}\circ\sigma = \sigma$.

Toutes ces propriétés qui définissent des liens entre les solutions symétriques et les deux types de symétries (symétries de solutions et symétries internes) aident a trouver un moyen afin d'exploiter les deux types de symétries a la fois. En effet, il n'ya pas de compatibilité entre les techniques de suppression de symétries de solutions et l'exploitation des solutions ayant des symétries internes particulières, et cela est dû à la suppression de symétries qui peut éliminer toutes les solutions contenant les symétries internes.

Prenons l'exemple précédent, soit la solution (3), cette solution contient la symétrie interne $\sigma_v \circ \sigma_{inv}$ qui inverse toutes les valeurs et applique la réflexion du carré par rapport à l'axe vertical :

1	4	13	16
14	15	2	3
8	5	12	9
11	10	7	6

σ_{inv} ⟹

σ_v ⟸

16	13	4	1
3	2	15	14
9	12	5	8
6	7	10	11

Supposons que la suppression de symétries élimine toutes les solutions de la meme classe symétrique que la solution (3) a l'exception de la solution symétrique qui fait la rotation du carré (solution (3)) de 90° dans le sens de l'horloge, cette solution ne contient pas la symétrie interne $\sigma_v \circ \sigma_{inv}$. *En fait, la rotation de la solution*

(3) contient la symétrie interne qui inverse toutes les valeurs et fait la réflexion du carré par rapport à l'axe horizontal.

11	8	14	1
10	5	15	4
7	12	2	13
6	9	3	16

$\sigma_v \circ \sigma_{inv}$ ⟺

16	3	9	6
13	2	12	7
4	15	5	10
1	14	8	11

Cependant y'a des cas particuliers où la rupture de symétrie ne change pas les symétries internes au sein des solutions. Si la rupture de symétrie supprime seulement les symétries qui commutent avec la symétrie interne contenu dans une solution particulière alors dans ce cas, comme la suppression de symétrie peut éliminer la solution donnée, elle doit laisser une solution symétrique contenant la symétrie interne.

Proposition 5 : Soit C un ensemble de contraintes, ∑ l'ensemble des symétries et S un ensemble non complet de contraintes de suppression de symétries et A une solution contenant la symétrie interne *σ,* si *σ* commute avec toute symétrie de ∑ alors il existe une solution d*e C U S* dans la même classe symétrique que A contenant aussi la symétrie inverse *σ.*

Preuve : comme S est un ensemble non complet, il existe une solution B de C U S et une symétrie τ Є ∑ / B= τ (A) et τ commute avec σ, par conséquent de la Proposition 4, B contient la symétrie interne σ.

Exemple : Soit la symétrie interne $\sigma_{inv} \circ \sigma_{180}$ contenues dans certains problèmes (mais pas tous) des carrés magiques. Cette symétrie particulière commute avec chaque symétrie de rotation, de réflexion et d'inversion de solution du problème. Par conséquent, s'il existe une solution avec la symétrie interne $\sigma_{inv} \circ \sigma_{180,}$ ceci reste vrai après la suppression des symétries de rotation, réflexion et inversion de solutions. Cependant, il ya des symétries internes contenus dans des certaines solutions (comme dans le dernier exemple : la réflexion sur l'axe vertical) qui ne commutent pas avec toutes les symétries du problème des carrés magiques.

3.5.4 L'exploitation des symétries internes

L'exploitation des symétries internes comporte deux étapes essentielles: trouver des symétries internes, puis limiter la recherche aux seules solutions qui contiennent ces symétries internes. La difficulté se situe dans la première étape, car l'idéal est de choisir les symétries internes qui ont un intérêt et qui sont susceptibles d'être contenues dans des cas non encore résolus de notre problème.

Une fois que les symétries internes qui semblent être contenues dans les solutions de d'autres instances du problème (comme les grandes instances aussi) sont identifiées, il s'agira de restreindre la recherche des solveurs de contraintes a des solutions de cette forme.

En général, si on veut trouver des solutions ayant la symétrie interne σ on poste des contraintes de symétrie de cette forme : $Z_i = j \Rightarrow \sigma(Z_i = j)$

L'idée consiste à limiter les décisions de branchement a un sous ensemble de variables de décisions qui génèrent un ensemble complets d'affectations, cela réduit significativement la taille de l'espace de recherche.

Exemple : Soit l'exemple des carrés magiques : les contraintes de suppression de symétries qui élimines la plupart des symétries de solutions de type rotation, réflexion et inversion des solutions sont les suivants :

$$X_{1,1} < \min(X_{1,n}, X_{n,1}, X_{n,n}), X_{1,n} < X_{n,1},$$
$$X_{1,1} < n^2 + 1 - \max(X_{1,1}, X_{1,n}, X_{n,1}, X_{n,n}) \quad (5)$$

D'autres contraintes de symétries sont ajoutées afin d'assurer qu'une simple symétrie interne existe dans une solution. Les carrées magiques d'ordre pair ou impair ont souvent des symétries internes différents alors différentes contraintes de symétries sont ajoutés selon le cas, pour les carrées magiques d'ordre n pair, la recherche se concentre sur les solutions ayant la symétrie interne $\sigma_v \circ \sigma_{inv}$. Pour n impaire, la recherche de solutions contenant la symétrie interne $\sigma_{inv} \circ \sigma_{180}$. Par conséquent, les contraintes de symétries suivantes sont utilisées pour $1 <= i, j <= n$:

Impair (n): $X_{n+1-j,\, n+1-i} = n^2 + 1 - X_{i,j}$

Paire (n): $X_{n+1-i,\, j} = n^2 + 1 - X_{i,j}$ (6)

Les résultats ont montré que les contraintes de suppression de symétries et les contraintes de symétries internes accélèrent la recherche et que l'exploitation des deux types de symétries à la fois donne des résultats meilleurs que dans le cas d'utilisation d'un seul type de symétrie. Les résultats ont montré aussi que le potentiel d'exploiter les symétries internes et l'identification d'un nouveau concept de symétrie étend l'état de l'art dans les deux cas (les symétries internes et les symétries de solutions), les deux types de symétries peuvent être exploitées ensembles à la fois.

3.6 Conclusion

Malgré la variété de tous ces approches, un certain nombre de questions importantes et difficiles sur la suppression de symétrie demeurent, y compris:

- *L'identification de d'autres cas traitables* : chercher l'existence d'autres types communs de symétrie qui peuvent se présenter en pratique et dont le traitement de leur suppression est polynomial.
- *Exploration d'autres ordres pour les contraintes de suppression statiques* : En effet, on peut considérer certains ordres qui ont donné de bons résultats comme le '' *Gray code ordering* '' ainsi '' *the multiset ordering* '' malgré que c'est un ordre partiel, ce dernier a donné de bon résultats. La question qu'on peut poser est : existe-t-il d'autres ordres à part l'ordre lexicographique avec lesquels on peut supprimer la symétrie efficacement ? un autre point consiste à étudier la possibilité d'utiliser différents ordres dans chaque classe symétrique.
- *Réduire le conflit de branchement* : un des grands problèmes des contraintes de suppression de symétries est qu'elles peuvent entrer en conflit avec la direction des heuristiques de branchement. Notre challenge est d'éviter ce conflit afin de bénéficier des contraintes de suppression de symétries et des heuristiques de branchement dynamiques. Pour cela, on doit chercher comment choisir des contraintes de suppression de symétries statiques qui alignent avec les heuristiques de branchement dynamiques.
- *Etudier et améliorer la combinaison des symétries* : On s'intéresse toujours à avoir une élimination de combinaisons de symétries qui soit efficace, pour cela nous devons poser trois questions :

- (**Soudness**) : Qu'elles sont les contraintes de suppression de symétries qui peuvent être combinés avec sureté.
- (**Complétude**) : Comment peut-on éliminer toutes les combinaisons des symétries.
- (**Traitabilité**) : Qu'elles sont les groupes symétriques qui sont traitable pour la suppression individuelle et qu'on peut combiner pour obtenir un groupe symétrique qui dont la suppression est aussi traitable.

- *La détection de dominance et l'élimination* : Dans les problèmes d'optimisation par contraintes, la notion de dominance joue un rôle très similaire à la symétrie. Une solution (partielle) domine une autre s'il est sûr que celle-là est au moins équivalente en qualité que l'autre. Le challenge ouvert consiste à chercher des méthodes pour identifier les solutions dominantes, par exemple, on peut chercher des méthodes génériques pour la suppression des solutions dominées comme la méthode générique Lex-Leader pour la suppression des solutions symétriques et chercher les cas traitables.

4 Gestion des tournois sportifs

4.1 Introduction

Le sport est devenu un grand business dans l'économie mondiale. Les tournois sont suivis par des millions de personnes à travers le monde. Les équipes font de gros investissements dans de nouveaux joueurs. Les droits de diffusion chiffrent à des centaines de millions de dollars dans certaines compétitions. Les pays et les villes luttent pour le droit d'organiser des manifestations à travers le monde comme les Jeux olympiques et la Coupe du Monde de Football. Les ligues de sport professionnel impliquent des millions de fans et d'importants investissements sur les joueurs, les droits de diffusion et la publicité, menant à des problèmes d'optimisation difficiles. De l'autre côté, les ligues amateurs impliquent moins d'investissements, mais aussi exigent une coordination et des efforts logistiques en raison du grand nombre de tournois et des concurrents.

Le principal problème dans le sport scheduling consiste à déterminer la date et le lieu dans lequel chaque match de tournoi sera joué.

Le domaine de la programmation sportive comprend un domaine de recherche de challenge avec une grande variété de problèmes et d'applications. La plupart des problèmes sont des problèmes d'optimisation combinatoire 'hard' offrant une plate-forme idéale pour développer et tester toutes sortes de méthodes de résolution. Exemples d'applications qu'on peut trouver dans le Schedule des tournois sportifs : le football, le baseball, le basketball, le cricket, et le hockey. Au cours de ces 30 dernières années, ces problèmes ont encouragé un grand nombre de recherches sur la construction de méthodes de résolution efficaces menant à un vaste répertoire d'approches, notamment la génération de colonnes, la décomposition de 'Benders', les méthodes de programmation par contraintes, les méthodes hybrides combinant programmation en nombres entiers et la programmation par contraintes, et des méthodes méta heuristiques comme la recherche tabou, les algorithmes génétiques et les méthodes 'simulated annealing'. Les algorithmes utilisés par les programmateurs dépendent de la complexité du schedule et des contraintes pour chaque problème. En effet, des contraintes additionnelles peuvent transformer un problème simple en un problème NP-Complet ce qui signifie que la taille de l'entrée du schedule joue un rôle

important pour déterminer la bonne approche algorithmique. Dans ce cas, lorsque le nombre de matches ou d'équipes ou de contraintes augmente des algorithmes sophistiqués sont nécessaires.

En plus, comme les ligues sportives peuvent considérer différentes contraintes et peuvent avoir différentes objectives, le domaine du sport scheduling fournit plusieurs applications pratiques. Les applications sont généralement caractérisées par une énorme quantité d'exigences contradictoires en provenance des équipes, des fans, etc, pour résoudre ces problèmes, des méthodes spécialisées capables d'intégrer les exigences spécifiques à l'application dans les algorithmes sont nécessaires. Dans la littérature, différentes approches sont présentées, cependant, plus les algorithmes s'améliorent, plus le nombre de contraintes augmente.

En effet, La définition d'un schedule «optimal» variait, tout comme l'ensemble des contraintes, mais le thème de la recherche d'un schedule optimal ou quasi-optimale qui répond aux contraintes demeuré constant. Certaines applications ont obligé les auteurs à reconstruire le schedule à chaque fois que de nouvelles contraintes sont ajoutées ou clarifiées [136]. En conséquence, l'optimisation de leur algorithme d'ordonnancement était essentielle afin d'être capable de produire de nouveaux schedules pour le commissaire de la ligue à revoir. Plusieurs aspects à prendre en considération dans la détermination du meilleur schedule pour un tournoi. Dans certaines situations, on cherche un schedule en minimisant la distance totale parcourue, comme dans le cas du problème du voyageur de tournoi TTP [137] et dans celui de sa variante miroir [138], qui est commun à de nombreux tournois en Amérique du Sud [139]. D'autres problèmes tentent de minimiser le nombre total de pauses, à savoir le nombre de paires de matchs consécutifs à domicile ou le nombre de matchs consécutifs à l'extérieur joué par la même équipe. La minimisation de la valeur des effets de 'carry-over' [140] est un autre critère d'équité qui conduit à une répartition uniforme de la séquence de matchs tout au long du schedule. En effet, La capacité d'obtenir des solutions de haute qualité pour les ligues sportives ne fournit pas seulement un schedule qui satisfait les besoins de l'équipe, mais peut aussi entraîner d'énormes bénéfices pour les ligues sportives. En plus, des niveaux d'optimalité différentes peuvent entraîner des différences de millions de dollars pour les clubs sportifs, les réseaux de télévision et les autres parties intéressées.

4. La Gestion des tournois sportifs

Quelques problèmes de sports scheduling ont un caractère multicritère. Ribeiro et Urrutia [141, 142] ont abordé le scheduling du tournoi de football annuel brésilienne, préalablement formulée comme un problème d'optimisation bicritère dont l'un des objectifs consistait à maximiser le nombre de matchs qui pourraient être diffusées par les chaînes de télévision ouvertes (pour augmenter les revenus provenant des droits de diffusion) et l'autre consistait à trouver un schedule équilibré avec un nombre minimum de pauses à domicile et de poses à l'extérieur (par souci d'équité). Une version multicritère d'un problème d'affectation arbitre découlant dans les ligues amateurs [143, 144] a été abordé par Duarte et Ribeiro [145].

Dans ce chapitre, nous allons présenter quelques définitions générales et terminologies necessaires pour comprendre le sport scheduling comme la notion du '*round robin tournament*' qu'on va considérer tout au long du chapitre ainsi que des modèles de présentations des tournois sportives, par la suite nous étudiant les grandes problèmes sur lequel se focalisent les recherches dans le domaine du sport scheduling et présenter le coté formel de quelques problèmes et les différentes approches et solutions proposées pour chaque classe de problème et nous terminons par présenter quelques applications et les travaux autour.

- ### Les Contraintes générales pour le sport scheduling des ligues :

De manière générale, les problèmes d'ordonnancement de sport se définirent comme suit: trouver un ordonnancement optimal pour un ensemble d'équipes qui jouent sur une période de temps qui satisfait à un ensemble de contraintes particulières. Dans certains scénarios, il n'y a pas de Schedule réalisable, pour d'autres, il peut y avoir beaucoup et dépendants des contraintes, et pour certains le Schedule peut être optimale. Un aspect frustrant de la programmation sportive est que les exigences des la ligues diffèrent et donc les techniques utilisées pour le schedule d'une ligue peuvent ne pas fonctionner pour une autre ligue. Ceci à amener à définir des problèmes de schedule académiques comme "le problème du voyageur de tournoi"(TTP), ou ça consiste à trouver un Schedule round robin optimal où chaque équipe joue contre chaque autre équipe une fois à domicile et une fois chez l'équipe adverse, tout en minimisant la distance parcourue par les équipes ainsi de satisfaire d'autres contraintes [146]. L'existence de tels problèmes permet la comparaison de

plusieurs techniques pour certains besoins, cependant, on ne peut pas affirmer ou dire que les techniques qui fonctionnent le mieux pour le TTP fonctionnera le mieux pour une vraie ligue de sport.

Le nombre de contraintes et leur type détermine souvent la difficulté du problème. Les contraintes peuvent être appliquées globalement à l'ensemble du Schedule ou localement à une partie de ce dernier ex : (seulement en Juin, week-end, etc.) on présente ici une liste des contraintes générales pour le sport scheduling des ligues sportives :

- ✓ *Les contraintes sur les lieux*: ou chaque match est soit "à domicile" soit "à l'extérieur". L'équité du lieu est souvent appliquée à l'ensemble du schedule et aussi pour des parties du schedule.
- ✓ *Les contraintes de pauses*: une pause est définie comme une séquence consécutive de matchs à domicile ou de matchs à l'extérieur (par opposition à «le cas ou pas de pause», une alternance de matches à domicile et matches à l'extérieur). Trouver un schedule 'min-break' avec un minimum de poses a été un sujet de recherche très actif.
- ✓ *Les contraintes de Temps et de Distance*: l'équité dans les temps des matchs (par exemple il faut éviter deux matchs dans 2 jours consécutives pour la même équipe ou le premier match se déroulera le premier jour le soir et le deuxième match se déroulera la matinée du deuxième jour). De plus, il faut tenir compte des différences des fuseaux horaires, et assurer l'équité à chaque équipe en ce qui concerne le temps et le coût de déplacement entre les sites.
- ✓ *Les contraintes de Variété ou de carry-over*: un bon calendrier doit avoir une «variété». À titre d'exemple, supposons que chaque équipe joue N fois contre chaque autre équipe: intuitivement, on souhaite diffuser les N fois au cours du calendrier tout en minimisant les "carry-over".
- ✓ *Les contraintes hiérarchiques*: un exemple pourrait être la division ou la structure de conférences d'une ligue, par exemple si une équipe doit jouer une certaine partie des matchs dans sa propre conférence, et une autre partie dans sa propre division, etc. Un autre exemple pourrait être l'application de contraintes de temps en termes de par mois et par semaine.
- ✓ *Les contraintes de matchs Désirées, fixes ou interdites*: souvent liées à certains moments de la saison.

Plus d'autres contraintes préférentielles spéciales.

4.2 Définitions et terminologies

Dans cette section, nous allons expliquer les terminologies du sport scheduling:

Un tournoi round robin est un tournoi où toutes les équipes rencontrent toutes les autres équipes un nombre fixe de fois. Chaque équipe fait face à l'autre équipe exactement une fois (resp. deux fois) en un tournoi single (resp. double) round robin (SRR) (resp. DRR) et joue au plus une fois à chaque tour. La plupart des ligues sportives jouent un tournoi double round robin où les équipes se rencontrent deux fois mais les tournois single rounds robin, triples et quadruples round robin peuvent se produire également.

Lors de la planification d'un tournoi, les matchs doivent être attribués à un certain nombre de créneaux horaires (slots) de telle sorte que chaque équipe joue au plus un match dans chaque créneau. Lorsque le nombre d'équipes n est pair alors il faut alouer au moins (n - 1) crénaux et si n est impaire au moins n créneaux sont nécessaires pour planifier un tournoi single round robin.

Nous considérons un tournoi joué par un nombre pair d'équipes n.

On dit qu'un tournoi round robin est compact si le nombre de tours est minimum et chaque équipe joue une seule fois à chaque tour. Chaque équipe a sa ville natale et sa propre salle et chaque match est joué sur le lieu de l'une des deux équipes qui s'affrontent.

L'équipe qui joue à son propre lieu est appelé l'équipe à domicile et on dit qu'elle joue un match à domicile, tandis que l'autre équipe est appelé l'équipe adverse et elle joue un match à l'extérieur.

On dit qu'il y'a une répétition lorsque pour la même paire d'équipes chaque équipe fait face à l'autre équipe deux fois en deux tours consécutifs. Si le nombre d'équipes est impair, alors dans chaque tour une équipe a un 'bye', c'est à dire qu'elle ne joue pas. Cette situation peut être réduite au cas d'un nombre pair d'équipes par l'ajout d'une équipe fictive. Ensuite, à chaque tour l'équipe jouant contre le équipe fictive aura un 'bye'.

Les tournois double round robin sont souvent divisés en deux phases, où chaque match doit avoir lieu exactement une fois dans chaque phase, mais avec des droits de 'jouer à domicile' différents.

Dans le cas schedules dites en miroir, les matchs joués par chaque équipe dans la deuxième phase suivent exactement le même ordre que ceux joués dans la première phase, mais avec des lieux échangés. Par conséquent, les deux matchs joués par chaque paire d'opposants ont lieu dans le même tour de la première et de la deuxième phase.

4.2.1 Le calendrier et le modèle Home/ Away

L'attribution des matchs aux slots (créneaux) peut être présentée comme par calendrier. Les lignes du calendrier correspondent aux équipes alors que les colonnes correspondent aux slots. L'intersection de la ligne i et de la colonne s correspond à l'adversaire de l'équipe i dans le slot s. La figure 3.1 montre un calendrier pour un tournoi round robin single et compact avec 6 équipes et un calendrier qui correspond à un tournoi avec 7 équipes.

Slots	1	2	3	4	5
Team 1	6	3	5	2	4
Team 2	5	6	4	1	3
Team 3	4	1	6	5	2
Team 4	3	5	2	6	1
Team 5	2	4	1	3	6
Team 6	1	2	3	4	5

Slots	1	2	3	4	5	6	7
Team 1	7		5	2	4	6	3
Team 2	5	6		1	3	7	4
Team 3	4	7	6		2	5	1
Team 4	3	5	7	6	1		2
Team 5	2	4	1	7	6	3	
Team 6		2	3	4	5	1	7
Team 7	1	3	4	5		2	6

Figure 4-1 Exemples de Calendriers pour les tournois avec 6 et 7 équipes

Dans la littérature, les équipes ont souvent un lieu associé et quand ils jouent dans leur propre lieu, on dit qu'ils jouent des matchs à domicile dans le cas contraire on dit qu'ils jouent des matchs a l'extérieur (dans les autres sites). Il est supposé que chaque fois que deux équipes se rencontrent l'une des équipes joue à domicile tandis que l'autre équipe joue à l'extérieur. Si une équipe ne joue pas dans un slot, on dit qu'il a un 'bye'. On définit la séquence de matchs à domicile, les matchs a l'exterieur et les 'bye' selon lesquelles une équipe joue pendant le tournoi par le modèle 'Home/ AwayPattern'.

Si les bye se produisent dans le tournoi, un modèle est normalement représenté par un vecteur avec une entrée pour chaque slot contenant soit un H, un A ou un B. Dans les tournois compacts avec un nombre pair d'équipes, toutes les équipes jouent dans chaque slot et B est omis. Dans ce cas, H et A sont souvent respectivement remplacés par 1 et 0.

Dans de nombreux tournois, il est considéré comme attractif d'avoir un modèle alternatif de matchs a domicile et de matchs a l'extérieur et on dit que le modèle a un break dans les slots différents d'une telle alternance de la séquence. Cela signifie que la rupture (break) correspond à deux consécutifs matchs à domicile ou deux consécutifs matches à l'extérieur. Ont dit que deux modèles peuvent être complémentaires si le premier modèle a un match à l'extérieur tandis que le deuxième modèle à un match à domicile et vice versa. La Figure 3.2 (a) montre deux modèles complémentaires pour un tournoi single round robin compact avec six équipes. Nous remarquons que les deux modèles ont une pause dans le slot 3.

1	0	0	1	0	
0	1	1	0	1	

Slots	1	2	3	4	5
Team 1	1	0	1	0	1
Team 2	1	0	0	1	0
Team 3	0	1	1	0	1
Team 4	1	0	1	0	0
Team 5	0	1	0	1	1
Team 6	0	1	0	1	0

Figure 4-2 (a) Deux modèles complémentaires, (b) Exemple d'n modèle défini pour un tournoi avec 6 équipes

Pour représenter les affectations des matchs a domicile et les matchs a l'extérieur pour un tournoi de n équipes nous utilisons le modèle Home/ Away (pattern set). Il s'agit d'un ensemble d'exactement n modèles et chaque modèle est associé à l'une des équipes. La Figure 3.2 (b) montre un exemple d'un ensemble de modèles établies pour un tournoi avec 6 équipes. Il faut noter que cet ensemble de modèles se compose exclusivement de paires de modèles complémentaires. Dans ce cas, l'ensemble des modèles satisfait la propriété de complémentarité et il est dit complémentaire. On dit que l'ensemble des modèles est équitable si toutes les équipes ont le même nombre de ruptures ou de pauses, et on dit qu'un ensemble de modèles

établi pour un tournoi single round robin est équilibré lorsque le nombre de matchs à domicile pour chaque équipe peuvent varier avec au plus un match.

En outre, l'ensemble de modèles peut être associé avec le calendrier de 6 équipes affichées dans la figure 3.1, puisque, à chaque match, l'un des adversaires joue à domicile, tandis que les autres jouent à l'extérieur. Un ensemble de modèles pour lesquels un calendrier correspondant existe est dit réalisable. La Figure 3.3 donne un exemple de trois modèles qui doivent donner un modèle établi impossible puisque les trois matchs communs ne peuvent être joués que dans les slots 1 et 2.

1	0	0	1	0
1	1	0	1	0
0	1	0	1	0

Figure 4-3 Exemple d'un sous-ensemble de modèle non réalisable

La combinaison de l'ensemble de modèles et du calendrier correspondant constitue le calendrier du tournoi. Un calendrier est mis en miroir lorsque la première et de la seconde moitié sont identiques avec permutation des matchs a domicile et des matchs a l'extérieur (c'est-à-dire les matchs joues a domiciles devient a l'extérieur et vice vers sa). Par ailleurs, nous disons qu'un calendrier est irréductible lorsque au plus un adversaire dans chaque match a une pause. Le calendrier peut être représenté comme dans la Figure 3.4 qui montre un à schedule double round robin miroir. Dans la figure, un (+) désigne un match à domicile tandis que le (-) désigne un match à l'extérieur. Une séquence de matchs consécutives à l'extérieur est appelée un 'trip' pendant qu'une séquence de matchs consécutifs à domicile est Appelé 'home stand'. une ligne entière du calendrier représente tout le tour pour l'équipe correspondante.

Slots	1	2	3	4	5	6	7	8	9	10
Team 1	+6	-3	+5	-2	+4	-6	+3	-5	+2	-4
Team 2	+5	-6	-4	+1	-3	-5	+6	+4	-1	+3
Team 3	-4	+1	+6	-5	+2	+4	-1	-6	+5	-2
Team 4	+3	-5	+2	-6	-1	-3	+5	-2	+6	+1
Team 5	-2	+4	-1	+3	+6	+2	-4	+1	-3	-6
Team 6	-1	+2	-3	+4	-5	+1	-2	+3	-4	+5

Figure 4-4 Exemple d'un tournoi miroir double round robin

4. La Gestion des tournois sportifs

Lors de la résolution d'un problème de gestion sportive, il peut être avantageux de reporter l'attribution des matchs jusqu'à l'obtention du calendrier. Dans ce cas les espaces réservés sont utilisés pour représenter les équipes dans l'ensemble des modèles et dans le calendrier jusqu'à ce que le calendrier soit trouvé.

4.2.2 Présentation graphique des tournois sportifs

Les tournois peuvent être représentés par des graphes qui offrent un bon modèle pour les formulations de planification et pour les algorithmes. Le graphe complet Kn peut être utilisé pour représenter un tournoi type single round robin ou l'une des phases d'un tournoi double round robin compact. Les nœuds du graphe représentent les équipes. Chaque match est représenté par une arrête, dont les extrémités sont associées aux deux équipes adversaires. La figure 1 montre un exemple illustrant la représentation graphique d'un tournoi type single round robin avec n = 4 équipes.

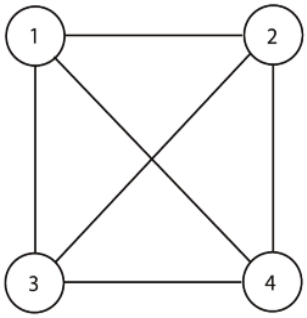
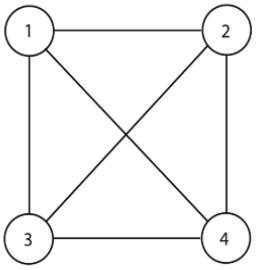
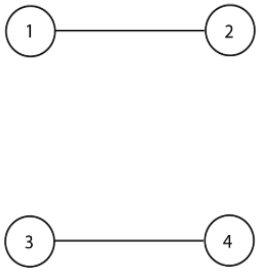


Figure 4-5 Exemple d'un tournoi type single round robin avec n = 4 équipes représenté par un graphe complet

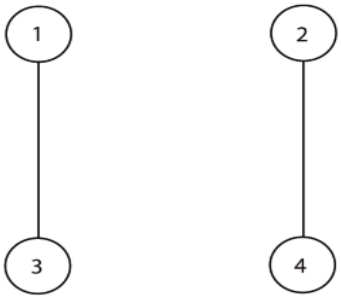
Une coloration des arêtes avec exactement n - 1 couleurs correspond à une 1-factorisation de Kn, Soit une partition de son jeu de bord en 1-facteurs F (constitués chacun de n / 2 bords non adjacents). Chaque 1-facteur correspond aux jeux programmés pour un donné ronde. Par conséquent, un 1-factorisation commandé détermine un calendrier pour l'tournoi, la définition de la tour dans laquelle chaque jeu est joué. Les figures 2 (a) à 2 (c), représenter un calendrier pour l'exemple présenté dans la Figure 1, dans laquelle chaque jeu est affecté à un tour.



(b) Premier round 1-Facteur F1.



(a) Deuxième round 1-Facteur F2.



(c) Troisième round
1-Facteur F3.

Figure 4-6 Calendrier d'un tournoi type single round robin avec n = 4 équipes représentant un de ses 1-factorisation: (a) 1-F1 facteur associé au premier tour, (b) 1-F2 facteur associé au second tour et (c) 1-facteur F3 associé au troisième tour

Un calendrier ne permet pas seulement de déterminer le tour ou chaque match sera joué, mais aussi son lieu. Si les matchs à domicile et les matchs à l'extérieur doivent être distingués, donc une orientation est affectée aux arêtes du graphe complet et à ses 1-facteurs. Dans le cas de l'exemple représenté sur les figures 1 et 2, nous supposons que l'équipe 2 joue tous ses matchs à l'extérieur, l'équipe 1 joue à l'extérieur avec l'équipe 3, l'équipe 4 joue à l'extérieur avec l'équipe 1 et joue à domicile avec l'équipe 3. Par conséquent, les quatre équipes ont les modèles de matchs joués « home/away » suivants:

- l'équipe 1 -home/away/home;
- l'équipe 2 - away / away / away;
- l'équipe 3 - away / home / home, et
- l'équipe 4 - home / home / away.

La figure 3 illustre cette situation, dans laquelle une orientation a été attribuée à chaque arrete de la figure 1 (ou, de façon équivalente, à chaque arrete dans les figures 2 (a) à 2 (c)) pour créer un calendrier complet défini par un graphe orienté: l'existence d'un arc du noeud i au noeud j signifie que l'équipe I joue à l'extérieur contre l'équipe j.

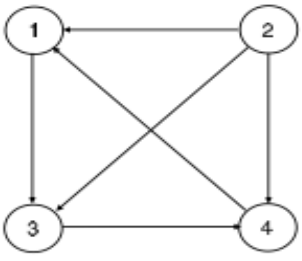


Figure 4-7 Exemple d'un tournoi type single round robin avec n = 4 équipes représenté par un graphe orienté dans lequel un arc du nœud i vers le nœud j qui signifie que l'équipe i joue contre l'équipe j dans le site de cette dernière.

L'ensemble HAP pour l'exemple précédent peut être représenté par une matrice comme dans le tableau 1, dans lequel la cellule correspondant à la ligne k et à la colonne j indique l'état de jeu de l'équipe j dans le round k.

	Teams			
Rounds	1	2	3	4
1	home	away	home	away
2	away	away	home	home
3	home	away	home	away

Table 3Exemple d'un ensemble HAP

Certains problèmes d'ordonnancement round robin impliquent la construction de fois le calendrier et le jeu de modèle chez-soi loin. Cependant, que ce soit le calendrier ou l'HAP ensemble peut être prédéfini et connu à l'avance dans certaines situations. Dans le premier cas, le calendrier est donné et le problème consiste à trouver un ensemble HAP possible optimiser une fonction objective. Dans le second cas, le jeu de modèle chez-soi loin est prédéterminé et un calendrier est demandé. Le problème de la construction d'un calendrier compatible avec un ensemble HAP donné et l'optimisation d'un certain objectif apparaît comme un sous-problème dans plusieurs approches pour résoudre les problèmes d'ordonnancement de la vie réelle, voir par exemple Nemhauser et Trick [136].

Pour tout programme donné ou chez-soi loin ensemble du motif, nous disons qu'il y'a une maison (resp. loin) pause dans k tour chaque fois qu'un équipes jouent deux matchs consécutifs à maison (resp. loin) dans les tours k - 1 et k.

Dans l'exemple représenté sur les figures 1 et 2, avec les orientations définies dans la figure 3, l'équipe 1 a un calendrier parfaitement alterné, sans interruption. Equipe 2 à deux pauses loin dans les deuxième et troisième rondes, tandis que les équipes 3 et 4 ont une maison briser chacune, respectivement dans les troisième et deuxième tours.

4.3 Les problèmes fondamentales

4.3.1 Minimisation de distance et le TTP

- ***Minimisation de Distance :***

Dans le cas des problèmes de schedule avec minimisation de distance, il ya une distance (ou un moment ou à un coût) associé à chaque paire d'équipes, ce qui correspond à la distance à parcourir entre leur ville d'origine. Il est donc demandé de trouver un schedule minimisant la distance totale parcourue par toutes les équipes. D'autres contraintes sont généralement imposées sur les voyages.

- ***Le 'Traveling Tournament Problem'(TTP) :***

Le problème du voyageur de tournoi (TTP) introduit dans l'article séminal de [137] est de loin le problème le plus emblématique dans ce domaine. Il s'agit d'un problème d'optimisation combinatoire difficile dans la programmation sportive qui fait abstraction des aspects les plus importants dans la création de calendriers chaque fois que les distances parcourues représentent une importante question.

Etant donné un nombre pair d'équipes n, les distances dij entre les sites de domicile des équipes i et j, pour chaque i,j=1,…,n (avec dij=0 si i=j), et deux nombres entiers L et U, le TTP appels à un schedule d'un tournoi double round robin minimisant la distance totale parcourue par les équipes et respectant un ensemble de contraintes, tout en assumant que chaque fois qu'une équipe joue deux matchs consécutifs à l'extérieur, elle passe directement du site du premier adversaire au site du second:

- chaque équipe commence le tournoi à son domicile et doit retourner à son site après son dernier match à l'extérieur;
- les répéteurs ne sont pas autorisés, c'est à dire deux équipes ne peuvent pas jouer les uns contre les autres en deux tours consécutifs;
- chaque séquence de matchs consécutifs joués à domicile par n'importe quelle équipe est formée par au moins L et au plus U matchs, et
- chaque séquence de matchs consécutifs joués à l'extérieur par n'importe quelle équipe est formée par au moins L et au plus U matchs.

4. La Gestion des tournois sportifs

La formulation du TTP par la programmation entière la plus directe fait usage aux variables de décision suivantes :

$$x_{ijk} = \begin{cases} 1, si\ l'équipe\ i\ joues\ a\ l'extérieur\ contre\ l'équipej\ dans\ le\ round\ k, \\ 0, sinon; \end{cases}$$

Et

$$y_{tijk} = \begin{cases} 1, si\ l'équipe\ t\ se\ déplace\ du\ site\ de\ l'équipe\ i\ a\ celui\ de\ l'équipej\ entre \\ les\ rounds\ k\ et\ k+1, \\ 0, sinon; \end{cases}$$

En utilisant ces variables, on obtient les formulations (1) à (12) pout le TTP :

(1) min

$\sum_{i=1}^{n}\sum_{j=1}^{n} dij\,.xij1 + \sum_{t=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{2n-3}$ dij . ytijk + $\sum_{i=1}^{n}\sum_{j=1}^{n} dji\,.xij, 2n-2$

(2) $x_{iik} = 0, \quad i = 1,\dots,n, \quad k = 1,\dots,2n-2,$

(3) $\sum_{j=1}^{n} (xijk + xjik) = 1, \quad i = 1,\dots,n, \ k = 1,\dots,2n-2,$

(4) $\sum_{k=1}^{2n-2} xijk = 1, \quad i,j = 1,\dots,n : \ i \neq j,$

(5) L $\leq \sum_{i=0}^{n}\sum_{i=0}^{n} xij, k+l \leq U,$ $\quad i = 1,\dots,n, \quad k = 1,\dots,2n-2-U,$

(6) $xijk + xjik + xij, k+1 + xji, k \leq 1,$ $\quad$ i, j = 1, … , n, $\quad$ k = 1, … , 2n − 3,

(7) $z_{iik} = \sum_{j=1}^{n} xjik,$ $\quad i = 1,\dots,n, \quad k = 1,\dots,2n-2,$

(8) $z_{jik} = x_{jik,} \quad i,j = 1,\dots,n: i \neq j, \quad k = 1,\dots,2n-2,$

(9) $y_{tijk} \geq z_{tik} + z_{tj,k+1} - 1,$ $\quad$ t, $i,j = 1,\dots,n, \quad k = 1,\dots,2n-3,$

(10) $x_{ijk} \in \{0,1\}, \quad i,j = 1,\dots,n, \quad k = 1,\dots,2n-2,$

(11) $z_{ijk} \in \{0,1\}, \quad i,j = 1,\dots,n, \quad k = 1,\dots,2n-2,$

(12) $y_{tijk} \in \{0,1\}, \quad t,i,j = 1,\dots,n, \quad k = 1,\dots,2n-2,$

- La fonction objective **(1)** représente la distance totale parcourue séparée dans trois termes: la distance parcourue par les équipes qui jouent à l'extérieur dans le premier tour, la distance parcourue après le premier et avant le dernier match joué par chaque équipe, et la distance parcourue pour rentrer à domicile par les équipes qui jouent à l'extérieur dans le dernier tour.
- La contrainte **(2)** stipule qu'aucune équipe ne peut jouer contre elle-même, tandis que la contrainte **(3)** impose que chaque équipe joue exactement une fois dans chaque tour, soit à domicile ou chez le site de l'équipe adverse.
- La contrainte **(4)** garantie que chaque équipe jouera contre chaque équipe adversaire à l'extérieur qu'une seule fois.
- La contrainte **(5)** est utilisée pour assurer que dans n'importe quelle séquence de matchs consécutifs *U + 1* une équipe jouera au moins *L* et au plus *U* matchs à l'extérieur.
- La contrainte **(6)** la non-occurrence de répéteurs.
- La contrainte **(7)** force z_{iik} quelle soit égale à 1 (respectivement. 0) si l'équipe i joue à domicile (respectivement. à l'extérieur) dans le tour k.
- Pour chaque deux équipes différentes i *et* j, la contrainte **(8)** vérifie que $z_{ijk} = x_{ijk}$, i.e. l'équipe i doit se trouver sur le site de l'équipe j si la première équipe joue contre la dernière à l'extérieur.
- La contrainte **(9)** impose sur l'équipe *T* de se rendre de la ville de l'équipe i à celle de l'équipe j si elle joue dans ces villes en deux tours consécutifs.
- Les contraintes **(10)** à **(12)** imposent les exigences d'intégralité.

Bien que ce qui précède donne une formulation complète pour le TTP, les bornes inférieures prévues par sa relaxation de programmation linéaire sont très faibles. Pour améliorer cette formulation, Trick [147] a suggéré d'ajouter les contraintes dites odd-set pour chaque semaine. Une approche alternative et meilleure consiste à reformuler le problème en redéfinissant les variables de décision, tel que décrit dans [148].

Le problème du voyageur de tournoi miroir [138] et le problème du voyageur de tournoi avec des lieux prédéfinis [149, 150] sont deux variantes du problème du voyageur de tournoi. Le premier a la contrainte supplémentaire que les matchs joués en round t sont exactement les mêmes matchs joués en ronde t + (n - 1) pour t = 1,. . . , N - 1, mais avec des lieux inversés.

Le second est une variante du TTP avec un unique round robin, dans lequel le lieu de chaque match qui sera joué est connu à l'avance.

Le TTP et ses variantes ont été abordés par les différentes méthodes de résolution exactes et approchées. La première approche de la programmation en nombres entiers pour résoudre exactement le TTP a été proposé par Easton et al. [151], où la borne inférieure à été amélioré plus tard par Urrutia et al. [152]. Rasmussen et Trick [153] ont développé une approche hybride exacte de deux phases qui génère tous les modèles réalisables dans une première phase en utilisant la programmation par contraintes et attribue aux équipes les modèles dans la deuxième phase en utilisant la programmation entier. Les instances Benchmarks TTP miroir et non-miroir avec huit équipes ont été résolus à l'optimalité par Cheung [154] et Irnich [155]. Uthus et al. [156] ont développé une approche itérative basée sur A* pour le TTP, cette dernière a été en mesure de trouver des solutions optimales aux plus grandes instances de Benchmarks résolus à ce jour, avec la participation de dix équipes.

4.3.2 Minimisation de pauses (Breaks)

L'un des buts les plus importants de la programmation sportive est la minimisation des pauses. Les ligues organisatrices souhaitent avoir des horaires avec un nombre minimum de pauses ou, au moins, avec un nombre équilibré de pauses (ie, les horaires où toutes les équipes ont le même nombre de ruptures).

Les problèmes de planification sportifs sont souvent résolus par l'une des deux approches de décomposition:

(1) "First-schedule, then-break": déterminer d'abord les matchs à jouer à chaque tour, par la suite le modèle ‘Home/ Away’ correspondant.

(2) "First-break, then-schedule": déterminer d'abord un modèle Home/ Away réalisable, par la suite déterminer les matchs correspondants à jouer pour chaque tour.

Les deux approches ont été étudiées dans la littérature pour différents paramètres de problème.

Dans les deux cas, un modele Home/ Away avec un nombre minimum de pauses est souvent recherchée. La minimisation de pauses dans ce contexte a été discutée par pas mal de travaux. Les résultats sur le nombre de pauses dans un tournoi round robin sont également apparus. Et d'autres travaux ont été faits sur l'optimisation et les approches de programmation par contraintes pour la minimisation de pauses.

Dans d'autres travaux on trouve une relation établie entre deux aspects des problèmes de gestion de tournois round robin: les voyages et les pauses. En particulier, ils ont montré que, pour tout calendrier S, le nombre total de voyages (ou trips) T(S) et le nombre total de pauses B (S) sont telles que T(S) = n ·R- B (S)/ 2, où R =n - 1(resp. R = 2(n - 1)) est le nombre de tours dans un tournois single (respectivement double) avec n équipes. Ce lien entre la maximisation de pauses et la minimisation de distance a été utilisée pour dériver des bornes inférieures pour certaines instances du problème du voyageur de tournoi en miroir et pour prouver l'optimalité des solutions trouvées par une heuristique pour le second.

4.3.3 Minimisation d'effets 'Carry over'

Dans la stratégie des équipes ou des athlètes, en particulier pour les longues compétitions, un problème majeur consiste à équilibrer les efforts des équipes à travers la compétition. Si une équipe joue contre une équipe adversaire faible, elle est susceptible d'être en meilleure forme pour jouer au prochain tour que si elle avait joué contre une équipe adversaire difficile avant. Les équipes qui jouent contre des adversaires forts seront très probablement plus fatigué pour leur prochain match. Par conséquent, il est probable qu'une équipe(ou un sportif athlète) fait beaucoup moins d'effort lorsque elle joue contre un adversaire qui a joué avant contre un concurrent

très fort, donc il doit faire face à un adversaire qui a jouer précédemment contre un concurrent facile.

La situation ci-dessus est particulièrement vrai dans le cas des sports qui nécessitent une grande quantité d'effort physique (comme la lutte, le rugbyet les arts martiaux). Dans ce genre de sport, on trouve souvent qu'une équipe (ou un sportif) joue plusieurs matchs d'affilée, mettant une séquence très fatigante (resp. bien reposante) contre adversaires très attractifs (resp.peu attrayants). Certains calendriers peuvent contenir plusieurs de ces séquences de matchs faciles ou plus difficiles affectés à une ou plusieurs équipes. Cette situation ne caractérise pas un calendrier équitable et est hautement indésirable dans n'importe quel tournoi.

Bien que certains auteurs préconisent que les effets de report ne jouent pas un rôle majeur dans les sports collectifs, Flatberget al. Ont montré une application réelle à une ligue de football en Norvège dans lesquelles les reports des effets déterminés par une équipe spécifique ou un joueur spécifique affectent fortement les résultats finaux de la compétition. En outre, ils ont également montré que la minimisation de ces effets conduit à une fixation plus juste et mieux du calendrier des matchs. Une autre situation réelle intéressante est illustrée par des problèmes dans le football collégial américain, où une équipe (Alabama) a été programmée à plusieurs reprises prévue de jouer contre des équipes ayant un « by » à la semaine précédente. La séquence de matchs a été très peu attrayante pour l'Alabama, parce qu'elle était censés rencontrer souvent une équipe reposante qui n'a pas joué au tour précédent.

Pour un calendrier compact de type single round robin avec n équipes, on dit que l'équipe i donne un effet de report à l'équipe j si une autre équipe joue en tours consécutifs
contre les équipes i et j. les tours sont considérés cycliques, c'est à dire, le premier tour suit le dernier tour (n – 1) et peut être considérée comme le tour n. Si l'équipe i est une équipe très forte (respectivement. très faible) et plusieurs équipes jouent consécutivement contre les équipes i et j, alors l'équipe j peut être favorisée (respectivement. handicapés) par rapport aux autres équipes.

4. La Gestion des tournois sportifs

Soit cij≥ 0 compte le nombre d'effets « carry over » donné par équipe i à l'équipe j, pour tout i,j=1,...,N avec i= !j. La qualité d'un calendrier en ce qui concerne les effets « carry over » est mesurée par la valeur d'effet « carry over » $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^{2}$.

Dans un calendrier idéal ou équilibrée par rapport aux effets de report, on ne peut pas avoir des équipes : i, j, p, q telles que les deux équipes p et q jouent contre l'équipe j immédiatement après avoir joué contre l'équipe i. Dans ce cas, on devrait avoir cij= 1, pour tout i, j = 1,...,N avec i= !j.

Pour chaque paire d'équipes i, j = 1,...,N (avec i = !j) et pour chaque tour k=1,. .., N (avec les tours représentés cycliquement, tel que le tour n-1 est suivie par le premier tour), la variable binaire Y_{kij} est définie par:

$$y_{kij} = \begin{cases} 1, si\ l'équipe\ i\ joues\ contre\ l'équipej\ dans\ le\ round\ k, \\ 0, sinon; \end{cases}$$

Le nombre d'effets « carry over » donné par l'équipe i à j est : $\sum_{\ell=1}^{n}\sum_{k=1}^{n-1} y_{k\ell i}\cdot y_{(k+1)\ell j}$, pour tout i= !j; zéro autrement. Par conséquent, le problème de minimisation de la

(13) $$min\sum_{i=1}^{n}\sum_{j=1}^{n}(\sum_{l=1}^{n}\sum_{k=1}^{n-1} y_{kli}\ \cdot\ y_{(k+1)lj})^{2}$$

valeur d'effet « carry over » (13) à (19) peut être formulée par la programmation en nombres entiers comme suit:

(14) $$y_{kij} = y_{kji,} \qquad i,j,k\ = 1,\dots,n,$$

(15) $$\sum_{j=1}^{n} y_{kij} = 1, \qquad j,k = 1,\dots,n,$$

(16) $$\sum_{k=1}^{n-1} y_{kij} = 1, \qquad i,j = 1,\dots,n: i\ \neq j,$$

(17) $$y_{kii} = 0, \qquad i,k\ = 1,\dots,n,$$

(18) $$y_{nij} = y_{1ij,} \qquad i,j\ = 1,\dots,n,$$

(19) $$y_{kij}\epsilon\{0,1\}, \qquad i,j,k\ = 1,\dots,n.$$

- ***La fonction objective (13)*** minimise la valeur d'effets carry over totale.
- ***La contrainte (14)*** assure que les variables $y_{kij}=y_{kji}$ sont identiques ou que le match entre les équipes i et j est le même que celui entre les équipes j et i (i.e. il ya un match unique entre les équipes i et j en un tournoi single round robin).
- La contrainte (15) assure que chaque équipe joue exactement une fois dans chaque tout d'un calendrier compact.
- Les contraintes (16) et (17) garantirent que chaque équipe joue exactement une fois contre toutes les autres équipes.
- La contrainte (18) impose que le nième tour est équivalent au premier.
- La contrainte (19) impose la variable soit binaire.

4.4 Les domaines d'application

Dans cette section, on présente les méthodes d'optimisation pour les problèmes de planification et d'ordonnancement pour les différentes disciplines sportives comme le football, le baseball, le basket-ball, le cricket et le hockey. Les disciplines sportives les plus activés sont le Basket-ball et le football.

➢ Football

Malgré le grand nombre de papiers discutant l'application de la programmation entière et les métas heuristiques pour le Schedule des tournois de football, on remarque que seuls quelques-uns se réfèrent à des applications réelles financés par des contrats de recherche et de développement. Parmi eux des travaux ont appliqué heuristiques et branch-and-bound pour planifier les ligues de football professionnel de l'Autriche et de l'Allemagne. D'autres travaux ont adapté l'approche de programmation entière pour planifier la ligue de football italienne.

On trouve aussi une élaboration d'une stratégie de décomposition en programmation entière pour résoudre le problème d'optimisation bi critère découlant du plan du tournoi national de football brésilien, dans lequel l'un des objectifs consiste à maximiser le nombre de matchs qui peuvent être diffusées par les chaînes de télévision ouvertes (pour augmenter les revenus des droits de diffusion) et l'autre consiste à trouver un plan équilibré avec un nombre minimum de pose 'home' (respectivement de pose away) par souci d'équité. L'approche a été utilisée en

pratique pour planifier les 2009, 2010, et 2011 éditions du tournoi. Un autre travail consiste à développer un modèle programmation entière résoluble par CPLEX qui a été utilisé pour la première fois en 2010 pour planifier un tournoi de football professionnel de type double round robin du Honduras, joué par dix équipes.

- Basketball

Le premier problème considéré est le problème de la planification d'un calendrier pour le basket-ball de dix équipes correspondant à un tournoi double round robin détendue. Les équipes ont été autorisés à jouer au plus deux matchs consécutifs à l'extérieur sans retour à leurs site.

Pour la National Basketball Association (NBA) aux États-Unis, des travaux ont été basés sur la construction de calendriers pour 22 équipes où chaque équipe a joué 82 matchs et les heures de repos ont été prises en compte. Les méthodes basées sur des heuristiques pour le problème du voyageur de commerce ont été proposées afin de réduire les frais de déplacement de la compagnie aérienne.

D'autres travaux ont combinés la programmation en nombres entiers et les techniques énumératives pour déterminer le calendrier des matchs des neuf universités de la Atlantic Coast Conference (ACC).

4.5 Conclusion

Dans ce chapitre, nous avons montré l'importance de gérer les tournois sportifs et l'importance de créer un calendrier optimal et son impact sur le stade économique, par la suite nous avons montré les différents types de contraintes pouvant exister dans un tournois sportif et qui différent d'un tournoi à un autre et d'une ligue sportive a une autre, ensuite nous avons présenté quelques notions générales dans le domaine des compétition sportives ou on a présenté les deux phases importantes dans la création d'un calendrier complet sui sont les modèles Home/ Away et le calendrier, ainsi que la présentation graphique des tournois, d'autre part nous avons montré les problèmes fondamentaux qu'on doit prendre en compte dans notre schedule, on s'intéresse dans notre travail au problème de minimisation de distance, et nous avons terminé par présenter quelques domaines d'application et les travaux faits.

5 Proposition d'une approche de résolution du C-TTP qui casse les symétries

5.1 Introduction

Dans cette section nous allons étudier un problème combinatoire réel et très connu, il s'agit du problème de compétition sportive C_TTP (circular travelin tournament problem). La propriété fondamentale de ce problème est sa structure symétrique, notre objectif consiste a exploiter cette dernière afin d'améliorer la résolution de ce problème. Donc ce chapitre sera divisé en deux parties : en premier lieu nous présentons le problème (contraintes,…etc.) et nous étudions les symétries que possède ce dernier. Dans la deuxième partie nous proposons une méthode de résolution au problème en prenons compte des symétries.

5.2 Le problème Circular Traveling Tournament Problem

5.2.1 Description du C-TTP

- ***Rappel :***

Le problème du voyageur de tournoi (***TTP***) consiste à trouver la distance minimale à un tournoi double round-robin où le nombre de pauses consécutives est borné **[**136**]**. Les équipes et leurs sites sont données par $T = \{1, 2, \ldots, n\}$ pour un nombre pair $n \in 2N$. Les indices $t, t' \in T$ représentent les équipes et les indices $i, j \in T$ représentent leurs sites. Soit $D = (dij)$ pour $i, j \in T$ représente la distance entre les sites i et j. chaque solution réalisable au TTP peut être représentée par les tours $p^t = (i^t_1, i^t_2, \ldots, i^t_{2(n-1)})$ pour tous les équipes $t \in T$. ici $i^t_s \in T$ décrit pour chaque créneau horaire $s \in \{1, 2, \ldots, 2(n-1)\}$ le site ou l'équipe t va jouer son match. Pour $i^t_s = t$, l'équipe t joue dans son lieu, autrement, t joue à l'extérieur, Il est clair qu'une solution ne peut être $\sum_{t\in T}\sum_{s=0}^{2(n-1)} d_{i^t_s, i^t_{s+1}}$ours sont compatibles. C'est-à-dire un match a l'exterieur $i^t_s = t' \neq t$ implique un match à domicile $i^{t'}_s = t'$.avec la définition $i^t_0 := i^t_{2n-1} :=$t pour tous les équipes $t \in T$, le cout d'une solution p est la distance totale parcourue :

En plus la contrainte 'pas de répétition' (NRCs) exige qu'un match a domicile de t contre t' ne peut pas être suivi par le match externe correspondant ou t' joue dans son domicile contre t.

- ***Définition du C-TTP (le problème de voyageur de tournoi circulaire) :***

5. Proposition d'une approche de résolution du C-TTP qui exploite les symétries.

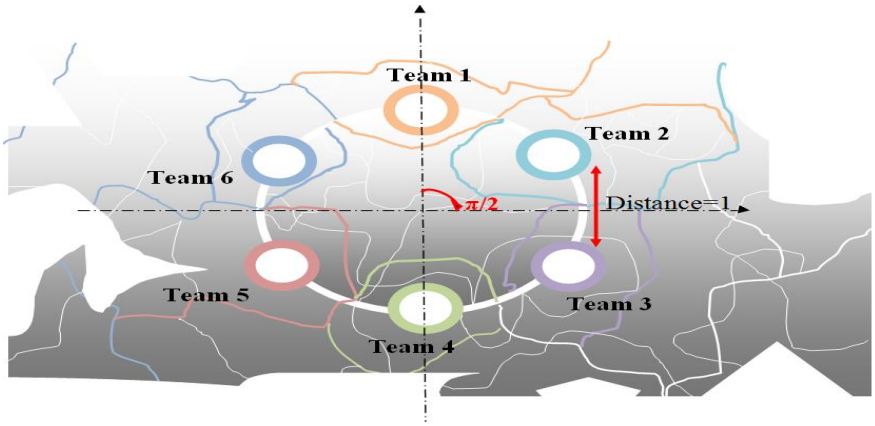


Figure 5-1 Exemple générique d'une instance circulaire d'un problème TTP avec 6 équipes.

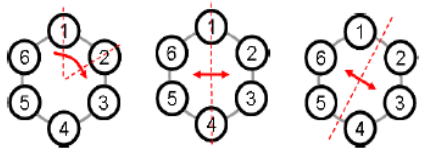
Easton et al. (2001) [136] ont introduit les instances TTP circulaires, ou les sites des équipes sont localisés sur un cercle. La distance entre les sites voisins est égale à un donc la distance entre deux équipes i>j est dij = min {i-j, n+j-i}. Il a été prouvé que ces instances sont difficiles à résoudre à cause de la présence des symétries.

Dans ce qui suit nous allons présenter les différents cas de symétries de notre problème et comment on peut les exploiter pour chaque cas.

5.2.2 Les symétries dans les instances circulaires du TTP

Pour une solution réalisable au ***C TTP***, il est possible de faire tourner les équipes et les lieux c'est-à-dire de faire leurs rotation, c'est à dire remplacer ***t*** par ***t +1*** (pour ***t*** <***n***) et ***n*** par ***1***. La solution résultante est également réalisable et a des coûts identi

$$(d_{ij}) = \begin{pmatrix} 0 & 1 & 2 & 3 & 2 & 1 \\ 1 & 0 & 1 & 2 & 3 & 2 \\ 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & 2 \\ 2 & 3 & 2 & 1 & 0 & 1 \\ 1 & 2 & 3 & 2 & 1 & 0 \end{pmatrix}$$



(a) (b) (c) (d)

Figure 5-2 Une instance circulaire pour n = 6; (a) Les distances, (b) la rotation par 2π/6, (c) et (d) les réflexions à travers les sommets ou les arrêtes.

réflexions à travers les sommets ou les arrêtes opposés, produisent aussi des solutions symétriques comme le montre la figure 1, et avec des coûts identiques.

5. Proposition d'une approche de résolution du C-TTP qui exploite les symétries.

Formellement, le groupe dièdre de symétrie ***Dn*** d'un polygone régulier à n côtés opère sur les (équipes/sites). ***Dn*** englobe les symétries suivantes :

- Les n rotations par multiples de $2\pi/n$ (y compris l'identité) et
- Les n réflexions à travers les sommets opposés et les arêtes opposés.

Cette opération peut être exploitée de différentes façons, soit en considérant les affectations ('Home/Away') ou en considérant les 1-facteurs orientés.

5.2.2.1 Symétries sur les affectations Home/ Away

Une affectation 'Home/ Away' (HAA) représente une solution partielle pour le TTP. Un HAA spécifie pour chaque équipe et pour chaque créneau horaire si l'équipe joue dans son site ou joue dans le site externe (le site de son adversaire). Lorsqu'on atteint l'égalité entre les équipes jouant à domicile et les équipes jouant à l'extérieur pour un créneau horaire, le nombre global de slots HAA est (***n n/2***). ***Dn*** donc opère sur l'ensemble des slots HAA pour un créneau ***s*** donné.

Exemple : pour n=6, on a (6 3)=20 slot HAA possibles. Les slots HAA qui sont symétriques au (hhhaaa) sont définis comme une orbite. Par exemple l'orbit de (hhhaaa) est {(hhhaaa), (ahhhaa), (aahhha), (aaahhh), (haaahh), (hhaaah)}. Pour cet exemple, les trois différents orbits avec les longeurs 2, 6 et 12 sont montrés dans la figure 5-3.

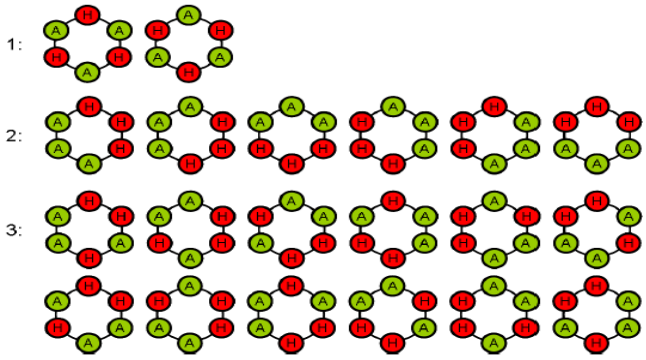


Figure 5-3 Les orbites de HAA pour n = 6.

Due à la symétrie présente dans les instances circulaires, il suffit de considérer une seule représentation de slot HAA pour chaque orbite pour un créneau horaire s donné. Ceci représente le facteur le plus important dans la réduction de symétrie. Donc prenant l'exemple de n=4, il y'a deux branchements possibles : le premier branchement peut fixer que les équipes 1 et 2 jouent dans leurs sites et que les autres équipes jouent à distance, par ailleurs, le second branchement peut fixer que les

équipes 2 et 4 jouent à domicile et que les équipes 1 et 3 jouent leurs matchs à l'extérieur de leurs sites, comme est présenté dans la figure 5-4.

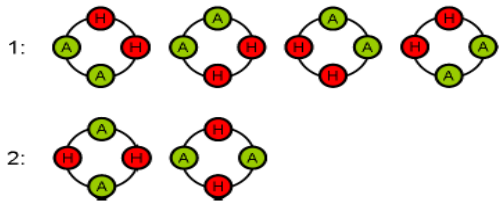


Figure 5-4 Les orbites de HAA pour n = 4.

La figure 2(c) présente le nombre de modèles HAA possibles et le nombre d'orbites correspondants à chaque nombre de joueurs dans une équipe.

5.2.2.2 Symétries des 1-Factors

Pour chaque intervalle de temps s, une solution impose un tournoi**.** Par exemple, ***f*** := {(3, 2), (1, 6), (5, 4)} signifie que l'équipe 3 joue à domicile contre l'équipe 2, l'équipe 1 joue dans son site contre l'équipe 6 et l'équipe 5 joue à son domicile contre l'équipe 4. Le groupe dièdre ***Dn*** opère naturellement sur les 1-facteurs orientés. Sur ***F***, ***Dn*** produit une orbite composé de quatre éléments:{f, {(2, 1), (4, 3), (6, 5)}, {(1, 2), (3, 4), (5, 6)}, {(2, 3), (4, 5), (6; 1)}}. Comme précédemment tous les éléments d'une orbite son équivalents donc, il suffit de considérer une seule représentation 1-factor orientée pour le slot s=1.

La figure 5-5 montre pour n=4 les trois orbites avec 4 éléments : le premier branchement affecte les matchs (1,2), (4,3) au slot 1, le second (1,2), (3,4) et le troisième affecte les matchs (1,3), (2,4).

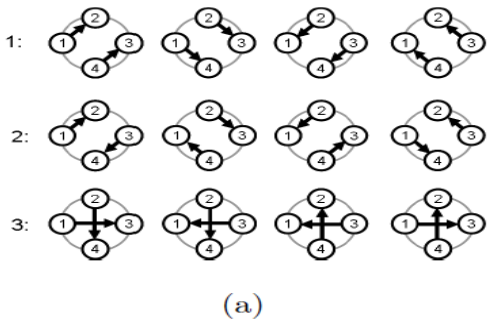


Figure 5-5 Les orbites des 1-Factors orientés pour n = 4.

5.2.2.3 *Inversement de tours*

Un autre type de symétrie peut être utilisé dans le cas où les distances d_{ij} sont symétriques, chaque solution avec les tours $p^t = (i^t_1, i^t_2, \ldots, i^t_{2\,(n-1)})$ des équipes $t \epsilon T$ à la solution inverse correspondante avec les tours inversés : $(i^t_{2(n-1)}, \ldots, i^t_2, i^t_1)$ pour tous les équipes t. Le groupe symétrique $Z_2 = \{id, rev\} = \{+1, -1\}$ composé de l'identité id=+1 et l'inverse rev=-1.

Plusieurs auteurs ont déjà exploité cette symétrie: Dans l'algorithme *branch-and-price* d'Irnich **[**155**],** les arcs sont retirés du réseau d'une équipe (par exemple l'équipe 1) de telle sorte que seule l'une des deux tours symétriques peut être générée. Uthus et al **[**156**]** en leur approche *branch-and-bound*, exigent que la première moitié du calendrier de l'équipe 1 (c'est-à-dire les matchs dans les slots 1 à $n - 1$) aura plus de matchs à domicile que des matchs à l'extérieur (ou vice versa).

Dans tous les cas, l'accélération qui peut être tirée de l'exploitation des tours inverses est limitée par un facteur 2.

5.2.2.4 *Symétries sur les pairs de HAA*

Les deux types de symétries dans le slot 1 et la symétrie introduite par l'inversement des tours peuvent être combinés. Le nouveau groupe symétrique est $Gn = Dn \times \{+1, -1\}$ avec $4n$ éléments. Tout élément du groupe $g=(d, e) \epsilon G$ opére sur une solution avec les tours $(i^t_1, i^t_2, \ldots, i^t_{2\,(n-1)})\ t \epsilon T$ comme suit : la solution résultante est donnée par les tours : $(i^{d(t)}_1, i^{d(t)}_2, \ldots, i^{d(t)}_{2\,(n-1)})$ pour e =+1 et $(i^{d(t)}_{2(n-1)}, \ldots, i^{d(t)}_2, i^{d(t)}_1)$ pour e =-1.

Afin d'exploiter cette symétrie, il faut deux slots à considérer simultanément. Par exemple le premier créneau horaire 1 ainsi que le dernier créneau 2(n-1). Les nombre total des pairs HAA est évalué donc a $(n\ n/2)^2$. Pour n=4, l'opération de $G4$ sur $(4\ 2)^2$=36 pairs de modèles HAA montrés sur la figure 4(a).

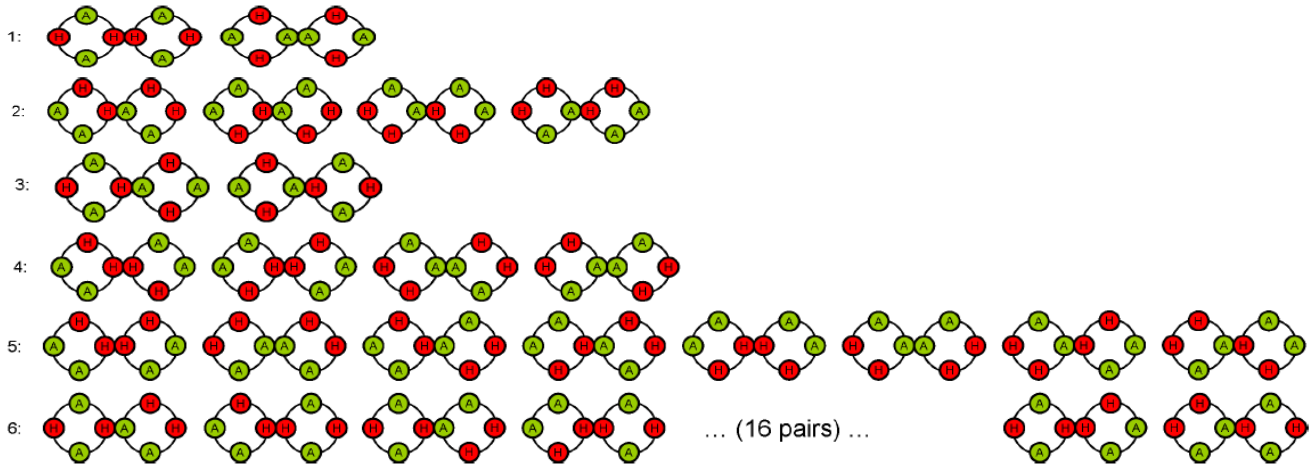


Figure 5-6 Opération de Gn = Dn * Z2: Orbits des pairs HAA pour n = 4.

En concernant la combinaison de la symétrie des 1-Factor et l'inversement des tours, lorsque le nombre des 1-facteurs augmente rapidement avec n, le nombre de pairs se développe encore plus vite, comme on le voit dans le tableau de la figure 4 (c). En raison du grand nombre de paires il n'était pas possible de calculer les orbites pour n = 10 et n = 12. Ainsi, la réduction de symétrie basée sur des paires de 1-facteurs n'est pas une approche prometteuse pour accélérer les algorithmes énumérative.

5.3 Modélisation et résolution du C-TTP

5.3.1 Modélisation

5.3.1.1 Représentation matricielle de la solution

Pour la solution nous choisissons une représentation matricielle n* 2(n-1) ou les lignes représentent les tours des équipes, la ligne i correspond au tour de l'équipe i, les colonnes correspondent aux slots ordonnées de 1 à 2(n-1) regroupés en deux rounds consécutives. Ou chaque round a n-1 slots dans les équipes jouent entre elles leurs premiers match.

L'intersection des lignes i avec les colonnes j représentent l'équipe adverse qui va jouer contre l'équipe i dans le slot j.

Pour la représentation Home / Away, nous avons choisis de représenter cette information par les symboles (+) et (-) ou le (+) désigne que l'équipe i joue à l'extérieur de son site et le symbole (-) désigne que l'équipe joue dans son domicile.

5. Proposition d'une approche de résolution du C-TTP qui exploite les symétries.

Donc nous avons choisis une représentation globale du Schedule, ou nous avons représenté le timetabling et le modèle Home/ Away à la fois dans la même représentation matricielle.

5.3.1.2 Les contraintes, variables et domaines de valeurs

- ***Les variables de décision du problème C-TTP :***

Les variables de décision du problème représentent les points d'intersection des lignes (tours) avec les colonnes (slots), et qui représentent en effet deux informations :

- Pour une équipe i elle représente le site de l'équipe adverse dans un slot j.
- Elle définit le modèle H/ A de l'équipe.

Donc les variables de décision de la Matrice Schedule sont : Schedule [team, slot].

Donc pour un problème de taille n on a (n * 2(n-1)) variables de décision.

- ***Les Domaines des variables du problème C-TTP :***

Le domaine D des variables sont des valeurs entières positives (lorsque l'équipe joue a domicile) et des valeurs entières négatives (lorsque l'équipe joue dans le site de l'équipe adversaire).

- ***Les contraintes du problème C-TTP :***

Soit une fonction Play (i, j, t) désigne i joues chez j dans le slot t,
alors les contraintes du problème considéré sont :

(1) L'équipe ne joue pas avec elle-même.

forall (i in TEAMS, t in SLOTS)
play(i,i,t) = 0 !

(2) Chaque équipe joue exactement une seule fois contre chaque autre équipe dans un round. (team i plays one team in each slot)

forall (i in TEAMS, t in SLOTS)
sum(j in TEAMS) (plays(i,j,t)+plays(j,i,t)) = 1

(3) DRRT: toute équipe doit jouer une seule fois dans un slot. (team i plays at team j exactly once)

forall (i,j in TEAMS | i <> j)

sum (t in SLOTS) plays(i,j,t) = 1

(4) Dans un slot si une équipe joue Home l'équipe adverse doit jouer Away et vice vers sa.

(5) La contrainte At most: l'équipe ne dépasse pas U fois Home consécutives repectivement (Away).

forall (i in TEAMS, t in 1..2*n-5)

1 <= sum(j in TEAMS) (plays(i,j,t)+plays(i,j,t+1)+

plays(i,j,t+2)+plays(i,j,t+3)) <= 3

(6) La contrainte (NRCs) : no repeaters

forall (i,j in TEAMS, t in 1..2*n-3)

plays(i,j,t)+plays(j,i,t)+

plays(i,j,t+1)+plays(j,i,t+1) <= 1

- ***Les extra-contraintes LDSB et la composition des symétries:***

La suppression des symétries se fera par l'ajout d'un ensemble d'extra contraints LDSB, la composition de ses contraintes est assurée par une répétition de tests de la propagation de ses contraintes et de vérifications. Elles seront implémentés dans des brancher indépendants traitant les symétries considérées.

5.3.2 Résolution

5.3.2.1 Choix de la méthode de résolution LDSB-B&B

Pour résoudre le CTTP nous avons choisis la méthode de de résolution énumérative et complète Branch and bound avec minimisation de la fonction objective qui la distance totale parcourue par les équipes et qui doit être minimisée.

Pour le traitement des symétries nous avons intégré la méthode dynamique (LDSB) à notre programme de recherche, nous avons choisi cette technique pour les raisons suivante :

- ✓ Efficacité de la méthode : elle supprime les types communs de symétries avec efficacité.
- ✓ Rapidité : elle réduit le temps d'exécution des problèmes dans tous les cas.

5. Proposition d'une approche de résolution du C-TTP qui exploite les symétries.

- ✓ Flexibilité : elle est implémentée comme une méthode dynamique mais a l'avantage qu'elle est comme tout ordre de recherche (statique ou dynamique).
- ✓ Elle évite le temps d'exécution additionnel et reduit la complexité de calcul du calcul du groupe.
- ✓ Elle ne tombe pas en conflit avec un petit ensemble de contraintes de suppression de symétries statiques.
- ✓ Elle regroupe les avantages des autres approches dynamiques alternatives.
- ✓ Elle traite les grandes instances et benchmarks des problèmes.

La méthode de résolution que nous allons appliquer est donc (Branch&Bound-LDSB).

5.3.2.2 Description de la méthode de suppression des symétries LDSB

5.3.2.2.1 Représentation des symétries dans LDSB

Afin de représenter nos symétries par LDSB nous devons suivre la syntaxe interne de ce dernier et qui représente la base des algorithmes de suppression des symétries.

5.3.2.2.1.1 Les modèles de LDSB et leurs syntaxes

Soit un CSP P = (X, D, C), la forme la plus générale pour toute symétrie de variable s est $< x_1, \dots x_n > \rightarrow < x'_1, \dots x'_n >$ ou $< x_1, \dots x_n > = < x'_1, \dots x'_n > = X$, et s $(x_i) = x'_i$. comme l'inverse de toute symétrie est aussi une symétrie, alors $< x'_1, \dots x'_n > \rightarrow < x_1, \dots x_n >$ est aussi une symétrie. Cela motive la syntaxe de LDSB qui représente s et son inverse simplement par : $\{ < x_1, \dots x_n > , < x'_1, \dots x'_n > \}$. Cette représentation est la même que la notation standard de *Cauchy* de deux lignes pour les permutations, a la seule exception et que l'on considère pour représenter la permutation et son inverse à la fois.

Naturellement, l'ordre dans lequel les deux couples (que nous appellerons comme des séquences) apparaissent dans cet ensemble n'est pas significatif. C'est la première des quatre modes de LDSB introduites dans cette section, et il sera désigné sous le motif de séquences variables interchangeables. Étant donné que toute valeur symétrie peut

être représentée de manière similaire en utilisant des séquences de valeurs de séquences (au lieu de variables), le second motif de LDSB est le motif de séquences interchangeable valeur équivalente.

Notez que, pour plus de simplicité, nous ne assumons aucune variable (ou la valeur) apparaît plus d'une fois dans une séquence d'LDSB. Cela ne provoque pas de perte de généralité puisque, par définition de symétrie, une variable (valeur) ne peut être mis en correspondance avec une variable (valeur). Ainsi, seule une occurrence est nécessaire.

Alors que toutes les symétries de variables et de valeurs peuvent être représentées en utilisant l'un des deux motifs ci-dessus, de nombreuses symétries des CSPs ont des propriétés spéciales qui leur permettent d'être représentés de façon plus compacte. Comme nous le verrons, c'est pour ces symétries que LDSB est capable de tailler des solutions symétriques plus efficacement et à détecter et à tailler plusieurs compositions des symétries. Nous discutons ces propriétés et leurs conséquences en termes de la syntaxe.

- Omettre l'auto mappages: La première propriété de symétrie est de cartographier un certain sous-ensemble des éléments (qu'il se agisse de variables ou des valeurs) à eux-mêmes. Pour ces types de symétries, LDSB utilise un raccourci dans lequel les séquences omettent ces éléments qui sont mappés sur eux-mêmes. Noter que, lorsque désignant toute symétrie en utilisant ces sténographies, les éléments des deux séquences sont les mêmes. Les valeurs qui sont mappés se sont omis dans les séquences, et que LDSB ne représentent pas explicitement la symétrie d'identité.

- Simplifiez involutions: La deuxième propriété de symétrie est égale à son inverse (dans la langue de permutations, ceux-ci sont appelés «involutions»). Pour ces symétries nous pouvons scinder les éléments en trois ensembles disjoints (Id), (EL1) et (EL2), Où (Id contient tous les éléments qui sont affectés à eux-mêmes, alors que tous les éléments (EL1) sont mappés à des éléments (EL2) et vice versa (et, par conséquent, on aura le même cardinal (EL1) = (EL2)). LDSB représente les involutions comme un ensemble avec deux séquences, un pour les éléments (EL1) et un pour les éléments (EL2).

5. Proposition d'une approche de résolution du C-TTP qui exploite les symétries.

- Réglez la représentation des ensembles complets de permutation: La troisième propriété est que toutes les permutations d'un ensemble particulier de séquences d'éléments (que ce soit des variables ou valeurs) sont symétries du CSP. Un tel groupe de symétries peuvent être représentés très compacte dans LDSB comme l'ensemble des séquences. Notez que cet ensemble représente non seulement toute permutation de deux éléments dans le problème mais aussi leurs compositions et, ainsi, la permutation d'ensemble complet.

- Des séquences de longueur égale à un: Si toutes les séquences dans une instance de notre modèle de séquences interchangeables contiennent une seule variable (valeur), alors nous pouvons simplifier ce cas en utilisant un ensemble de variables (valeurs) plutôt que d'un ensemble de tuples de variables (les valeurs). Cette version particulière du motif de séquence sera désigné sous le motif de variables interchangeable, et un équivalent pour la valeur sera appelé le motif valeurs interchangeables. Ces deux, avec nos deux tendances générales séquences interchangeables, forment les quatre modèles utilisés par LDSB.

- ***La syntaxe de LDSB :*** La syntaxe formelle utilisée par LDSB pour représenter les symétries est : étant donné un CSP P = (X, D, C). Un exemple de motif de LDSB (W) doit être une instance de l'une des variables du motif interchangeables (valeurs) ou la variable interchangeable (valeur) des séquences motif. Dans le premier cas, (W) doit être un ensemble de vari-recevables (valeurs) tel que $W \subseteq X$ ($W \subseteq D$). Dans ce dernier cas, (W) doit être un ensemble de variables (valeur) des séquences telles que les cinq propriétés suivantes s'appliquent. Soit seqi(p) qui désigne l'élément (variavle ou valeur) dans la position p de seqi. Premièrement, on a $var\ (W) \subseteq X$ ($vals\ (W) \subseteq D$), Deuxièmement, toutes les séquences doivent avoir la même longueur k. Troisièmement, tous les éléments d'une séquence doivent être différents, i.e., $\forall\ i. |vars(seqi)| = k$ ($|vals(seqi)| = k$). Quatrièmmement, aucunes dex sequences seqi , seqj avec $i \neq j$, ne peuvent avoir les meme elements dans une position donnée, i.e., $\forall p. seqi(p) \neq seqj(p)$. T finalement, toute paire de sequences seqi et seqj doivent etre dijoints, i.e., $vars(seqi) \cap vars(seqj) = \emptyset. (vals(seqi) \cap$

$vals(seqj) = \emptyset$), ou bien ils ont le meme ensemble d'éléments, i.e., $vars(seqi) = vars(seqj)$ $\big(vals(seq) = vals(seqj)\big)$.

5.3.2.2.2 Les symétries représentées par les modèles LDSB-B&B

Parlons maintenant des symétries représentées par chaque instance de pattern. Considérant un CSP P = (X, D, C) qui a une instance de pattern W et laisser (GW; P) représentent le groupe de permutation sur (allumé (P)) représenté par W. si W est une instance du modèle des variables interchangeables (valeurs), alors $\Upsilon_{w,p}$ est identique a $p_{w,p}$. Si W = {seq1, …, seqm}est une instance du modèle de sequence de valeurs interchangeables, alors $\Upsilon_{w,p}$ est induite par l'ensemble $\{ s_{i,j} \ i,j \epsilon 1..m\}$ de permutations sur valeurs, ou :

$$s_{i,j} = \begin{cases} si,j\big(seqi(p)\big) = seqj(p) \quad \forall\, p \in 1..k \\ si,j\ \big(seqj(p)\big) = seqi(p)\ p \in 1..k\ si\ vals(seqi) \cap vals(seqj) = \emptyset \\ si,j(d) = d \quad pour\ tout\ d \in D \backslash (vals(seqi) \cup vals(seqj)) \end{cases}$$

et k est la longueur de chaque séquence dans W. Notez que, étant donné les cinq propriétés des instances de valeur interchangeables séquences modèle introduit dans la section 4.1, chaque sij est une permutation sur certains D(x) et {sij | i,j ϵ 1.. m} forme un groupe de permutation. L'ensemble $\Upsilon_{w,p}$ peut être définie de manière similaire, lorsque W est un exemple du motif de séquences variables interchangeables.

Alors, étant donné un CSP P avec l'ensemble Patts = {W1, …, Wm} de instances du modèle, Patts représente toute permutation du groupe de permutations $[\Upsilon_{w^1,p}, \ldots, \Upsilon_{w^m,p}]$, où $\Upsilon_{w^i,p}$ est le groupe représenté par permutation exemple de motif Wi.

A noter que ne importe quelle permutation de variable (et de valeur) peut être représentée en utilisant un LDSB en position de l'un des quatre modèles introduits ci-dessus. Ceci distingue déjà LDSB des autres méthodes, telles que les méthodes SSB incomplets, qui ne peut casser les permutations représentés par des positions des motifs de variables interchangeables et de valeurs interchangeables (et non pas avec

ceux qui sont représentés par les schémas généraux des séquences plus interchangeables).

5.3.2.2.3 Algorithmes de suppression des symétries pendant la recherche

L'algorithme de suppression des symétries de LDSB est similaire a celui de SBDS avec les différences suivantes :

- ✓ Les types de symétries utilisées.
- ✓ La manière dont ces symétries sont représentées et traitées.
- ✓ Seulement les symétries qui sont actives a un nœud de recherche n qui sont utilisées pour calculer les contraintes de suppression des symétries au nœud n.

Algorithm 1: `search(`$X, Patts$`)`

input : Set X of search variables; Set *Patts* of pattern instances

```
1  if all variables in X are fixed then stop;
2  ;
3  Select variable x ∈ X and value d ∈ D(x);
4  Create choice point
5      Left branch:
6          Assert x = d;
7          search(X,update(x = d,Patts));
8      Right branch:
9          L ← some_orbit(x = d,Patts);
10         foreach literal l ∈ L do Assert ¬(l);;
11         search(X,Patts);
```

La recherche commence avec un appel au search(X, Patts), décrite dans l'algorithme 1, ou X représente l'ensemble de variables à affecter, et Patts est l'ensemble des instances modèles représentant les symétries connues comme actives au nœud racine (root node)(ie toutes les symétries du problème vu que l'affectation au nœud racine est vide). Dans le cas où toutes les variables dans X sont fixés la recherch est complète, dans le cas contraire, la recherche choisie la prochaine variable x et valeur d non fixées. Et cela selon la stratégie de recherche choisie et elle explore la branche x=d (comme par defaut), la seule difference ici est que avant d'appliquer ça la recherche peut avoir besoin de modifier Patts puisque le post de l'affectation x = d peut causer certaines symétries dans les instances modèle pour etre non acitves (ou meme supprimées). Apres retour arrière (bachtracking), la recherche post la négation de (certains) littéraux qui par rapport au patts, ils sont symétriques a x = d , ie, (certaisn) de ceux dont l'orbite de x = d pour chacun des instances modèles(qui lui-

même inclut toujours x = d). donc en résumant, la seule différence avec la recherche normale, c'est que l'ensemble des symétries soit modifié en explorant la partie gauche et qu'on post la négation de l'orbite en explorons la partie droite. L'algorithme 1 n'impose pas des restrictions sur la sélection de x ou de d, par conséquent, toutes stratégies de recherche de variable où de valeur peut etre utilisée.

La manière dont chaque instance modèle W ϵ Patts est modifiée et la suppression atteinte dépendent du modèle associé à W. comme c'est indiqué par les algorithmes 2 et 3 respectivement. Ces algorithmes représentent le résultat de l'application de la théorie du groupe aux modèles particuliers de symétries à traiter.

Algorithm 2: `update`$(x = d, Patts)$

input : Set *Patts* of pattern instances before literal $x = d$ is posted
output: Set of pattern instances after literal $x = d$ is posted

```
1  foreach symmetry pattern W ∈ Patts do
2      if W is a set of interchangeable variables then W ← W \ {x};
3      ;
4      else if W is a set of interchangeable values then W ← W \ {d};
5      ;
6      else if W is a set of interchangeable value sequences then
7          W ← W \ {sq | d ∈ sq};
8      else if W is a set of interchangeable variable sequences then
9          do nothing;
10     end
11 end
12 return Patts;
```

Algorithm 3: `some_orbit`$(x = d, Patts)$

input : Set *Patts* of pattern instances
output: Some subset L of the literals in the orbit of $x = d$ according to *Patts*

```
1  L ← {x = d};
2  foreach pattern instance W ∈ Patts do
3      if W is a set of interchangeable variables AND x ∈ W then
4          foreach w ∈ W \ {x} do L ← L ∪ {w = d};
5          ;
6      if W is a set of interchangeable values AND d ∈ W then
7          foreach e ∈ W \ {d} do L ← L ∪ {x = e};
8          ;
9      if W is a set of interchangeable value sequences AND ∃seq^1 ∈ W, seq^1[i] = d then
10         foreach seq^2 ∈ W \ {seq^1} do L ← L ∪ {x = seq^2[i]};
11         ;
12     if W is a set of interchangeable variable sequences AND ∃seq^1 ∈ W, seq^1[i] = x then
13         foreach seq^2 ∈ W \ {seq^1} do
14             if active(seq^1, seq^2) then L ← L ∪ {seq^2[i] = d};
15             ;
16         end
17     end
18 end
19 return L;
```

Algorithm 4: `active`(seq^1, seq^2)

input : Sequences seq^1 and seq^2 of interchangeable variables
output: true if $seq^1 \leftrightarrow seq^2$ is known to be active, false otherwise

```
1 for j ← 1 to |seq¹| do
2     if fixed(seq¹[j]) and fixed(seq²[j]) and seq¹[j] = seq²[j] then
3         do nothing;
4     else if not fixed(seq¹[j]) and not fixed(seq²[j]) then
5         do nothing;
6     else return false ;
7     ;
8 end
9 return true;
```

5.3.2.2.4 Quelques propriétés théorique de la technique

- **Correctnes et complétude de notre cas d'utilisation (VariableSequenceSymmetry/ set variableSequenceSymmetry).**

Théorème : Pour un CSP P, Soit W = {seq^1, …, seq^m} est l'ensemble de m séquences de variables interchangeables et $\Upsilon_{W,\ P}$ est le groupe des symétries représentés par W. l'ensemble some_orbit(x = d, {W}) calculé par LDSB par l'affectation A est le sous ensemble de l'ensemble obtenu par Orbit(X= d, SetStab(A, $\Upsilon_{W,\ P}$)).

Preuve : Soit $\Upsilon_{WA,\ P} \subseteq \Upsilon_{W,\ P}$ définit comme l'ensemble des symétries $\sigma_{i\ j}$ qui mappe les variables dans seq^i a ceux dans seq^j, et pour lequel active(seq^i, seq^j) est Vrai pour l'affectation A. de l'algorithme 4, toute $\sigma_{i\ j} \in \Upsilon_{WA,\ P}$ stabilise A. de l'algorithme 3, pour tout litéral l $\in$ some_orbit(x = d, {W}), on a l = (y = d) pour certains y distincts de x. aussi, de l'algorithme 3, il doit y exister une séquence seq$^i \in$ W tel que x est dans la position p, et une autre séquence différente seq^j ou y aussi se produit dans la position p, tel que $\sigma_{i\ j} \in \Upsilon_{WA,\ P}$ (et par conséquent dans $\Upsilon_{W,\ P}$) et puisque tous les elemeents de $\Upsilon_{WA,\ P}$ stabilisent A, alors $\sigma_{i\ j}$ est aussi dans SetStab(A, $\Upsilon_{W,\ P}$).

Par conséquent, l = $\sigma_{i\ j}$ (x = d) doit être dans Orbit(x = d, SetStab(A, $\Upsilon_{W,\ P}$)).

Ce théorème peut être facilement étendu aux ensembles de séquences de variables interchangeables.

Théorème : (Corectness pour ensemble de séquences de variables interchangeables) : Pour un CSP P, soit W^1, …, W^m des instances du modèle des

séquences de variables interchangeables, et W_A^1, …, W_a^m les instances du modèle calculés par LDSB à partir de W^1, …, W^m, respectivement, a l'assignement A. Alors, l'ensemble retourné par some_orbit(x =d, { W_A^1, …, W_A^m }) est un sous ensemble de celui obtenu par Orbit(x = d, SetStab(A, [$\Upsilon_{W^1,P}$, …, $\Upsilon_{W^m,P}$])).

Preuve : le theorème est simple et de l'algorithme 3 :

Some_orbit(x =d, { W_A^1, …, W_a^m }) = $U_{i\in 1..m}$ some_orbit(x =d, { W_A^i}).

- **Composition des symétries.**

Lorsque la fonction some_orbit(x = d, patts) retourne l'ensemble L de littéraux qui peuvent être supprimer de la recherche, on peut en effet supprimer plus de littéraux si en plus des symétries représentées par chacun des instances modèle dans patts on considère aussi ceux obtenus par composition de ces symétries. L'algorithme 5 cherche certains de ces extra littéraux par l'application des symétries représentés dans Patts mais non seulement au littéral original x = d mais aussi a chaque nouveau littéral symétrique trouvé. Afin d'atteindre ceci l'algorithme construit une queue de nouveaux littéraux a traiter (initialisée a x = d). chaque litéral dans Queue est dont retiré de la queue et (certains) des littéraux de son orbite sont calculés. Et chaque littéral qui n'a pas été calculé auparavant est ajouté a L et aussi inséré a la queue afin d'être traité par la suite. Le processus s'exécute et se termine lorsque y'aura plus de littéraux à retirer de la queue. Ce processus peut être imaginé comme une traversé largeur d'abord d'un graphe ou chaque littéral est représenté par un sommet et il existe un arc qui relie le littéral a au littéral b si pour certaine symétrie active σ on a σ(a) = b.

En effet, cet algorithme ne compose réellement pas les symétries mais il applique les symétries d'une façon répétée sur les littéraux. Cette différence est importante pour l'efficacité : lorsqu'on compose même un petit nombre de symétries on peut arriver à un nombre exponentiel de symétries, cependant cet algorithme calcule au plus un nombre quadratique de littéraux (|X| × |D|). Cependant l'algorithme peut oublier certaines symétries. En effet LDSB calcule d'abord les stabilisateurs de chaque symétrie séparément par la suite il compose leur éléments. Cela induit a la perte de certaine opportunités : soit deux groupes de permutation G1 et G2, [SetStab(A, G1), SetStab(A, G2)] peut etre un sous ensemble propre de SetStab(A, [G1, G2]).

Algorithm 5: `composed_orbit` $(x = d, Patts)$

```
input  : Set Patts of pattern instances before literal x = d is posted
output: Set L of literals symmetric to x = d according to Patts
1  Queue ← [x = d];
2  L ← {x = d};
3  while Queue is not empty do
4      l ← serve(Queue);
5      foreach literal sl ∈ some_orbit(l,Patts) do
6          if sl ∉ L then
7              L ← L ∪ {sl};
8              append(Queue, sl);
9          end
10     end
11 end
```

- **Correctness de la composition de LDSB.**

Pour un CSP P, Soit $\Upsilon_{W^1_{A,P}}$, …, $\Upsilon_{W^m_{A,P}}$ les goupes representés par chacun des instances du modele dans $Patts_A = \{W^1_A, \ldots, W^m_A\}$respectivement, calculés par LDSB a l'affectation A. alors l'ensemble des literaux s l_A=composed_orbit (x = d, $Patts_A$) est un sous ensemble de l'ensemble Orbit(x = d, [SetStab(A, $\Upsilon_{W^1_{A,P}}$), …, SetStab(A, $\Upsilon_{W^m_{A,P}}$)]).

Avec la composition, dans la brnche droite x ≠ d de chaque nœud de recherche A, DSB doit executr $O(|Patts_A| \times |L_A \cup \{x = d\}|)$ travail. Ou $|Patts_A|$ represente la some de toutes les tailles des instances du modèle dans $Patts_A$ et $|L_A|$ est l'ensmeble des literaux calculés par some_orbit(x = d, Patts A), ie l'ensemble des litéraux qui avec x = d vont etre éliminés.

- **Correctness de LDSB.**

Theorème : Pour un CSP P, Soit $Patts_A = \{W^1, \ldots, W^m\}$ l'ensemble des instances modèle pour P, $W^1_A, \ldots, W^m_A$ les instances modèles calculés par LDSB a partir de $W^1, \ldots, W^m$, respectivement, a toute affectation A, et $\Upsilon_{W^1_{A,P}}, \ldots, \Upsilon_{W^m_{A,P}}$ est le groupe représenté par chacun des instances modelé associées. L'appel a la fonction search(x =d, Patts) va supprimer a chaque branche droite x = d au nœud associé a A le sous ensemble des littéraux dans Orbit(x = d, [SetStab(A, $\Upsilon_{W^1_{A,P}}$), …, SetStab(A, $\Upsilon_{W^m_{A,P}}$)].

La preuve de ce théorème se déduit directement à partir des théorèmes précédents.

Application de LDSB sur le CTTP :

Pour la symétrie Z2, nous avons appliqué le quatrième type définit par LDSB (ie. l'interchangeabilité de séquences de variables. Dans notre cas il s'agit de l'interchangeabilité entre colonnes selon la définition de Z2 dans le modèle matriciel). Pour Dn, c'est l'utilisation du type d'interchangeabilité entre variables précises dans un slot de l'équipe).

Un autre traitement ajouté et qui doit supprimer plusieurs symétries dans le modèle, qui consiste à considérer une contrainte supplémentaire qui représente une restriction sur l'équipe 1 qui doit jouer away dans le premier round et sa distance de parcours soit la plus petite dans ce round.et elle joue toujours contre les équipes dans l'ensemble 2..n/2 (i.e., l'équipe 1elle voyage selon l'horloge).

5.4 Conclusion

Dans ce chapitre nous avons présenté le problème de compétition sportive C_TTP, ses différentes contraintes, et nous avons fait une étude approfondie des différentes symétries que présente le problème, nous avons présenté les formes de symétries du problème et nous avons montré que le nombre de symétries augmente avec la taille du problème (le nombre d'équipes et ou le nombre de sites sur le cercle). Nous avons contribué à la résolution de ce problème par une méthode de recherche énumérative dite Branch and Bound à laquelle nous avons intégré une méthode de suppression de symétries dynamique LDSB, qui théoriquement semble la plus efficace pour la suppression de toutes les formes de symétries présentés dans le C_TTP. Le prochain chapitre consistera à l'implémentation de l'approche proposée.

6 Expérimentation

6.1 Introduction

Dans ce chapitre nous allons présenter l'implémentation de la méthode (*Branch and Bound-LDSB*) proposée dans le chapitre précèdent sur les instances circulaires du TTP. Le chapitre est divisé comme suit : nous commençons par présenter le système et l'environnement du travail sur lequel nous avons implémenté notre méthode ainsi que les benchmarks, nous présentons certains pseudo code et nous terminons par présenter et discuter les résultats obtenus.

6.2 Calcul, environnement de travail et benchmarks

Nous avons testé notre méthode de réduction de la symétrie sur les différentes variantes des instances de TTP circulaires, à savoir les cas limité, où le nombre de jeux Home/ Away consécutives est bornée par trois, ainsi que les cas sans contrainte (avec ou sans CNR).

L'algorithme a été codé en C ++, compilé en mode release avec MS-Visual studio Express C ++ 2012. La plupart des essais ont été effectués sur un PC standard avec processeur Intel Core I5 CPU avec 2.76 GHz, 4 Go de mémoire principale, sur Windows 7.

De manière générale, l'un des facteurs les plus importants dans la résolution des problèmes combinatoires est l'environnement d'exécution de ces derniers (environnement matériel et logiciel) qui a un grand impact sur les paramètres (temps d'exécution et espace mémoire), donc avant de procéder à résoudre un problème x donné, il est nécessaire de bien choisir ses outils, il faut noter qu'il a été montrer que la résolution d'un mêmeproblème avec des outils différents et sur des solveurs différents à donner également des résultats différents et dans des délais d'exécution différents [157, 158].

6. Expérimentation.

Description de l'outil des contraintes Gecode 

Gecode (GenericConstraintDevelopmentEnvironment) est un outil pour développer des systèmes et des applications basées sur les contraintes. Avec Gecode on peut développer une application de contraintes modulaire et extensible.

- **Caractéristiques :**
  - ✓ **Générique**: elle doit supporter l'exécution de tous les problèmes.
  - ✓ **Extensible** : elle donne la possibilité de modifier le problème en rajoutant par exemple des contraintes ou en ajoutant des modules pour l'exécution sans avoir à refaire tous l'algorithme ou tout le modèle (pour cela elle doit être modulaire).

  - ✓ ***Ouvert :***Gecode est un système ouvert pour la programmation: il peut être facilement relié à d'autres systèmes. Il prend en charge la programmation de nouvelles contraintes, les stratégies de branchement, et les moteurs de recherche. Les nouveaux domaines de variables peuvent être programmés au même niveau d'efficacité que les variables prédéfinis dans Gecode.
  - ✓ ***Complet :***Gecode dispose d'un ensemble complet de fonctionnalités:
    - les contraintes sur les entiers, les booléens, les ensembles, et les réels (il met en œuvre plus de 70 contraintes du catalogue de contrainte globale et d'autres contraintes additionnels non définis dans le catalogue globale des contraintes);
    - Une couche de modél[4]isation C + +;
    - Des heuristiques de branchements avancés,
    - de nombreux moteurs de recherche (parallèles et interactives graphiquement, redémarrage);
    - Il supporte le langage de modélisation MiniZinc,
    - et d'autres caractéristiques.
  - ✓ ***Efficacité :***Il offre d'excellentes performances en ce qui concerne le temps d'exécution et l'utilisation de l'espace mémoire. Il a remporté

[4]FlatZinc est une langue d'entrée au solveur de bas niveau qui représente la langue cible pour MiniZinc. Il est conçu pour être facile à traduire dans la forme requise par un solveur.

toutes les médailles d'or dans toutes les catégories sur les défis de MiniZinc '*the MiniZinc Challenges*' [159].

- ✓ ***libre :***Gecode est distribué sous la licence MIT et est répertorié comme un logiciel libre par la FSF.
- ✓ ***Portable:*** Il est implémenté en C + + qui suit attentivement la norme C + +. Il peut être compilé avec des compilateurs C + + modernes et fonctionne sur une large gamme de machines (y compris les machines 64 bits).
- ✓ ***Parallèle :***Gecode compile avec exploitation des cœurs multiples du matériel de base d'aujourd'hui pour la recherche parallèle, ce qui donne un système de base efficace, déjà c'est un atout supplémentaire.
- ✓ ***Testé :*** Il utilise une suite de tests avec près de 50000 différents cas de test atteignant une couverture de test de près de 100%.

- ***La version installée* :** Gecode4.2.0.
- ***Liens de la page principale de Gecode* [**160**].**
- ***Référence* :** Catalogue des solveurs de contraintes [161].

Instances et Benchmarks :

On va tester notre méthode sur différentes variantes des instances TTP circulaires,

Les instances benchmarks TTP sont sur le site : http://mat.tepper.cmu.edu/TOURN.

6.3 Discussion et analyse des résultats

	Instances avec contraintes		Instances sans contraintes avec NRCs		Instances sans contraintes sans NRCs	
Instance	Temps de calcul	Sol. Opt	Temps de calcul	Sol. Opt	Temps de calcul	Sol. Opt
Circ 4	0s	20	0s	20	0s	20
Circ 6	1.6s	64	2.4s	58	2s	54
Circ 8	89s	140	224280s	110	189182s	100

Table 4 Temps d'exécution pour les instances circulaires du TTP (avec contraintes sans contraintes avec NRCs sans contraintes sans NRCs).

Notre première observation est que le cout de traitement des symétries et de traiter la composition des différents formes de symétries du C_TTP est négligeable.

Notre résultat le plus important est que toutes les réductions de symétrie réduit le temps de calcul de l'algorithme branch-and-bound significativement. Le tableau 1 résume les résultats concernant les instances avec contraintes et les instances sans contraintes (avec ou sans NRCs). Pour chaque cas, il indique le temps de calcul global et la solution optimale trouvée.

Avec la complexité croissante des instances nous observons une accélération croissante en temps d'exécution.

Les résultats suggèrent que des accélérations similaires peuvent également être obtenues pour de plus grandes instances (si nous sommes en mesure de les résoudre).

Une autre observation est que Le modèle 1-Factor donne de meilleurs résultats pour les instances sans contraintes dans le cas où le modèle Home/ away est le mieux pour les instances avec contraintes.

En comparant les instances circulaires avec contraintes aux instances sans contraintes,

On remarque que les dernières sont considérablement lourdes à résoudre. Cependant, avec les méthodes de réduction de symétries nous avons obtenu des résultats moyennement bons avec la représentation des 1-Factor.

6.4 Conclusion

Dans ce chapitre, nous avons fourni un aperçu de la nature de la symétrie inhérente dans les cas du TTP circulaire.

Le groupe symétrique est Gn = Dn × Z2, qui est le produit cartésien du groupe dièdre d'ordre 2n (n est le nombre d'équipes) et le groupe d'ordre 2. Nous avons montré que Gn opère sur les paires d'affectations home/ away et peut réduire l'espace de recherche par un facteur de 4n par approximation lorsqu'il s'agit d'une suppression complète. Nous avons considéré une suppression partielle de symétries dont l'avantage est la suppression de grand nombre de solutions symétriques et dont le traitement des réductions des symétries et des compositions de différentes formes de symétries présentés soit négligeable.

6. Expérimentation.

En utilisant une approche branch-and-bound nous avons démontré que les vitesses augmentent avec le nombre n des équipes. Une analyse complète pour plus d'équipes est impossible en raison des grands temps de calcul, mais les résultats empiriques suggèrent que la réduction de la symétrie peut conduire à des accélérations qui atteignent approximativement les environs de 3n pour les plus grandes instances. Pour le modèle des 1-facteurs (impliqués par les matchs d'un slot de temps), l'opération de Z2 est exploitée efficacement dans l'approche branch-and-bound. Dans ce cas, les accélérations semblent rapprocher un facteur de 3n.

7 Conclusion Générale

Dans ce mémoire, nous nous sommes penchés sur l'intérêt de l'exploitation des symétries dans la résolution des problèmes combinatoires. Cette propriété a été étudiée dans des domaines divers tel que la programmation logique et la programmation par contraintes, donc, nous avons présenté l'exploitation de la symétrie dans les deux domaines, ou nous avons commencé par présenter les différents concepts de la symétrie ainsi que la relation de cette dernière avec la théorie du groupe, en effet les symétries diffères d'un problème a un autre. Donc, chaque problème peut admettre des types particuliers de symétrie, nous avons vu que parfois la symétrie peut être introduite par le modèle choisi pour modéliser le problème considéré. Par la suite nous avons étudié et analyser les différents approches et travaux faits dans ce domaine, ou chaque approche présente des avantages et des inconvénients, nous avons étudié aussi la possibilité de combiner ces méthodes. Dans tous les cas, l'exploitation de la symétrie s'avère être efficace de point de vue du temps d'exécution même pour de grandes instances de problèmes. Cependant, lorsque le traitement des symétries devient coûteux (par exemple le cas où le nombre de contraintes de suppression de symétries statiques ajoutés est exponentiel), des solutions ont été proposés dans la littérature qui visent une réduction partielle des symétries. Dans ce travail, nous avons considéré le problème de compétition sportive Circular traveling tournament problem (C-TTP). Ou nous avons cherché à améliorer sa résolution en exploitant les symétries présentes dans les instances circulaires du problème. Pour réaliser cet objectif, nous avons fait le choix de combiner une méthode complète Branch and Bound minimisant la distance totale parcourue par les équipes avec une méthode de suppression de symétrie partielle dynamique dite LDSB. Ce choix est justifié par le nombre d'avantages que présente la méthode (Efficacité, Rapidité, Flexibilité, sa compatibilité avec les autres algorithmes tel que les heuristiques de recherche, le temps ajouté pour le traitement de la symétrie est presque négligeable). Les résultats obtenu par l'implémentation de cette méthode sont prometteurs. Sauf lorsque la taille de l'instance augmente le calcul devient complexe a cause de l'utilisation d'une méthode de résolution complète.

***Perspectives* :**

7. Conclusion Générale.

- **Sur les types de symétries à considérer :** nous allons étudier la possibilité de trouver des symétries internes au problème considéré afin d'exploiter à la fois les symétries internes aux solutions et les symétries entre solutions. Nous pensons que cela va aéliorer la résolution du problème.
- **Sur les techniques à utiliser :** nous avons vu que malgré l'efficacité de la méthode proposée, l'utilisation d'une méthode de résolution complète empêche la résolution de grands instances, pour palier à ce problème, nous optons (dans le futur) pour une utilisation des méthodes approchés comme les heuristiques de recherche.

8 Bibliographie

[1] C. Papadimitriou. Computational Complexity, Addison Wesley, 1994.

[2] P. Cheeseman, B. Kanelfy, and W. T aylor, Where the really hard problems are, In Proceedings of IJCAI'91, Morgan Kaufmann, Sydney , Australia, pages 331–337, 1991.

[3] T . Hogg. Refining the phase transition in combinatorial search, Artificial Intelligence, pages 127–154, 1996.

[4] E. MacIntyre, P.Prosser, B.Smith, and T.Walsh, Random constraints satisfaction: theory meets practice. In CP98, LNCS 1520, pages 325–339, Springer Verlag, Berlin, Germany, 1998.

[5] A. Davenport. A comparison of complete and incomplete algorithms in the easy and hard regions, In Proceedings of CP'95 workshop on Studying and Solving Really Hard Problems, pages 43–51, 1995.

[6] D.A. Clark, J. Frank, I.P . Gent, E. MacIntyre, N. T omv , and T . Walsh. Local search and the number of solutions, In Proceedings of CP'96, LNCS 1118, Springer V erlag, Berlin, Germany, pages 119–133, 1996.

[7] I.P . Gent, E. MacIntyre, P . Prosser, and T. Walsh, The constrainedness of search. In Proceedings of AAAI-96, AAAI Press, Menlo P ark, California, 1996.

[8] T . Hogg, Exploiting problem structure as a search heuristic, Intl. J. of Modern Physics C, 9:13–29, 1998.

[9] T . Asselmeyer H. Rosé, W. Ebeling, Density of states-a measure of the difficulty of optimization problems, In Proceedings of PPSN'96, pages 208–217, LNCS 1141, Springer Verlag, 1996.

[10] P.F. Stadler. T owards a theory of landscapes, In Complex Systems and Binary Networks, volume 461, pages 77–163, Springer Verlag, 1995.

[11] C. Fonlupt, D. Robilliard, P . Preux, and E. Talbi, Fitness landscape and performance of meta-heuristics, 1999.

[12] T . Jones and S. Forrest, Fitness distance correlation as a measure of problem difficulty for genetic algorithms, In Proceedings of International Conference on Genetic Algorithms, Morgan Kaufmann, Sydney , Australia, pages 184–192, 1995.

[13] P . Merz and B. Freisleben, Fitness landscapes and memetic algorithm design, In D. Corne, M. Dorigo, and F . Glover, editors, New Ideas in Optimization, pages 245–260. McGraw Hill, UK, 1999.

[14] M.R. Garey and D.S. Johnson, A guide to the theory of NP-Completeness, Freeman, 1979.

8. Bibliographie.

[15] E. Freuder and R. Wallace. Partial constraint satisfaction, Artificial Intelligence, 58 :21–70, 1992.

[16] T . Shiex, H. Fargier, and G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, In Proceedings of IJCAI-95, MIT Press, Cambridge, MA, pages 631–637, 1995.

[17] S. Bistarelli, U. Montanari, and F . Rossi, Semiring-based constraint satisfaction and optimization, J. ACM, 44(2) :201–236, 1997.

[18] I.P . Gent and T . Walsh. Csplib : a benchmark library for constraints, Technical report, APES-09-1999, 1999.available from http ://csplib .cs.strath.ac.uk/. A shorter version appears in CP99.

[19] V . Chvatal. Linear Programming, W.H. Freeman and Co, 1983.

[20] E.P .K. Tsang. Foundations of Constraint Satisfaction. Academic Press, London, UK, 1993.

[21] F. Fages. Programmation Logique par Contraintes, Collection Cours de l'Ecole Polytechnique. Ed. Ellipses, 1996.

[22] K. Marriott and P .J. Stuckey, Programming with Constraints : an Introduction, MIT Press, 1998.

[23] M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization : A critical survey, Annals of Operations Research, 131 :373–395, 2004.

[24] R. Battiti and M. Protasi. Reactive local search for the maximum clique problem. Algorithmica, 29(4):610–637, 2001.

[25] B. Neveu, G. Trombettoni, and F. Glover, Id walk : A candidate list strategy with a simple diversification device. In Proceedings of CP'2004, volume 3258 of LNCS, pages 423–437, Springer Verlag, 2004.

[26] N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuris-tics. Artificial Intelligence, 139, 2002.

[27] C. Pralet and G. Verfaillie, Travelling in the world of local searches in the sapce of partial as-signments, In Proceedings of CP-AI-OR 2004, volume 3011 of LNCS, pages 240–255. Springer Verlag, 2004.

[28] B. Estellon, F . Gardi, and K. Nouioua, Ordonnancement de véhicules : une approche par recherche locale à grand voisinage, In Actes des premières Journées Francophones de programmation par Contraintes (JFPC), pages 21–28, 2005.

[29] P . Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts : T owards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology , Pasadena, California, USA, 1989.

8. Bibliographie.

[30] L. Pitsoulis and M. Resende, Handbook of Applied Optimization, chapter Greedy randomized adaptive search procedures, pages 168–183, Oxford University Press, 2002.

[31] D.B. Shmoys. Computing near-optimal solutions to combinatorial optimization problems, DI-MACS Series in Discrete Mathematics and Theoretical Computer Science, 20: 355–397, 1995.

[32] B. Krishnamurty, Short proofs for tricky formulas, Acta information, (22):253-275.1985.

[33] B. Benhamou. Etude des symétries et de la cardinalité en Calcul propositionnel, 1993.

[34] 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.

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

[36] B. Benhamou, T. Nabhani, R. Ostrowski, S. Mohamed Réda, Détection et élimination dynamique de la symétrie dans le problème de satisfiabilité, 2010.

[37] T. Nabhani, Symétries locales et globales en logique propositionnelle et leurs extensions aux logiques non monotones, thèse de doctorat, 09 décembre 2011.

[38] Cook, S. A, The complexity of theorem proving procedures, In *STOC*, pages 151–158, 1971.

[39] James Crawford, Matthew L. Ginsberg, Eugene Luck, and Amitabha Roy, Symmetry breaking predicates for search problems. In *KR'96 : Principles of Knowledge Representation and Reasoning*, pages 148–159, Morgan Kaufmann, San Francisco, California, 1996.

[40] F. A. Aloul, A.Ramani, I. L. Markov, and K. A, Sakallak. Solving difficult sat instances in the presence of symmetry, *In IEEE Transaction on CAD, vol. 22(9)*, pages 1117–1137, 2003.

[41] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallak, Symmetry breaking for pseudo boolean satisfiabilty. *In ASPDAC'04*, pages 884–887, 2004. *****

[42] Genevieve Bossu and Pierre Siegel, Nonmonotonic reasoning and databases, In Advances in Data Base Theory, pages 239–284, 1982.

[43] B. Benhamou, T. Nabhani, P. Siegel, Symétries dans les logiques non monotones, 2010.

[44] B. Benhamou, T. Nabhani, R. Ostrowski, S. Mohamed, Amélioration de l'apprentissage des clauses par symétrie dans les solveurs SAT, 2010.

[45] Genevieve Bossu and Pierre Siegel. Saturation, nonmonotonic reasoning and the closed-world assumption. Artif. Intell., 25(1) :13–63, 1985.

8. Bibliographie.

[46] Yoav Shoam. A semantical approach to nonmonotonic logic. In IJCAI, pages 388–392, 1987.

[47] P. Besnard and P. Siegel. The preferential-models approach in nonmonotonic logics - in non-standard logic for automated reasoning. In Academic Press, pages 137–156. ed. P. Smets, 1988.

[48] Sarit Kraus, Daniel J. Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. Artificial Intelligence, 44(1- 2):167–207, 1990.

[49] P. Siegel, L. Forget, and V. Risch. Preferential logics are x-logics. Journal of Logic and Computation, 11(1):71–83, 2001.

[50] Ray Reiter. A logic for default reasoning. Artificial Intelligence, pages 81–132, 1980.

[51] Aggoun A. et Beldiceanu N. Extending chip in order to solve complex scheduling and placement problems, Mathematical Computing and Modelling , 17(7) :57/ 73, 1993.

[52] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey and Thierry. Global Constraint Catalog: Past, Present and Future *Constraints* Volume 12, Issue1: pp 21-62. Journal**,** 2007.

[53] Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1968.

[54]Gurari, Eitan. Backtracking algorithms "CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms". 1999.

[55] J.C. Régin, A filtering algorithm for constraints of difference in CSPs, in: Proceedings of the 12th National Conference on Artificial Intelligence AAAI-1994. AAAI Press, pp. 362–367, 1994.

[56] Bessiere et als, An Optimal Coarse-grained Arc Consistency Algorithm, (description de AC-2001/3.1), 2005.

[57] Prosser, Patrick. "Hybrid Algorithms for the Constraint Satisfaction Problem". Computational Intelligence 9(3), 1993.

[58] Matthew L. Ginsberg. Dynamic Backtracking. Journal of Articial Intelligence Research 1, 25-46, 1993

[59] G. Katsirelos and F. Bacchus. Unrestricted nogood recording in CSP search. In Proceedings of CP'03, pages 873–877, 2003.

[60] DEMASSEY Sophie. Méthodes hybrides de programmation par contraintes et de programmation linéaire pour le problème d'ordonnancement de projet à contraintes de ressources.

8. Bibliographie.

[61] B. M. Smith, K. E. Petrie, and I. P. Gent. Models and symmetry breaking for 'peaceable armies of queens'. In J.-C. R´egin and M. Rueher, editors,CPAIOR, volume 3011 ofLecture Notes in Computer Science, pages 271–286. Springer, 2004. ISBN 3-540-21836-X.

[62] C. A. Brown, L. Finkelstein, and P. W. Purdom. Backtrack Searching in the Presence of Symmetry. In T. Mora, editor,Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, LNCS 357, pages 99–110. Springer, 1988. [34] 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.

[63] E. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. InAAAI-91, pages 227–233, 1991.

[64] B. Benhamou. Study of symmetry in constraint satisfaction problems. InProceed-ings of the 2nd workshop on Principles and Practices of Constraint Programming -PPCP'94, pages 246–254, 2003.

[65] A. Lal, B. Y. Choueiry, and E. C. Freuder. Neighborhood interchangeability and dynamic bundling for non-binary finite CSPs. In M. M. Veloso and S. Kambhampati, editors,AAAI, pages 397–404. AAAI Press AAAI Press / The MIT Press, 2005. ISBN 1-57735-236-X.

[66] I. P. Gent, P. Nightingale, and K. Stergiou. Qcsp-solve: A solver for quantified constraint satisfaction problems. In Kaelbling and Saffiotti [68], pages 138–143.

[67] J.-F. Puget. On the satisfiability of symmetrical constraint satisfaction problems. In Methodologies for Intelligent Systems (Proceedings of ISMIS'93), LNAI 689, pages 350–361. Springer, 1993.

[68] P. Roy and F. Pachet. Using Symmetry of Global Constraints to Speed up the Resolution of Constraint Satisfaction Problems. InWorkshop on Non Binary Constraints, ECAI-98, August 1998.

[69] P. Meseguer and C. Torras. Exploiting Symmetries within Constraint Satisfaction Search.Artificial Intelligence, 129:133–163, 2001.

[70] The GAP Group. GAP – Groups, Algorithms, and Programming, Version 4.4.7, 2006. http ://www.gap-system.org. Cité p. 14, 50

[71] J. Molony. Symmetry and Complexity in Propositional Reasoning. PhD thesis, University of Edinburgh, 1999.

[72] A. Smaill. Symmetry in boolean constraints: Some complexity issues. In Tenth Workshop on Automated Reasoning, 2003.

8. Bibliographie.

[73] B. M. Smith. Reducing Symmetry in a Combinatorial Design Problem. Technical report, School of Computer Studies, University of Leeds, Jan. 2001.

[74] C. Jefferson and A. M. Frisch. Representations of sets and multisets in constraint programming. In B. Hnich, P. Prosser, and B. Smith, editors,The Fourth International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, pages 102–116, 2005.

[75] I. P. Gent, T. Kelsey, S. Linton, I. McDonald, I. Miguel, and B. M. Smith. Conditional symmetry breaking. In van Beek [118], pages 256–270. ISBN 3-540-29238-1.

[76] I. Gent, I. McDonald, and B. Smith. Conditional symmetry in the all-interval series problem. In B. Smith, I. Gent, and W. Harvey, editors,Proceedings of the Third International Workshop on Symmetry in Constraint Satisfaction Problems, pages 55–65, 2003.

[77] C. M. Roney-Dougal, I. P. Gent, T. Kelsey, and S. Linton. Tractable symmetry breaking using restricted search trees. In de M´antaras and Saitta [25], pages 211–215. ISBN 1-58603-452-9.

[78] P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. van Hentenryck, editor,Principles and Practice of Constraint Programming - CP 2002, LNCS 2470, pages 462–476. Springer, 2002.

[79] L. G. Proll and B. M. Smith. ILP and Constraint Programming Approaches to a Template Design Problem.INFORMS Journal on Computing, 10:265–275, 1998.

[80] J. Crawford, M. L. Ginsberg, E. Luks, and A. Roy. Symmetry-breaking predicates for search problems. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, KR'96: Principles of Knowledge Representation and Reasoning, pages 148–159. Morgan Kaufmann, San Francisco, California, 1996.

[81] F.A.Aloul, K.A.Sakallah, andI.L.Markov. Efficient symmetry breaking for boolean satisfiability. In Gottlob and Walsh [57], pages 271–276.

[82] I. P. Gent, W. Harvey, and T. Kelsey. Groups and Constraints: Symmetry Breaking During Search, 2002.

[83] E. Luks and A. Roy. The complexity of symmetry-breaking formulas. Ann. Math. Artif. Intell., 41(1):19–45, 2004.

[84] A. M. Frisch and W. Harvey. Constraints for breaking all row and column symmetries in a three-by-two matrix. InProceedings of the Third International Workshop on Symmetry in Constraint Satisfaction Problems",, September 2003.

[85] J.-F. Puget. Breaking symmetries in all-different problems. InIJCAI, pages 272–277, 2005.

8. Bibliographie.

[86] I. J. Lustig and J. F. Puget. Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming. INTERFACES, 31 (6):29–53, 2001.

[87] K. E. Petrie and B. M. Smith. Symmetry breaking in graceful graphs. In Rossi [104], pages 930–934. ISBN 3-540-20202-1.

[88] J.-F. Puget. Breaking symmetries in all-different problems. InIJCAI, pages 272–277, 2005.

[89] A. Seress. Permutation group algorithms. Number 152 in Cambridge tracts in mathematics. Cambridge University Press, 2002.

[90] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah. Solving difficult sat instances in the presence of symmetry. InDAC, pages 731–736. ACM, 2002. ISBN 1-58113-461-4.

[91] I. Shlyakhter. Generating effective symmetry-breaking predicates for search problems. Discrete Applied Mathematics, To appear. Earlier version presented at the SAT 01 Workshop.

[92] I. P. Gent, K. E. Petrie, and J.-F. Puget, Symmetry in constraint programming, in Handbook of Constraint Programming, F. Rossi, P. van Beek, and T. Walsh, eds., Elsevier, 2006, pp. 329–376.

[93] A. M. Frisch and W. Harvey, Constraints for breaking all row and column symmetries in a three-by-two matrix, in Proceedings of the Third International Workshop on Symmetry in Constraint Satisfaction Problems (SymCon ’03), 2003.

[94] A. Grayland, C. Jefferson, I. Miguel, and C. Roney -Dougal, Minimal ordering constraints for some families of variable symmetries, Annals of Mathematics and Artificial Intelligence, 57 (2009), pp. 75–102.

[95] Katsirelos, G., and Walsh, T. 2010. Symmetries of symmetry breaking constraints. In Proc. of the 19th European Conf. on Artificial Intelligence (ECAI-2010). IOS Press.

[96] Huczynska, S.; McKay, P.; Miguel, I.; and Nightingale, P. 2009. Modelling equidistant fre-quency permutation arrays: An application of constraints to mathematics. In Gent, I. P., ed., Proc. of 15th International Conf. on Principles and Practice of Constraint Program-ming (CP 2009), LNCS 5732, 50–64. Springer.

[97] Flener, P.; Frisch, A.; Hnich, B.; Kizil-tan, Z.; Miguel, I.; Pearson, J.; and Walsh, T. 2001a. Symmetry in matrix models. Technical Report APES-30-2001, APES group. Presented at SymCon’01 (Symmetry in Con-straints), CP2001 post-conference workshop.

[98] Flener, P.; Frisch, A.; Hnich, B.; Kizil-tan, Z.; Miguel, I.; and Walsh, T. 2001b. Matrix modelling. Technical Report APES-36-2001, APES group. Presented at Formul’01 Workshop on Modelling and Problem Formula-tion, CP2001 post-conference workshop.

8. Bibliographie.

[99] Flener, P.; Frisch, A.; Hnich, B.; Kizil-tan, Z.; Miguel, I.; Pearson, J.; and Walsh, T. 2002. Break-ing row and column symmetry in matrix models. In 8th International Conf. on Principles and Practices of Constraint Programming (CP-2002). Springer.

[100] Katsirelos, G.; Narodytska, N.; and Walsh, T. 2010. On the complexity and completeness of static constraints for breaking row and column symmetry. In Cohen, D., ed., Proc. of the 16th International Conf. on the Principles and Practice of Constraint Programming (CP 2010), LNCS 6308, 305–320. Springer.

[101] Narodytska, N., and Walsh, T. 2012. Beyond lex leader: Breaking symmetry with other orderings. Under review for Proc. of the 18th International Conf. on the Principles and Practice of Constraint Program-ming (CP 2012), LNCS. Springer.

[102] Grayland, A.; Miguel, I.; and Roney-Dougal, C. 2009. Snake lex: An alternative to double lex. In Gent, I. P., ed., Proc. of 15th In-ternational Conf. on Principles and Practice of Constraint Programming, LNCS 5732, 391–399. Springer.

[103] Puget, J.F.: Breaking all value symmetries in surjection problems. In van Beek, P., ed.: Proceedings of 11th International Conference on Principles and Practice of Constraint Pro-gramming (CP2005), Springer (2005).

[104] Law, Y., Lee, J.: Global constraints for integer and set value precedence. In: Proceedings of 10th International Conference on Principles and Practice of Constraint Programming (CP2004), Springer (2004) 362–376.

[105] Walsh, T.: Symmetry breaking using value precedence. In Brewka, G., Coradeschi, S., Perini, A., Traverso, P., eds.: ECAI 2006, IOS Press (2006) 168–172.

[106] Puget, J.-F. 2005b. Breaking all value symmetries in surjection problems. In van Beek, P., ed., Proc. Of 11th International Conf. on Principles and Practice of Con-straint Programming (CP2005). Springer.

[107] Sellmann, M., and Hen-tenryck, P. V. 2005. Structural symmetry breaking. In Proc. of 19th International Joint Conf. on AI (IJCAI 2005), 298–303. IJCAI.

[108] Flener, P.; Pearson, J.; Sellmann, M.; and Hentenryck, P. V. 2006. Static and dynamic structural sym-metry breaking. In Proc. of 12th International Conf. on Prin-ciples and Practice of Constraint Programming (CP2006). Springer.

[109] Law, Y.-C.; Lee, J.; Walsh, T.; and Yip, J. 2007. Breaking symmetry of interchangeable variables and values. In 13th International Conf. on Principles and Prac-tices of Constraint Programming (CP-2007). Springer.

[110] R. Backofen and S. Will. Excluding symmetries in concurrent constraint programming. InWorkshop on Modeling and Computing with Concurrent Constraint Programming held in conjunction with CP 98, 1998.

[111] R. Backofen and S. Will. Excluding symmetries in constraint-based search. In J. Jaffar, editor,Principles and Practice of Constraint Programming - CP ’99,vol-ume Lecture Notes in Computer Science 1713, pages 73–87. Springer, 1999.

8. Bibliographie.

[112] I. P. Gent and B. M. Smith. Symmetry breaking in constraint programming. In Proceedings of European Conference on Artificial Intelligence - ECAI 2000,pages 599–603. IOS press, 2000.

[113] I. Gent, S. Linton, and B. Smith. Symmetry breaking in the alien tiles puzzle. Technical Report APES-22-2000, APES Research Group, October 2000. Available from http://www.dcs.st-and.ac.uk/˜apes/apesreports.html.

[114] I. McDonald and B. Smith. Partial symmetry breaking. In P. V. Hentenryck, editor, Principles and Practice of Constraint Programming, pages 431–445, 2002.

[115] F.Focacci and M.Milano. Global cut framework for removing symmetries. In T. Walsh, editor,Principles and Practice of Constraint Programming - CP 2001, volume Lecture Notes in Computer Science 2239, pages 77–92. Springer, 2001.

[116] T. Fahle, S. Schamberger, and M. Sellmann. Symmetry breaking. In T. Walsh, editor, Principles and Practice of Constraint Programming - CP 2001, volume Lecture Notes in Computer Science 2239, pages 93–107. Springer, 2001.

[117] J.-F. Puget. Symmetry breaking revisited. Constraints, 10(1):23–46, 2005.

[118] M. Sellmann and P. V. Hentenryck. Structural symmetry breaking. In Kaelbling and Saffiotti [68], pages 298–303. ISBN 0938075934.

[119] I. P. Gent, W. Harvey, T. Kelsey, and S. Linton. Generic SBDD Using Computational Group Theory. In F. Rossi, editor, Principles and Practice of Constraint Programming - CP2003, LNCS 2833, pages 333–347. Springer, 2003.

[120] W. Harvey. Symmetry Breaking and the Social Golfer Problem. In Proceedings SymCon-01: Symmetry in Constraints, pages 9–16, 2001.

[121] Ian P. Gent, Warwick Harvey, and Tom Kelsey. Groups and constraints: Symmetry breaking during search. In Van Hentenryck [29], pages 415–430.

[122] Ian P. Gent, Warwick Harvey, Tom Kelsey, and Steve Linton. Generic SBDD using computational group theory. In Rossi [27], pages 333–347.

[123] The GAP Group. GAP – Groups, Algorithms, and Programming, Version 4.4.9, 2006.

[124] P. V. Hentenryck, P. Flener, J. Pearson, and M. Agren. Tractable Symmetry Breaking for CSPs with Interchangeable Values. In International Joint Conference on Artificial Intelligence - IJCAI 2003, pages 277–282, 2003.

[125] Ian P. Gent and Barbara M. Smith. Symmetry breaking in constraint programming. In Werner Horn, editor, ECAI, pages 599–603. IOS Press, 2000.

[126] Jean-Francois Puget. Symmetry breaking using stabilizers. In Rossi [27], pages 585–599.

[127] Steve D. Prestwich, Brahim Hnich, Helmut Simonis, Roberto Rossi, and S. Armagan Tarim. Partial symmetry breaking by local search in the group. Constraints, 17:148–171, 2012.

8. Bibliographie.

[128] C.Mears, M.Garcia de la Banda, B. Demoen, M. Wallace, Lightweight dynamic symmetry breaking, article, 2013.

[129] Barnier, N., & Brisset, P. (2002). Solving the Kirkman's schoolgirl problem in a few seconds. In P. Van Hentenryck (Ed.),Proceedings of CP'02, LNCS(Vol. 2470. pp. 477–491). New York: Springer.

[130] Meseguer, P., & Torras, C. (2001). Exploiting symmetries within constraint satisfaction search. Artificial Intelligence, 129(1–2), 133–163.

[131] Puget, J.-F. (2002). Symmetry breaking revisited. In P. Van Hentenryck (Ed.),Proceedings of CP'02, LNCS(Vol. 2470, pp. 446–461). New York: Springer.

[132] Smith, B. M., Brailsford, S. C., Hubbard, P. M., & Williams, H. P. (1996). The progressive party problem: Integer linear programming and constraint programming compared.Constraints, 1, 119–138.

[133] Flener, P., Pearson, J., Sellmann, M., Van Hentenryck, P., Ågren, M.: Dynamic structural sym-metry breaking for constraint satisfaction problems. Constraints14(4), 506–538 (2009, Union and extension of [18] and [17])

[134] Flener, P., Pearson, J., Sellmann, M., & Van Hentenryck, P. (2006). Static and dynamic structural symmetry breaking. In F. Benhamou (Ed.),Proceedings of CP'06, LNCS(Vol. 4204, pp. 695– 699). New York: Springer.

[135] M. Heule, T.walsh, Symmetry within solutions, journal, 2010.

[136] "Scheduling a Major College Basketball Conference", a paper by George L. Nemhauser and Michael A. Trick published in 1997, available online at http://mat.gsia.cmu.edu/trick/acc.pdf

[**137**] K. Easton, G. Nemhauser, and M. A. Trick. The travelling tournament problem:
Description and benchmarks. In T. Walsh, editor, Principles and Practice of
Constraint Programming, volume 2239 of Lecture Notes in Computer Science, pages 580–585. Springer, Berlin, 2001.

[**146**] "Constraint Programming for the Travelling Tournament Problem", a paper by Gan Tiaw Leong from the National University of Singapore in 2002/03, found online at http://www.comp.nus.edu.sg/~henz/students/gan_tiaw_leong.pdf

[**138**] C. C. Ribeiro and S. Urrutia. Heuristics for the mirrored traveling tournament problem. European Journal of Operational Research, 179:775–787, 2007b.

[**139**] G. Dur´an, T.F. Noronha, C.C. Ribeiro, S. Souyris, and A. Weintraub. Branch-and cut for a real-life highly constrained soccer tournament scheduling problem. In E. Burke and H. Rudov´a, editors, Practice and Theory of Automated Timetabling VI, volume 3867 of Lecture Notes in Computer Science, pages 174–186. Springer, 2007b.

8. Bibliographie.

[**140**] K. G. Russell. Balancing carry-over effects in round robin tournaments. Biometrika, 67:127–131, 1980.

[**141**] C. C. Ribeiro and S. Urrutia. Scheduling the Brazilian soccer tournament with fairness and broadcast objectives. In E. K. Burke and H. Rudov´a, editors, Practice and Theory of Automated Timetabling VI, volume 3867 of Lecture Notes in Computer Science, pages 147–157. Springer, Berlin, 2007a.

[142] C. C. Ribeiro and S. Urrutia. Bicriteria integer programming approach for scheduling the Brazilian national soccer tournament. In Proceedings of the Third International Conference on Management Science and Engineering Management, pages 46–49, Bangkok, 2009.

[**143**] A. R. Duarte, C. C. Ribeiro, and S. Urrutia. A hybrid ILS heuristic to the referee assignment problem with an embedded MIP strategy. In T. Bartz-Beielstein, M. J. B. Aguilera, C. Blum, B. Naujoks, A. Roli, G. Rudolph, and M. Sampels, editors, Hybrid Metaheuristics, volume 4771 of Lecture Notes in Computer Science, pages 82–95. Springer, Berlin, 2007a.

[**144**] A. R. Duarte, C. C. Ribeiro, S. Urrutia, and E. H. Haeusler. Referee assignment in sports leagues. In E. K. Burke and H. Rudov´a, editors, Practice and Theory of Automated Timetabling VI, volume 3867 of Lecture Notes in Computer Science, pages 158–173. Springer, Berlin, 2007b.

[**145**] R. Duarte and C. C. Ribeiro. Referee assignment in sports leagues: Approximate and exact multi-objective aproaches. In 19th International Conference on Multiple Criteria Decision Making, pages 58–60, Auckland, 2008.

[**146**] "Constraint Programming for the Travelling Tournament Problem", a paper by Gan Tiaw Leong from the National University of Singapore in 2002/03, found online at http://www.comp.nus.edu.sg/~henz/students/gan_tiaw_leong.pdf

[**147**] M. A. Trick. Integer and constraint programming approaches for round robin tournament scheduling. In E. Burke and P. de Causmaecker, editors, Practice and Theory of Automated Timetabling IV, volume 2740 of Lecture Notes in Computer Science, pages 63–77, Berlin, 2003. Springer.

[**148**] M. A. Trick. Formulations and reformulations in integer programming. In R. Bartak and M. Milano, editors, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 3524 of Lecture Notes in Computer Science, pages 833–836. Springer, 2005.

[**149**] F. N. Costa, S. Urrutia, and C. C. Ribeiro. An ILS heuristic for the traveling tournament problem with predefined venues. Annals of Operations Research, to appear. doi: 10.1007/s10479-010-0719-9.

8. Bibliographie.

[**150**] R. A. Melo, S. Urrutia, and C. C. Ribeiro. The traveling tournament problem with predefined venues. Journal of Scheduling, 12:607–622, 2009.

[**151**] K. Easton, G. L. Nemhauser, and M. A. Trick. Solving the travelling tournament problem: A combined integer programming and constraint programming approach. In E. K. Burke and P. de Causmaecker, editors, Practice and Theory of AutomatedTimetabling IV, volume 2740 of Lecture Notes in Computer Science, pages 100–109, Berlin, 2003. Springer.

[**152**] Urrutia, C. C. Ribeiro, and R. A. Melo. A new lower bound to the traveling tournament problem. In Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling, pages 15–18, Honolulu, 2007. IEEE.

[**153**] R. V. Rasmussen and M. A. Trick. The timetable constrained distance minimization problem. In J. C. Beck and Smith B. M., editors, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 3990 of Lecture Notes in Computer Science, pages 167–181. Springer, 2006.

[**154**] K. K. H. Cheung. Solving mirrored traveling tournament problem benchmark instances with eight teams. Discrete Optimization, 5:138–143, 2008.

[155] S. Irnich. A new branch-and-price algorithm for the traveling tournament problem. European Journal of Operational Research, 204:218–228, 2010.

[156] D. C. Uthus, P. J. Riddle, and H. W. Guesgen. Solving the traveling tournament problem with iterative-deepening A*. Journal of Scheduling, 2011. To appear.

[157] Les conferences CP, *the 19th International Conference on Principles and Practice of Constraint Programming* (http://cp2013.a4cp.org/),

[158] Les conférences CP-AI-OR, http://www.minizinc.org/

[159] http://www.minizinc.org/challenge.html, 2008-2012.

[160] www.Gecode.org.

[161] http://openjvm.jvmhost.net/CPSolvers/