UNIVERSITÉ DE PAUL VERLAINE - METZ

ÉCOLE DOCTORALE IAEM Lorraine

Thèse

Présentée pour l'obtention du titre de

Docteur en Sciences de l'Université de Paul Verlaine - Metz

Mention : Informatique

par

Belaïd SAAD

Intégration des problèmes de satisfaction de contraintes distribués et sécurisés dans les systèmes d'aide à la décision à base de connaissances

Soutenue le 10 décembre devant le jury composé de :

M. Jean-Jacques CHABRIER	Rapporteur
M. Yves DEVILLE	Examinateur
Mme Francine HERRMANN	Examinatrice
M. Dieter KRATSCH	Directeur de thèse
M. Yann LANUEL	Examinateur
M. Lakhdar SAÏS	Rapporteur
M. Thomas SCHIEX	Président du jury
M. Thomas TAMISIER	Examinateur

Thèse préparée au sein du
Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine - Metz
Ile du Saulcy 57045 METZ CEDEX 1, France
et du
Centre de Recherche Public Gabriel Lippmann, Département Informatique,Systèmes et Collaboration
41, rue du Brill, L-4422 Belvaux, Luxembourg

Résumé

Any beginning has an end.

Une large gamme de problèmes pratiques nécessite une diversité de représentation et de modélisation des données et de développer des modèles dans lesquels les différentes données peuvent être représentées. Dans cette thèse, nous nous intéressons à l'hybridation de deux modèles : le modèle des Systèmes Experts (SE) et le modèle des Problèmes de Satisfaction de Contraintes (CSP). L'objectif de notre travail est de proposer un système distribué et sécurisé pour intégrer des contraintes dans un moteur d'inférence. Pour ce faire, nous avons d'une part développé un outil de communication qui facilite la collaboration entre SE et CSP. D'autre part, nous avons conçu un algorithme qui permet la communication entre plusieurs agents situés dans un environnement distribué. Enfin, un de nos buts, et non des moindres, est d'assurer la protection des données privées de chaque entité. La thèse est donc constituée de trois axes principaux :

Le premier axe vise l'élaboration d'une méthode de communication entre les deux modèles. Tout d'abord, nous décrivons une procédure de transformation automatique entre un système expert à base de règles vers un nouveau modèle de CSP dynamique nommé DDCSP (Dynamic Domain CSP) que nous avons au préalable défini. Cette procédure de transformation automatique permettra l'injection des résultats de l'un des deux modèles en entrée de l'autre. Cette procédure joue donc un rôle essentiel pour assurer la collaboration qui s'appuie sur l'échange d'informations.

Le deuxième axe est consacré à la prise en compte de la distribution d'un problème CSP sur plusieurs sites. Nous proposons un algorithme basé sur la notion de coopération et de parallélisme qui assure une résolution distribuée entre plusieurs agents. Notre approche consiste à construire un anneau d'agents autonomes, responsable chacun d'une partie des variables et des contraintes du problème. Chacun de ces agents va initier un processus qui explore une branche différente de l'arbre de recherche. Des heuristiques sont proposées pour garantir une diversification des explorations, en d'autres termes pour éviter que les branches explorées ne se recouvrent.

Enfin, nous présentons une technique de sécurisation de cet algorithme dans l'environnement distribué basée sur l'utilisation judicieuse des propriétés de cryptographie asymétrique pour préserver la confidentialité des instanciations. Afin d'effectuer une validation expérimentale de nos travaux, une implémentation dans les langages de programmation C/C++ ou Java est décrite dans chacun de ces trois axes.

Mots-clés : CSP, système expert, DisCSP, DCSP, système distribué, confidentialité.

Abstract

A wide range of practical problems requires a diversity of data representation and to develop models in which different data can be represented. In this thesis, we focus on the hybridization of two models : Expert System (ES) and Constraint Satisfaction Problem (CSP). The aim of our work is to propose a secure distributed system that allows to integrate constraints in an inference engine. To reach this goal, on one hand we developed a communication tool that facilitates collaboration between ES and CSP. On the other hand, we designed an algorithm that allows communication between multiple entities in a distributed environment. Finally, one of our goals, not the least, is to protect private data of each entity. The thesis is composed of three main axis.

The first priority is to develop a method of communication between the two models. First, we describe an automatic transformation procedure of the rule based expert system into the new dynamic CSP model called DDCSP (Dynamic Domain CSP) that we have designed. This procedure will automatically transform and inject the result of one of the two models as input to the other one. This process plays an essential role for collaboration based on the exchange of information.

Our second axe proposes an algorithm based on the concept of cooperation and parallelism which provides a resolution distributed among several entities. Our approach is to build a ring of autonomous agents, each responsible for some of the variables and constraints of the problem. Each of these agent will initiate a process that explores a different branch of the search tree. Heuristics are proposed to ensure a diversification of exploration, in other words to prevent overlapping of the explored branches.

Finally, we present a technique for securing this distributed algorithm based on a judicious use of the properties of asymmetric encryption to protect the confidentiality of instantiations. To perform an experimental validation of our work, an implementation in the programming languages C/C++ or Java is described in each of these three axis.

Keywords : CSP, expert system, DisCSP, DCSP, distributed system, privacy.

Remerciements

Je voudrais, tout d'abord, remercier tous les membres du jury : M. Jean-Jacques Chabrier et M. Lakhdar Saïs qui m'ont fait l'honneur d'être les rapporteurs de cette thèse, et qui m'ont permis d'améliorer la qualité de ce rapport par leurs remarques et commentaires précieux. Je remercie vivement M. Yves Deville et M. Thomas Schiex qui ont accepté de participer à mon jury.

Je remercie de même M. Stéphane Vialle pour l'intérêt qu'il a porté à mon travail et pour avoir accepté d'être le référent interne de cette thèse.

J'adresse également mes remerciements à mon directeur de thèse Dieter Kratsch pour sa disponibilité et ses conseils.

Je ne remercierais jamais assez Francine Herrmann pour tout ce qu'elle a fait pour moi durant ces années. Elle m'a fourni des conditions idéales pour mener à bien cette thèse. Elle est toujours parvenue à me consacrer du temps, et cela malgré sa charge de travail. Elle a su guider mon travail vers des voies que, sans elle, je n'aurais pas explorées, ce mémoire ne serait pas aussi abouti sans son aide.

Je remercie aussi mes co-encadrants Thomas Tamisier et Yann Lanuel pour leur disponibilité, leurs conseils, et leur soutien. Merci à tous les deux pour la relecture minutieuse de la thèse, leurs précieuses remarques et pour m'avoir accompagné tout au long de ces années.

J'adresse mes remerciements à Fernand Feltz directeur de département Informatique, Systèmes et Collaboration (ISC), CRP-Gabriel Lippmann d'avoir accepté de m'accueillir au sein de son département.

Je tiens à remercier tous les membres du département Informatique, Système et Collaboration, Centre de Recherche Public Gabriel Lippmann, Luxembourg, et plus particulièrement les membres du projet Cadral. Je remercie de même tous les membres du Laboratoire d'Informatique Théorique et Appliquée de l'Université de Paul Verlaine-Metz et en particulier M. Anass Nagih qui a toujours montré un réel intérêt à mes travaux et qui m'a donné maintes fois des conseils judicieux.

Merci aux amis Arezki, Assia, Boutiche, Brahim, Farida, Fide, Menouar, Sabrina, Samia, Séverine, Sonia, Toufik, Yasmine, Yagouni..., et la liste est loin d'être exhaustive pour leur soutien et leurs encouragements.

Une pensée affectueuse à ma famille qui m'a soutenu tout au long de mes études.

Enfin, je remercie les membres de ministère de l'Enseignement Supérieur et de la Recherche du Grand-Duché de Luxembourg, et les membres du FNR (Fonds National de la Recherche) Luxembourg pour le financement de cette thèse.

Dédicaces

À la mémoire de ma sœur Sadjia

Table des matières

1 Contexte de l'étude **4**

1.1 Les systèmes experts 5

1.1.1 Définition d'un système expert 5

1.1.2 Composants d'un système expert 6

1.1.3 Organisation des connaissances 8

1.1.4 Techniques de raisonnement 9

1.1.5 Quelques architectures ou langages de développement des systèmes experts 11

1.1.6 Limitations des systèmes experts 13

1.2 Les problèmes de satisfaction de contraintes 14

1.2.1 Représentation graphique d'un CSP binaire 15

1.2.2 Représentation des contraintes 16

1.2.3 Méthodes de résolution des problèmes de satisfaction de contraintes 17

1.2.4 Filtrage au cours de la résolution 18

1.2.5 Résolution parallèle des CSP 20

Évaluation des performances d'un algorithme parallèle . . 20

Parallélisation des algorithmes existants 21

Approche concurrente 21

Parallélisation et décomposition 22

1.3 Conclusion 22

2 Liens entre CSP et systèmes experts **23**

2.1 Renforcement des systèmes experts par des CSP 24

2.1.1 Le projet KRAFT 24

2.1.2 Utilisation des techniques de consistance dans les systèmes experts 25

2.1.3 Utilisation des techniques de consistance pour la validation de la base de connaissances 25

2.1.4 Les CSP^T [113] 26

2.2 CSP dynamiques (DCSP : Dynamic Constraint Satisfaction Problem) . . . 27
2.2.1 Les DCSP . . . 28
2.2.2 Les CCSP [128] . . . 31
2.2.3 Les CSP* [2] : . . . 31
2.2.4 Les CSPe [152] : . . . 31
2.2.5 Les ACSP [48] . . . 32
2.3 Techniques de résolution des DCSP . . . 33
2.4 Conclusion . . . 35

3 Vers des systèmes à noyaux multiples 36
3.1 Motivations pour un système à noyaux multiples . . . 37
3.2 Architecture à noyaux multiples . . . 38
3.2.1 Présentation du principe . . . 38
3.2.2 Fonctionnement général du système . . . 39
3.3 Le mécanisme d'inférence hybride . . . 39
3.3.1 Hybridation filtrante . . . 40
3.3.2 Hybridation concurrente . . . 40
3.3.3 Hybridation cyclique . . . 41
3.4 Implantation . . . 41
3.4.1 Interface du système . . . 42
3.5 Conclusion . . . 44

4 Protocole de communication entre SE et CSP 45
4.1 Introduction . . . 45
4.2 Définitions . . . 46
4.2.1 Logique du premier ordre . . . 46
Syntaxe . . . 46
Sémantique . . . 47
4.2.2 Définition des règles et des faits . . . 48
4.3 Construction d'un nouveau CSP dynamique . . . 50
4.3.1 Illustration sur un exemple . . . 50
4.3.2 Définition d'un DDCSP . . . 51
4.4 La procédure de résolution des DDCSP . . . 58
4.4.1 Satisfaction des contraintes . . . 58
4.4.2 Résolution des DDCSP . . . 59
4.4.3 Discussion . . . 63
4.5 Transformation d'un système expert en un DDCSP . . . 63
4.6 Mise en œuvre . . . 67
4.6.1 Jeux de tests . . . 67

4.6.2 Choix des outils 69
4.6.3 Résultats 70
4.7 Conclusion 72

5 CSP distribués 73
5.1 Introduction 73
5.2 Définitions et outils 74
5.2.1 Système Multi-Agent (SMA) 74
5.2.2 Système distribué 75
5.2.3 DisCSP 76
5.3 Les défis des DisCSP 77
5.4 Algorithmes de résolution des DisCSP 78
5.4.1 "La famille ABT" 78
5.4.2 Les différentes variantes de l'algorithme ABT 78
5.5 Conclusion 83

6 Approche parallèle et coopérative 84
6.1 Approche Parallèle et Coopérative (PC) 84
6.1.1 Présentation générale de l'architecture 84
6.1.2 Principe de l'approche PC 85
6.1.3 Algorithme PC_BT 88
6.1.4 Illustration de principe sur un exemple : le problème des quatre reines 91
6.1.5 Validation de l'algorithme PC_BT 93
6.1.6 Complexité 96
6.2 Extension de l'approche 96
6.2.1 Variante de l'algorithme de filtrage 97
6.2.2 Variante de l'heuristique de choix 97
6.3 Résultats expérimentaux 98
6.3.1 Validation expérimentale de PC_BT 98
6.3.2 Évaluation des performances parallèles de l'algorithme PC_BT 100
6.3.3 Comparaisons de différentes versions de PC_BT 100
6.3.4 Tests sur des CSP aléatoires classiques 101
6.4 Conclusion 103

7 CSP sécurisés distribués 104
7.1 Définitions et présentation des outils 105
7.1.1 Cryptographie 105
Algorithmes symétriques ou à clé secrète 106
Algorithmes asymétriques ou à clé publique 106

7.2 DisCSP sécurisés et résolution 107
7.2.1 Méthodes sans utilisation des techniques cryptographiques 107
7.2.2 Méthodes avec utilisation des techniques cryptographiques 108
7.3 Conclusion 111

8 Approche parallèle, coopérative, et sécurisée 112
8.1 La stratégie proposée 112
8.1.1 Principe général 112
8.1.2 Détail de la stratégie 113
8.2 Discussion 114
8.3 Conclusion 116

9 Conclusion & perspectives 117

Liste des tableaux

4.1 Correspondance entre les valeurs des booléens *Fin* et *Trouve* et les états Pending, Failing, Satisfaction 61
4.2 *Caractéristiques du système utilisé pour l'exécution des jeux de tests* 70
4.3 *Temps de résolution des problèmes du zèbre et des reines pour Clips et pour le solveur DDCSP* 71
4.4 *Temps de résolution des problèmes Miss Manners pour Clips et pour le solveur DDCSP* 71

6.1 *Résultats d'exécution de l'algorithme PC_BT sur le problème de coloriage d'un graphe.* 99
6.2 Exécution de l'algorithme PCBT 100
6.3 *Évaluation des performances parallèles du PC_BT* 100
6.4 Comparaison entre PC_BT, PC_BT$_2$, et PC_BT$_3$ en utilisant des problèmes DIMACS consistants 101

Table des figures

1.1 Architecture principale des systèmes experts 6
1.2 Problème de coloriage de carte et son graphe des contraintes associé. 15

3.1 Architecture d'un système à noyaux multiples 39
3.2 Interface principale du système 43

4.1 Hypergraphe associé au eDDCSP de l'exemple 4.5 52
4.2 Hypergraphe associé au eDDCSP de l'exemple 4.6 53
4.3 Hypergraphe associé au eDDCSP de l'exemple 4.7 avant et après la résolution 55
4.4 Hypergraphe associé au DDCSP de l'exemple 4.8 57

5.1 Exemple d'un CSP distribué binaire 76
5.2 Concurrent Backtracking 82

6.1 Le déroulement de l'algorithme sur le problème des quatre reines 93
6.2 Comparaison de PC_BT et PC_FC avec l'algorithme ABT . . . 102

Introduction générale

A good management, a good optimization, says what are powerful tools to ensure this.
Anonymous

Le concept de Système Expert (SE) est un concept assez ancien apparu dans le début des années 60 comme une discipline de l'intelligence artificielle. L'énorme travail des chercheurs a mené à la commercialisation des premiers systèmes experts au début des années 80. Ces SE exploitent une base de connaissances, où la représentation des données dépend fortement du domaine d'application. Généralement, la base de connaissances permet par son aspect modulaire la mise au point de la connaissance et facilite son évolution dans le temps. Un SE doit pouvoir par ailleurs justifier son raisonnement a postériori. La diversité des données conduit à concevoir des noyaux de raisonnement spécialisés qui peuvent résoudre des problèmes particuliers. Toutefois, la conception des SE à domaines spécifiques diminue l'intérêt commercial de ces systèmes.

Motivations et cadre de recherche

Ces travaux s'intègrent dans un projet plus global nommé Cadral, du Centre de Recherche Public Gabriel Lippmann, visant à mettre au point un outil générique et à montrer la faisabilité de l'intégration des CSP dans les SE. L'objectif de cette thèse est donc de proposer un système distribué et sécurisé pour intégrer des contraintes dans un moteur d'inférence.

Le système Cadral est destiné au traitement des allocations familiales au Luxembourg. Ce système utilise un mécanisme de raisonnement qui s'appuie sur un moteur d'inférence à base de règles de production. Ce système est actuellement déployé dans un cadre centralisé. Le Centre du Recherche Public Gabriel Lippmann souhaite étendre les fonctionnalités de Cadral, ses potentialités de réutilisation ainsi que ses champs d'exploitation pratiques. Ce système se caractérise par une syntaxe limitée, conséquence directe du mode de représentation de connaissances utilisé. Il faut noter également que le traitement des allocations familiales au Luxembourg, exige d'accéder à différentes bases de

données situées sur des sites distincts, transfrontaliers, compte tenu du nombre important de travailleurs frontaliers. Ces bases de données contiennent des informations confidentielles et sensibles qui justifient la mise en œuvre de mesures de protection des données.

Afin de répondre à ces besoins, nous nous focalisons dans un premier temps sur le renforcement du mécanisme de raisonnement d'un SE par un noyau de résolution des problèmes de satisfaction de contraintes (Constraint Satisfaction Problem : CSP). Les CSP offrent un cadre puissant et performant pour résoudre et pour représenter de nombreux problèmes notamment de l'intelligence artificielle et de l'ingénierie informatique. La puissance de ce formalisme CSP repose essentiellement sur les contraintes. Notre objectif réside donc dans la possibilité d'exprimer sous forme de contraintes certains modules d'un système expert et de les résoudre efficacement avec les techniques de résolution dédiées aux CSP. La structure des problèmes traités, généralement non statiques, justifie l'utilisation des CSP dynamiques pour conditionner la prise en compte de l'évolution du problème. Nous avons analysé et étudié les différentes variantes des CSP dynamiques et observé que ces formalismes ne répondent pas à nos besoins. Ce qui nous a conduit à la proposition d'un nouveau formalisme, nommé DDCSP (Dynamic Domain Constraint satisfaction Problem) qui est caractérisé par des domaines et des relations variables. Dans la suite de ce travail, nous avons associé ce formalisme à un moteur de raisonnement d'un SE afin d'élaborer un système multi-noyaux. Plus précisément, nous nous sommes intéressés aux différentes stratégies pour combiner les techniques de résolution des CSP et les techniques de raisonnement des systèmes experts. On a ainsi abouti à la définition d'une architecture qui permet la combinaison des deux formalismes.

Par ailleurs, nos travaux de recherche portent sur la construction d'une plateforme répartie permettant de prendre en compte la répartition d'un CSP et son traitement distribué. Nous nous intéressons en particulier aux problèmes de satisfaction de contraintes distribués (Distributed Constraints Satisfaction Problems : DisCSP) pour l'élaboration d'un algorithme de coopération entre plusieurs entités distribuées. La sécurisation de notre prototype constitue également une de nos priorités. Nous présentons un nouveau protocole pour résoudre les DisCSP sécurisés sans utilisation d'un tiers de confiance. Ce protocole nécessite moins de communications, garantit le même niveau de confidentialité que les protocoles existants et utilise un chiffrement à clé publique.

En résumé, les principales motivations qui nous ont incités à mener ce travail sont :

- L'intégration des problèmes de satisfaction de contraintes (CSP) dans le formalisme standard des systèmes experts ;
- L'élaboration d'un prototype de communication lorsque le problème est

distribué favorisant le parallélisme et la coopération ;
- L'étude de techniques de protection des données pour assurer la sécurisation de ce prototype ;
- La validation des résultats par des tests expérimentaux.

Organisation de la thèse

Globalement, la thèse est divisée en deux parties. La première partie contient quatre chapitres. Nous commençons par étudier dans le chapitre 1, les travaux relatifs aux systèmes experts et aux CSP standards. Le chapitre 2 est consacré à l'étude des liens déjà mis en évidence dans la littérature entre les deux modèles et décrit en particulier les CSP dynamiques. Le chapitre 3 présente notre première contribution, qui consiste à décrire l'architecture générale d'un système multi-noyaux, en renforçant le système expert par le formalisme CSP. Ensuite, il décrit les différentes approches possibles de coopération. Le chapitre 4 présente le cœur de notre contribution : une technique de transformation des systèmes à base de règles en un modèle à base de contraintes, appelé DDCSP. Il aborde aussi les algorithmes de résolution des DDCSP. Enfin, il contient également une validation expérimentale d'une architecture multi-noyaux utilisant les DDCSP.

La deuxième partie comporte cinq chapitres. Le chapitre 5 présente un tour d'horizon des articles relatifs aux problèmes de satisfaction de contraintes distribués. Ce chapitre a pour objectif de positionner nos résultats dans le contexte des travaux existants. Le chapitre 6 décrit notre approche parallèle et coopérative qui permet la communication entre plusieurs entités dans un environnement distribué. Le chapitre 7 est consacré aux outils cryptographiques et à leur utilisation dans le contexte des problèmes de satisfaction de contraintes distribués sécurisés. Le chapitre 8 décrit notre politique afin de protéger les données privées dans notre approche parallèle et distribuée, et présente une analyse du niveau de sécurité de cette politique par rapport aux approches classiques existant dans la littérature.

Dans le dernier chapitre de cette thèse, nous présentons le bilan de nos travaux et donnerons quelques perspectives d'études, d'exploitation et de recherche qui permettraient de compléter ou d'approfondir nos résultats.

Chapitre 1

Contexte de l'étude

Artificial intelligence is an engineering discipline built on an unfinished science.
Steve Polyak

Les travaux de cette thèse abordent deux domaines de l'Intelligence Artificielle (IA) : les techniques de raisonnement des systèmes experts et les techniques de résolution des problèmes de satisfaction de contraintes. La conception des systèmes experts constitue un domaine majeur en IA. Ces systèmes sont conçus pour atteindre des performances d'experts humains dans des domaines spécifiques en exploitant un ensemble de connaissances acquises auprès de ces experts humains. Ces systèmes, aussi complexes qu'ils soient, demeurent toutefois limités sur de nombreux points, tant théoriques que pratiques. Ces limitations restent particulièrement visibles dans le domaine juridique (Cadral [1]) comme dans tous les domaines où les méthodes de résolution des problèmes ne relèvent pas d'une simple logique formelle. Nous proposons de renforcer ces systèmes par un formalisme, appelé CSP (Constraint Satisfaction Problem ou Problème de Satisfaction de Contraintes). Ce formalisme offre un cadre puissant et performant pour résoudre et représenter de nombreux problèmes notamment les problèmes combinatoires.

Ce premier chapitre décrit la terminologie et les concepts autour desquels s'articulent nos travaux. Il présente les notions relatives aux systèmes experts (l'architecture, les techniques de raisonnement), et celles relatives aux CSP (le formalisme, les techniques de résolution).

1. Logiciel développé au Centre de Recherche Public Gabriel Lippmann au Luxembourg, le logiciel est spécifique au traitement des dossiers d'allocation familiales

1.1 Les systèmes experts

Les systèmes experts ont comme finalité de reproduire le comportement de l'être humain dans ses activités de raisonnement, afin de parvenir à un gain en temps et en efficacité dans le traitement de tâches de décision complexes. Dans cette partie, nous allons présenter quelques définitions relatives aux systèmes experts : les composants, les modes de représentation des connaissances ainsi que les techniques de raisonnement.

1.1.1 Définition d'un système expert

Il existe plusieurs définitions des systèmes experts dans la littérature [80]. Certains mettent l'accent sur les tâches que ces systèmes peuvent effectuer, d'autres accordent plus d'importance à la façon dont les tâches sont mises en œuvre.

Dans [95] *un système expert est défini comme un programme informatique qui peut conseiller, analyser, classer, communiquer, consulter, concevoir, explorer, anticiper, former, identifier, interpréter, justifier, gérer, planifier, apprendre et tester. Ces tâches constituent des problèmes complexes qui se réfèrent généralement à l'expertise humaine pour l'élaboration d'une solution.*

Dans [45], *un système expert est un programme interactif qui a un mécanisme de raisonnement qui intègre jugements, expériences et règles de décision dans le but de faire des suggestions logiques pour des types de problèmes variés.*

Ces deux premières définitions insistent sur les tâches à réaliser. Par contre Feigenbaum [35] met en évidence les mécanismes de raisonnement dans sa définition : *un système expert est un programme intelligent qui utilise des connaissances et des procédures d'inférences pour résoudre des problèmes.*

Ces définitions présentent un système expert comme un programme informatique capable de réagir à des questions d'un utilisateur selon certaines règles. Cependant, ces définitions restent toujours informelles puisque généralement ces systèmes dépendent du modèle choisi pour la représentation des connaissances. Ces modèles de représentation des connaissances forment un élément clé pour identifier le mode de raisonnement pertinent. Nous allons décrire tout d'abord les composants d'un système expert. Nous précisons ensuite les types d'organisation des données dans ces systèmes. Puis, nous décrivons les différents types de raisonnement existant dans la littérature. Enfin nous passerons en revue les principales plate-formes de systèmes experts existantes et leurs limitations.

1.1.2 Composants d'un système expert

L'architecture générale d'un système expert [59] est illustrée par la figure 1.1. Nous pouvons classifier ces éléments en deux parties : les acteurs et les composants logiciels. On distingue deux principaux types d'acteurs : l'expert et les utilisateurs. La partie "composants logiciels" présente la base de connaissances et le moteur d'inférence.

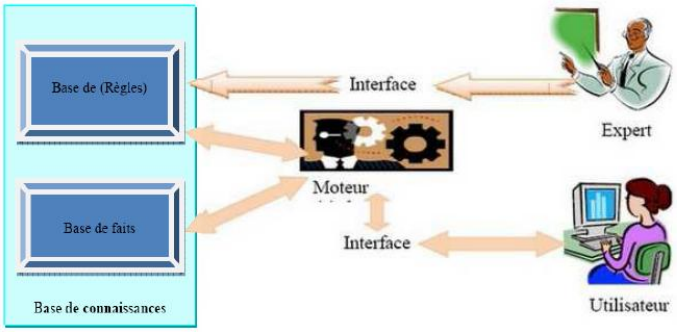


FIGURE 1.1 – Architecture principale des systèmes experts

Les acteurs :

– **L'expert :** enrichit la base de connaissance du système. Cette phase appelée "la phase d'acquisition de la connaissance" est une première tâche à effectuer pour constituer un système expert. Le système possède en général une interface pour l'introduction et la mise à jour des connaissances. Il est préférable d'assister l'expert par un cogniticien. Un cogniticien est une personne qui a une double compétence. Il a des connaissances approfondies dans le domaine de l'expertise. De plus, il possède les connaissances du domaine de l'IA afin de proposer des représentations adéquates pour formaliser et pour structurer les données de l'expertise.

- **L'utilisateur :** interroge le système à travers une interface dédiée. Il n'est pas forcément un expert dans le domaine traité par l'application. Bien entendu, l'interaction entre l'utilisateur et le système doit se faire au travers d'une interface de dialogue dont l'ergonomie n'exige pas de connaissances informatiques.

Les composants logiciels :

- **La base de connaissances :** se décompose en deux parties. La première partie contient des faits spécifiques au domaine d'application. On parle alors de la base de faits.
  La seconde partie, contient l'ensemble des règles qui vont permettre au système de raisonner à partir de la base de faits. Il existe différents modes possibles de représentation des données (voir section 1.1.3)

  1. **La base de faits :** constitue la mémoire de travail du système à base de connaissances. Elle contient les données initiales et les données recueillies par les hypothèses émises ainsi que les nouveaux faits prouvés. Par conséquent, la base de faits, spécifique au problème à résoudre, s'enrichit au cours de la résolution de manière dynamique.
     Les faits peuvent prendre des formes plus ou moins complexes. Les faits élémentaires peuvent avoir des valeurs booléennes, symboliques, ou réelles. Un système qui n'utilise que des faits booléens est dit d'ordre 0. Un système qui utilise des faits symboliques ou réels, sans utiliser de variables, est d'ordre 0+. Un système utilisant toute la puissance de la logique du premier ordre est d'ordre 1.
  2. **La base de règles :** contient l'ensemble des règles de raisonnement du système. Elle peut être modélisée par différents modes de représentation des données (voir section 1.1.3). Cette base rassemble la connaissance et le savoir-faire de l'expert. Elle n'évolue donc pas au cours d'une session de travail et constitue la partie statique du système.

- **Le moteur d'inférence :** constitue la partie du système qui fournit des réponses aux questions des utilisateurs. Il met en œuvre le mécanisme de raisonnement chargé d'exploiter les connaissances. Il effectue les déductions nécessaires afin de proposer une réponse au problème posé. De façon générale, le moteur d'inférence est capable de répondre à des questions, de raisonner et de déduire des conséquences impliquées par la connaissance incluse dans le système.

1.1.3 Organisation des connaissances

La représentation des connaissances est considérée comme la manière de mémoriser les connaissances dans un ordinateur. Il y a plusieurs modes de représentation des données qui peuvent être utilisés séparément ou en conjonction pour construire un système. Chaque technique offre certains avantages de performance, de lisibilité ou de modularité. Les modes de représentation les plus utilisés dans les systèmes experts sont : les règles de production, la structure d'objets, la logique d'ordre 0 et 1, la représentation procédurale et heuristique.

- **Les règles de production [70] :** ce modèle de représentation des connaissances est très répandu. Il est ainsi dénommé, car les règles produisent de nouveaux faits au cours de la déduction. Ces règles sont représentées sous la forme suivante :

  Si (condition) **alors** (conclusion).

  La partie *condition*, appelée aussi *prémisse*, correspond à une expression qui doit être vérifiée pour que la règle se déclenche. La partie *conclusion* peut correspondre au déclenchement d'un nouveau *fait*.
- **La structure d'objets :** le concept a été introduit pour la première fois dans le domaine IA par Minsky [96] sous le nom de frames. L'idée consiste à offrir un support permettant de regrouper l'ensemble des informations disponibles et de structurer les données similaires dans une entité, appelée objet. Un objet représente un élément utilisé au cours du raisonnement. Différents types d'informations sont attachées à l'objet : ses caractéristiques (ses propriétés) et comment utiliser l'objet (ses méthodes). Par exemple, dans le contexte de la conception assistée par ordinateur, une voiture constitue un objet. Les propriétés représentent les caractéristiques de l'objet (nom de la voiture, constructeur, type, la vitesse...). Une méthode "carburation" définit l'accélération de la voiture en fonction de la richesse du mélange fourni au moteur.
- **La logique [54] :** ce formalisme est un outil de représentation des connaissances, l'un des premiers formalisme utilisé en IA. Les données représentées sont déclaratives et se basent essentiellement sur la logique mathématique. Dans la littérature, nous distinguons : (1) la logique des propositions, (2) la logique des prédicats.

  1. **La logique des propositions [91] :** est constituée par un ensemble de propositions qui peuvent prendre la valeur "vrai" ou "faux". La détermination des propositions repose sur l'utilisation de connecteurs logiques (et, ou, non, implique, équivalent). La logique des propositions n'utilise pas les notions de variables, de fonctions, de prédicats ni les quantificateurs.

2. **La logique des prédicats [33] :** les principes de base du langage ne sont plus des propositions, mais des prédicats. Un prédicat peut être vu comme une proposition qui contient une ou plusieurs variables. La logique des propositions est donc incluse dans la logique des prédicats. Nous y reviendrons plus longuement dans le chapitre 4.

– **La représentation procédurale et heuristique [159] :** le comportement procédural dans un système expert consiste à utiliser les notions classiques de procédures informatiques. Ainsi, pour une base de connaissances qui contient un nombre très important de règles, la quantité d'opérations à effectuer pour parcourir l'ensemble de règles dépasse souvent la capacité des ordinateurs, c'est le problème de l'explosion combinatoire. C'est là qu'intervient la notion d'heuristique. Ces heuristiques procèdent par l'instauration de coupures dans l'arbre de recherche et évitent l'exploration de certaines directions jugées infécondes. Elles orientent la recherche en fonction du but à résoudre.

1.1.4 Techniques de raisonnement

La base de connaissances n'explique pas la manière dont les règles sont reliées entre elles. Il est simple de comprendre les interactions entre les règles lorsque leur nombre est restreint, mais la complexité de la tâche augmente avec la taille de la base de connaissances. Le rôle d'un moteur d'inférence est justement de donner un sens à ces règles en les reliant entre elles de manière à déterminer les conclusions appropriées aux questions posées. Le mécanisme de raisonnement utilisé par le moteur d'inférence est appelé *la procédure d'inférence*. La procédure d'inférence est un algorithme qui trouve les conséquences logiques d'un ensemble de prémisses. Selon leur principe de fonctionnement, on peut classer ces procédures de raisonnement en deux classes : les techniques d'inférence et les stratégies de contrôle.

– **Techniques d'inférence :** ces techniques permettent de guider le SE pour prouver la vérité ou la fausseté d'une proposition, ou bien pour déduire des conclusions à partir des prémisses. C'est l'application des règles d'inférence (ou règles de déduction logique) qui guident le SE dans ce sens. Nous caractérisons trois techniques d'inférence : (1) le Modus Ponens, (2) le Modus Tollens et (3) l'inférence par résolution.

1. **Modus Ponens :** c'est un mode de raisonnement logique qui consiste à déduire B à partir des deux formules propositionnelles A et $A \Longrightarrow B$. Ce mode de raisonnement est utilisé lorsque les connaissances du système sont représentées par la logique du premier ordre. La règle d'inférence de Modus Ponens s'écrit formellement :

A, $A \Longrightarrow B \vdash B$. Si on a, par exemple, les propositions suivantes : (1) aujourd'hui il pleut, (2) s'il pleut alors je prends mon parapluie, on peut déduire que je prends mon parapluie.

2. **Modus Tollens :** c'est une règle presque similaire à la règle Modus Ponens, et s'écrit formellement : $\neg B$, $A \Longrightarrow B \vdash \neg A$. Exemple, (1) je suis sorti, (2) s'il pleut je reste chez moi, on peut en déduire qu'il ne pleut pas.
3. **Inférence par résolution :** cette règle consiste à appliquer de façon itérative la règle de résolution suivante lorsque A, B, et C sont des propositions : $(A \vee B) \wedge (\neg A \vee C) \Longrightarrow (B \vee C)$. Elle permet de combiner des propositions qui contiennent une proposition et sa négation en une seule proposition qui sera rajoutée à l'ensemble des connaissances. Par exemple de (il pleut ou je reste chez moi) et (il ne pleut pas ou je sors avec mon parapluie) on peut déduire (je reste chez moi ou je sors avec mon parapluie)

Pour une définition complète et exhaustive des termes et des notions utilisés dans le paragraphe ci-dessus, nous conseillons au lecteur intéressé de consulter [81, 3].

– **Stratégies de contrôle :** typiquement, le moteur d'inférence d'un système expert utilise des stratégies de contrôle des différentes règles d'un SE lorsque les connaissances sont représentées sous forme des règles de production. Nous pourrons distinguer trois types de stratégies de contrôle : (1) le chaînage avant, (2) le chaînage arrière et (3) le chaînage mixte.

1. Le **chaînage avant [135] :** permet de déduire les nouveaux faits à partir des faits initiaux. Le mécanisme consiste à appliquer toutes les règles possibles à l'ensemble des faits connus, en ajoutant chaque nouvelle conclusion à cet ensemble. Par application itérative du processus, chaque conclusion peut elle-même satisfaire les conditions d'une autre règle, ce qui conduit à un chaînage avant des règles. Le processus s'arrête lorsqu'aucune nouvelle règle n'est applicable à l'ensemble des faits, ou quand une solution satisfaisante est trouvée.La plupart des moteurs d'inférence en chaînage avant utilisent l'algorithme de RETE [42] pour optimiser les performances.
2. Le **chaînage arrière [135] :** dans le mécanisme de chaînage arrière, le moteur d'inférence part du but à démontrer. Il sélectionne alors les règles qui permettent de prouver ce fait. De proche en proche, on considère les sous-conditions, qui sont les prémisses de ces règles, comme de nouveaux buts à prouver. Le processus s'arrête dès qu'une sous-condition a été prouvée ou lorsque la liste des sous-conditions à

prouver est vide. Prolog est basé sur le chaînage arrière.

3. Le **chaînage mixte [135] :** ce chaînage consiste à utiliser les deux types de chaînages présentés ci-dessus (chaînage avant et chaînage arrière). On peut alors aussi bien raisonner à partir des prémisses que des conclusions.

   Ce sont les caractéristiques du problème qui vont conditionner le chaînage qu'il est judicieux d'utiliser. Ainsi, lorsque les faits sont peu nombreux ou que le but est inconnu, il est préférable d'employer un chaînage avant. Par contre, dans le cas où les faits sont nombreux et les buts précis, le chaînage arrière est préconisé car il évite l'explosion combinatoire.

1.1.5 Quelques architectures ou langages de développement des systèmes experts

Nous décrivons dans cette section quelques architectures ou langages de développement des systèmes experts. Notre objectif est de donner une vue générale de ces différents outils, de montrer les limitations rencontrées et d'examiner les exigences requises par ces systèmes.

Les systèmes experts ont été utilisés dans de multiples domaines comme par exemple dans le cadre de la médecine [36, 100], de la finance [105, 163], de l'éducation [107, 148] ou de l'administration [146]. Dans la pratique, la plupart des systèmes experts développés utilisent la représentation par règles de production. On les nomme généralement "systèmes de production". Parmi les plateformes qui utilisent les règles de production, nous pouvons citer Mycin [90], Prolog [175], Clips [161], Soar [176], et Cadral [146]. Nous allons décrire ici, le principe et le rôle général de chacune de ces plateformes.

- **Mycin [90]** est l'un des premiers systèmes experts, développé à l'université de Standford à partir du début des années 70. Ce système a été conçu pour le domaine spécifique des antibiotiques. Les chercheurs ont observé que les antibiotiques agissent différemment en fonction du germe bactérien impliqué. Seules certains spécialistes connaissent bien ce domaine rendu complexe par l'immense variété de germes bactériens et de molécules antibiotiques. Mycin est fondé sur des règles très simples de type *si condition alors conclusion* et utilise le chaînage avant. Pourtant, les résultats obtenus [86] par ce système expert dans le diagnostic médical ont montré des performances très acceptables par comparaison à des experts humains.
- **Soar [104]** est une architecture cognitive destinée à développer des systèmes experts. Les premiers travaux ont démarré en 1983 à l'université du Michigan. Il utilise également une représentation des connaissances sous

forme de règles de production et l'algorithme de chaînage avant comme mécanisme d'inférence. Il est indépendant d'un domaine d'application qui pourra varier en fonction de la base de connaissances.

- **Clips [161]** est un environnement et un langage de développement de systèmes experts développé à l'origine par la Direction de la Technologie des Logiciels et par le Johnson Space Center de la NASA. Depuis sa première version en 1986, les performances de CLIPS ont été largement améliorées. Il utilise un raisonnement dirigé par les faits en chaînage avant. Il est développé en langage de programmation C/C++. Les données sont représentées par des règles de production.
- **Prolog [175]** a été crée à Marseille en 1972 par Alain Colmerauer et Philippe Roussel. Son nom, abréviation de "PROgrammation en LOGique", désigne l'outil informatique conçu pour implanter un système de communication homme/machine en langage naturel. Prolog est un langage qui utilise une description logique des fonctions à calculer et un mécanisme de preuve pour les évaluer. Ainsi, il exploite la logique pour la représentation des connaissances. Les concepts fondamentaux de Prolog sont l'unification, la récursivité et le backtrack. Contrairement à CLIPS, Prolog utilise un raisonnement dirigé par les buts et une résolution basée sur le chaînage arrière.
- **Cadral [146]** est développé au sein du département ISC du CRP-GL (Centre de Recherche Public Gabriel Lippmann) en partenariat avec la Caisse Nationale des Prestations Familiales (CNPF) du Luxembourg pour le traitement des dossiers de demande d'allocations. Ce logiciel utilise actuellement un noyau de résolution Soar et une représentation graphique des règles qui facilitent l'acquisition des connaissances. Un interpréteur permet de faire un lien entre les connaissances saisies graphiquement et les règles de type Soar.

Soar, Prolog, CLIPS peuvent être considérés comme des plateformes de programmation destinées au développement de SE. En effet, ces plateformes restent indépendantes du domaine d'application et pourront être utilisées dans divers domaines d'expertise simplement en modifiant la base de connaissances. Néanmoins, ces systèmes nécessitent de formaliser les connaissances dans un mode approprié.

1.1.6 Limitations des systèmes experts

Ces systèmes apportent des solutions nouvelles ou parfois plus rapides mais soulèvent plusieurs difficultés [65, 27] :

- **Acquisition des connaissances :** l'acquisition des connaissances requiert le codage des connaissances dans la base, ce qui mobilise l'intervention de deux spécialistes. Le premier, expert du domaine d'application, fournit les connaissances nécessaires à la réalisation de l'application. Le second, expert en IA, extrait le savoir-faire du premier et le code dans la base. L'expert du domaine a souvent des difficultés pour exprimer explicitement ses connaissances, ce qui nuit à l'élaboration du système.
- **Cohérence de la base de connaissances :** la vérification de la correction et de la complétude de la base de connaissances est une tâche complexe. Selon le domaine d'application, cette tâche peut même s'avérer impossible dans la mesure où des incohérences et des contradictions sont parfois inhérentes ou consubstantielles du domaine, par exemple dans le cadre juridique.
- **La rigidité de la base de connaissances :** l'autre défi des systèmes experts est une conséquence de la représentation uniforme des connaissances. Toutes les connaissances sont représentées dans un formalisme unique. Les règles de la base de connaissances sont alors dédiées à un type de raisonnement particulier. Il en résulte la difficulté de construire la base pour des problèmes qui s'expriment naturellement dans des formalismes différents.
- **La faiblesse de raisonnement :** cette limitation est une conséquence directe du codage de la base connaissances, puisque généralement la procédure de raisonnement dépend du mode de représentation de la base connaissances.

Cependant, si l'on considère la part d'efficacité d'un système expert, ce résultat repose sur la possibilité de construire une base de connaissances complète et cohérente, de la développer de manière qu'elle soit réutilisable et adaptable, avec des techniques de raisonnement appropriées. Afin d'atteindre ces objectifs, nous allons renforcer le moteur du système expert avec un formalisme de CSP que nous introduisons dans les chapitres 3 et 4. Après avoir présenté les différents modes de représentation des connaissances, les types de raisonnement existants et quelques exemples de plateformes de SE, nous allons dans la seconde partie de ce chapitre présenter le deuxième modèle essentiel que nous avons étudié : les problèmes de satisfaction de contraintes.

1.2 Les problèmes de satisfaction de contraintes

Nous présentons dans cette seconde partie quelques notions et définitions relatives aux problèmes de satisfaction de contraintes. Nous allons rappeler les notions de vocabulaire utilisées dans le contexte des CSP. Puis, nous verrons de manière très générale les différentes techniques de résolution des CSP. Un accent particulier sera mis également sur les techniques de filtrage et de propagation.

Définition 1.1 (CSP [99]) *Les problèmes de satisfaction de contraintes sont définis par un quadruplet (X, D, C, R) où :*

- *$X = \{x_1, x_2, ..., x_n\}$ est un ensemble fini de variables ;*
- *$D = \{d_1, d_2, ..., d_n\}$ est un ensemble fini de domaines. Quel que soit i, d_i est l'ensemble des valeurs pouvant être attribuées à la variable x_i ;*
- *$C = \{c_1, c_2, ..., c_m\}$ un ensemble fini de contraintes. Chaque contrainte c_i est définie par un tuple $(x_{i1}, x_{i2}, ..., x_{ik})$ de k variables de X liées par c_i. On nomme $scope(c_i)$ l'ensemble $\{x_{i1}, x_{i2}, ..., x_{ik}\}$;*
- *$R = \{r_1, r_2, ..., r_m\}$ est un ensemble fini de relations associées aux m contraintes de C. Chaque relation r_i représente un ensemble de n-uplets de valeurs autorisées par la contrainte c_i.*

Cette définition met en évidence l'ensemble des relations R qui est souvent implicite dans les définitions de la littérature. Cette distinction est volontaire dans la mesure où elle sera importante pour les définitions liées aux DDCSP que nous introduisons dans le chapitre 4.

De plus, cette définition utilise volontairement des relations exprimées en extension qui sont plus générales (voir plus loin les définitions 1.7 et 1.8).

Définition 1.2 (arité) *L'arité d'une contrainte c est le nombre de variables sur lesquelles elle est définie. On la note $|c|$. Une contrainte est dite **unaire** (respectivement **binaire**) si elle d'arité un (respectivement deux). Dans les autres cas, on la qualifie de **n-aire**, où n est l'arité de la contrainte.*

Définition 1.3 (CSP binaire) *Un CSP dont toutes les contraintes sont unaires ou binaires, est appelé un CSP binaire.*

Définition 1.4 (CSP n-aire) *Un CSP est n-aire s'il comporte des contraintes d'arité supérieure à deux.*

1.2.1 Représentation graphique d'un CSP binaire

Un CSP binaire peut être représenté par un graphe de contrainte $G(X, C)$ dont les sommets sont les variables et les arcs sont les contraintes du CSP.

Exemple d'un CSP (coloriage d'une carte)

L'objectif consiste à colorier une carte, de sorte que deux régions ayant des frontières en commun soient coloriées avec des couleurs différentes.

Ce problème peut-être modélisé par le CSP suivant :

- L'ensemble des variables $X = \{x_1, x_2, ..., x_6\}$ est l'ensemble des régions à colorer ;
- $D = \{d_1, d_2, ..., d_6\}$ est l'ensemble des domaines de chaque variable. Quel que soit i, d_i est l'ensemble des couleurs pouvant être attribuées à la variable x_i. $d_i = \{rouge, jaune, bleu\}$;
- $C = \{c_1, c_2, ..., c_{10}\}$ est l'ensemble fini des contraintes. Chaque contrainte c_i est définie par un couple (x_{i1}, x_{i2}) de variables de X représentant des régions voisines sur la carte. $c_1 = (x_1, x_2), c_2 = (x_1, x_3), \ldots, c_{10} = (x_5, x_6)$
- $R = \{r_1, r_2, ..., r_{10}\}$ est l'ensemble des valeurs autorisées respectivement par $\{c_1, c_2, ..., c_{10}\}$. Chaque relation r_i représente l'ensemble de couples de couleurs différentes. Donc quel que soit i, $r_i = \{(rouge, jaune), (rouge, bleu), (jaune, rouge), (jaune, bleu), (bleu, rouge), (bleu, jaune)\}$;

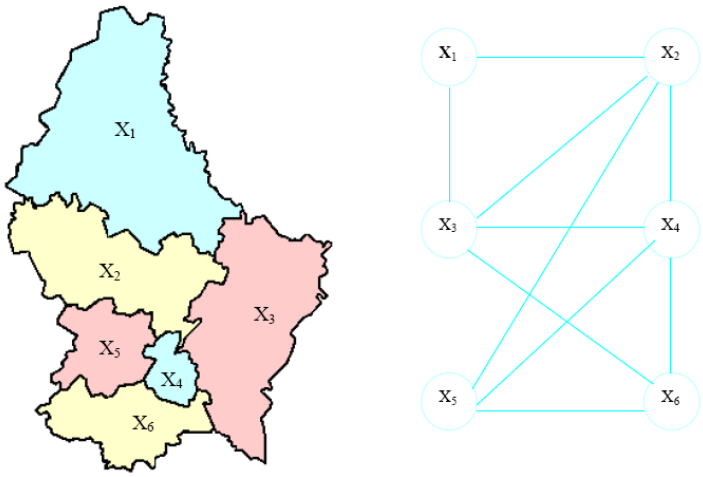


FIGURE 1.2 – Problème de coloriage de carte et son graphe des contraintes associé.

La figure 1.2 illustre cet exemple et sa représentation graphique. Les domaines qui sont les couleurs possibles de chaque variable n'apparaissent pas dans cette représentation graphique (il existe une représentation en microstructure qui permet de voir les valeurs des domaines des variables en explicitant les n-uplets autorisés par chacune des contraintes). Les contraintes sont représentées par les arcs du graphe. Elles expriment la différence des valeurs affectées aux variables.

Cette représentation ne fait pas apparaitre le détail des domaines et des valeurs respectant chaque contrainte. Mais elle donne une idée générale de la structure du problème.

Remarque 1.1 *La représentation graphique d'un CSP n-aire s'exprime par un hypergraphe.*

1.2.2 Représentation des contraintes

Une contrainte peut être représentée de différentes façons. Une contrainte portant sur des variables numériques peut être représentée par des équations d'égalité ou d'inégalité, un prédicat, une fonction booléenne, par exemple, $x_i \neq x_j$. On qualifie ces contraintes de contraintes primitives.

La définition des contraintes globales est sujet à discussion dans la communauté des contraintes [10] où les notions de globalité sémantique, opérationnelle et algorithmique sont distinguées. Néanmoins, dans la suite de ce document, nous définissons la notion de globalité dans un sens simple de combinaison de contraintes primitives.

Définition 1.5 (contrainte globale) *Une contrainte globale est une contrainte dont la sémantique est équivalente à une conjonction de plusieurs contraintes primitives. Exemple : soit le système de contraintes suivant* $(x_1 \neq x_2) \vee (x_2 \neq x_3) \vee (x_1 \neq x_3)$. *Ces trois contraintes peuvent être modélisées par une unique contrainte globale* $alldiff(x_1, x_2, x_3)$.

Définition 1.6 (méta-contrainte) *On utilisera le terme de "méta-contrainte" pour qualifier une contrainte de contraintes, définie à l'aide d'opérateurs logiques. Soit c la contrainte suivante : si* c_1 *est satisfaite alors* c_2 *et* c_3 *doivent être satisfaites. On écrira* $c : \neg c_1 \vee (c_2 \wedge c_3)$.

Définition 1.7 (contrainte en intension) *Une contrainte est représentée en intention lorsqu'elle exprime la relation entre plusieurs variables en utilisant les opérateurs arithmétiques ou logiques (par exemple* $x <= y$, $A \wedge B \Rightarrow \neg C$*)).*

Définition 1.8 (contrainte en extension) *Une contrainte est représentée en extension lorsque tous les n-uplets autorisés (goods) ou bien non autorisés (nogoods) sont énumérés.*

Définition 1.9 (contrainte satisfaite) *Une contrainte $c = (x_{i_1}, \ldots, x_{i_k})$ est satisfaite par un n-uplet $\tau \in d_{x_{i_1}} \times \ldots \times d_{x_{i_k}}$ si et seulement si $\tau \in r$, où r est la relation associée à c. On dit alors que τ satisfait c. Dans le cas contraire on dit que τ viole la contrainte c.*

Définition 1.10 (instanciation) *Soit $x \in X$ une variable et $v \in d(x)$. On dit que x est instanciée avec la valeur v lorsque v a été affectée à x (on dit également que $x = v$). Soit un ensemble de variables $Y \subseteq X$ tel que $\forall\ y_i \in Y$, y_i est instanciée avec la valeur v_i. L'ensemble des couples $I = \{(y_i, v_i)\}$ est une instanciation partielle de X. I est dite complète si elle porte sur toutes les variables de X.*

Remarque 1.2 *On emploie souvent dans la littérature scientifique le terme d'affectation au lieu d'instanciation.*

Définition 1.11 (instanciation consistante) *Étant donné un problème $P = (X, D, C, R)$, une instanciation I des variables de X est dite consistante si et seulement si elle ne viole aucune contrainte de C. Elle est dite inconsistante sinon.*

Définition 1.12 (solution d'un CSP) *Une solution d'un CSP est une instanciation complète consistante.*

Définition 1.13 (CSP consistant) *Un CSP P est dit consistant s'il admet au moins une solution. Il est qualifié d'inconsistant sinon.*

La résolution d'un CSP dépend du résultat attendu. Résoudre un CSP peut consister à : :

- **rechercher toutes les solutions :** Il s'agit de déterminer toutes les solutions (si le CSP est consistant).
- **rechercher une solution :** L'algorithme de résolution s'achève dès qu'une solution est trouvée.
- **vérifier la consistance d'une instanciation complète :** il s'agit de la vérification de la consistance de l'instanciation donnée.

1.2.3 Méthodes de résolution des problèmes de satisfaction de contraintes

Les méthodes complètes de résolution de CSP explorent de manière systématique l'espace de recherche et sont capables de fournir toutes les solutions d'un problème. L'algorithme de recherche de base le plus souvent utilisé est l'algorithme de retour arrière ou backtrack [62]. Cet algorithme met en place une stratégie de parcours de l'arbre de recherche en profondeur d'abord avec

un mécanisme de retour arrière lorsque l'affectation partielle courante n'est pas consistante.

Dans ce cas, l'algorithme backtrack tente de modifier la dernière affectation de la variable courante. Lorsque toutes les valeurs ont été utilisées, l'algorithme revient en arrière pour modifier l'affectation de la variable précédente. Par contre, rien ne garantit que cette variable soit la cause du conflit rencontré. Pour remédier à ce défaut, plusieurs variantes comme les algorithmes Backjumping [44] ou Conflict-directed Backjumping [110] ont proposé de mémoriser les variables déjà instanciées susceptibles d'être causes de conflit et proposent, en cas de retour arrière, de remonter directement à ces variables. D'autres variantes, de type Backmarking [58], se focalisent sur la notion de mémorisation dans l'objectif de ne pas visiter plusieurs fois les mêmes sous-branches de l'arbre de recherche. À partir d'algorithmes provenant de ces deux voies on peut produire plusieurs algorithmes hybrides [110].

Ces algorithmes peuvent également être améliorés par des heuristiques déterminant l'ordre des variables à instancier et l'ordre des valeurs à tester pour minimiser le nombre de branches à explorer [129, 11].

Les méthodes de résolution dites incomplètes n'explorent pas de façon systématique l'espace de recherche. Elles sont basées sur une exploration opportuniste de l'ensemble des instanciations complètes et ne fournissent qu'un sous-ensemble de solutions. Elles nécessitent une fonction d'évaluation et de comparaison des affectations. Ces méthodes incomplètes sont généralement utilisées pour résoudre des problèmes de taille élevée. Elles fournissent un ensemble de solutions. Il faut alors choisir la "meilleure" solution, la mieux adaptée au problème de départ. Ce choix final requiert un effort de réflexion et d'analyse supplémentaire. Ces méthodes sont donc habituellement combinées à des mécanismes d'optimisation. Nous pouvons citer la méthode de recherche Tabou [61] ou le Recuit Simulé [82].

1.2.4 Filtrage au cours de la résolution

Dans les techniques de résolution présentées précédemment, les contraintes sont utilisées de manière passive. Elles sont exploitées uniquement pour tester la consistance des affectations partielles et complètes. Les techniques de filtrage utilisent les contraintes de manière active pour effectuer des déductions sur le problème. L'objectif principal des techniques de filtrage est la détection précoce d'instanciations partielles localement ou totalement incohérentes. Une des techniques les plus utilisées est la technique de renforcement de la consistance locale ou consistance d'arc [99].

Le premier algorithme qui utilise ce concept de filtrage est nommé Forward

Checking (FC) [69]. Lors de chaque instanciation d'une variable courante, FC supprime les valeurs des variables non encore instanciées inconsistantes avec l'instanciation courante. L'algorithme FC limite la suppression de valeurs aux variables liées à la variable courante. D'autres algorithmes propagent en plus les suppressions. Par exemple, l'algorithme Maintaining Arc-Consistency (noté MAC [127]) applique un filtrage par arc-consistance après chaque instanciation.

Définition 1.14 (Arc-consistance) *Soit P un CSP binaire (X, D, C, R). Une contrainte binaire $c \in C$ liant les variables x_i et x_j est arc-consistante si et seulement si pour toute valeur $v_i \in d_i$, il existe une valeur $v_j \in d_j$ telle que le couple de valeurs (v_i, v_j) soit autorisé par c. On dit que P est arc-consistant si et seulement si toutes les contraintes c de C sont arc-consistantes.*

Pour les contraintes n-aires, la consistance d'arc est généralisée dans [7] et est appelé consistance d'arc généralisée.

Définition 1.15 (consistance d'arc généralisée (GAC)) *Soit P un CSP (X, D, C, R). Une contrainte $c \in C$ de relation r est GAC (arc consistante généralisée) si et seulement si $\forall x_i \in scope(c)$, $d_i \neq \emptyset$ et $\forall v \in d_i$, $\exists \tau \in r$ tel que $v = \tau[x_i]$. On dit que P est GAC si et seulement si toutes les contraintes c de C sont GAC.*

Remarque 1.3 *La notation $\tau[x_i]$ représente la projection de τ sur x_i. L'opérateur de projection est défini ci-dessous.*

Définition 1.16 (projection de la contrainte n-aire) *Soit $c_{1,...,k}$ une contrainte d'arité k, définie sur un sous-ensemble de variables $\{x_1, ..., x_k\}$. On appelle la projection de la contrainte $c_{1,...,k}$ sur un sous-ensemble $\{x_1, ..., x_l\}$, avec $l < k$ et $\{x_1, ..., x_l\} \subset \{x_1, ..., x_k\}$, la nouvelle contrainte $c_{1,...,l}$ d'arité l, dont les tuples autorisés sont les sous-tuples consistants de la contrainte $c_{1,...,k}$. On la note $c_{1,...,k}[x_1, ..., x_l]$. De la même manière on peut définir la projection d'une affectation de $\{x_1, ..., x_k\}$ sur $\{x_1, ..., x_l\}$.*

Il existe plusieurs degrés de filtrage qui permettent de vérifier la consistance de n-uplets de valeurs de variables. Le degré de filtrage correspond au nombre de variables participant à la vérification de la cohérence locale. Plus le degré est grand, meilleure sera la détection des combinaisons inconsistances, mais plus il faudra de temps pour les détecter.

Définition 1.17 (k-consistance) *Un CSP est k-consistant si pour tout k-uplet de variables $(x_1, x_2, \ldots, x_k)$ de domaines respectifs $(d_1, d_2, \ldots, d_k)$ et pour tout $(k-1)$-uplet de valeurs consistantes $(v_1, v_2, \ldots, v_{k-1})$, où $(v_1 \in d_1, v_2 \in d_2, \ldots, v_{k-1} \in d_{k-1})$, il existe une valeur $v_k \in d_k$ telle que toutes les contraintes liant le k-uplet de variables $(x_1, x_2, \ldots, x_k)$ soient satisfaites.*

Définition 1.18 (k-consistance forte) *Un CSP P est fortement k-consistant si $\forall j \leq k$, P est j-consistant.*

Ces méthodes de filtrage permettent de réduire la taille du problème courant en éliminant les valeurs devenues inconsistantes. Ces techniques de propagation permettent d'améliorer les algorithmes de résolution dont [84] dressent un inventaire. Les méthodes de résolution les plus efficaces [110, 63, 127] combinent les différents filtrages par consistance d'arc avec un algorithme de retour arrière.

1.2.5 Résolution parallèle des CSP

Nous venons de voir qu'une manière d'accélérer la résolution d'un CSP est de s'appuyer sur les notions de consistance. Appliquer une technique de filtrage permet de réduire le nombre de valeurs dans les domaines de définition donc de réduire l'espace de recherche.

Une autre méthode consiste à utiliser le parallélisme. Trois types d'approches principales sont utilisées pour paralléliser la résolution des CSP : (1) la parallélisation des étapes de filtrage ou de résolution ; (2) le lancement de plusieurs solveurs sur le même problème (concurrence) ; (3) la décomposition de problème en plusieurs sous-problèmes.

Avant de décrire ces différentes approches, nous rappelons quelques notions pour mesurer les performances d'un algorithme parallèle.

Évaluation des performances d'un algorithme parallèle

D'un point de vue expérimental, on peut considérer deux critères d'évaluation de la performance d'un algorithme parallèle. Le premier consiste à évaluer les performances de l'algorithme parallèle par rapport aux algorithmes séquentiels standards existants dans la littérature, alors que le second concerne la qualité du parallélisme.

En fait, l'évaluation de performance d'un algorithme parallèle est donc une simple comparaison parallèle des résultats de l'algorithme parallèle avec les principaux algorithmes séquentiels. Elle permet d'évaluer l'intérêt pratique de l'utilisation de parallélisme. La parallélisation d'un algorithme séquentiel a pour objectif d'accélérer l'exécution de la tâche accomplie par cet algorithme. Lorsqu'on souhaite évaluer la qualité d'un algorithme, on procède aux calculs d'accélération et l'efficacité. Nous évoquons ci-dessous les définitions de ces deux notions.

Définition 1.19 (accélération) *Soit T_1 (respectivement T_p) le temps d'exécution d'un algorithme en utilisant un processus (resp. p processus). On appelle accélération le rapport $\frac{T_1}{T_2}$. Une accélération est dite linéaire si elle est égale à*

p, superlinéaire si elle est supérieure à p, et sublinéaire sinon. L'accélération est dite absolue si T_1 est le temps du meilleur algorithme séquentiel, ou relative si T_1 est le temps du même algorithme parallèle exécuté sur un seul processus.

Définition 1.20 (efficacité) *Soit T_1 (respectivement T_p) le temps d'exécution d'un algorithme utilisant un processus (respectivement p processus). On appelle efficacité le rapport de l'accélération divisée par le nombre de processus, c'est-à-dire $\frac{T1}{p.T_p}$. Bien sûr, l'idéal est d'obtenir une efficacité de 1, en pratique, une efficacité supérieure ou égal à 0,95 apparaît raisonnable. Au niveau expérimental, l'efficacité peut parfois devenir supérieure à 1, lors de l'occurrence d'accélérations superlinéaires. Notons enfin que l'efficacité dépend du nombre de processus employés et qu'elle a généralement tendance à décroître avec l'augmentation du nombre de processus.*

Parallélisation des algorithmes existants

La parallélisation de la résolution a été initiée par la communauté de la programmation logique [102] et par celle de la recherche opérationnelle [79]. Ces travaux sont en relation avec les techniques de résolution du backtacking, mais ne sont pas directement liés avec les CSP. Par contre, plusieurs études ont été effectuées pour paralléliser les techniques de filtrages [77, 78]. La mise en œuvre de ces algorithmes comme pré-traitement est une technique intéressante qui permet d'assurer un niveau de consistance.

Approche concurrente

Cette approche consiste à lancer plusieurs solveurs indépendants dirigés par des heuristiques différentes. Les solveurs résolvent le même problème mais, les heuristiques étant différentes, les résultats obtenus pourront varier. L'objectif est en général de déterminer l'existence d'une solution. L'algorithme se termine dès qu'un solveur a trouvé une solution ou qu'il a prouvé que le problème était inconsistant. En pratique, cette approche est basée sur la différence des stratégies de résolution de chaque solveur, avec l'espoir que l'un des solveurs soit mieux adapté au problème à résoudre. Un des avantages du point de vue du parallélisme est que les solveurs s'arrêtent dès que l'un d'entre eux a trouvé une solution ce qui évite tout problème de famine. Mais un inconvénient majeur réside dans le fait que plusieurs parties de l'espace de recherche seront visitées plusieurs fois par les différents solveurs. Cette approche a été expérimentée sur le problème SAT [52]. Les résultats présentés dans ces travaux montrent une légère amélioration par rapport à un solveur unique. Cependant, les accélérations sont majoritairement sub-linéaires. Quelques accélérations linéaires ou super-linéaires sont toutefois observées sur des benchmarks spécifiques.

Durant la résolution, ces solveurs peuvent travailler de manière coopérative en échangeant des informations. Les solveurs pourraient s'entraider et ainsi déterminer plus rapidement si le problème possède ou non une solution. Ce principe constitue la base de toute approche coopérative [21, 22, 68].

Parallélisation et décomposition

Cette approche commence par une phase de pré-traitement qui consiste à décomposer le problème en plusieurs sous-problèmes indépendants. En général, la phase de construction des sous-problèmes reste séquentielle. Par contre, les sous-problèmes peuvent être résolus en parallèle par différents solveurs, puisqu'ils sont indépendants les uns des autres. La décomposition permet de dégager une nouvelle heuristique liée aux différents ordres possibles de résolution des sous-problèmes [77, 78]. De nombreuses techniques de décomposition ont été développées comme les techniques de décomposition structurelles telles que le tree clustering et la méthode coupe-cycle [31] ou encore la décomposition de domaines [73]. Ces méthodes de décomposition arborescente ont également été utilisées dans des contextes proches [1, 162]. La difficulté majeure de ce type d'approche réside généralement dans la construction d'une solution globale à partir des solutions de chaque sous-problème.

1.3 Conclusion

Dans ce chapitre, nous avons rappelé les principes généraux des systèmes experts et des problèmes de satisfaction de contraintes. Nous avons décrit dans un premier temps les modèles de représentation des connaissances des systèmes experts ainsi que les modes de raisonnement. Dans un second temps, nous avons donné les définitions du formalisme CSP et ses modes de résolution.

La comparaison de ces deux modèles et les études permettant des interactions et des coopérations entre CSP et SE seront abordées dans le chapitre suivant.

Chapitre 2

Liens entre CSP et systèmes experts

The more constraints one imposes, the more one frees one's self. And the arbitrariness of the constraint serves only to obtain precision of execution.
Igor Stravinsky

Le deuxième chapitre de cette thèse est dédié aux liens entre les CSP et les SE, et contient deux parties. La première partie présente quelques liens existants dans la littérature scientifique entre les SE et les CSP. En particulier, nous évoquons l'utilisation des contraintes dans les SE. Dans la seconde partie de ce chapitre, nous introduisons les formalismes des problèmes de satisfaction de contraintes dynamiques (DCSP : Dynamic Constraints Satisfaction Problems). Nous analyserons les formalismes proposés et nous montrerons leur complémentarité avec les systèmes experts. Enfin, nous conclurons ce chapitre en synthétisant les limitations des systèmes experts et en présentant des perspectives de renforcement de ces systèmes par les CSP.

2.1 Renforcement des systèmes experts par des CSP

Le concept du rapprochement des CSP avec les systèmes experts a commencé avec l'apparition des systèmes à base de contraintes et la programmation par contraintes. L'utilisation des contraintes dans le domaine des systèmes experts n'est pas une idée nouvelle, mais remonte à la naissance des langages de programmation logique comme Prolog dont certaines variantes comme Gnu-Prolog utilisent des méthodes de résolution issues des techniques de satisfaction de contraintes. De même, tout le domaine CLP (Constraint Logic Programming [88]) définit des programmes logiques qui contiennent des clauses dont les prémisses sont constituées de contraintes. De nombreux travaux de recherche ont été effectués dans les deux directions autant du côté des SE que des CSP pour élargir le formalisme, pour améliorer les techniques de résolution et l'efficacité d'exécution. Dans cette section, nous abordons le sujet crucial de l'hybridation entre systèmes experts et CSP. La question clé est de savoir si les modèles dont nous disposons permettent d'analyser et de faire coopérer de façon satisfaisante les SE et les CSP.

2.1.1 Le projet KRAFT

Nous trouvons la première tentative de rapprochement entre les deux modèles dans le projet nommé Knowledge Reuse And Fusion Transformation (KRAFT) [64, 153]. Ce projet a été lancé entre les universités de Aberdeen, Cardiff, Liverpool et Britsh Telecom vers la fin des années 90. Il vise à fusionner des données et des contraintes de différentes sources, dans un environnement distribué. Les bases de données et les solveurs sont contrôlés par un médiateur. Le principe de cette architecture consiste en premier lieu, à localiser et à contrôler la validité des contraintes et des données dans un environnement distribué et hétérogène. Ensuite, ces connaissances sont converties dans une syntaxe spécifique et homogène pour générer un problème composite. Avant de fusionner les contraintes, il faut réécrire les contraintes conformément à un schéma d'intégration afin que toutes les contraintes utilisent les mêmes variables et les mêmes valeurs. Dans ce but, une procédure ("wrapper") effectue une réécriture déclarative des données en contraintes homogènes. Seules les contraintes qui peuvent être réécrites dans une syntaxe composite propre à l'intégration sont exportées et partagées. Puis les contraintes sont placées sur différents nœuds du réseau (architecture Kraft) et résolues par un solveur de contraintes.

2.1.2 Utilisation des techniques de consistance dans les systèmes experts

Cette approche présentée dans [108], consiste à mettre en œuvre des techniques de consistance pour calculer les solutions partielles d'un système expert. Le but d'une telle approche est de construire un modèle pour réduire la quantité de calculs effectués par un SE. On exploite la structure des connaissances spécifiques au domaine. Le calcul de la solution combine des techniques de la programmation logique (système expert) et de propagation de contraintes.

L'auteur démontre qu'un ensemble de clauses d'un système expert peut être représenté par des graphes ET/OU. Ensuite, il recherche dans ce graphe un sous-graphe représentant le chemin vers la solution en établissant au préalable une consistance de nœud, d'arc et de chemin. Cette technique qui est en fait la technique de propagation de contraintes permet de filtrer les arcs consistants du système et d'accéder à l'objectif final en un temps réduit. De plus cette méthode va permettre d'avoir un système plus consistant et plus stable. En cas d'échec sur un nœud inconsistant du graphe, l'auteur utilise un algorithme de backtrack qui permet de remonter par ses nœuds parents à un nœud consistant.

2.1.3 Utilisation des techniques de consistance pour la validation de la base de connaissances

Dans la même direction, dans [14], les auteurs s'intéressent à la consistance de la base de connaissances et plus précisément à la consistance d'une fusion de bases de connaissances. Le problème est de déterminer la consistance lors de la fusion et de la mise en interaction de plusieurs bases de connaissances. La méthode proposée consiste à construire de façon incrémentale une nouvelle base de connaissances à partir des bases fusionnées puis d'utiliser des algorithmes de recherche locale sur les bases de connaissances instanciées. Le fait d'instancier les prédicats permet d'établir la consistance et la stabilité du système. Les résultats expérimentaux présentés dans [14] prouvent l'efficacité de cette méthode de résolution, pour construire une base de connaissance consistante et stable. En définitive, cette approche fusionne plusieurs bases de connaissances de systèmes experts en conservant leur consistance puis utilise des méthodes proches des techniques de résolution de CSP sur ces bases de connaissances instanciées.

2.1.4 Les CSP^T [113]

Simultanément, d'autres travaux ont été développés dans [114]. Ces travaux consistent à reformuler certains problèmes exprimés en logique du premier comme des problèmes de satisfaction de contraintes. La transformation d'un problème du calcul propositionnel, exprimé sous forme normale conjonctive en un CSP est relativement simple [113] : on transforme les variables propositionnelles en un ensemble de variables de CSP qui sont définies sur le domaine fini bi-valué $\{0, 1\}$, c'est-à-dire le domaine booléen $\{vrai\ ou\ faux\}$, avec la valeur 1 correspondant à la valeur $vrai$, et 0 à $faux$. Les clauses sont traduites en un ensemble des contraintes de cardinalité.

Définition 2.1 (Contrainte de cardinalité) *[113] On note une contrainte de cardinalité par $\# < \alpha, \beta, L_i >$, où α et β sont des entiers, et L_i est un ensemble de littéraux. La contrainte est satisfaite si au moins α et au plus β littéraux de L_i sont interprétés à vrai (avec $0 \leq \alpha \leq \beta \leq n$).*

Définition 2.2 *[113] Tout problème écrit dans le langage du calcul propositionnel sous forme normale conjonctive $P = \{c_1, c_2, \ldots, c_m\}$ composé de m clauses et n variables propositionnelles $\{x_1, x_2, \ldots, x_n\}$, peut être transformé en un problème de satisfaction de contraintes de type CSP^B défini par $CSP^B = (X, D, C)$, où :*

- *$X = \{x_1, x_2, \ldots, x_n\}$ est un ensemble de variables composé par l'ensemble des variables propositionnelles de l'ensemble P.*
- *$D = \{d_1, d_2, \ldots, d_n\}$ est composé des domaines booléens $\{Vrai,\ Faux\}$ associés aux variables de X.*
- *$C = \{c_1, c_2, \ldots, c_m\}$ est composé de m contraintes de cardinalité $\# < 1, k_i, L_i >$, en associant une contrainte de cardinalité à chaque clause c_i de P, de longueur k_i, et L_i représente l'ensemble des littéraux de la clause c_i .*

Le CSP^B est donc le résultat de la transformation d'un problème écrit naturellement sous forme normale conjonctive en langage propositionnel. Une autre approche CSP^T consiste à étendre le formalisme des CSP^B afin de traiter des problèmes du calcul des prédicats avec les CSP. Dans cette approche, certaines contraintes sont représentées sous forme de clauses unitaires qui permettent de maintenir la consistance logique et d'autres sont représentées sous forme de contraintes de cardinalité afin d'exprimer les clauses non unitaires [114]. Les contraintes de cardinalité définies ci-dessus permettent de borner le nombre de littéraux interprétés à vrai. Bien entendu, le CSP^T tel qu'il est défini ne couvre pas la notion de flexibilité du problème initial exprimé dans la logique du premier ordre. Face à ce besoin de flexibilité des données, l'extension du formalisme CSP^T est imposée. Cette extension nécessite l'évolution des domaines

des variables durant la résolution. Les valeurs des domaines de variables sont représentées par des n-uplets qui associent les variables aux prédicats "vrai" ou "faux". Il en résulte un formalisme appelé CSP^T *étendu* qui est présenté dans [113]. La seule différence des CSP^T *étendus* par rapport au formalisme CSP^T se situe dans le fait que les domaines évoluent durant la résolution. Les défis de ce formalisme sont la difficulté de satisfaire les contraintes de cardinalité et la réactivation de toutes les contraintes de cardinalité à chaque changement d'un domaine.

Nous venons donc de voir que les recherches de liens entre les SE et CSP sont considérables. Toutefois, la question cruciale est exprimée par la manière d'exploiter et d'utiliser les deux modèles (SE et CSP) dans la pratique. C'est pourquoi il est impératif d'examiner les questions suivantes : Quelles sont les types de données formalisées par les CSP et les types de données formalisées par les SE? Lors de la fusion ou de la collaboration de ces deux systèmes, comment faut-il classifier et structurer l'ensemble des données pour bénéficier des avantages de chaque modèle? Quels sont les critères pour justifier l'échange et la transmission des données? Quelles sont les dispositions liées à la protection et à la sécurité des données?

Ces questions sont incontournables pour la conception d'un système hybride qui combine les deux noyaux et nous donnerons des éléments de réponse dans les chapitres 3 et 4.

Il est essentiel de remarquer que, dans les SE, les faits sont modifiés et mis à jour régulièrement alors que les CSP classiques sont statiques. Pour cette raison, nous nous sommes intéressés aux CSP dynamiques et dans la section suivante, nous rappelons les principaux modèles de CSP dynamiques (DCSP) présentés dans la littérature scientifique. Ces DCSP constituent par leurs caractéristiques dynamiques inhérentes, un modèle plus proche des SE.

2.2 CSP dynamiques (DCSP : Dynamic Constraint Satisfaction Problem)

Le formalisme DCSP est particularisé par des contraintes ou des variables dynamiques activées ou désactivées durant la résolution. Ce formalisme a été introduit afin de répondre aux exigences des problèmes qui réclament des environnements dynamiques, comme la conception, la configuration, la planification ou encore l'ordonnancement. Les nombreuses variantes de DCSP proposées visent à remplir certains de ces besoins. Nous allons ici nous focaliser sur la description de chacune de ces variantes.

2.2.1 Les DCSP

Dans [98] Mittal et Falkenhainer ont été les premiers à proposer une approche pour gérer la pertinence des variables dans un modèle à base de contraintes. Ils ont défini les Dynamic CSP (DCSP), comme un formalisme qui permet l'autorisation ou l'interdiction de la participation de variables grâce à l'introduction des contraintes d'activité. Un DCSP est défini par un ensemble de variables (qui comporte un sous-ensemble de variables initialement actives) et un ensemble de contraintes. Chaque variable peut prendre une valeur choisie dans le domaine qui lui est associé. Les contraintes, quant à elles, sont divisées en deux sous-ensembles qu'on appelle les contraintes de compatibilité et les contraintes d'activité. L'objectif est d'attribuer une valeur à chaque variable active de sorte que toutes les contraintes soient satisfaites.

Définition 2.3 (DCSP) *Un DCSP est un quadruplet* (X, X_i, D, C) *où :*

- X *est un ensemble de* n *variables ;*
- X_i *est un ensemble* $\{x_1, x_2, ..., x_m\}$ *de variables initialement actives, tel que* $m < n$ *et* $X_i \subset X$ *;*
- D *est un ensemble de domaines finis* $\{d_1, d_2, ..., d_n\}$*, tel que chaque* d_i *est associé à la variable* x_i *de* X *;*
- C *est un ensemble de contraintes, tel que* $C = C_a \cup C_c$ *où* C_a *contient les contraintes d'activité responsables de l'activation et de la désactivation des variables, et* C_c *est un sous-ensemble de contraintes classiques, appelées les contraintes de compatibilité.*

Définition 2.4 (solution d'un DCSP) *Dans un DCSP, une solution est une instanciation de toutes les variables* actives *telle que toutes les contraintes de compatibilité et toutes les contraintes d'activité sont satisfaites.*

Ce formalisme DCSP apporte une grande flexibilité qui constitue l'un de ses principaux attraits et qui réside dans la possibilité d'activer et de désactiver des variables grâce aux contraintes d'activité. Par contre, il convient d'attirer l'attention sur les problèmes que génère cette flexibilité durant la résolution et la vérification de la validité des contraintes, en particulier, des contraintes d'activité. Les deux sous-ensembles de variables actives et inactives évoluent durant la résolution et ces évolutions diffèrent selon le type de contraintes impliquées. Mittal et Falkenhainer ont introduit une typologie des contraintes d'activation en les classifiant en quatre catégories [98] : (1) Require variable (RV), (2) Always require variable (ARV), (3) Require not (RN), et (4) Always require not (ARN).

1. **La contrainte RV :** active une variable lorsqu'un prédicat dépendant d'un ensemble d'autres variables est "vrai". On note ce type de contrainte de la façon suivante :

$$C_a(RV) : P(x_i, \ldots, x_j) \underset{RV}{\Longrightarrow} x_k \, (avec \; x_k \notin \{x_i, \ldots, x_j\})$$
$$\Longleftrightarrow$$
$$C_a(RV) : P(x_i, \ldots, x_j) \xrightarrow{\text{ACT}} x_k$$

   En d'autres termes, le prédicat $P(x_i, \ldots, x_j)$ déclenche l'activité de la variable x_k si et seulement si toutes les variables sur lesquelles porte le prédicat $x_i, \ldots, x_j$ sont actives et que $P(x_i, \ldots, x_j)$ est satisfait.

2. **La contrainte ARV :** active une variable lorsqu'un ensemble d'autres variables sont activées. Ce type de contrainte prend la forme :

$$C_a(ARV) : x_i \wedge \ldots \wedge x_j \underset{ARV}{\Longrightarrow} x_k (avec \; x_k \notin \{x_i, \ldots, x_j\})$$
$$\Longleftrightarrow$$
$$C_a(ARV) : \{x_i = v_{i,1} \vee \ldots \vee x_i = v_{i,m_i}\} \wedge$$
$$\ldots \wedge$$
$$\{x_j = v_{j,1} \vee \ldots \vee x_j = v_{j,m_j}\} \xrightarrow{\text{ACT}} x_k$$
$$\text{où } m_i \text{ est la taille du domaine } d_i$$

   L'activité des variables $\{x_i, \ldots, x_j\}$ déclenche l'activité de la variable x_k. Ce type de contrainte considère uniquement l'activité de certaines variables (et non leur instanciation) afin d'activer ou non d'autres variables.

3. **La contrainte RN :** désactive ou empêche l'activation d'une variable lorsqu'un prédicat dépendant d'un ensemble d'autres variables est "vrai". Ce qui peut être noté sous la forme suivante :

$$C_a(RN) : P(x_i, \ldots, x_j) \underset{RN}{\Longrightarrow} x_k (avec \; x_k \notin \{x_i, \ldots, x_j\})$$
$$\Longleftrightarrow$$
$$C_a(RN) : P(x_i, \ldots, x_j) \xrightarrow{\neg ACT} x_k$$

   On remarque que ces types de contraintes servent à empêcher l'activation des variables. Donc, un sous-ensemble des variables actives peut engendrer la désactivation d'une variable.

4. **La contrainte ARN :** désactive ou empêche l'activation d'une variable

lorsqu'un ensemble d'autres variables sont activées :

$$C_a(ARN) : x_i \wedge \ldots \wedge x_j \underset{ARN}{\Longrightarrow} x_k (avec\ x_k \notin \{x_i, \ldots, x_j\})$$
$$\Longleftrightarrow$$
$$C_a(ARN) : \{x_i = v_{i,1} \vee \ldots \vee x_i = v_{i,m_i}\} \wedge$$
$$\ldots \wedge$$
$$\{x_j = v_{j,1} \vee \ldots \vee x_j = v_{j,m_j}\} \xrightarrow{\neg ACT} x_k$$
où m_i est la taille du domaine d_i

Ces contraintes fonctionnent de la même façon que les contraintes ARV qui modélisent l'activation des variables lorsqu'une condition est vérifiée mais elles sont responsables de la désactivation des variables dès qu'un sous-ensemble de variables sont actives.

Ces nouveaux types de contraintes (contraintes d'activité) permettent d'étendre les problèmes pouvant être résolus par des CSP à une gamme très large de problèmes, en particulier, les problèmes qui sont de nature évolutive. En revanche, la modélisation du problème se complique par les relations de dépendance entre les contraintes. De plus, la satisfaction de ce type de contraintes est coûteuse en temps d'exécution puisque le problème devient sensiblement plus difficile quand on souhaite construire un algorithme pour ce formalisme. Une autre définition a été proposée par Soininen et Gelle dans [144] permettant de mélanger les contraintes de type RV, ARV, RN et ARN pour établir des prédicats autorisant ou interdisant l'activation d'une variable, en se basant sur une combinaison de l'activité, de l'inactivité "et" ou "ou" de la valeur d'autres variables. Ce formalisme est basé sur le même principe de contraintes d'activité et de compatibilité, mais utilise des algorithmes générant un nouveau CSP « classique » à chaque nouvelle activation ou désactivation de variables. L'intérêt est alors de pouvoir utiliser dans un processus itératif les algorithmes usuels de résolution de CSP au prix de devoir réinitialiser et redémarrer le processus de résolution à chaque activation ou désactivation de variable.

Entre ces contraintes d'activité et les règles de production d'un SE, il existe une ressemblance de nature fondamentale qui respecte l'ensemble des propriétés que l'on souhaite voir satisfaites tant par le SE que par le DCSP. Néanmoins, cette analyse n'a jamais été signalée ni par la communauté CSP ni par la communauté SE. Nous y reviendrons à la fin de cette section, après avoir présenté différentes variantes de DCSP.

2.2.2 Les CCSP [128]

Les CCSP (Composite CSP) ont été introduits dans l'article [128] par Sabin et Freuder. La principale spécificité est l'ajout de sous-problèmes grâce à des méta-variables, ce qui permet d'établir une hiérarchie de variables, de contraintes et de sous-problèmes et ainsi, de gérer la pertinence de ces éléments. Les variables et les contraintes qui appartiennent à un sous-problème n'existent pas nécessairement dans le problème de départ.

Un CCSP est défini par un triplet (X, D, C), de la même façon qu'un CSP "classique". Cependant, la différence entre CCSP et CSP « classique » est que les valeurs des variables peuvent être des sous-problèmes entiers (et pas seulement des valeurs "standards").

Définition 2.5 (CCSP) *Un CCSP est défini par un problème* $P = (X, D, C, R)$. *Soit le sous-problème* $P' = (X', D', C', R')$ *avec* $P' \in d_i$. *Lorsque* x_i *prend pour valeur* P', *le problème* P *à résoudre est modifié et devient,* $P" = (X", D", C", R")$ *avec :*

- $X" = X' \cup X \setminus x_i$
- $D" = D' \cup D \setminus d_i$
- $C" = C' \cup C \setminus c(x_i)$, *où* $c(x_i)$ *est l'ensemble des contraintes sur* x_i.
- $R" = R' \cup R \setminus r(x_i)$, *avec* $r(x_i$ *est l'ensemble des relations sur* x_i.

Les variables dont les valeurs sont des sous-problèmes entiers sont appelés des méta- variables. Lorsqu'une méta-variable est valuée, elle est remplacée dans le problème par le sous-problème correspondant à cette valeur. La structure du modèle est donc augmentée dynamiquement de l'ensemble des variables et des contraintes présentées dans le sous-problème.

2.2.3 Les CSP* [2] :

Les CSP* [2], essentiellement utilisés dans des problématiques de conception assistée par ordinateur, proposent de gérer la pertinence d'une variable par l'ajout d'une valeur spéciale "*" au domaine. Cette méthode n'ajoute pas d'éléments au cours du traitement du problème. Elle fait donc partie des CSP conditionnels. La valeur "*" est ajoutée dans le domaine des variables. Ainsi, lorsqu'une variable prend la valeur "*", cela signifie que cette variable ne participe pas à la solution du CSP ; elle est inactive au sens de Mittal et Falkenhainer.

2.2.4 Les CSPe [152] :

Les auteurs de l'article [152] ont introduit les CSPe (CSP à états) pour permettre une prise en compte explicite de la pertinence des variables. En asso-

ciant à chaque variable, une variable d'état qui permet de gérer explicitement l'activité de la variable.

Définition 2.6 (CSPe) *Un CSPe est défini par un triplet (X, D, C) tel que :*
- *X est un ensemble de variables comprenant chacune un attribut d'état ;*
- *D est l'ensemble des domaines de définition des variables de X ;*
- *C est un ensemble de contraintes sur les variables de X.*

Définition 2.7 (contrainte d'un CSPe) *Une contrainte dans un CSPe est une condition logique sur les valeurs ou les états d'un ensemble de variables noté $(x_i, x_j, \ldots)$. Une contrainte est donc une formule logique $F(x_i, x_j, \ldots)$ avec, pour chaque variable impliquée dans la formule, une condition sur la valeur de cette variable ou sur son attribut d'état.*

Le concept de CSPe se base sur les DCSP de [98], en réifiant l'activité des variables. Cette activité est attachée à un attribut d'état booléen (actif ou inactif). Il est possible de contraindre dans une même formule logique des conditions sur l'état des variables et sur leurs valeurs.

2.2.5 Les ACSP [48]

Les ACSP (Activity CSP) sont eux aussi basés sur les DCSP. Ils font donc également partie des CSP conditionnels. Les ACSP utilisent des notions développées auparavant pour gérer la pertinence d'éléments du CSP avec des variables d'activité.

Définition 2.8 (ACSP) *Un ACSP est un sextuplet (X, X_i, X_a, D, C, A) , où :*
- *Les symboles X, X_i et D sont équivalents à ceux d'un DCSP (variables, variables initialement actives et domaines) ;*
- *X_a est un ensemble de variables d'activité (dont le domaine est booléen), avec $X_a \subset X_i$;*
- *A est l'ensemble des conditions d'activation pour chaque variable non active initialement. Autrement dit, A associe à toute variable $x \in X \backslash X_i$ une condition d'activation $A(x)$ exprimée en fonction des variables d'activité de X_a ;*
- *C est un ensemble de contraintes. On ne distingue plus les contraintes de compatibilité des contraintes d'activité puisque toute contrainte peut porter sur une variable d'activité.*

Définition 2.9 (pertinence) *Toute variable non initialement active est associée à une variable d'activité dont le domaine est booléen. Une variable est dite*

pertinente si elle appartient à X_i ou qu'elle est associée à une variable d'activité égale à "vrai". On dit qu'une contrainte est pertinente si elle porte sur des variables pertinentes.

Définition 2.10 (solution ACSP) *Une solution S d'un ACSP est une valuation d'un ensemble de variables X_s ($X_s \subset X$) tel que :*

- *Toutes les variables pertinentes sont valuées ;*
- *Toutes les contraintes pertinentes sont satisfaites.*

La pertinence d'une variable doit être vue du point de vue de l'utilisateur. Ainsi, certaines variables peuvent ne participer à aucune solution du problème. Ces variables sont considérées comme non-pertinentes. Les ACSP explicitent la notion de pertinence par le biais du jeu de variables X_a. Par rapport aux CSPe, les ACSP ne cherchent pas à attribuer un attribut d'état à chaque variable, mais chaque élément (contrainte ou groupe) non initialement actif est associé à une variable d'activité. Ils permettent ainsi de modéliser des paramètres ou composants optionnels, mais aussi des sous-problèmes définis hiérarchiquement, grâce à la factorisation de la variable active. Dans leur article, Geller et Veksler ont montré l'équivalence du modèle ACSP avec les CCSP.

2.3 Techniques de résolution des DCSP

Rappelons que dans certaines définitions de DCSP, un DCSP est vu comme une suite incrémentale de CSP classiques. Résoudre un CSP consiste donc à résoudre successivement une suite de CSP classiques. Les algorithmes de résolution dédiés aux CSP classiques ne sont pas adaptés au traitement de ces problèmes dynamiques. En effet, toute modification conduit à la définition d'un nouveau problème qui est traité par le redémarrage du processus de recherche [57]. On peut donc appliquer directement les algorithmes de résolution connus pour les CSP classiques de manière récursive. En revanche, lorsque le problème à résoudre est très difficile ou lorsque de nombreuses modifications interviennent, une telle approche devient très coûteuse. Une autre façon de résoudre les DCSP est de proposer des techniques pour prendre en compte la dynamicité du problème. Les premiers travaux sont basés sur l'approche standard proposée dans [98]. Elle limite la résolution à trouver les solutions minimales. Une solution minimale est trouvée lorsque toutes les variables actives sont instanciées et aucune contrainte d'activité ne peut être activée. Une façon de prendre avantage des solutions déjà trouvées dans les étapes précédentes de la résolution est de procéder au mécanisme de mémorisation. La mémorisation de l'information est classifiée en deux catégories [133] : la mémorisation des solutions et la mémorisation des nogoods.

Les méthodes basées sur la mémorisation des solutions exploitent ces solutions durant l'évolution du problème, c'est-à-dire en fonction de l'activation et de la désactivation des contraintes ou des variables. Une première méthode, décrite dans [97], exploite l'heuristique de minimisation des contraintes insatisfaites. Dans [151], les auteurs ont proposé une méthode appelé "local change" qui utilise l'heuristique qui consiste à traiter en priorité les variables dont la valeur est incompatible avec la valeur instanciée à la variable courante.

D'autres méthodes sont basées sur la mémorisation des nogoods. Cette approche ne se contente pas de mémoriser tout simplement les nogoods durant la résolution mais elle effectue également la mémorisation de l'ensemble des contraintes qui sont responsables de la génération de ce nogood, afin d'exploiter ces informations pour valider temporairement les nogoods lorsque les mêmes contraintes apparaissent. Ces algorithmes sont décrits dans [150, 151].

2.4 Conclusion

Indépendamment du formalisme, un CSP dynamique est caractérisé par l'ajout où la suppression de variables, de valeurs des domaines ou de contraintes. Une question essentielle est celle des équivalences entre ces différents modèles.

Les DCSP ont été conçus en ajoutant aux CSP classiques des contraintes d'activité qui sont responsables de l'activation et la désactivation des variables. La vérification de ces contraintes nécessite donc de prendre en compte l'activité des variables et de leur contraintes associées durant la résolution.

Puisque l'activité des variables des DCSP est matérialisée par le symbole * dans les CSP*, par les variables d'états des CSPe et par les variables d'activité dans les ACSP, et puisque l'équivalence entre les ACSP et les CCSP a été établie dans [48]. On peut en déduire que tous les formalismes de CSP dynamiques proposés : DCSP, CCSP, CSP*, CSPe, ACSP sont équivalents et n'apportent pas une nouveauté par rapport aux DCSP classiques. Ils offrent simplement un autre moyen de représentation des données du problème tout en présentant les mêmes caractéristiques de calcul et de résolution.

En revanche, les CSP^T permettent d'exprimer certains problèmes de la logique du premier ordre dans le formalisme CSP, et la présence des nouvelles déductions logiques impose l'élargissement de ce formalisme au "CSP^T étendu".

Nous remarquons que les différents formalismes DCSP et les modes de représentation classiques d'un SE constituent des moyens d'exprimer des connaissances. Certains problèmes sont mieux exprimés par un formalisme que par l'autre. Par exemple les problèmes combinatoires sont mieux exprimés et structurés par les DCSP que par les SE. Par conséquent les DCSP et le SE sont deux alternatives qui vont permettre d'exprimer et de résoudre différents type de problèmes au mieux.

Nous avons évoqué dans ce chapitre les liens et les travaux existants entre des systèmes experts et des CSP, en donnant leurs principales propriétés. Nous avons présenté par la suite les différentes variantes des CSP dynamiques qui constituent des approches plus proches du formalisme d'un SE.

Néanmoins, les CSP dynamiques ne permettent pas de couvrir tous les problèmes qui peuvent être modélisés sous forme d'un système expert. Dans le prochain chapitre, nous décrirons une architecture dont l'objectif est d'hybrider et de combiner le moteur d'inférence d'un système expert et le noyau de résolution d'un CSP. Puis dans le chapitre 4, nous définirons un nouveau formalisme qui nous permettra de réaliser cette hybridation.

Chapitre 3

Vers des systèmes à noyaux multiples

We must encourage the commercialization of technologies that are compatible with existing infrastructure. What makes plug-in hybrids promising is that they are ; we don't need a Manhattan Project to make this happen.

James Woolsey

Dans ce chapitre, nous proposons une architecture générale basée à la fois sur le formalisme d'un CSP et le formalisme d'un SE. Cette architecture mixte nous permet de définir trois approches d'hybridation : filtrante, cyclique et concurrente. En premier lieu, nous décrivons l'architecture globale à noyaux multiples, puis nous discutons les différentes possibilités de communication et l'intérêt pratique de cette hybridation. En second lieu, nous présentons le mécanisme d'inférence hybride, et nous donnons le principe de ces différentes approches. Puis, nous analysons ces approches en soulignant les propriétés de chacune.

3.1 Motivations pour un système à noyaux multiples

Certains problèmes peuvent être modélisés en même temps soit par le formalisme CSP, soit par le formalisme de système expert. En terme d'efficacité, certains types de problèmes seront traités plus efficacement par l'un ou l'autre solveur. Par exemple, des problèmes de diagnostiques médicaux sont souvent traités par des systèmes experts alors que des problèmes de configuration ou combinatoires gagnent à être modélisés et résolus par des CSP.

Dans la pratique, il existe aussi des problèmes spécifiques qui revêtent à la fois des aspects combinatoires et des fonctionnements régis par des règles.

Imaginons par exemple un jeu d'échec un peu particulier entre deux joueurs A et B. Ce jeu se joue sur un échiquier divisé en $n*n$ cases. Nous supposons que A dispose de n reines, et B dispose de n cavaliers. Les pièces sont dispersées sur l'échiquier. Les reines et les cavaliers se déplacent et se menacent conformément aux règles standards du jeu d'échec classique, où la reine se déplace horizontalement, verticalement et en diagonale, et le cavalier lui se déplace selon un L majuscule, qu'il y ait un obstacle ou non.

Initialement les pièces sont dispersées sur l'échiquier de façon à trouver la meilleure configuration pour chaque joueur. Il est très clair que la meilleure configuration pour chaque joueur est celle qui permet d'assurer le recouvrement de l'échiquier. Trouver une meilleure disposition des pièces d'un joueur est un problème qui peut être modélisé sous forme d'un CSP. Par contre, les règles de déplacement et de capture suivent, elles, des règles plus naturellement modélisées sous forme de règles de production. A chaque temps t, toutes les pièces du joueur A effectuent un et un seul mouvement, puis au temps suivant, ce sera le tour du joueur B. Le but est de capturer le maximum de pièces du joueur adverse. L'objectif est donc de recouvrir l'échiquier de manière efficace.

Les mouvements des pièces engendrent deux problèmes de nature distincte. A chaque instant, la capture des pièces du joueur adverse peut être traitée de manière efficace avec un système expert. En effet, la capture est une simple vérification de la règle de déplacement. Les autres pièces qui ne peuvent pas capturer une pièce puisqu'elles ne sont pas dans une position adéquate, doivent effectuer un mouvement de manière à trouver une meilleure configuration en tenant compte des mouvements des autres pièces du joueur et de la disposition des pièces du joueur adverse. Il est très clair que ce jeu nécessite une alternance d'appel aux deux solveurs (SE et CSP) à chaque mouvement.

Ce jeu est donc une illustration qui permet de montrer que cette architecture à noyaux multiples a un intérêt pratique dans plusieurs applications, en particulier dans le domaine de la théorie des jeux.

3.2 Architecture à noyaux multiples

Nous présentons dans cette section notre première contribution qui consiste à élaborer une architecture, appelée architecture à noyaux multiples, parce qu'elle est constituée de deux noyaux de résolution, et nous montrons par la suite les différentes variantes possibles de coopération. Notre but en proposant cette approche est motivé par l'existence de problèmes complexes composites, qui contiennent plusieurs parties dont certaines sont mieux exprimées et plus faciles à résoudre par un noyau que par l'autre. L'objectif de ce travail est donc de renforcer les noyaux l'un par l'autre, en faveur d'un meilleur équilibre.

3.2.1 Présentation du principe

L'idée principale est de renforcer l'architecture standard et le moteur d'inférence d'un système expert avec un deuxième moteur de résolution basé sur les CSP. Nous avons restreint notre intérêt aux systèmes experts à base de règles.

Le problème qui se pose est donc de définir comment faire coopérer ces deux moteurs. Le formalisme CSP permet de représenter les données de types combinatoires et comme on l'a vu dans l'exemple introductif peut améliorer les résultats des systèmes experts.

L'architecture d'un système à noyaux multiples représentée par la figure 3.1 peut être structurée selon trois parties :

- la construction et la mise à jour des connaissances selon le mode de représentation de connaissances approprié constituent la première partie du système ;
- La deuxième partie est constituée par le double noyaux de raisonnement : un moteur d'inférence standard d'un système expert et un solveur standard de CSP ;
- le stockage et le partage des données durant la résolution, le choix entre les deux noyaux, et l'échange d'informations entre ces deux noyaux constituent la troisième partie.

Dans ce schéma, on retrouve les bases de faits et de règles usuelles des systèmes experts. On voit cependant que les deux moteurs ont chacun accès à ces bases. La base de décision permet d'orienter le traitement vers l'un ou l'autre des deux noyaux. L'étude détaillée de cette base de décision dépasse le cadre de notre étude et ne sera pas abordée dans ce document.

Dans notre système les connaissances sont représentées sous forme d'un ensemble de règles de production et d'un ensemble de variables avec des domaines finis qui peuvent évoluer durant la résolution.

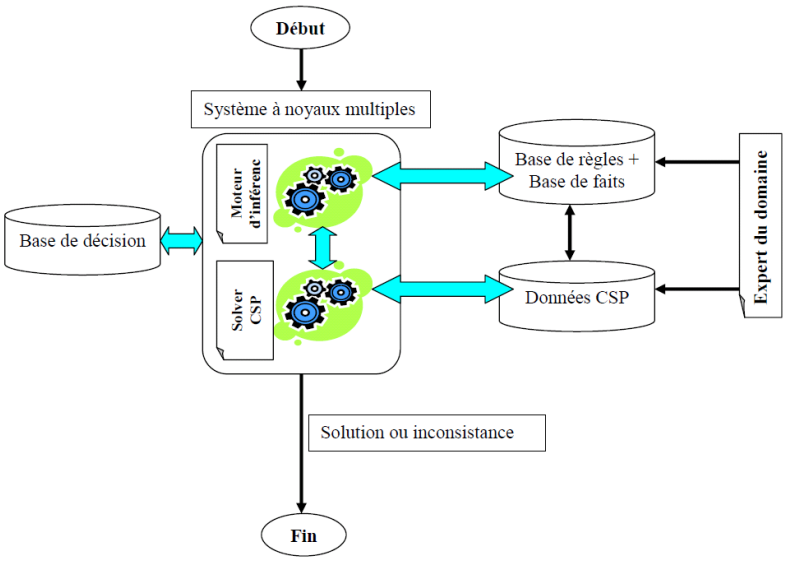


FIGURE 3.1 – Architecture d'un système à noyaux multiples

3.2.2 Fonctionnement général du système

Nous pouvons diviser le fonctionnement de ce système en deux tâches : la représentation des connaissances et le raisonnement.

- **Représentation des connaissances :** La représentation des connaissances constitue une phase de pré-traitement pour l'élaboration du système. Ces connaissances peuvent être structurées au choix sous forme de règles de production ou sous forme d'un ensemble de variables à domaines finis liées par un ensemble de contraintes.
- **Raisonnement :** L'objectif du raisonnement est de manipuler les connaissances stockées de façon à pouvoir répondre aux questions posées au système. Ce qui revient à dire qu'on cherche une affectation des variables impliquées dans la question de l'utilisateur.

3.3 Le mécanisme d'inférence hybride

Le mécanisme d'hybridation définit le moyen d'interaction entre les deux formalismes. Ces interactions permettent de faire des échanges entre les deux modèles durant la résolution et nécessitent une stratégie efficace et performante de collaboration. Les faits établis par un solveur peuvent se révéler utiles pour l'autre solveur. En échangeant des informations entre eux, les solveurs peuvent

s'entraider et ainsi déterminer plus rapidement la solution. Trois types d'hybridation peuvent être envisagées : l'hybridation filtrante, où un solveur constitue un filtre pour l'autre solveur, l'hybridation concurrente qui consiste à lancer en parallèle les deux solveurs afin de répondre à la même question le plus rapidement possible, et enfin l'hybridation cyclique qui consiste à échanger en cours de traitement des informations entre les deux solveurs afin d'accélérer la recherche de résultats.

3.3.1 Hybridation filtrante

Un problème est résolu séquentiellement par les deux noyaux. Par exemple, le noyau de résolution CSP peut assurer un pré-traitement de données destinées au moteur du SE. On peut dans ce type d'hybridation rappeler les travaux de [108] qui vérifie l'arc consistance au sein du moteur d'inférence. A l'inverse, le pré-traitement du graphe de contraintes pourrait être effectué par le système expert pour identifier les variables pertinentes qui participeront à la solution du CSP. D'une certaine manière, les règles d'activation des DCSP peuvent être vues comme des règles de production d'un système expert qui vont filtrer les variables et les contraintes utiles avant leur résolution par le solveur CSP. Cette approche d'intégration est fondée sur des interactions du type pré-traitement ou post-traitement, où les sous-systèmes sont reliés entre eux dans une architecture en pipeline. En effet, la résolution d'un problème peut être composée de plusieurs étapes de résolution. Chaque étape comporte des spécificités et peut être traitée par des outils appropriés. L'hybridation filtrante proposée permet d'organiser les différents modules de résolution et de les exploiter au mieux de leur performance : l'établissement de consistance pour les solveurs CSP et le calcul de déductions pour les moteurs d'inférence. De plus, dans un environnement parallèle, cela permet d'optimiser la résolution de plusieurs problèmes en tirant avantage des architectures pipeline à la manière des composants électroniques.

3.3.2 Hybridation concurrente

Dans cette approche, les deux noyaux : le moteur d'inférence et le solveur CSP travaillent de manière séparée sur le même problème jusqu'à ce que l'un des noyaux trouve une solution. Bien entendu, cette hybridation n'a de sens que dans un environnement parallèle ou distribué d'une part, et d'autre part, le même problème doit être modélisé dans les deux formalismes afin d'être traité par les deux noyaux. Néanmoins, si on dispose des procédures de transformation d'un modèle dans le second, l'exploitation de cette hybridation est directe. Un mécanisme de transformation fait l'objet du chapitre 4. Par contre, un inconvénient majeur réside dans la redondance de l'espace de recherche visité par les

deux noyaux et des traitements effectués. Une façon de réduire ces redondances serait de faire communiquer ces deux noyaux. Mais le surcoût induit par les communications risque de compenser le gain en efficacité.

3.3.3 Hybridation cyclique

Dans cette approche, les deux noyaux travaillent de manière collaborative jusqu'à ce que la solution soit trouvée. Ils sont des partenaires égaux dans le traitement à effectuer et coopèrent pour résoudre le problème. En cas de décomposition, l'un des deux noyaux pourra par exemple s'intéresser à un sous-problème particulier du problème initial, ou encore les deux pourront proposer des solutions qui seront ensuite évaluées par un noyau tiers (base de décision). Chaque noyau peut dans cette architecture, interagir directement avec la base de connaissance. Si nous considérons à nouveau notre système constitué de deux joueurs présenté au paragraphe 3.1. Cet exemple montre clairement la notion d'hybridation cyclique. Il utilise des appels alternés entre les formalismes pour atteindre le but.

Au cours de la résolution, les solutions partielles établies par le noyau CSP pourront être communiquées au noyau du système expert. De même en cours de fonctionnement, les nouveaux faits prouvés par le noyau du système expert pourront eux aussi être communiqués au noyau de résolution du CSP. Il s'avère donc nécessaire de définir un mécanisme de communication et d'adaptation des solutions d'un formalisme dans l'autre. Le mécanisme présenté dans le chapitre 4 pourra apporter une réponse.

3.4 Implantation

Ce système multi noyaux a été implémenté dans le langage de programmation Java, en utilisant une plateforme de développement Eclipse version 3.5.1. Dans notre système le mécanisme d'exploitation des connaissances est constitué par deux noyaux de résolution. Un noyau de résolution CSP (le solveur Choco) et un moteur d'inférence des systèmes à base de règles de production (Clips). Nous expliquons ici les composants généraux de notre système, nous reviendrons sur le concept des deux systèmes Choco et Clips par la suite. La plateforme du système est composée d'une base de décision qui permet l'orientation de la requête selon le type d'information recherchée. La base de décision apporte une réponse aux changements fréquents de données. Les données seront représentées soit dans un formalisme CSP sous forme d'un ensemble de variables, de domaines, et de contraintes, soit sous forme d'un ensemble de règles de production "si" *conditions* "alors" *action*. Le choix d'un de ces deux formalismes, est effectué

par l'expert du domaine d'application. Une interface (téléchargeable sur notre site web) a été conçue pour faciliter l'introduction des données et l'interrogation du système.

La figure 3.1 montre le schéma général de notre système. Notre procédé consiste, à partir des données introduites par un expert du domaine, à transformer un ensemble de règles de production sous forme d'un problème à base de contraintes et inversement. Ces informations seront exploitées pour résoudre le problème posé.

Pour la mise en œuvre de ce système, nous avons utilisé deux outils open-source : le solveur de CSP Choco et le moteur d'inférence Clips. Ces deux applicatifs sont téléchargeables sur le site Sourceforge, et offrent une API Java.

Choco

Choco est une librairie Java développée spécifiquement pour résoudre des problèmes de satisfaction de contraintes (CSP). Il est construit sur un mécanisme de propagation basé sur les événements avec des techniques de résolution complètes. Une version du code source est téléchargeable en ligne à l'adresse suivante [1]

Clips

CLIPS est un outil de développement de systèmes experts développé à l'origine par la Direction de la Technologie des Logiciels (STB), NASA/Lyndon B. Johnson Space Center. Depuis sa première version en 1986, de multiples améliorations ont été apportées à Clips. Il est développé en langage de programmation C/C++. Les données sont représentées par des règles de production. Il a aussi une grammaire qui lui est propre. Une version complète du code source est téléchargeable en ligne à l'adresse suivante [2].

3.4.1 Interface du système

Cette interface est conçue pour faciliter l'acquisition des données d'un expert. Elle offre aussi un accès simple aux données.

La figure 3.2 représente l'interface principale de notre système. Elle est complètement dédiée à la communication avec les experts afin de construire les données de système. Cette interface permet à l'utilisateur d'interroger cette base pour obtenir les informations qu'il souhaite. Le menu "System" permet soit de construire un nouveau système, ou bien d'ouvrir un système déjà existant. Le

1. https ://sourceforge.net/projects/choco/files/
2. http ://clipsrules.sourceforge.net/

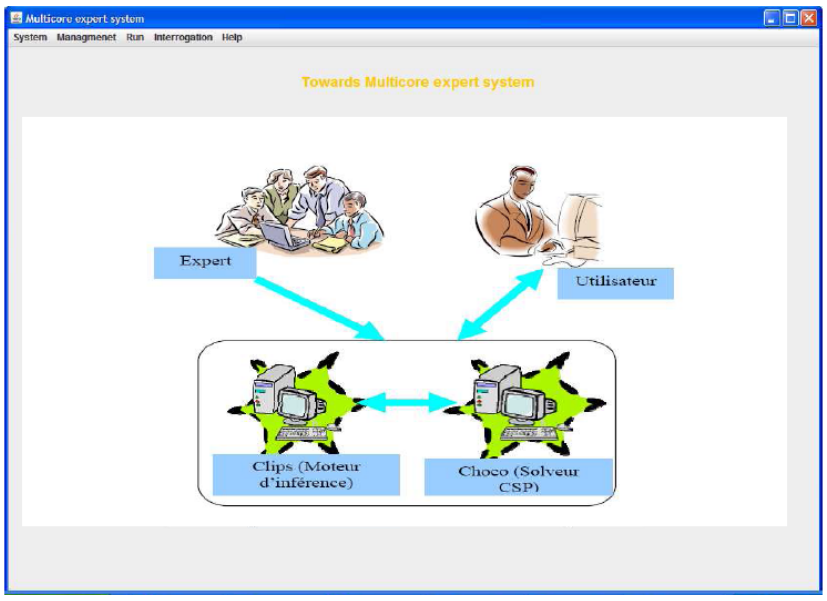


FIGURE 3.2 – Interface principale du système

menu "Management" effectue des différentes manipulations sur notre base de données d'un système. Le menu "Run" permet de lancer l'exécution de notre système en sélection la base de données sur laquelle nous allons travailler. Le menu "Interrogation" offre une interface d'interaction entre l'utilisateur et le système.

Nous trouverons une version de l'interface de notre système sur le site suivant [3]

La mise en œuvre de la communication entre le solveur Choco et le moteur d'inférence Clips est décrite dans le chapitre suivant après avoir décrit le mécanisme de communication mis en œuvre.

3. `http://www.lita.univ-metz.fr/~saad/MulticoreSystem`

3.5 Conclusion

Dans ce chapitre, nous avons présenté un système à noyaux multiples et ses principales caractéristiques. En résumé, on peut dire qu'un système à noyaux multiples est un système qui hybride le formalisme des systèmes expert à base de règles et le formalisme des CSP.

Cette hybridation induit plusieurs stratégies de coopération : filtrante, concurrente ou cyclique qui nécessitent le transfert des connaissances entres les deux modèles. Généralement les connaissances transférées sont des instanciations partielles. Dans le chapitre suivant, nous présentons un nouveau modèle qui permet de transformer automatiquement un système expert vers un modèle à base de contraintes.

Chapitre 4

Protocole de communication entre SE et CSP

The only possible interpretation of any research whatever in the social sciences' is : some do, some don't.
Ernest Rutherford

Dans ce chapitre, nous introduisons un nouveau formalisme à base de contraintes, nommé DDCSP (Dynamic Domain Constraint Satisfaction Problem). Ce formalisme se distingue par des domaines et des relations dynamiques. Nous étudierons certaines caractéristiques des DDCSP puis nous proposerons des techniques de résolution de ce formalisme. Puis, nous proposons une procédure de transformation automatique d'un système à base de règles de production vers un formalisme à base de contraintes. Cette transformation joue un rôle très important dans la communication entre le moteur d'inférence standard d'un système expert et le noyau de résolution des CSP.

4.1 Introduction

Dans un premier temps, nous définissons une procédure de transformation des problèmes modélisés par des règles d'inférence vers les CSP. A cette fin, une extension du formalisme CSP est proposée, nommée DDCSP (Dynamic Domain Constraint Satisfaction Problem). Les DDCSP permettent d'associer des domaines dynamiques aux variables du problème, et des relations dynamiques aux contraintes du problème pour prendre en compte un nouveau fait lorsqu'il est déduit en cours de résolution. Nous étudions par la suite les différentes propriétés des DDCSP. Nous proposons en outre deux mécanismes qui permettent l'adaptation des techniques de résolution standards des CSP classiques aux DDCSP.

Le principal argument pour motiver cette étude est de permettre de résoudre certains problèmes plus complexes en tirant avantage des algorithmes de résolution dédiés aux CSP et en particulier en concevant des architectures hybrides ainsi que des architectures distribuées et sécurisées. Comme nous venons de le voir dans le chapitre 1, il existe plusieurs représentation possibles de la base de connaissances. Nous restreignons notre étude aux problèmes modélisés par la logique des prédicats du premier ordre sans fonctions. Nous commençons par donner certaines définitions qui constituent un pré-requis. Puis nous décrivons les mécanismes de la transformation.

4.2 Définitions

Dans cette section, nous allons définir le vocabulaire qui décrit l'ensemble des concepts présents dans ce chapitre et leurs relations. Ces définitions sont extraites de [32, 26].

4.2.1 Logique du premier ordre

La logique du premier ordre est un formalisme fréquemment utilisé pour la modélisation de la connaissance dans les systèmes experts. Nous décrivons la syntaxe des formules de la logique du premier ordre et leur sémantique.

Syntaxe

Définition 4.1 *Un alphabet pour un langage du premier ordre consiste en un ensemble de symboles spécifiés par :*

- *Des connecteurs logiques :*
  - *Les connecteurs du calcul propositionnel* $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\neg$ *;*
  - *les quantificateurs* $\forall$ *("pour tout") et* $\exists$ *("il existe") ;*
  - *le symbole d'égalité (=).*
- *Des variables : un ensemble infini dénombrable de variables* $X = \{x_1, x_2, \ldots\}$ *;*
- *Des parenthèses "(" et ")" ;*
- *Des symboles de fonctions : un ensemble dénombrable éventuellement vide* $S_f = \{f, g, h, \ldots\}$ *de fonctions, ainsi qu'une fonction d'arité ar :* $S_f \mapsto \mathbb{N}^*$ *spécifiant le nombre d'arguments d'une fonction ;*
- *Des symboles de constantes : un ensemble dénombrable éventuellement vide* $S_c = \{a, b, c, \ldots\}$ *de constantes. Si a est une constante alors l'arité de a vaut* 0 *;*
- *Des symboles de prédicats : un ensemble dénombrable éventuellement vide* $S_p = \{P, Q, R, \ldots\}$*, ainsi qu'une fonction d'arité ar :* $S_p \mapsto \mathbb{N}$ *donnant le*

nombre d'arguments de chaque prédicat.

Définition 4.2 *L'ensemble des termes est défini inductivement comme suit :*
- *Les variables et les constantes sont des termes ;*
- *Si $t_1, \ldots, t_n$ sont des termes, si f est un symbole de fonction d'arité n, alors $f(t_1, \ldots, t_n)$ est un terme.*

Définition 4.3 *L'ensemble des formules est défini par :*
- *Si $t_1, \ldots, t_n$ sont des termes et si P est un symbole de prédicat tel que $ar(P) = n$ alors $P(t_1, ..., t_n)$ est une formule atomique ;*
- *Si t_1 et t_2 sont des termes alors $t_1 = t_2$ est une formule atomique ;*
- *Si A est une formule alors $\neg A$ est une formule ;*
- *Si A et B sont des formules alors $A \wedge B$, $A \vee B$, $A \Rightarrow B$, $A \Leftrightarrow B$ sont des formules ;*
- *Si A est une formule et si x est une variable, alors $\forall x A$ et $\exists x A$ sont des formules.*

Sémantique

Définition 4.4 *Une structure d'interprétation $SI = (E, I)$ pour un langage du premier ordre est la donnée d'un ensemble E appelé domaine et d'une fonction d'interprétation I associant des fonctions et des prédicats sur E aux symboles de fonctions et de prédicats de ce langage, telle que :*
- *Pour chaque symbole de fonction f d'arité n, $I(f) : E^n \mapsto E$;*
- *Pour chaque constante c, $I(c) \in E$.*
- *Pour chaque symbole de prédicat P d'arité n, $I(P) \subseteq E^n$;*

On se retrouve donc avec des symboles d'un côté, donnés par l'alphabet du langage du premier ordre et des objets de E (éléments, sous-ensembles de tuples, fonctions) de l'autre côté, donnés par la structure d'interprétation, qui permettent de donner une signification à ces symboles.

Définition 4.5 *L'affectation des valeurs aux variables est une fonction ϕ : $X \mapsto E$. On note $\phi[x := e]$ l'affectation de la valeur e à la variable x définie par :*
- *$\phi[x := e](y) = \phi(y)$ si $x \neq y$;*
- *$\phi[x := e](x) = e$;*

Définition 4.6 *La valeur d'un terme t par rapport à une structure d'interprétation $SI = (E, I)$ et à une affectation de valeurs aux variables ϕ, est notée $[t]_{SI,\phi}$. Cette valeur est définie inductivement par :*
- *$[x]_{SI,\phi} = \phi(x)$ si x est une variable*

- $[a]_{SI,\phi} = I(a)$ *si* a *est une constante*
- $[f(t_1, \ldots, t_n)]_{SI,\phi} = I(f)([t_1]_{SI,\phi}, \ldots, [t_n]_{SI,\phi})$ *si* f *est un symbole de fonction d'arité* n.

L'interprétation I sert donc ici à évaluer les symboles de fonctions et les constantes, alors que l'affectation de valeurs aux variables ϕ sert à évaluer les variables.

Définition 4.7 *La valeur de vérité d'une formule* A *par rapport à une structure d'interprétation* $SI = (E, I)$ *et une affectation* ϕ*, est l'application* $v(A, I, \phi)$ *dans* $\{vrai, faux\}$*. On définit l'application* v *de la manière suivante :*

- *Si* A *est un atome* $P(t_1, t_2, \ldots, t_n)$ *alors* $v(A, I, \phi) = vrai$ *ssi il existe* $(d_1, \ldots, d_n) \in I(P)$ *avec, pour tout* $i \in \{1, \ldots, n\}$, $d_i = [t_i]_{SI,\phi}$ *;*
- *Si* A *est une égalité entre deux termes* $t_1 = t_2$, $v(A, I, \phi) = vrai$ *ssi* $[t_1]_{SI,\phi} = [t_2]_{SI,\phi}$
- *Si* A *est une différence entre deux termes* $t_1 \neq t_2$, $v(A, I, \phi) = vrai$ *ssi* $[t_1]_{SI,\phi} \neq [t_2]_{SI,\phi}$
- *si* $A = \neg B$ *alors* $v(A, I, \phi) = NON(v(B, I, \phi))$*, la négation étant interprétée comme en logique des propositions ;*
- *si* $A = (B \wedge C)$ *alors* $v(A, I, \phi) = v(B, I, \phi)$ *ET* $v(C, I, \phi)$*, la conjonction étant interprétée comme en logique des propositions.*
- *si* $A = (B \vee C)$ *alors* $v(A, I, \phi) = v(B, I, \phi)$ *OU* $v(C, I, \phi)$*, la disjonction étant interprétée comme en logique des propositions.*
- *si* $A = (B \Rightarrow C)$ *alors* $v(A, I, \phi) = v(B, I, \phi)$ $IMPLIQUE$ $v(C, I, \phi)$*, l'implication étant interprétée comme en logique des propositions.*
- *si* $A = (B \Leftrightarrow C)$ *alors* $v(A, I, \phi) = v(B, I, \phi)$ $EQUIVALENT$ $v(C, I, \phi)$*, l'équivalence étant interprétée comme en logique des propositions.*
- *si* $A = \forall x B$ *alors* $v(A, I, \phi) = vrai$ *ssi pour tout élément* $d \in E$, $v(B, I, \phi[x := d]) = vrai$.
- *si* $A = \exists x B$ *alors* $v(A, I, \phi) = vrai$ *ssi pour un élément* $d \in E$, $v(B, I, \phi[x := d]) = vrai$.

Dans la suite de notre travail, nous nous restreignons notre étude à l'utilisation des termes atomiques, sans utilisation de symbole de fonction.

4.2.2 Définition des règles et des faits

Définition 4.8 (clause) *Une clause est une formule de la forme* $\forall t_1, \ldots \forall t_k$, $A_1 \vee \ldots \vee A_n \vee \neg B_1 \vee \ldots \vee \neg B_p$ *où les* A_i*,*B_i *sont des atomes. Les quantificateurs étant tous universels, ils sont souvent omis.*

Un atome est un littéral positif. Sa négation est un littéral négatif. La clause vide notée $\square$ ne contient aucun littéral. Elle est toujours fausse.

Définition 4.9 (clause de Horn) *Une clause de Horn contient au plus un littéral positif.*

Définition 4.10 (règle) *Une règle est une clause de Horn. Une règle peut donc s'écrire sous la forme* $B_1 \wedge \ldots \wedge B_n \Rightarrow A$, *où les* B_i *et* A *sont des atomes, ou bien sous la forme* $A \leftarrow B_1 \wedge \ldots \wedge B_n$.

Exemple 4.1 (Règle)
père$(x, y) \wedge$ *frère*$(y, z) \Rightarrow$ *père*(x, z)

Dans la suite de ce chapitre, les règles que nous utilisons sont des clauses de Horn. Par abus de langage, on les appellera règles de Horn. Si les littéraux L et $\neg L$ s'avèrent utiles, on introduit deux littéraux positifs L et NON_L et on rajoute la règle $L \wedge$ NON_L$\Rightarrow \square$.

Remarque 4.1
- Une règle est donc une formule de la forme "si Condition *alors* Conclusion*"*
- On se restreint dans la suite à l'étude des règles sûres (safe rules) où toute variable apparaît dans au moins un prédicat de la prémisse.

Définition 4.11 (fait) *Un fait est l'interprétation et l'affectation des arguments d'un prédicat* P. *Autrement dit un fait représente un tuple de valeurs qui rendent ce prédicat vrai.*

Exemple 4.2 (Faits)
père(Jacques,Charles)
frère(Charles,François)

Définition 4.12 (but) *Un but est défini par un ensemble d'atomes positifs qui sont reliés entre eux par le connecteur logique* $\wedge$.

Un but peut être considéré comme une question de l'utilisateur. L'objectif d'un système expert est de répondre à la question de l'utilisateur. Autrement dit, l'objectif d'un système expert est de trouver dans la base de connaissances des faits qui satisfont le but.

Exemple 4.3 (But)
? :- père(Jacques,Charles)
? :- père(x,y)

Définition 4.13 (arité de la règle) *On appelle arité de la règle la somme des arités des prédicats utilisés dans la règle.*

Exemple 4.4 (Arité) *L'arité de la règle* R : *père*$(x, y) \wedge$ *frère*$(y, z) \Rightarrow$ *père*(x, z) *est* 6 *et sera notée* $|R| = 6$.

4.3 Construction d'un nouveau CSP dynamique

Notre objectif est d'élaborer une procédure de transformation des problèmes du système expert écrits dans la logique des prédicats du premier ordre fini, en problèmes à base de contraintes. Pour parvenir à ce but, nous présentons d'abord un petit exemple puis nous décrivons une nouvelle classe de CSP dynamique nommée DDCSP. La suite de ce chapitre sera consacrée à l'étude des mécanismes de résolution et des propriétés des DDCSP.

4.3.1 Illustration sur un exemple

Soit un système expert à base de règles : $p_1(x, y) \wedge p_2(y, z) \Rightarrow p_1(x, z)$ et les faits $p_1(1, 2)$, $p_2(2, 3)$.

Ce SE peut être transformé en un CSP tel que

- $X = \{x, y, z\}$
- $D = \{d_x, d_y, d_z\}$ avec $d_x = \{1\}$, $d_y = \{2\}$, $d_z = \{3\}$
- $C = \{c_{p_1}, c_{p_2}\}$
- $R = \{r_1, r_2\}$ avec $r_1 = \{(1, 2)\}$ est l'ensemble des tuples autorisés de c_{p_1} et $r_2 = \{(2, 3)\}$ est l'ensemble des tuples autorisés de c_{p_2}

L'instanciation $x = 1$, $y = 2$, $z = 3$ est une solution de ce CSP. Lorsque cette solution est trouvée, la relation r_1 doit être modifiée dynamiquement pour lui ajouter le couple $(1, 3)$. Cette mise à jour dynamique de r_1 provoque également la mise à jour des domaines des variables x et y de la contrainte c_{p_1}.

La transformation que nous venons d'effectuer peut être généralisée et nous verrons qu'elle permet de transformer toute règle d'un SE à base de règles en un CSP dynamique dont les relations et les domaines évoluent en cours de résolution.

Dans la suite de cette section, nous donnons d'abord une définition du CSP dynamique obtenu que nous appelons DDCSP, puis nous prouvons que tout SE à base de règles peut être transformé en un DDCSP.

4.3.2 Définition d'un DDCSP

Définition 4.14 (eDDCSP) *Un eDDCSP (elementary dynamic domain constraint satisfaction problem) est défini par* $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$ *où* (X, D, C, R) *désigne un CSP, où* X_s *est un sous-ensemble éventuellement vide de* X, D_s *est l'ensemble des domaines des variables de* X_s *et* c_s *est une contrainte portant sur les variables de* X_s. *La relation* r_s *associée à* c_s *est initialement connue, éventuellement vide.*

Définition 4.15 (résoudre un eDDCSP) *Résoudre un eDDCSP* $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$, *c'est trouver une solution du CSP* (X, D, C, R). *Pour toute solution* σ *du CSP* (X, D, C, R), *la relation* r_s *est modifiée dynamiquement par* $r_s = r_s \cup \{\sigma[X_s]\}$ *où* $\sigma[X_s]$ *est la projection de* σ *sur* X_s. *On dit alors que* c_s *et* r_s *sont "deltadépendantes de* δ*" ou simplement "deltadépendantes".*

Exemple 4.5 (eDDCSP) *Soit* δ *un eDDCSP* $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$, *avec*
$X = \{x_1, x_2, x_3\}$
$D = \{d_1, d_2, d_3\}$,
$d_1 = \{a, b\}$, $d_2 = \{b, c\}$, $d_3 = \{b, c\}$
$C = \{c_1, c_2, c_3\}$
$c_1 = (x_1, x_2)$, $c_2 = (x_2, x_3)$, $c_3 = (x_1, x_3)$
$R = \{r_1, r_2, r_3\}$
$r_1 = \{(a, b), (b, b), (b, c)\}$, $r_2 = \{(b, c), (c, b)\}$, $r_3 = \{(a, c)\}$
$X_s = \{x_1, x_2, x_3\}$
$c_s = (x_1, x_2, x_3)$
$r_s = \emptyset$

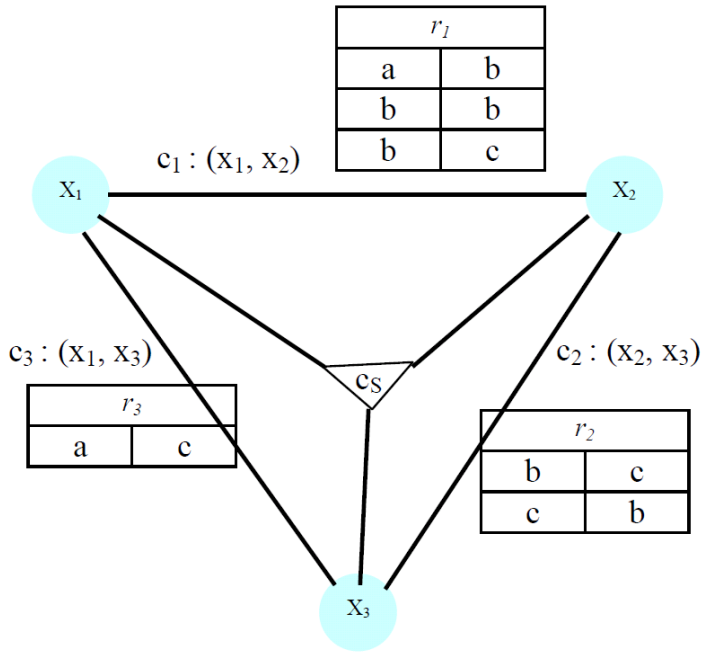


FIGURE 4.1 – Hypergraphe associé au eDDCSP de l'exemple 4.5

Ce eDDCSP est représenté dans la figure 4.1 par un hypergraphe dont les nœuds représentent l'ensemble des variables, les arcs représentent les contraintes binaires et les hyper-arcs schématisés par des polygones représentent les contraintes n-aires.

Dans cet exemple l'instanciation $\sigma = \{x_1 = a, x_2 = b, x_3 = c\}$ est consistante dans le CSP (X, D, C, R) dont elle est une solution. Pendant la résolution, la relation r_s est modifiée dynamiquement par $r_s = r_s \cup \{(a, b, c)\}$.

Exemple 4.6 (eDDCSP) *Modifions à présent l'exemple précédent. On a à présent un nouveau eDDCSP* $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$, *avec*
$X = \{x_1, x_2, x_3\}$
$D = \{d_1, d_2, d_3\}$
$d_1 = \{a, b\}$, $d_2 = \{b, c\}$, $d_3 = \{b, c\}$
$C = \{c_1, c_2\}$
$c_1 = (x_1, x_2)$, $c_2 = (x_2, x_3)$
$R = \{r_1, r_2\}$
$r_1 = \{(a, b), (b, b), (b, c)\}$, $r_2 = \{(b, c), (c, b)\}$
$X_s = \{x_1, x_3\}$
$c_s = (x_1, x_3)$
$r_s = \emptyset$

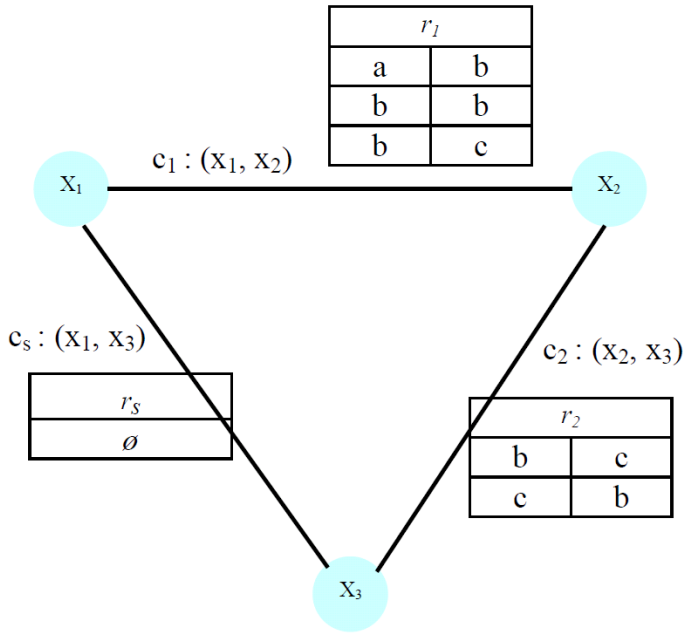


FIGURE 4.2 – Hypergraphe associé au eDDCSP de l'exemple 4.6

Ce eDDCSP est représenté dans la figure 4.2. Dans cet exemple les instanciations $\{x_1 = a, x_2 = b, x_3 = c\}$, $\{x_1 = b, x_2 = b, x_3 = c\}$ et $\{x_1 = b, x_2 = c, x_3 = b\}$ sont consistantes. Pendant la résolution, la relation r_s est modifiée dynamiquement par $r_s = r_s \cup \{(a, c), (b, c), (b, b)\}$.

Définition 4.16 (eDDCSP dépendant) *Un eDDCSP* $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$ *est dit dépendant si et seulement si l'une au moins des contraintes de* C *ou l'une au moins des relations de* R *est deltadépendante.*

Définition 4.17 (relation deltadépendance) *Soient deux eDDCSP* $\delta_1 = (X_1, D_1, C_1, R_1, X_{s_1}, D_{s_1}, c_{s_1}, r_{s_1})$ *et* $\delta_2 = (X_2, D_2, C_2, R_2, X_{s_2}, D_{s_2}, c_{s_2}, r_{s_2})$ *, on dit que* δ_1 *est deltadépendant de* δ_2 *si et seulement si l'une au moins des contraintes de* C_1 *est deltadépendante de* δ_2*.*

Dans un eDDCSP dépendant, les contraintes deltadépendantes sont dynamiques. En effet, conformément à la définition 4.15, de nouveaux tuples consistants sont ajoutés à une relation d'une contrainte deltadépendante chaque fois qu'une solution du eDDCSP associé est trouvée. Au contraire, dans un eDDCSP indépendant, par définition aucune contrainte de C n'est deltadépendante. La seule relation dynamique d'un eDDCSP indépendant est r_s. Toutes les contraintes de C sont donc statiques et on peut en déduire que $c_s \notin C$.

Les deux eDDCSP des exemples 4.5 et 4.6, considérés séparément, sont tous les deux indépendants contrairement au eDDCSP suivant :

Exemple 4.7 (eDDCSP dépendant) *Modifions à nouveau l'exemple 4.5. Et considérons le eDDCSP* $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$*, avec*
$X = \{x_1, x_2, x_3\}$
$D = \{d_1, d_2, d_3\}$
$d_1 = \{a, b\}$, $d_2 = \{b, c\}$, $d_3 = \{b, c\}$
$C = \{c_1, c_2\}$
$c_1 = (x_1, x_2)$, $c_2 = (x_2, x_3)$
$R = \{r_1, r_2\}$
$r_1 = \{(a, b), (b, b)\}$, $r_2 = \{(b, c)\}$
$X_s = \{x_1, x_3\}$
$c_s = (x_1, x_3)$
$r_s = r_1$

Ce eDDCSP est représenté dans la figure 4.3. Nous attirons l'attention sur le fait $r_s = r_1$ n'est pas une simple commodité d'écriture mais signifie que la relation r_s partage avec r_1 la même liste de couples autorisés. Donc modifier

r_s revient à modifier r_1 et réciproquement. Dans cet exemple les instanciations $\{x_1 = a, x_2 = b, x_3 = c\}$, $\{x_1 = b, x_2 = b, x_3 = c\}$ sont consistantes. Puisque $r_s = r_1$, pendant la résolution, la relation r_1 est modifiée dynamiquement par $r_1 = r_1 \cup \{(a, c), (b, c)\}$.

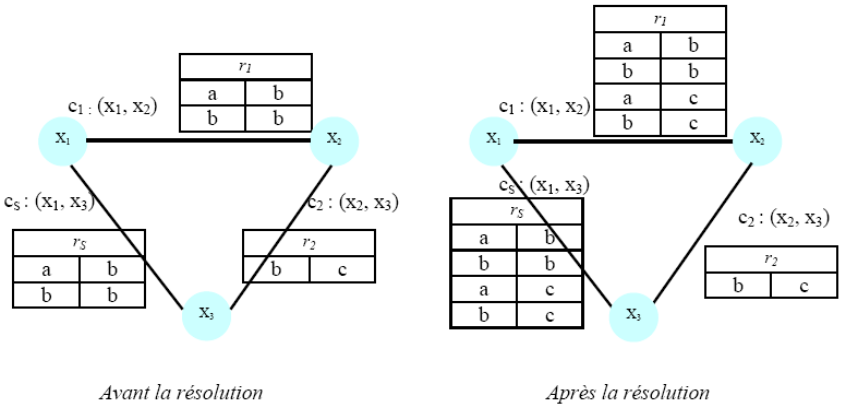


FIGURE 4.3 – Hypergraphe associé au eDDCSP de l'exemple 4.7 avant et après la résolution

Définition 4.18 (DDCSP) *Un DDCSP est défini par* (X, D, C, R, Δ), *où* $\Delta = \{\delta_0, \delta_1, \ldots, \delta_n\}$ *est un ensemble de eDDCSP, tels que* $\forall i \in \{0, \ldots, n\}$, $\delta_i = (X_i, D_i, C_i, R_i, X_{s_i}, D_{s_i}, c_{s_i}, r_{s_i})$, *avec* $X = \cup_{i=0}^{n} X_i$, $D = \cup_{i=0}^{n} D_i$, $C = \cup_{i=0}^{n} C_i \cup \{c_{s_0}, \ldots, c_{s_n}\}$ *et* $R = \cup_{i=0}^{n} R_i \cup \{r_{s_0}, \ldots, r_{s_n}\}$.

Définition 4.19 (résoudre un DDCSP) *Résoudre un DDCSP, c'est résoudre* δ_0.

Si $X_0 = \emptyset$, alors par convention on cherche toutes les solutions de tous les eDDCSP de Δ.

Exemple 4.8 (DDCSP) *Soit le DDCSP* (X, D, C, R, Δ) *de la figure 4.4, avec* $\Delta = \{\delta_0, \delta_1, \delta_2\}$, *tels que :* $\delta_0 = (X_0, D_0, C_0, R_0, X_{s_0}, D_{s_0}, c_{s_0}, r_{s_0})$, $\delta_1 = (X_1, D_1, C_1, R_1, X_{s_1}, D_{s_1}, c_{s_1}, r_{s_1})$, *et* $\delta_2 = (X_2, D_2, C_2, R_2, X_{s_2}, D_{s_2}, c_{s_2}, r_{s_2})$.

δ_0 *est défini par :*

$X_0 = \{x_{0_1}, x_{0_2}, x_{0_3}\}$
$D_0 = \{d_{0_1}, d_{0_2}, d_{0_3}\}$
$d_{0_1} = \emptyset$, $d_{0_2} = \emptyset$, $d_{0_3} = \emptyset$
$C_0 = \{c_{0_1}\}$
$c_{0_1} = (x_{0_1}, x_{0_2}, x_{0_3})$
$r_{0_1} = r_{s_2} = \emptyset$
$X_{s_0} = \emptyset$
$c_{s_0} = \emptyset$
$r_{s_0} = \emptyset$

δ_1 *est défini par :*

$X_1 = \{x_{1_1}, x_{1_2}, x_{1_3}\}$
$D_1 = \{d_{1_1}, d_{1_2}, d_{1_3}\}$
$d_{1_1} = \{a, b\}$, $d_{1_2} = \{b, c\}$, $d_{1_3} = \{b, c\}$
$C_1 = \{c_{1_1}, c_{1_2}\}$
$c_{1_1} = (x_{1_1}, x_{1_2})$, $c_{1_2} = (x_{1_2}, x_{1_3})$
$R_1 = \{r_{1_1}, r_{1_2}\}$
$r_{1_1} = \{(a, b), (b, b)\}$, $r_{1_2} = \{(b, c)\}$
$X_{s_1} = \{x_{1_1}, x_{1_3}\}$
$c_{s_1} = (x_{1_1}, x_{1_3})$
$r_{s_1} = r_{2_1}$

δ_2 *est défini par :*

$X_2 = \{x_{2_1}, x_{2_2}, x_{2_3}\}$
$D_2 = \{d_{2_1}, d_{2_2}, d_{2_3}\}$
$d_{2_1} = \{a, c\}$, $d_{2_2} = \{b\}$, $d_{2_3} = \{b, c\}$

$C_2 = \{c_{2_1}, c_{2_2}\}$
$c_{2_1} = (x_{2_1}, x_{2_2})$, $c_{2_2} = (x_{2_2}, x_{2_3})$
$R_2 = \{r_{2_1}, r_{2_2}\}$
$r_{2_1} = r_{s_1} = \{(a, b), (c, b)\}$, $r_{2_2} = \{(b, c)\}$
$X_{s_2} = \{x_{2_1}, x_{2_2}, x_{2_3}\}$
$c_{s_2} = (x_{2_1}, x_{2_2}, x_{2_3})$
$r_{s_2} = r_{0_1} = \emptyset$

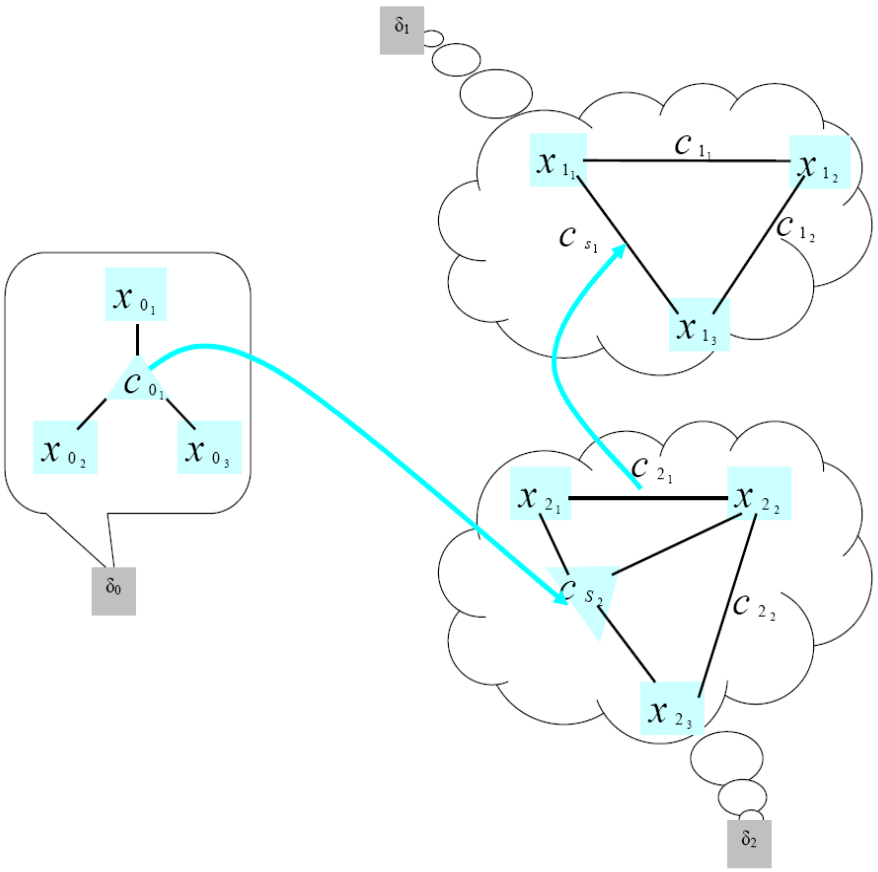


FIGURE 4.4 – Hypergraphe associé au DDCSP de l'exemple 4.8

La figure 4.4 est la représentation graphique de l'exemple 4.8. Ce DDCSP est constitué par deux eDDCSP principaux qui sont δ_1 *et* δ_2. Le eDDCSP δ_0 représente le but associé à la question de l'utilisateur. Donc, ce δ_0 peut varier en fonction de l'objectif souhaité. Les flèches représentent les relations de "delta-dépendance" des contraintes. On voit que δ_2 est deltadépendant de δ_1 puisque

la contrainte c_{2_1} dépend de la contrainte c_{s_1}. En effet, elles ont la même relation $r_{2_1} = r_{s_1}$. De même, δ_0 est deltadépendant de δ_2 car c_{0_1} dépend de c_{s_2} puisqu'elles ont même relation $r_{s_0} = r_{s_2}$.

Propriété 4.1 *Tout CSP est un eDDCSP indépendant.*

Preuve 4.1 *Soit un CSP $P = (X, D, C, R)$. P est égal au eDDCSP $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$ où X_s, D_s, c_s et r_s sont vides.*

Propriété 4.2 *Tout eDDCSP est un DDCSP.*

Preuve 4.2 *Soit un eDDCSP $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$ et sa relation associée r_s deltadépendante. On peut toujours construire le DDCSP $\delta' = (X', D', C', R', \Delta)$ où $\Delta = \{\delta\}$ et $X' = X$, $D' = D$, $C' = C \cup \{c_s\}$, $R' = R \cup \{r_s\}$. Résoudre δ' est équivalent à résoudre δ.*

4.4 La procédure de résolution des DDCSP

Les DDCSP sont spécifiques dans le sens où les relations des contraintes du problème évoluent dynamiquement. Dans cette partie nous allons décrire précisément comment satisfaire les contraintes d'un DDCSP.

A partir des méthodes de satisfaction, nous déduirons un algorithme général de résolution des DDCSP.

4.4.1 Satisfaction des contraintes

Une relation d'une contrainte c d'un DDCSP δ est une relation définie en extension. On dispose donc d'une liste r de tuples autorisés pour c. On dit qu'un tuple t de valeurs satisfait c si $t \in r$.

Remarque 4.2 *Si $t \notin r$, la contrainte c n'est pas satisfaite mais il n'est pas impossible que dans une étape ultérieure de la résolution, un nouveau fait vienne compléter la liste r. En effet si r est deltadépendante d'un eDDCSP , cette liste est par construction dynamique.*

On dit qu'un eDDCSP $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$ est satisfait si on a trouvé une instanciation consistante de toutes les variables du CSP (X, D, C, R). Dès lors, si X_s n'est pas vide, on active l'ajout du nouveau tuple dans la relation deltadépendante r_s.

Exemple 4.9 *Reprenons le DDCSP présenté dans l'exemple 4.8. Le but de ce DDCSP consiste à trouver une instanciation des variables $(x_{0_1}, x_{0_2}, x_{0_3})$ qui satisfait δ_0. La relation r_{0_1} associée à la contrainte c_{0_1} qui lie $(x_{0_1}, x_{0_2}, x_{0_3})$ est*

ici deltadépendante du eDDCSP δ_2. Avant la résolution la relation r_{0_1} égale à r_{s_2} est vide, donc la résolution consiste à déterminer les différents tuples possibles de r_{0_1}. Il faut donc résoudre δ_2. La résolution de δ_2 nous donne $r_{s_2} = r_{s_2} \cup \{(a,b,c),(c,b,c)\} = \{(a,b,c),(c,b,c)\} = r_{0_1}$. Mais δ_2 dépend de δ_1. La résolution de δ_1 permet de déduire $r_{s_1} = r_{s_1} \cup \{(a,c),(b,c)\} = \{(a,b),(c,b),(a,c),(b,c)\} = r_{2_1}$. On résout donc à nouveau δ_2. Cette nouvelle résolution de δ_2 laisse r_{s_2} inchangé. On n'a donc plus de nouvelles solutions et le DDCSP est stabilisé. A la fin de la résolution, on obtient donc $r_{0_1} = \{(a,b,c),(c,b,c)\}$

4.4.2 Résolution des DDCSP

L'objectif de ce paragraphe est de proposer un algorithme de résolution dédié à ce formalisme. L'arbre de recherche qui représente ce formalisme est spécifique par rapport à un arbre de recherche d'un CSP standard dans le sens où chaque nœud peut être caractérisé par trois états : les états "mise en attente", "satisfaction" et "conflit".

État "mise en attente" (pending) : cet état est déclenché par l'exploration totale de l'arbre de recherche du eDDCSP. Cet état n'implique pas la fin de l'exploration définitive du eDDCSP. En effet, nous avons déjà mentionné que lorsqu'un domaine a été exploré, il peut se produire que dans une étape ultérieure de la résolution, un nouveau tuple soit ajouté à une relation et de nouvelles valeurs consistantes sont dans ce cas détectées dans les domaines. L'exploration d'un arbre en état pending sera déclenchée à chaque modification d'une relation d'une contrainte portant sur une des variables de l'arbre. Suite à un état 'pending', la fin de l'exploration du DDCSP est détectée lorsque plus aucune modification de relation ne peut se produire. Dans ce cas, soit des solutions ont été trouvées et le DDCSP est satisfait, soit le DDCSP est inconsistant.

État "satisfaction" : cet état résulte de la satisfaction d'un eDDCSP, et implique la mise à jour de la relation deltadépendante du eDDCSP.

État "conflit" (failing) : cet état se produit en cours de résolution d'un eDDCSP lorsqu'il n'y a plus d'instanciation consistante possible d'une variable de X, et nécessite un retour arrière pour essayer de modifier les valeurs des variables déjà instanciées.

Propriété 4.3 *La résolution d'un DDCSP se termine.*

Preuve 4.3 *Il suffit de remarquer qu'en cours de résolution, chaque relation r est soit inchangée, soit remplacée par r' avec $r \subset r'$. Les relations ainsi obtenues forment une suite croissante d'ensembles finis et majorés par les produits cartésiens des domaines, donc d'après le théorème de Tarski cette suite converge vers un point fixe. Lorsque ce point fixe est atteint, toutes les relations sont statiques et le DDCSP est alors un CSP classique.*

Deux stratégies de résolution peuvent être envisagées en regard des propriétés de ce formalisme : la première stratégie consiste à explorer l'espace de recherche jusqu'à la fin puis à mettre à jour les relations qui ont été modifiées. Ensuite, le processus de recherche est réitéré. Ces procédures sont répétées jusqu'à la stabilité du système démontrée dans la propriété 4.3. La seconde stratégie consiste à modifier les relations en cours de résolution.

Stratégie 1

Le principe fondamental de cette stratégie est d'explorer l'ensemble de l'arbre de recherche des différents eDDCSP, de construire une liste des modifications à effectuer, puis de réitérer la résolution sur l'espace de recherche global tant que la solution associée à δ_0 n'est pas encore trouvée. Ce processus est réitéré jusqu'à ce qu'on atteigne une stabilité (qui correspond à la saturation des relations des variables).

Dans l'algorithme 1, on s'attache à ne résoudre que les eDDCSP qui peuvent potentiellement apporter de nouvelles solutions.

Pour ce faire, on construit un ensemble *DeltaSet* initialisé avec tous les eDDCSP δ_i.

En cours de résolution, cet ensemble *DeltaSet* est mis à jour avec les eDDCSP dont les relations deltadépendantes ont été modifiées et qui sont donc susceptibles d'avoir de nouvelles solutions.

La boucle **Pour** des lignes 5 à 7 recherche les solutions des eDDCSP en utilisant un solveur CSP standard matérialisé par l'appel de la procédure de résolution "Resolution_CSP" qui est une procédure de résolution classique des CSP statiques.

Le test des lignes 8 à 10 permet de vérifier qu'une solution de δ_0 a été trouvée et le cas échéant s'arrête. Pour trouver toutes les solutions du DDCSP, il suffit de supprimer ce test et le booléen *Trouve* de la condition de réitération de la ligne 24.

La boucle **Pour** des lignes 13 à 21 permet de mettre à jour les relations deltadépendantes et les domaines des variables associées par l'appel de la fonction $Mise_a_jour$. Cette fonction a pour objet d'ajouter des nouveaux tuples dans la relation r_{s_i} et dans les domaines des variables liées par cette relation. Le paramètre de sortie *Set* de cette fonction contient les eDDCSP deltadépendants de δ_i impactés par cette mise à jour et qui devront donc être à nouveau résolus.

La condition de réitération de la ligne 24 est composée de deux variables booléennes *Fin* et *Trouve* dont les valeurs correspondent aux états définis plus haut et dont le tableau 4.1 dresse un résumé.

Fin	$Trouve$	Etat
Faux	Faux	Pending
Vrai	Faux	Failing
Faux	Vrai	Satisfaction
Vrai	Vrai	Impossible

TABLE 4.1 – Correspondance entre les valeurs des booléens Fin et $Trouve$ et les états Pending, Failing, Satisfaction

---

Détail de l'algorithme de la stratégie 1 :
procédure Résoudre($P : DDCSP$, $Y : Liste_solution$, $Trouve$: booléen)
Entrée : un DDCSP (X, D, C, R, Δ) avec $\Delta = \{\delta_0, \delta_1, ..., \delta_n\}$;
Sortie : Y est la liste des solutions de δ_0 ; Trouve est vrai si Y n'est pas vide et faux sinon ;

1. **Début**
2. $Fin = Faux$, $Trouve = Faux$, $Y = \emptyset$
3. $DeltaSet = \{\delta_0, \delta_1, ..., \delta_n\}$
4. **Répéter**
5. **Pour** $\delta_i \in DeltaSet$ **faire**
6. $Liste_solution_{\delta_i}$ = Resolution_CSP(δ_i)
7. **FinPour**
8. **si** $Liste_solution_{\delta_0} \neq \emptyset$ **alors**
9. $Trouve = Vrai$
10. **sinon**
11. $Fin = Vrai$
12. $DeltaSet' = \emptyset$
13. **Pour** $\delta_i \in DeltaSet$ **faire**
14. **si** $Liste_solution_{\delta_i} \neq \emptyset$ **alors**
15. Mise_a_jour(δ_i, $Liste_solution_{\delta_i}$, Set)
16. **si** $Set \neq \emptyset$ **alors**
17. $Fin = Faux$
18. $DeltaSet' = DeltaSet' \cup Set$
19. **fsi**
20. **fsi**
21. **FinPour**
22. $DeltaSet = DeltaSet'$
23. **fsi**
24. **Jusqu'à** (Fin **ou** $Trouve$)
25. **si** (Trouve) **alors**
26. Y =$Liste_solution_{\delta_0}$
27. **fsi**
28. **Fin**

Stratégie 2

Cette stratégie est caractérisée par le redémarrage immédiat du processus de recherche à chaque fois qu'une relation est modifiée. La figure suivante illustre cette stratégie.

Détail de l'algorithme de la stratégie 2 :
procédure Résoudre(P : $DDCSP$, Y : $Liste_solution$, $Trouve$: booléen)
Entrée : un DDCSP (X, D, C, R, Δ) avec $\Delta = \{\delta_0, \delta_1, ..., \delta_n\}$;
Sortie : Y est la liste des solutions de δ_0 ; $Trouve$ est vrai si Y n'est pas vide et faux sinon ;
Début

1. $Fin = Faux$, $Y = \emptyset$, $Trouve = Faux$
2. $DeltaSet = \{\delta_0, \ldots, \delta_n\}$
3. **Tantque** (**Non** Fin **et Non** $Trouve$ **et** ($DeltaSet \neq \emptyset$)) **faire**
4. $Fin = Vrai$
5. δ_i = selection($DeltaSet$)
6. $DeltaSet = DeltaSet - \delta_i$
7. $Liste_solution_{\delta_i}$ = Resolution_CSP(δ_i)
8. Mise_a_jour($\delta_i, Liste_solution_{\delta_i}, Set$)
9. **si** $Set \neq \emptyset$ **alors**
10. $Fin = Faux$
11. $DeltaSet = DeltaSet \cup Set$
12. **fsi**
13. $Y = Liste_solution_{\delta_0}$
14. $Trouve = (Y \neq \emptyset)$
15. **FinTantque**
16. **Fin**

L'algorithme 2 utilise les mêmes fonctions que l'algorithme 1 et leurs structures sont proches. La différence majeure entre les deux approches réside dans le fait que :

- on sélectionne un seul $delta_i$ de l'ensemble $DeltaSet$;
- on effectue la résolution CSP standard sur ce δ_i ;
- l'ensemble des eDDCSP deltadépendants de ce δ_i est ajouté à l'ensemble $DeltaSet$.

De plus, dans cet algorithme apparaît une fonction **selection** qui choisit dans l'ensemble $DeltaSet$ le δ_i à étudier. Plusieurs stratégies de sélection peuvent être envisagées de la plus simpliste (prendre le premier $delta_i$ de l'ensemble) à la plus

subtile (prendre le δ_i qui a le plus de contraintes et donc la plus haute probabilité d'échec). Les heuristiques peuvent être nombreuses mais ne feront pas l'objet d'une étude dans la suite de ce travail.

4.4.3 Discussion

Nous avons proposé deux types d'approches pour la résolution des DDCSP.

Dans les deux stratégies, les ensembles de eDDCSP à traiter *DeltaSet* évoluent de manière différente. Dans tous les cas, à chaque étape un eDDCSP $(X, D, C, R, X_s, D_s, c_s, r_s)$ est susceptible d'être remplacé par un eDDCSP $(X, D', C, R', X_s, D'_s, c_s, r'_s)$. Ce qui signifie que, entre deux étapes, seules les relations et les domaines de chaque eDDCSP seront éventuellement modifiés. Ainsi en cours de résolution, un arbre de eDDCSP est construit.

La stratégie 1 consiste à résoudre à chaque étape tous les eDDCSP. On obtient ainsi un nouvel ensemble de eDDCSP modifiés qui seront à leur tour tous résolus à la prochaine étape. D'une certaine manière, cette stratégie implémente un parcours en largeur de l'arbre des eDDCSP.

Alors que dans la stratégie 2, on a une liste de eDDCSP. On résout séquentiellement les éléments de la liste. Et dès qu'une modification apparaît, on met à jour la liste et on redémarre au début de la liste. Donc cette stratégie implémente une sorte de parcours en profondeur d'abord de l'arbre des eDDCSP.

La stratégie 1 a l'avantage de permettre une parallélisation naturelle en exploitant l'indépendance de tous les eDDCSP. Dans cette stratégie parallèle, l'ensemble des domaines et des relations des eDDCSP est en mémoire partagée. Chaque eDDCSP est résolu de façon indépendante par un appel à la fonction **Resolution_CSP**. Les solutions obtenues sont transmises à un superviseur qui met à jour les variables et les relations par l'appel de la fonction **Mise_a_jour**. La terminaison de l'algorithme est détectée par la stabilité des relations (l'absence de modifications) et l'état Pending de tous les processus de résolution. En cas de modification de relations, le superviseur détermine les eDDCSP concernés et le cas échéant active leur résolution.

4.5 Transformation d'un système expert en un DDCSP

Nous décrivons à présent le détail de la procédure de transformation des systèmes experts écrits sous forme des règles de Horn sûres en un problème de satisfaction de contraintes.

Propriété 4.4 *Soit ρ une règle sûre exprimée en logique du premier ordre fini. ρ peut être transformée en un eDDCSP δ.*

Preuve 4.4 *Soit ρ une clause de Horn. ρ peut donc s'écrire sous la forme $B_1 \wedge \ldots \wedge B_n \Rightarrow A$, où les B_i et A sont des atomes. Donc ρ peut s'écrire $B_1(x_{1,1}, \ldots, x_{1,k_1}) \wedge B_2(x_{2,1}, \ldots, x_{2,k_2}) \wedge \ldots \wedge B_n(x_{n,1}, \ldots, x_{n,k_n}) \Longrightarrow A(x_{n+1,1}, \ldots, x_{n+1,k_{n+1}})$*

Soit le eDDCSP défini par $\delta = (X, D, C, R, X_s, D_s, c_s, r_s)$ avec :

- *X est l'ensemble des variables de la règle. C'est à dire que $X = \{x_{1,1}, \ldots, x_{n,k_n}\}$. En effet, puisque la règle est sûre, les variables de la conclusion apparaissent nécessairement dans les prémisses.*
- *On associe un domaine à chaque variable donc $D = \{d_{x_{1,1}}, \ldots, dx_{n,k_n}\}$, où chaque $d_{x_{i,j}}$ est construit dynamiquement à partir des faits de la base de faits. Plus précisément, si $B(a_1, \ldots, a_k)$ est un fait, si $a_1, \ldots, a_k$ sont des valeurs, et si $B(x_1, \ldots, x_k)$ est un atome de la règle, alors on ajoute les valeurs $a_1, \ldots, a_k$ respectivement aux domaines de $x_1, \ldots, x_k$.*
- *$C = \{c_{B_1}, c_{B_2} \ldots c_{B_n}\}$. À chaque prédicat B_i est associée une contrainte $c_{B_i} = (x_{i,1}, \ldots, x_{i,k_i})$.*
- *$R = \{r_{B_1}, r_{B_2} \ldots r_{B_n}\}$. À chaque contrainte est associée une relation construite à partir des tuples de valeurs utilisées dans la base de faits pour le prédicat associé. Plus précisément, si $B(a_1, \ldots, a_k)$ est un fait, alors le tuple $(a_1, \ldots, a_k)$ est ajouté à r_B.*
- *$X_s = \{x_{n+1,1}, \ldots, x_{n+1,k_{n+1}}\}$ est l'ensemble des variables de la conclusion de ρ. On a bien $X_s \subset X$ puisque ρ est une règle de production sûre.*
- *$D_s = \{d_{x_{n+1,1}}, \ldots, d_{x_{n+1,k_{n+1}}}\}$.*
- *$c_s = \{c_A\}$ est la contrainte associée au prédicat de la conclusion de ρ.*
- *r_s est la liste éventuellement vide des tuples autorisés de c_s.*

Remarque 4.3 *Lorsqu'un prédicat apparaît plusieurs fois dans des règles (par exemple $\exists(i,j)|B_i = B_j$), plusieurs contraintes distinctes lui seront associées (c_{B_i} et c_{B_j}). En revanche, ces contraintes seront liées par une unique relation (r_B).*

Propriété 4.5 *Soit ϕ un fait d'un système expert à base de règles de Horn sûres en logique du premier ordre fini. ϕ peut être transformé en un tuple de valeurs appartenant à une relation associée à une contrainte d'un DDCSP.*

Preuve 4.5 *D'après la preuve constructive de la propriété 4.4, $B(a_1, \ldots, a_n)$ est un fait de la base de faits si et seulement si $(a_1, \ldots, a_n)$ est un tuple de valeurs de la relation r_B associée à B.*

Propriété 4.6 *Tout système expert $S = (\mathcal{R}, \mathcal{A})$ exprimé par un ensemble $\mathcal{R}$ de règles de Horn sûres en logique du premier ordre fini et par un ensemble de faits $\mathcal{A}$ peut être transformé en un DDCSP P. On dit que P est le DDCSP associé à S.*

Preuve 4.6 *La preuve constructive est directe :*

- *On renomme les variables de règles différentes afin que deux règles distinctes ne partagent jamais la même variable.*
- *On applique respectivement les propriétés 4.4 et 4.5 à chaque règle de $\mathcal{R}$ et à chaque fait de $\mathcal{A}$.*
- *Lorsque plusieurs contraintes sont associées au même prédicat, on associe à ces contraintes distinctes la même liste de tuples autorisés.*

On obtient ainsi un DDCSP associé à S.

Propriété 4.7 *Tout but $\mathcal{B}$ peut être transformé en un eDDCSP δ_0.*

Preuve 4.7 *On rappelle qu'un but $\mathcal{B}$ d'un système expert $S = (\mathcal{R}, \mathcal{A})$ est défini par un ensemble d'atomes positifs qui sont reliés entre eux par le connecteur logique $\wedge$ et par le quantificateur $\exists$. Un but a la forme d'une condition d'une règle. On peut donc toujours construire le eDDCSP défini par $\delta_0 = (X, D, C, R, X_s, D_s, c_s, r_s)$ associé au but $\mathcal{B}$ en appliquant la procédure de transformation décrite en 4.4. On aura ici X_s, D_s, c_s et r_s vides.*

Remarque 4.4 *Dans le cas particulier où un prédicat du but $\mathcal{B}$ est de la forme $P(t_1, ..., t_n)$ où certains termes $t_1, ..., t_n$ sont des constantes, on peut toujours se ramener à un prédicat qui ne contient que des variables en remplaçant chaque constante t_i par une variable x_i et en rajoutant la contrainte d'égalité $x_i = t_i$.*

Propriété 4.8 *Soit S un système à base de règles et P son DDCSP associé. Soit ρ une règle de S et soit $\delta = (X, D, C, R, X_s, D_s, r_s, c_s)$ le eDDCSP associé à ρ. On peut appliquer la règle ρ dans S si et seulement si (X, D, C, R) admet une solution.*

Preuve 4.8 *(X, D, C, R) admet une solution si et seulement si toutes les variables de X sont instanciées et vérifient toutes les contraintes de C. Donc il existe un tuple consistant pour chaque contrainte de C. D'après la propriété 4.5, chacun de ces tuples correspond à un fait de la condition de ρ. La condition de ρ est donc vérifiée par ces tuples puisque l'unification dans ρ est garantie par la conservation de l'égalité des variables de ρ lors de la construction du eDDCSP.*

Propriété 4.9 *Soit S un système expert exprimé par un ensemble de règles de Horn sûres en logique du premier ordre fini, soit P son DDCSP associé et $\mathcal{B}$ son but. S admet une solution si et seulement si P admet une solution.*

Preuve 4.9 *Une solution d'un système expert $S = (\mathcal{R}, \mathcal{A})$ associé à la question $\mathcal{B}$ peut être vue comme une liste de faits qui appartiennent à la base de connaissances de S et qui répondent à la question $\mathcal{B}$. D'après la propriété 4.4, toute règle r de S est associée à un eDDCSP δ_r de P.*

D'après les propriétés 4.5 et 4.8, un nouveau fait dans S est déduit de r, par construction, si et seulement si la condition de la règle r est vérifiée. Autrement dit si et seulement si une solution de δ_r est trouvée et qu'un nouveau tuple est ajouté à la relation δ_r-dépendante.

On vient donc d'établir qu'un nouveau fait est déduit dans le système expert si et seulement si on trouve un nouveau tuple dans une relation du DDCSP.

On en déduit qu'un fait vérifie un but $\mathcal{B}$ si et seulement si le tuple correspondant est solution du eDDCSP associé à $\mathcal{B}$.

On en déduit l'équivalence entre S et P.

Exemple 4.10 *Nous reprenons l'exemple 4.8. Donc soit le système expert suivant :*
$p_1(1,2)$
$p_2(2,3)$
$R_1 : p_1(x,y) \wedge p_2(y,z) \Longrightarrow p_1(x,z)$
et le but associé $p_1(1,3)$.
En appliquant la règle R_1, on déduit $p_1(1,2) \wedge p_2(2,3) \Longrightarrow p_1(1,3)$ d'où $p_1(1,3)$ est prouvé.
De même nous allons résoudre le DDCSP associé à ce problème

- $X = \{x, y, z\}$;
- $D = \{d_x, d_y, d_z\}$ *avec* $d_x = \{1\}$, $d_y = \{2\}$, $d_z = \{3\}$;
- $C = \{c_{p_1}, c_{p_2}\}$ *avec* $c_{p_1} = (x, y)$ *et* $c_{p_2} = (y, z)$;
- $R = \{r_1, r_2\}$ *avec* $r_1 = \{(1,2)\}$ *est l'ensemble des tuples autorisés de* c_{p_1} *et* $r_2 = \{(2,3)\}$ *est l'ensemble des tuples autorisés de* c_{p_2} ;
- $X_s = \{x, z\}$;
- $D_s = \{d_x, d_z\}$ *avec* $d_x = \{1\}$, $d_z = \{3\}$;
- $c_s = (x, z)$
- $r_s = r_1$

Pendant la résolution, la solution $x = 1, y = 2, z = 3$ du CSP (X, D, C, R) est trouvée. Donc le tuple $(1,3)$ est rajouté à la relation r_s ce qui équivaut à le rajouter à la relation r_1. On en déduit que $p_1(1,3)$ est vérifié.

On souhaite dans la suite de cette section comprendre les liens entre la consistance des DDCSP et les systèmes experts à base de règles. L'objectif principal des techniques de filtrage est la détection des affectations partielles inconsistantes. Pour cela, toutes les valeurs des variables sont testées afin de détecter et d'éliminer celles qui ne peuvent jamais mener à une solution. Il

existe différents degrés de filtrage selon le nombre de variables participant à l'affectation partielle. Plus celui-ci est élevé, meilleur sera le filtrage, mais plus le temps consacré à l'élimination des valeurs de variables incompatibles avec une solution risque d'être long.

Propriété 4.10 *Soit S un système à base de règles et P son DDCSP associé. Soit ρ une règle de S et soit $\delta = (X, D, C, R, X_s, D_s, r_s, c_s)$ le eDDCSP associé à ρ. Appliquer la règle du modus ponens sur une instanciation de ρ équivaut à trouver une solution de (X, D, C, R).*

Preuve 4.10 *Conséquence directe de la propriété 4.8.*

Remarque 4.5 *On peut construire le graphe orienté de dépendance du DDCSP noté $\mathcal{G}$. Dans $\mathcal{G}$, les sommets sont les eDDCSP et un arc entre deux eDDCSP (δ_i, δ_j) exprime que δ_i est deltadépendant de δ_j. Lorsqu'on résout un DDCSP, on cherche toutes les solutions de δ_0. δ_0 appartient à une composante connexe de $\mathcal{G}$. Les eDDCSP à l'extérieur de cette composante connexe n'ont aucune influence sur la résolution de δ_0. On peut donc effectuer un préfiltrage avant la résolution afin de se limiter aux DDCSP de la composante connexe de δ_0.*

Propriété 4.11 *Soit ρ une règle d'un SE et soit $\delta = (X, D, C, R, X_s, D_s, r_s, c_s)$ le eDDCSP associé à ρ. Si l'arité de ρ est k, alors établir la k-consistance forte de (X, D, C, R) permet de résoudre δ avec un algorithme glouton.*

Preuve 4.11 *Il suffit de remarquer que $|X| \leq k$. D'après la définition 1.18, si δ est fortement k-consistance, on peut construire sans backtrack une solution de (X, D, C, R). On a dans ce cas la consistance globale de δ.*

4.6 Mise en œuvre

4.6.1 Jeux de tests

Notre objectif est de tester l'outil que nous avons développé et qui permet, conformément aux modèles et aux algorithmes que nous venons de décrire, de transformer un système expert à base de règles en un DDCSP. Nous avons effectué des tests sur différentes instances de jeux de tests usuels : le problème des reines, le problème du zébre et Miss Manners. Nous nous contenterons ici de donner la modélisation du problème des reines. Ces jeux de tests sont utilisés pour valider la faisabilité de notre système hybride et pour tester la résolution expérimentale des DDCSP. Les tests ont été exécutés sur les configurations décrites dans la suite de cette section.

Le problème des reines

Dans ce problème, on essaie de placer n reines sur un échiquier classique (comportant n lignes et n colonnes) de telle façon qu'aucune reine n'en menace une autre. Deux reines se menacent ou "sont en prise" si elles se trouvent sur une même diagonale, une même ligne ou une même colonne de l'échiquier.

Ce problème se formalise sous forme de CSP par n variables x_1 à x_n (une par ligne) de n valeurs. La valeur de chaque variable correspond à la colonne où la reine représentée par la variable doit être placée. Les contraintes peuvent être définies implicitement par la formule $|x_i - x_j| \neq |i - j| \wedge x_i \neq x_j, \forall (i,j) \in [1..n]^2 | i \neq j$. Nous avons modélisé ce problème dans un système à base de règles de production. La règle du système expert est constituée par la conjonction de toutes les contraintes du problème et admet pour conclusion un prédicat $solution(x_1, ..., x_n)$. La base de faits est constituée par les tuples autorisés pour chaque contrainte. Dès que ce prédicat est prouvé, il caractérise une configuration correcte des reines.

En utilisant la même démarche, nous avons modélisé les autres problèmes énoncés ci-dessous.

Le problème du zèbre

"**Le problème du zèbre**" consiste à résoudre l'énigme suivante : on a cinq maisons consécutives, de couleurs différentes, habitées par des hommes de différentes nationalités. Chacun possède un animal différent, a une boisson préférée différente et fume des cigarettes différentes. De plus, un ensemble de contraintes doivent être respectées.

1. Le Norvégien habite la première maison ;
2. Le Norvégien demeure à côté de la maison bleue ;
3. On boit du lait dans la maison du milieu ;
4. L'Anglais habite la maison rouge ;
5. On boit du café dans la maison verte ;
6. On fume des Kools dans la maison jaune ;
7. La maison verte est voisine de droite de la maison blanche ;
8. Le chien appartient à l'Espagnol ;
9. L'Ukrainien boit du thé ;
10. Le Japonais fume des Cravens ;
11. Le fumeur de Old Golds a un escargot ;
12. Le fumeur de Gitanes boit du vin ;

13. Le voisin du fumeur de Chesterfields a un renard ;
14. Le voisin du fumeur de Kools a un cheval.

On cherche alors à savoir : qui boit de l'eau ? Et à qui appartient le zèbre ?

Le problème "Miss Manners"

Le problème "Miss Manners" consiste à trouver une solution au problème de répartition d'un ensemble d'invités autour d'une table lors d'un dîner, tel que les sexes doivent être alternés et que chacun ait au moins un hobby en commun avec chacun de ses voisins.

4.6.2 Choix des outils

La motivation de cette partie est de valider notre approche.

En particulier, on a fait le choix de ne pas développer de nouveaux outils, ni un nouveau moteur d'inférence pour résoudre les systèmes experts, ni un nouveau solveur de CSP.

Notre approche s'est orientée vers le choix d'outils standard et open source afin de prouver la validité des principes et des algorithmes présentés dans les sections qui précèdent.

Cette approche a l'avantage de demander un minimum d'effort d'implémentation et de prouver expérimentalement que cette transformation et les algorithmes de la section 4.4, en particulier la procédure Resolution_CSP, peuvent être réalisés en s'appuyant sur des outils existants.

Cependant, l'inconvénient majeur de cette approche est de ne pas pouvoir optimiser certains aspects notamment la gestion des données, l'initialisation des solveurs et la résolution.

Pour tester la transformation d'un problème modélisé dans une base de connaissances en un DDCSP, de nombreuses solutions techniques ont été envisagées.

Pour le choix du moteur d'inférence, nous nous sommes orientés vers Clips [178] dont les qualités sont reconnues. Cependant, afin de respecter l'expression des règles et des prédicats tels qu'ils ont été définis dans le paragraphe 4.2, nous nous sommes limités à une partie restreinte de la syntaxe de Clips. Il va de soi que ce choix a des conséquences sur les performances de Clips durant la résolution de ces problèmes. Malgré cette limitation, il nous est apparu important de n'utiliser que la syntaxe de la logique du premier ordre, syntaxe universelle commune à un grand nombre de systèmes experts.

Pour le choix du solveur de CSP, Choco [177], outre ses qualités de solveurs, présente différents avantages, notamment celui de s'intégrer facilement dans l'environnement de test choisi. L'inconvénient majeur de Choco réside dans une

phase d'initialisation lors de chaque appel. Cette phase d'initialisation introduit un temps non négligeable.

De plus, Clips est programmé en C/C++, par contre, Choco est développé en Java, ce qui induit une échelle de temps d'exécution différente pour les deux applications.

Il est certain que des solutions peuvent améliorer l'intégration de Choco dans les algorithmes de résolution des DDCSP et éviter cette étape pénalisante d'initialisation, mais ce développement n'a pas été mené dans le cadre de cette thèse.

4.6.3 Résultats

Les trois problèmes ont été modélisés sous forme d'un ensemble de règles d'un système expert qui ont été générées automatiquement à partir d'une des modélisations standard.

La seule stratégie de résolution de DDCSP que nous avons implémentée est la stratégie 1 qui consiste à résoudre à chaque étape tous les eDDCSP. On obtient ainsi un nouvel ensemble de eDDCSP éventuellement modifiés qui seront à leur tour résolus à la prochaine étape, jusqu'à ce que plus aucune modification n'ait lieu.

Le tableau 4.2 décrit les principales caractéristiques de l'environnement matériel qui a servi à la réalisation des tests.

Système d'exploitation	Linux
Processeur	Intel core 2 Duo
Vitesse	2*2 GHz
RAM	2 GB
Version J2SE	1.6

TABLE 4.2 – *Caractéristiques du système utilisé pour l'exécution des jeux de tests*

Benckmarks	Clips (version 3.5) temps (ms)	DDCSP Solveur Choco (version 2.1.1) temps (ms)
Zèbre	Overflow	1150
Reine n*n		
4	0.039	5,3
5	0.120	111
6	0.081	469
7	0.640	153
8	1.68	4260
9	7.50	27300

TABLE 4.3 – *Temps de résolution des problèmes du zèbre et des reines pour Clips et pour le solveur DDCSP*

Nombre d'invite	Clips (version 3.5) temps (s)	DDCSP Solveur Choco (version 2.1.1) temps (s)
4	0,005	0,02
8	0,015	0,19
16	0,045	2,01
32	0,120	16

TABLE 4.4 – *Temps de résolution des problèmes Miss Manners pour Clips et pour le solveur DDCSP*

Les tableaux 4.3 et 4.4 montrent les tests de faisabilité de notre solveur de DDCSP en les comparant avec Clips.

La ligne 1 de la table 4.4 (le problème du zèbre) est intéressante parce qu'elle montre que certains problèmes, même s'ils peuvent être exprimés facilement par la logique du premier ordre, ne permettent pas à un moteur d'inférence de trouver une solution en un temps raisonnable. En revanche, la transformation a permis d'obtenir un DDCSP qui cette fois conduit à un résultat.

Certes, le système de règles que nous avons utilisé dans Clips pour modéliser le problème, n'est sûrement pas optimal, il reste un exemple de ce qui peut arriver dans la réalité, à savoir qu'un problème exprimé sous forme de système expert peut être difficilement solvable dans cet environnement. On peut alors constater que le transformer en une autre modélisation peut permettre de trouver de meilleurs résultats.

Premièrement, ces tests prouvent la correction de notre solveur. En effet, toutes les solutions trouvées par les deux systèmes sont identiques.

Les temps de résolution des DDCSP sont très nettement supérieurs à ceux de Clips. L'explication réside dans les problèmes d'intégration de Choco déjà

mentionnés dans la section précédente. En particulier, dans Choco, aucun accès direct à la structure de données en mémoire n'est possible. Donc toute modification dynamique des domaines des variables et des relations associées aux contraintes nécessite une nouvelle initialisation du solveur et la régénération du modèle ce qui coûte très cher en temps.

Mais nous insistons sur le fait que, dans ce travail de développement, le point fondamental est de valider la transformation et non pas de construire un système performant. Les temps d'exécution indiqués dans les tableaux ont pour seul but de montrer la terminaison des deux algorithmes : une comparaison des valeurs des temps obtenus par Choco et Clips serait vide de sens du fait des grandes différences entre ces deux outils.

4.7 Conclusion

La première partie de cette thèse s'est focalisée sur l'intégration des CSP dans l'architecture d'un système expert. L'étude des systèmes experts et des CSP a permis d'introduire un "système multi-noyaux" et nous a conduit à la définition de trois modèles d'hybridation entre les systèmes experts et les CSP. Nous avons ensuite proposé une procédure de transformation d'un problème à base de règles de production sûres en logique du premier ordre fini, vers un nouveau formalisme "DDCSP" à base de contraintes. Cette transformation est nécessaire pour implémenter les hybridations proposées. Pour résoudre les DDCSP, deux algorithmes ont été conçus. Durant cette résolution, les DDCSP ont la particularité d'avoir des domaines de variables et des relations dynamiques. Nous avons étudié les caractéristiques et les propriétés des DDCSP. Puis nous avons prouvé que tout problème à base de règles de production sûres en logique du premier ordre fini est équivalent à un DDCSP.

Enfin, nous avons décrit l'implémentation et présenté des premiers résultats expérimentaux d'un solveur de DDCSP et de la stratégie d'hybridation basée sur l'approche concurrente.

Un des avantages des DDCSP est de permettre des stratégies de résolution distribuées qui constituent une voie prometteuse. Les chapitres suivants présentent les algorithmes de résolution de CSP distribués et sécurisés, et proposent de nouvelles approches qui mettent l'accent sur le parallélisme et la coopération.

Notre objectif final consistera à exploiter ces nouvelles approches pour améliorer la résolution des DDCSP.

Chapitre 5

CSP distribués

The real work of domain name system happens in a very distributed, very decentralized way.
Andrew McLaughlin

Nous avons présenté et justifié dans les chapitres précédents l'intérêt d'hybrider des problèmes de satisfaction de contraintes avec des systèmes experts en donnant des stratégies de coopération entre les deux modèles et en présentant un protocole qui permet le passage d'un modèle à l'autre. Dans cette deuxième partie de la thèse nous nous intéressons à la résolution de problèmes dans un environnement distribué. En particulier, nous étudions les problèmes de satisfaction de contraintes distribués.

Dans ce chapitre nous présentons une synthèse des principaux travaux concernant les problèmes de satisfaction de contraintes distribués.

5.1 Introduction

Les problèmes naturellement distribués sont nombreux, en particulier dans le contexte du commerce électronique et les problèmes de planification. Ces applications ont favorisé l'étude des systèmes distribués et l'élaboration de modèles et de prototypes de communication dédiés à ces systèmes. Dans cette thèse nous focalisons nos recherches sur le paradigme des problèmes de satisfaction de contraintes distribués (DisCSP : **Dis**tributed **C**onstraints **S**atisfaction **P**roblems). Un DisCSP [165] est un CSP dans lequel les variables et les contraintes sont distribuées entre un ensemble d'agents. Chaque agent possède un sous-ensemble de variables. Les sous-ensembles des variables des agents forment une partition de X. La répartition de l'ensemble des variables entre les agents conduit à distinguer deux types de contraintes : (1) les *intra-contraintes*

constituent le sous-ensemble de C des contraintes entre les variables locales d'un agent ; (2) les *inter-contraintes* est le sous-ensemble de contraintes de C qui lient les variable(s) de deux agents distincts. Comme dans le cadre d'une résolution centralisée d'un CSP, une solution d'un DisCSP est une affectation des valeurs aux variables tel qu'elle ne viole aucune contrainte.

5.2 Définitions et outils

Cette section présente les systèmes multi-agents qui sont utilisés dans les applications distribuées et en particulier dans les DisCSP.

5.2.1 Système Multi-Agent (SMA)

Nous pouvons décrire un SMA comme étant un ensemble d'entités actives et autonomes, les agents, qui évoluent dans un environnement.

Dans [117], les auteurs décrivent un agent de la manière suivante : "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon the environment through effectors".

D'après cette description, la notion d'agent regroupe aussi bien des êtres humains, des robots ou encore des agents virtuels (incarnés par un programme informatique). L'environnement est considéré comme un contexte pour les activités et les interactions des agents. Il peut être le monde réel (dans le cas de robots par exemple) ou bien un univers virtuel modélisé de manière informatique. En outre, il est important de préciser qu'un agent n'est capable de percevoir que partiellement son environnement.

Il n'y a pas de consensus absolu pour donner une définition exacte du terme « agent ». Il existe plutôt différentes définitions acceptées, chacune étant plus adaptée à un contexte particulier. Nous nous contenterons donc ici de reprendre la définition de Jennings et d'énumérer un ensemble de propriétés communément admises pour les agents.

Définition 5.1 (un agent [160]) *Un agent est un système informatique, situé dans un environnement, qui agit de façon autonome et flexible pour atteindre les objectifs pour lesquels il a été conçu.*

Dans cette définition, nous retrouvons trois concepts clés qui doivent être présents lorsque nous parlons d'agents intelligents :

1. **Autonome :** l'agent doit pouvoir prendre des initiatives et agir sans l'intervention directe d'un humain (ou d'un autre agent). Il a le contrôle de ses actions et de son état interne.

2. **Situé dans un environnement :** l'agent peut recevoir des informations de l'environnement au moyen de capteurs. Il peut aussi effectuer des actions susceptibles de modifier l'environnement dans lequel il se trouve.
3. **Flexible :** par flexible, on entend que l'agent est :
   (a) *Capable de répondre à temps* : il peut percevoir son environnement et répondre rapidement aux changements qui s'y produisent.
   (b) *Proactif* : il est capable d'avoir un comportement opportuniste, dirigé par ses buts ou sa fonction d'utilité, et prendre des initiatives au moment approprié.
   (c) *Social* : il est capable d'interagir avec les autres agents présents dans l'environnement afin de compléter ses tâches ou d'aider les autres à compléter les leurs.

5.2.2 Système distribué

Un système distribué est composé de plusieurs nœuds qui coopèrent et échangent des informations. L'échange d'informations entre les nœuds peut être effectué par le biais d'une mémoire partagée (accès en lecture et écriture sur des variables communes). On discerne deux modèles de systèmes distribués : les systèmes fortement couplés (ou systèmes à mémoire partagée), et les systèmes faiblement couplés (ou systèmes par envois de messages). Le deuxième modèle de programmation est utilisé par de très nombreuses applications de calcul scientifique. Plusieurs librairies de programmation à base d'envois de messages ont été développées. PVM (Parallel Virtual Machine) [56] et MPI (Message Passing Interface) [41] sont les plus répandues. Autrement dit, un système distribué est un ensemble de machines (nœuds) autonomes et asynchrones unies par un réseau de communication. Nous pouvons définir un système distribué ainsi :

Définition 5.2 (Système distribué) *Un système distribué est défini par un ensemble de nœuds indépendants, communiquant via un réseau, coopérant pour fournir un service.*

Parmi les paradigmes de systèmes distribués, on peut citer le peer-to-peer, le desktop sharing, le calcul à haute performances, les grilles de calcul. La résolution distribuée de problèmes par des systèmes distribués nécessite une communication et une interaction entre les différents nœuds.

5.2.3 DisCSP

Définition 5.3 *Formellement, un DisCSP est défini par Yokoo et al. [165] comme un quintuplet (X, D, C, A, $\mathcal{F}$) , où*

X : est un ensemble $\{x_1, x_2, ..., x_n\}$ *de variables*

D : est un ensemble de domaines finis $\{d_{x_1}, d_{x_2}, ..., d_{x_n}\}$ *tel que le domaine* d_{x_i} *soit associé à la variable* x_i .

C : est un ensemble $\{c_1, c_2, ..., c_m\}$ *de contraintes.*

$A = \{A_1, A_2, ..., A_p\}$ *est un ensemble de p agents et*

$\mathcal{F} : X \rightarrow A$ *est une fonction qui associe chaque variable à un agent.*

La distribution des variables forme une bipartition de C en C_{intra} *et* C_{inter}*, qu'on nomme ensemble des contraintes intra-agents et inter-agents, respectivement. Une contrainte intra-agents* c_i *n'est connue que par l'unique agent détenteur de toutes les variables impliquées dans la contrainte* c_i*, et une contrainte inter-agents est connue par tous les agents qui détiennent l'une au moins des variables de* c_i *[167, 66].*

Comme dans le cadre centralisé, une solution est une affectation de valeurs aux variables qui ne viole aucune contrainte. Les CSP distribués sont résolus par l'action collective et coordonnée des agents de A, chacun exécutant un processus de résolution et de satisfaction de contraintes. Les agents communiquent par des envois de messages.

1. Un agent ne peut envoyer de message que s'il connaît l'adresse du destinataire.
2. Le délai de réception d'un message est aléatoire, fini
3. Les messages sont reçus dans l'ordre dans lequel ils ont été envoyés.

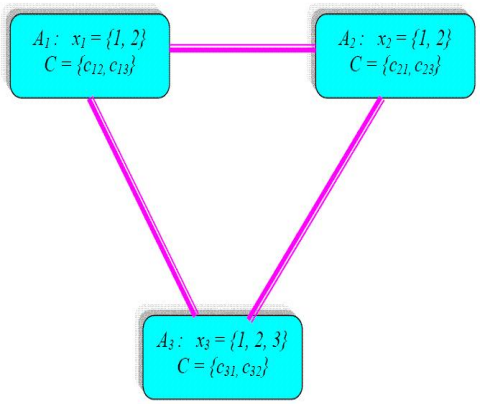


FIGURE 5.1 – Exemple d'un CSP distribué binaire

5.3 Les défis des DisCSP

Les problèmes de satisfaction de contraintes distribués posent les mêmes problèmes que les CSP centralisés (explosion combinatoire, backtrack, filtrage). Mais de nouveaux défis spécifiques à la distribution apparaissent tels que la synchronisation des messages échangés, le coût des communications, la complexité de l'algorithme et également la confidentialité des données.

- **Synchronisation.** Les DisCSP sont sujets aux problèmes traditionnels des applications distribuées. En particulier, l'ordre et les temps d'arrivée des messages ne sont pas garantis. Les algorithmes DisCSP doivent donc tenir compte de ces problèmes et mettre en place des stratégies asynchrones.
- **Le coût des messages échangés.** La vérification des contraintes constitue un goulot d'étranglement pour les algorithmes DisCSP. Puisque les variables et les contraintes sont distribuées entre différents agents, cette vérification nécessite souvent des échanges de messages. Ce coût de communication a un impact direct sur l'efficacité d'un algorithme. Par conséquence, un grand défi des algorithmes DisCSP est de réduire le nombre de communications entre les agents.
- **Problème de distribution.** Par définition des DisCSP, toutes les variables et toutes les contraintes sont distribuées sur un ensemble d'agents. Quelques éléments du problème sont publics et d'autres privés. Ceci complique la recherche d'une solution et réduit l'efficacité. Quand un agent n'a pas une vue globale sur le problème, il peut proposer une valeur à sa variable qui est incompatible avec la variable privée d'un autre agent. L'incompatibilité des deux valeurs exige des communications supplémentaires afin que l'un des agents change la valeur instanciée à sa(ou ses) variable(s), contrairement aux CSP centralisés, où les échecs peuvent être détectés localement, sans coût de communication.

Ces défis ont justifié le développement de nombreux algorithmes de résolution de DisCSP que nous étudions dans la section suivante.

5.4 Algorithmes de résolution des DisCSP

Dans cette section, nous nous intéressons aux techniques de résolution des CSP distribués, et décrivons brièvement les différents travaux existant.

5.4.1 "La famille ABT"

La plupart des recherches sur les algorithmes de résolution de DisCSP ont été focalisées sur un principe d'exploration complète et asynchrone de l'espace de recherche. Ces algorithmes font agir les agents en parallèle et leurs communication s'effectuent, si besoin est, pour maintenir la consistance des instanciations de variables qu'ils contrôlent. L'algorithme fondateur est appelé "Asynchronous BackTracking" (ABT) et fut proposé par Yokoo et ses collaborateurs [165]. L'algorithme ABT effectue une exploration complète de l'espace de recherche. Cet algorithme est considéré comme une référence pour la conception de tout nouvel algorithme de résolution de DisCSP. L'algorithme ABT utilise un mode asynchrone pour la communication et procède par des envois de messages qui contiennent soit :

- une instanciation courante des variables de l'agent émetteur, ces messages sont notés "ok ?" ;
- Une demande de BackTrack. Un tel message est appelé nogood et noté "ngd" ;
- Une demande d'ajout de lien qui permet de ne pas revisiter plusieurs fois la même partie de l'espace de recherche. Ces liens sont ajoutés de manière dynamique. Ce type de message "add-link-request", est noté "adl".

L'algorithme impose un ordre statique entre les agents selon un ordre de priorité. Cet ordre permet de déterminer le receveur d'un nogood (i.e. l'agent devant changer sa valeur à cause d'un BackTrack). Quand un agent choisit une valeur à sa variable il la communique à ses voisins successeurs dans le graphe de contrainte via un message "ok ?". Le choix de cette valeur fait en sorte de ne pas violer les contraintes reliant l'agent à ses prédécesseurs dans le graphe de contraintes. Quand un agent reçoit un message "ok ?" il vérifie que sa valeur ne rentre pas en conflit avec la nouvelle valeur reçue, sinon il change sa valeur afin de ne plus être en conflit et prévient ses successeurs du changement. Si le domaine d'une de ses variables est vidé suite au dernier message reçu, l'agent envoie un nogood au plus proche prédécesseur qui est responsable de cet échec.

5.4.2 Les différentes variantes de l'algorithme ABT

Distributed Synchronous Backtracking L'algorithme synchrone backtracking distribué (DBT) constitue la méthode de base pour la résolution des

DisCSP. Cet algorithme a été développé par Yokoo et al. [165]. Il est basé sur un algorithme de backtracking simple. Dans cet algorithme, chaque agent a exactement une variable. L'ordre des agents est défini de manière statique lors d'une phase de pré-traitement en appliquant une simple heuristique pour définir l'ordre d'instanciation des variables (Max-Deg, Min-Deg, ...). Chaque agent effectue un traitement synchrone, ce qui signifie ici qu'à un temps donné, un seul agent est actif. Le principe est le même que celui du Backtracking centralisé et la seule différence réside dans la vérification des contraintes qui exige une communication avec les agents qui détiennent une partie du problème. Quand l'agent reçoit une instanciation partielle, il essaye d'étendre cette instanciation en lui ajoutant sa propre variable affectée d'une valeur de son domaine. Si c'est réussi, la nouvelle instanciation est transmise à l'agent suivant. Si aucune valeur n'est trouvée pour l'instanciation partielle reçue, alors l'agent renvoie un message de backtracking à l'agent émetteur. Quand le dernier agent (dans l'ordre prédéfini au départ) a trouvé une affectation à sa propre variable, il annonce à tous les autres agents que la solution a été trouvée. Le défaut de cet algorithme est qu'il ne profite pas des avantages du parallélisme, et que l'ordre d'instanciation des variables est statique.

Asynchronous Weak-Commitment (AWC) Cette extension de l'algorithme ABT, Asynchronous Weak Commitment (AWC) [166], étend au cadre distribué la méthode séquentielle dite Weak Commitment search. Le principe de cette méthode séquentielle consiste à réviser l'ordre d'instanciation des variables lors de chaque échec rencontré. Le même principe est utilisé dans le cadre distribué. Nous signalons que dans le cadre distribué, chaque changement local de priorité est diffusé à l'ensemble des agents, ce qui entraîne une augmentation des communications. Ces diffusions entraînent une perte de localité pour les informations locales de chaque agent. De plus, deux agents du système peuvent échanger successivement leurs niveaux de priorité. Les simulations de l'algorithme en versions gros grain (plusieurs variables par agents), sont présentées dans [167]. Ces résultats montrent AWC comme supérieur à ABT et à ABT muni de l'heuristique min-conflict.

Dynamic Backtracking(DiBT) Cette approche a été proposée au départ par Hamadi et al. mais n'était pas complète [66]. Puis elle a été étendue par plusieurs auteurs [9, 12]. C'est un algorithme calqué sur le modèle du Backtrack dynamique distribué. Il réalise des sauts dynamiques sur l'ensemble des agents en conflit. La recherche d'une solution nécessite la mise en place d'un ordre total entre les agents. Cette phase est exécutée avant la phase de résolution. Au départ, une phase d'échanges entre les agents leurs permet de connaître la valeur de l'heuristique d'ordonnancement de chaque agent, et les contraintes sont orientées pour former un graphe acyclique à l'aide de l'algorithme DisAO [66]. Il

construit une hiérarchie d'agents en utilisant des critères heuristiques. Cette hiérarchie induit un ordre partiel sur les agents, qui doit être complète pour former ensuite un ordre total et garantir la complétude. Le principe de cette heuristique consiste à définir pour chaque agent, appelé *self*, $\Gamma^{-}(self)$ qui désigne l'ensemble des agents partageant une contrainte avec *self* et qui sont classés plus haut dans la hiérarchie. Réciproquement, l'heuristique définit $\Gamma^{+}(self)$, l'ensemble des voisins de *self* classés plus bas dans la hiérachie.

Chacun des agents exécute l'algorithme DiBT et mémorise un ensemble de nogoods. Le contexte de *self* est l'ensemble des valeurs affectées (à sa connaissance), aux agents le précédant dans l'ordre. Il est toujours cohérent avec l'ensemble des nogoods mémorisés localement. Les agents échangent des affectations et des nogoods. Les affectations sont toujours acceptées, et le contexte mis à jour en conséquence. Un nogood est accepté s'il est cohérent avec le contexte de l'agent et sa propre instanciation, sinon il est considéré comme caduc et obsolète. S'il est accepté, il pourra justifier le retrait de la valeur courante.

Asynchrone Forward Checking (AFC)

Dans [93] Zivan et al. proposent un algorithme de type Forward Checking. Ils s'inspirent des travaux de l'algorithme Forward Ckecking centralisé et des propriétés de l'algorithme ABT.

L'algorithme AFC combine l'avantage du filtrage des valeurs des domaines des agents voisins (un agent est voisin d'un autre si il existe au moins une inter-contrainte entre les variables des deux agents). Lors de la réception d'une affectation partielle d'un agent voisin, le récepteur propose des valeurs pour ses variables et envoie ensuite un message à ses voisins inférieurs pour qu'ils puissent filtrer leurs domaines. Les agents qui reçoivent une demande de filtrage mettent à jour les domaines de leurs variables, en supprimant toutes les valeurs qui entrent en conflit avec l'affectation partielle reçue. Lorsqu'un domaine est vide, l'agent initie un backtrack par l'envoi de messages non OK qui contiennent les affectations partielles incompatibles. Le message non OK est envoyé à tous les agents *avec des variables non affectées dans l'affectation incompatible*. Cet algorithme induit un surcoût de communications dû aux filtrages des valeurs des variables des agents moins prioritaires et qui restent à être instanciées selon l'ordre préétabli.

Concurrent Backtracking (ConBT) Cet algorithme a été développé par Zivan et al. [174, 173]. Dans ce travail, les auteurs s'inspirent des travaux effectués dans le cadre d'un CSP centralisé, où la concurrence consiste à lancer en parallèle plusieurs solveurs sur le même problème. Avant tout démarrage, une phase de pré-traitement est exécutée dans le but de définir l'ordre d'instanciation des variables ainsi qu'une phase de lancement des processus de recherche par l'agent le plus prioritaire. Autrement dit, par l'agent qui possède la première variable à instancier selon l'ordre prédéfini. Le nombre de processus lancés K doit être inférieur ou égal à la taille du domaine de la première variable à instancier. Chaque processus a en charge l'exploration exhaustive de son sous espace. La Figure 5.2 illustre cette exploration. Nous supposons que le problème est distribué entre trois agents A_1, A_2 *et* A_3. Chaque agent possède une variable x_1, x_2 *et* x_3 respectivement, et l'ordre d'exécution est le suivant A_1, A_2 et A_3, donc l'agent A_1 crée deux processus de recherche P_1 et P_2. Le premier processus explore l'espace engendré par l'instanciation $X_0 = a$, alors que le second explore le sous espace restant. Le problème dans cette version que les agents plus prioritaires dans l'ordre d'instanciation de leurs variables sont plus chargés par rapport aux autres (moins prioritaires). Le second problème est de ne pas exploiter les avantages de parallélisme ni de la coopération entre les différents processus de recherche. Ces inconvénients influent sur les performances de l'algorithme.

La simulation des résultats de cet algorithme a été présentée dans [92]. Ces

résultats montrent l'efficacité de ConBT par rapport ABT.

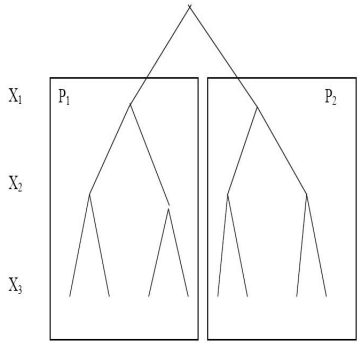


FIGURE 5.2 – Concurrent Backtracking

Il existe d'autres travaux sur la résolution des DisCSP, qu'on ne va pas présenter dans notre étude. Ces travaux s'inspirent de la distribution canonique des systèmes distribués [145] G. Solotorevsky et al. proposent deux schémas de méthodes basées sur ce type de distribution de problème. La première approche est nommée Central First Peripheral After (CFPA). Dans cette version, lorsque la solution est trouvée par le composant principal, elle est diffusée vers les autres éléments, et la seconde approche, nommée Peripheral First Central After (PFCA), est une approche inverse de la précédente. Tout d'abord, les agents périphériques instancient leurs variables. Quand un agent périphérique trouve une solution, il informe le composant principal.

5.5 Conclusion

Dans ce chapitre, nous avons réalisé un tour d'horizon des méthodes de résolutions des problèmes de satisfaction contraintes distribués. On remarque que la plupart de ces algorithmes utilisent le principe de base de l'algorithme ABT proposé par Yokoo, et tentent de pallier ses lacunes. Parmi les difficultés apparues dans les techniques de résolution de DisCSP, on peut relever l'encombrement du graphe de communication et le surcoût associé en terme d'envoi de messages. Une solution peut consister à attendre qu'un conflit soit effectivement détecté pendant la recherche, et à créer alors des liens dynamiquement. Parmi les inconvénients de ces algorithmes, l'espace mémoire nécessaire pour sauvegarder les valeurs « good » et « nogood » est exponentiel. Il est également possible de rencontrer un problème de famine, c'est-à-dire qu'un agent peut être inactif ce qui réduit l'efficacité de l'algorithme. Un autre inconvénient de ces algorithmes est de ne pas exploiter pleinement le parallélisme ni les heuristiques d'ordre d'instanciation des variables et des valeurs. Nos contributions pour améliorer ces différents points sont présentées dans le chapitre suivant.

Chapitre 6

Approche parallèle et coopérative

Society must take every means at its disposal to defend itself against the emergence of a parallel power which defies the elected power.

Pierre Elliott Trudeau

Dans ce chapitre, nous proposons une architecture pour la résolution de DisCSP, fondée sur l'algorithme de backtracking distribué et un couplage de deux notions : la coopération et le parallélisme. Le principe consiste à lancer plusieurs processus de recherche, appelés agents mobiles. Ces processus de recherche travaillent de manière coopérative et simultanée en assurant leur dispersion dans l'espace de recherche. Après une description de cette architecture, nous montrons les caractéristiques de cette approche : la complétude, la terminaison et la correction. Enfin, nous proposons une extension de cette approche en l'hybridant avec d'autres algorithmes de résolution des CSP classiques.

6.1 Approche Parallèle et Coopérative (PC)

Cette section est consacrée à la description et à l'étude des différentes caractéristiques de la nouvelle approche proposée.

6.1.1 Présentation générale de l'architecture

La topologie utilisée est une topologie en anneau et chaque nœud dispose d'un prédécesseur et d'un successeur. Il est important de noter qu'on ne parle pas ici de la topologie réelle mais de la topologie virtuelle du réseau. Chaque noeud

initie un processus de recherche. Lorsqu'un processus de recherche s'exécutant sur un nœud a trouvé une instanciation partielle consistante, il envoie cette instanciation partielle à son successeur sur l'anneau, se poursuivant ainsi sur le nœud voisin. Chaque nœud contribue ainsi à plusieurs explorations simultanées. En effet, le principe est de faire exécuter plusieurs processus de recherche par différents nœuds qui travaillent de manière coopérative et parallèle.

Un processus de recherche est exécuté successivement sur différents nœuds indépendants du système. Il peut alors manipuler des données locales au nœud sur lequel il s'exécute.

Grâce à la topologie en anneau utilisée, les messages de même type transitent toujours dans la même direction. La topologie en anneau permet également de réduire le temps d'attente des nœuds participants. Afin d'éviter autant que possible, d'explorer plusieurs fois une même partie de l'espace de recherche, nous proposons un mécanisme local de gestion d'une mémoire partagée par les différents processus de recherche exécutés sur un même nœud. Ce mécanisme favorise une bonne diversification dans l'espace de recherche.

6.1.2 Principe de l'approche PC

Étant donné un groupe de p agents avec ses données (variables, contraintes, domaines), tous devront collaborer afin d'élaborer une solution globale du problème puisque chacun d'eux ne détient qu'une partie du problème. La recherche d'une solution nécessite la mise en place d'un ordre total entre les agents. En effet, cet ordre est important pour réduire le temps de repos des agents, et pour éviter ainsi les déséquilibres de charge et la surcharge de certains agents. Toutefois, dans notre approche, nous utilisons un ordre statique défini dans une phase de pré-traitement. Nous notons que deux notions peuvent influencer l'intérêt de cette topologie :

1. les heuristiques d'ordre d'instanciation des variables : dans les DisCSP comme dans les CSP, l'ordre d'instanciation des variables a une grande influence sur la taille de l'arbre de recherche. Il existe deux types d'heuristiques pour le choix de l'ordre d'instanciation des variables : les heuristiques statiques (Max-deg [50], MW [38]) et les heuristiques dynamiques (par exemple MC [50], MRV [69]). Toutefois, l'influence de ces heuristiques reste marginale dans notre approche, car chacun des p processus de recherche lancés aura son propre ordre d'instanciation de variables. Les différents ordres d'instanciation mis en œuvre découlent automatiquement de la topologie de communication en anneau.

2. la topologie physique du réseau peut aussi influencer l'efficacité de cette approche.

Après la phase de pré-traitement, chaque agent est capable de déterminer localement son prédécesseur et son successeur. Un identifiant est associé à chaque agent. Ce même identifiant est également attribué au processus de recherche lancé par l'agent. Classiquement, la stratégie de parcours de l'arbre de recherche d'un processus de recherche au sein d'un agent est cruciale puisque l'efficacité de l'exploration de l'arbre de recherche en dépend. Afin de mieux guider cette orientation, on introduit des indicateurs au niveau de chaque agent. Un indicateur est associé à chaque valeur qui peut être attribuée à une variable. Ces indicateurs représentent un mécanisme d'orientation des processus de recherche. Chaque processus de recherche choisit à chaque étape la valeur consistante la moins utilisée par d'autre processus de recherche, puis incrémente l'indicateur associé à la valeur sélectionnée. Les agents s'échangent la valeur de l'instanciation courante via des messages, et poursuivent leur exécution en fonction des types de messages reçus. Nous décrivons ici tout d'abord cette approche avec un algorithme de Backtraking, que nous avons nommé PC_BT (Parallel and Cooperative BackTracking). On distingue quatre types de messages :

- **Fail** : il n'y a pas de solution et les destinataires arrêtent la recherche.
- **Success** : une solution a été trouvée et les destinataires arrêtent la recherche.
- **Info** : le message contient une instanciation partielle (CPA : Current Partial Assignment), et est adressé à l'agent suivant selon l'ordre prédéfini. Ainsi, à chaque réception d'un message de type **Info** qui contient une instanciation partielle CPA, l'agent tente d'étendre l'instanciation partielle à une solution. Le choix de la valeur utilise l'heuristique de la "moindre utilisation" que nous avons définie et qui consiste à étendre l'instanciation partielle avec la valeur consistante la moins utilisée par d'autres processus de recherche. L'heuristique de "moindre utilisation" garantit une exploration plus diversifiée donc plus rapide de l'espace de recherche.
- **Back** : le message de retour arrière contient également une instanciation partielle CPA et adresse à l'agent prédécesseur sur l'anneau une demande de type Backtrack. L'agent qui reçoit ce message, essaie de changer la dernière valeur instanciée inconsistante. S'il n'a pas de prédécesseur, cette valeur sera éliminée du domaine global à tous les agents du système. Si le domaine local est vide, l'agent propage à son tour un message de type **Back** à son prédécesseur. Si le domaine est vide et que l'agent n'a pas de prédécesseur, cela signifie que toute la recherche est inconsistante et un message **Fail** est envoyé à tous les agents.

Les domaines des variables sont définitivement réduits à chaque fois qu'un agent reçoit un message de type **Back** en provenance du processus de recherche **qu'il a lui-même initié** puisque on sait dans ce cas que la branche enracinée à cette valeur a été complètement explorée. L'algorithme s'achève lorsqu'une solution a été trouvée par l'un des agents, ou lorsque l'un des agents a un domaine vide, ce qui signifie que le problème est insoluble.

L'algorithme 6.1.3 : Conformément à la définition classique des DisCSP, on dispose d'un ensemble de p agents $A = \{A_1, A_2, ..., A_p\}$ et pour des raisons de simplicité nous associons chaque variable à un agent, sans perte de généralité, puisque le même algorithme peut-être appliqué dans le cas où chaque agent gère plusieurs variables. L'agent A_i gère le domaine global initial de sa variable x_i, et il gère également p domaines courants (un par processus de recherche). La procédure principale commence par une phase d'initialisation qui associe un indicateur initialement nul à chaque valeur qui peut être attribuée à la variable x_i de A_i. Cet indicateur donne une information sur la fréquence d'utilisation de cette valeur par d'autres processus de recherche. Ensuite, chaque agent fait appel à la procédure **Initialization**, qui commence par initialiser un processus de recherche, c'est-à-dire que chaque agent crée une CPA. Il choisit pour cela une valeur pour sa variable x_i, et incrémente l'indicateur de cette valeur. Après cette phase d'initialisation, l'agent envoie un message à son successeur et attend la réception d'un nouveau message dont dépendra la suite de l'exécution. Plus précisément :

- Lors de la réception d'un message de type **Fail** ou **Success**, la procédure s'achève.
- Lors de la réception d'un message de type **Info**, la procédure **Assign_CPA()** est exécutée. Elle tente d'étendre l'instanciation partielle reçue. Pour ce faire, elle essaie de trouver la valeur consistante la moins utilisée. Si une telle valeur v existe, alors la procédure **Assign_CPA()** ajoute à la CPA l'instanciation de $x_i = v$. Si la CPA est complète, un message de type **Success** est diffusé à tous les agents. Sinon un message de type **Info** est envoyé à l'agent suivant. Si aucune valeur consistante v n'est trouvée, un message de type **Back** est renvoyé à l'émetteur (l'agent précédent).
- Lors de la réception d'un message de type **Back**, on appelle d'abord la procédure **Remove_last_assignment()** qui enlève la dernière valeur instanciée de la CPA reçue. La valeur v est éliminée du domaine local courant de la variable x_i puis la procédure **Assign_CPA()** est à nouveau exécutée.

Notre algorithme se distingue des autres techniques de backtracking distribués par les points suivants :

1. L'ordre d'instanciation des variables est propre à un processus de recherche. Il dépend de la position de l'agent dans l'anneau ;
2. L'ordre d'instanciation des valeurs dépend des indicateurs. Choisir la valeur la moins utilisée permet de diversifier l'exploration dans l'arbre de recherche ;
3. Certaines parties de l'espace de recherche seront visitées plus d'une fois car on ne mémorise pas les goods et les nogoods ;
4. La topologie en anneau est également un moyen de diversification dans l'espace de recherche, et contribue à un équilibrage de la charge entre les agents ;
5. La coopération entre les différents processus de recherche est introduite par le choix de la valeur la moins utilisée et par la réduction des domaines initiaux des variables.

6.1.3 Algorithme PC_BT

Nous présentons ici une description en pseudo code de l'algorithme PC_BT. Chaque agent du système possède les structures suivantes :

- Variable : est un tableau qui contient l'ensemble des variables locales de l'agent ;
- Domaine : un tableau d'ensembles où chaque élément du tableau Domaine[i] est le domaine global de la variable x_i, $\forall x_i \in Variable$;
- Indicateurs : un tableau d'ensembles où chaque élément Indicateurs[i] est l'ensemble des indicateurs des valeurs de x_i. L'indicateur d'une valeur est le nombre de fois que cette valeur a été sélectionnée et affectée à x_i par un processus de recherche.
- DomaineCourant : un tableau de même type que Domaine où chaque élément constitue les domaines courants des variables locales du problème pour chaque processus lancé.

On peut noter que Variable, Domaine, Indicateurs et DomaineCourant ont le même nombre d'éléments. De plus, Domaine[i], Indicateurs[i] et DomaineCourant[i] ont le même nombre d'éléments et sont tous trois associés à la même variable x_i.

On peut noter également que Variable, Domaine et Indicateurs sont partagés par tous les processus d'un agent, alors qu'un tableau DomaineCourant distinct est géré par chaque processus.

Pour des raisons de clarification et de simplification, nous supposons que chaque agent ne possède qu'une variable. Au niveau de chaque agent on va

donc avoir le tableau Localvariable réduit à une seule variable. Initialement, les indicateurs de Indicateurs sont initialisés à zéro.

Les primitives suivantes ne sont pas détaillées car elles sont simples et s'exécutent toutes en temps constant :

- Taille(S) : renvoie le nombre d'éléments de la structure S ;
- ValeurConsistanteMinUsed(x, I, D) : retourne une valeur v de x choisie dans le domaine D, consistante avec l'instanciation partielle I, et dont l'indicateur de Indicateurs est minimal. Par convention $v = -1$ signifie qu'on ne peut pas trouver une valeur de x consistante avec l'instanciation I.
- IncrementeIndicateur(x, v) : incrémente l'indicateur associé à la valeur v de x ;
- EtendreCPA(I, x, v) insère $x = v$ dans l'instanciation partielle I.
- ReduireCPA(I, x, v) supprime l'affectation de x ($x = v$) de l'instanciation partielle I. v est une valeur en sortie de ReduireCPA.
- MAJ(x_i, v) : supprime la valeur v du domaine courant DomaineCourant[i] et son indicateur dans Indicateurs[i]. De plus, si le domaine courant est vide et que l'agent a initialisé ce processus de recherche, alors MAJ supprime la valeur v de Domaine[i].
- Reinitialise(x_i) : remplace le domaine courant de x_i DomaineCourant[i] par son domaine global Domaine[i] ;
- ReceptionMsg() : retourne le premier message disponible ;
- Envoie(Message, Type, ListeAgent) : le message Message étiqueté Type est envoyé à l'ensemble des agents de liste ListeAgent ;

Le pseudocode ci-dessous est donné en fonction de i, le numéro de l'agent qui l'exécute.

```
1. Programme Principal (PC_BT)
2. Début
3.     Initialisation() ;
4.     Fin = faux ;
5.     Tantque(!Fin) Faire
6.         msg = ReceptionMsg() ;
7.         Cas (msg.type) Parmi
8.                 Info : Assign_CPA( ) ;
9.                 Back : Remove_last_assignment() ; Assign_CPA( ) ;
10.                Success : Fin = vrai ;
11.                Fail : Fin = vrai ;
12. Fin
13.
14. Procédure Assign_CPA( )
15. Début
16.    Trouve = Assign_local()
17.    Si (Trouve) Alors
18.        Si (Taille(CPA) = N) Alors
19.                Envoie(CPA, Success, all_agents) ;
20.        Sinon
21.                Envoie ( CPA, Info, next_agent) ;
22.    Sinon
23.        Si (Taille(CPA) = 0) Alors
24.                Envoie(CPA, Fail, all_agents) ;
25.        Sinon
26.                Reinitialise(Variable[i]) ;
27.                Envoie ( CPA, Back, previous_agent) ;
28. Fin
```

29. **Procédure Assign_local( )**
30. **Début**
31. x_i = Variable[i] ;
32. v = ValeurConsistanteMinUsed($x_i, CPA, DomaineCourant[i]$) ;
33. **si** (v != -1) **alors**
34. IncrémenteIndicateur(Indicateurs[i], v) ;
35. EtendreCPA(CPA, x_i, v)
36. **Retourne**(vrai) ;
37. **sinon**
38. **Retourne**(faux) ;
39. **Fin**
40. **Procédure Remove_last_assignment( )**
41. **Début**
42. ReduireCPA(CPA, x_i, v)
43. MAJ(x_i, v) ;
44. **Fin**

6.1.4 Illustration de principe sur un exemple : le problème des quatre reines

Afin d'illustrer le comportement de cet algorithme, nous considérons le problème des quatre reines. La figure 6.1 décrit les 6 cycles d'exécution de l'algorithme PC_BT sur ce problème. Chaque cycle du calcul décrit la réception des messages. Pour simplifier cette illustration, nous supposons que dans un cycle, un agent ne peut traiter qu'un seul message. Chaque symbole dans la figure 6.1 représente un processus de recherche mobile qui s'exécute tour à tour sur chaque agent.
$\triangle$ désigne les affectations effectuées par le premier processus de recherche P_1
$\bigtriangledown$ désigne les affectations effectuées par le deuxième processus de recherche P_2
$\otimes$ désigne les affectations effectuées par le troisième processus de recherche P_3
$\oslash$ désigne les affectations effectuées par le quatrième processus de recherche P_4

Rappelons qu'une modélisation usuelle du problème des quatre reines consiste à définir ce problème comme un CSP de 4 variables $\{x_1, x_2, x_3, x_4\}$. Chaque variable x_i représente la ligne de la reine dans la colonne i. Le domaine des variables est $\{1, 2, 3, 4\}$. On a une contrainte entre chaque couple de reine. La contrainte entre deux reines consiste à interdire qu'elles se menacent c'est-à-dire à interdire qu'elles soient sur la même ligne ou la même diagonale. La solution est une affectation de $\{x_1, x_2, x_3, x_4\}$ qui vérifient toutes les contraintes.

Lorsque la case $[i, j]$ contient le symbole d'un processus de recherche P_r, cela signifie que l'instanciation $x_i = j$ a été effectuée par l'agent A_i au sein d'un processus de recherche mobile P_r.

Ici l'algorithme se déroule sur un réseau de 4 nœuds. La topologie en anneau est constituée de $A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4 \rightarrow A_1$.

- Dans le premier cycle, les processus P_1, P_2, P_3, P_4 s'exécutent respectivement sur A_1, A_2, A_3, A_4. Chaque agent instancie sa variable par la première valeur de son domaine. Autrement dit chaque agent crée une CPA et lance un processus de recherche. Ensuite, chaque agent envoie la CPA qu'il vient de créer à son successeur : par exemple A_1 envoie $x_1 = 1$ à A_2, A_2 envoie $x_2 = 1$ à A_3, A_3 envoie $x_3 = 1$ à A_4 et A_4 envoie $x_4 = 1$ à A_1.
- Dans le deuxième cycle, les processus P_1, P_2, P_3, P_4 s'exécutent respectivement sur A_2, A_3, A_4, A_1. Lorsqu'un agent reçoit une instanciation partielle, il essaie d'assigner à sa variable une valeur moins utilisée par d'autre processus de recherche et consistante avec l'instanciation partielle reçue, puis il envoie lui-même cette instanciation à l'agent suivant. Par exemple dans ce cycle l'agent A_2 reçoit de A_1 la CPA $x_1 = 1$. L'agent A_2 complète la CPA en ajoutant $x_2 = 3$ puisque les autres valeurs $x_2 = 1$ et $x_2 = 2$ ne sont pas consistantes. De la même manière les autres agents traitent leurs messages reçus. Dans ce cycle tous les agents réussissent à assigner leurs variables.
- Dans le cycle trois, les processus P_1, P_2, P_3, P_4 s'exécutent respectivement sur A_3, A_4, A_1, A_2. Les agents A_1 et A_2 étendent leurs CPA et respectivement envoient des messages de type **Info**, c'est-à-dire $x_3 = 1, x_4 = 3, x_1 = 4$ et $x_4 = 1, x_1 = 2, x_2 = 4$ à leur successeur A_2 et A_3 respectivement. Par contre, aucune valeur consistante n'est trouvée par A_3 et A_4. Ces agents ne peuvent pas étendre leurs instanciations partielles. Donc A_3 envoie un message de type **Back** $x_1 = 1, x_2 = 3$ à l'agent A_2. De la même façon, l'agent A_4 envoie un message de type **Back** $x_2 = 1, x_3 = 3$ à A_3.
- Dans le cycle quatre, les processus P_1, P_2, P_3, P_4 s'exécutent respectivement sur A_2, A_3, A_2, A_3. L'agent A_3 reçoit le message de type **Back** émis par A_4. Il change la valeur instanciée à sa variable, puis, il cherche une autre valeur consistante pour sa variable. Si cette valeur est trouvée, et c'est notre cas, il envoie le message de type **Info** à l'agent suivant avec une nouvelle CPA $x_2 = 1, x_3 = 4$. de même, l'agent A_2 reçoit le message de type **Back** émis par A_3 et effectue un traitement similaire.
- Dans le cycle cinq, les processus P_1, P_2, P_3, P_4 s'exécutent respectivement sur A_4, A_1, A_2, A_3. Le cycle cinq s'exécute comme le cycle quatre, l'agent A_1 traite le message reçu de l'agent A_4 de processus P_3, en affectant $x_1 = 2$ donc la CPA de P_3 devient $x_3 = 1, x_4 = 3, x_1 = 2$, l'agent A_2 traite le

message Back qui provient de A_3, en changeant la valeur ($x_2 = 4$) dans la CPA de processus de recherche P_1 de même manière A_3 et A_4 traitent leurs messages reçus (voir les modifications dans le schéma).

– Dans le cycle six, l'algorithme termine avec succès. L'agent A_2 réussit à trouver une solution : $x_3 = 1, x_4 = 3, x_1 = 2, x_2 = 4$.

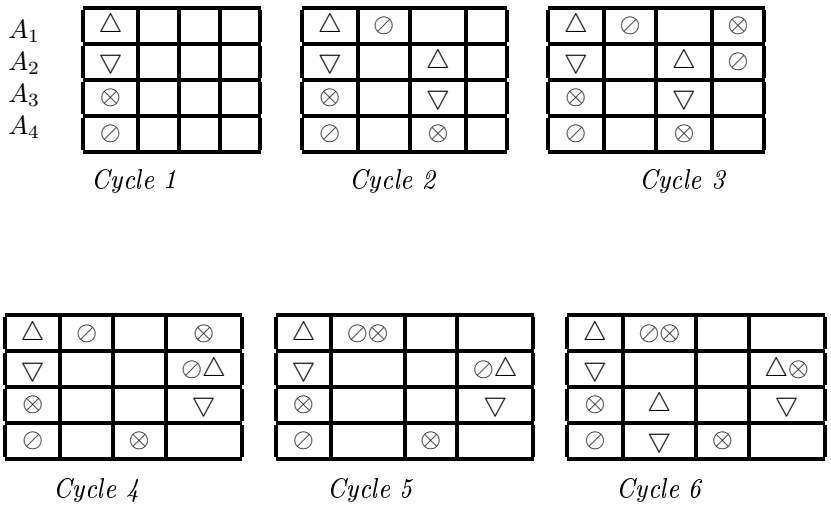


FIGURE 6.1 – Le déroulement de l'algorithme sur le problème des quatre reines

6.1.5 Validation de l'algorithme PC_BT

En analysant la structure de l'algorithme de la section 6.1.3 ci-dessus, on peut effectuer un certain nombre de remarques fondamentales qui caractérisent le comportement de l'algorithme.

Remarque 6.1 *L'ordre est circulaire entre les agents. Chaque agent envoie des messages de type* ***Info*** *à son successeur et des messages de type* ***Back*** *à son prédécesseur. Lorsqu'un agent a une instanciation complète consistante, ou lorsque le domaine de sa variable est vide, il envoie un message d'arrêt, respectivement* ***Success*** *ou* ***Fail****, à tous les agents.*

L'objectif est ainsi de réduire le nombre de messages circulant, et de favoriser l'équilibrage de la charge entre les agents comme nous le verrons dans la section 6.3.

Remarque 6.2 *Un agent demande à son prédécesseur de modifier la dernière affectation dans la CPA, lorsqu'il ne trouve aucune une valeur susceptible d'étendre l'instanciation courante.*

Remarque 6.3 *Un agent décide de supprimer une valeur du domaine global initial de sa variable dès qu'il reçoit un message de type* ***Back*** *au sein du processus de recherche qu'il a lui-même initié.*

Remarque 6.4 *Lorsqu'un agent essaye d'étendre une instanciation, il cherche une valeur la moins utilisée et consistante avec cette instanciation partielle reçue.*

L'objectif de ces indicateurs est de diversifier au mieux les explorations de l'espace de recherche.

Propriété 6.1 *L'algorithme PC_BT assure une diversification dans l'arbre de recherche.*

Preuve 6.1 *En effet, chaque processus de recherche effectue une exploration diversifiée et différente. Lorsqu'un agent reçoit un message de type* ***Info****, il doit satisfaire la remarque 6.4, avant de préparer un nouveau message de type Info vers l'agent suivant. Si plusieurs chemins consistants existent, ils seront empruntés par des processus de recherche distincts (ligne 32 du pseudo code de la section 6.1.3). Ce choix assure une diversification dans l'espace de recherche.*

Remarque 6.5 *Lorsqu'un agent reçoit un message de type* ***Back*** *dans le processus de recherche qu'il a initié, il peut supprimer cette valeur définitivement du domaine global initial de sa variable. En effet, lorsqu'un agent reçoit un message de type* ***Back****, il contrôle s'il est l'initiateur de ce processus de recherche. Si oui, la branche enracinée par cette valeur a été complètement explorée par le processus de recherche et aucune solution n'a été trouvée : cette branche ne contient aucune solution consistante. L'agent décide donc de supprimer cette valeur définitivement du domaine global initial de sa variable. Ceci est naturellement vrai quel que soit l'ordre dans lequel on considère les variables du problème. Par conséquent, cette suppression induit progressivement une réduction de l'espace de recherche puisque les autres processus ne pourront plus, eux non plus, utiliser cette valeur.*

Propriété 6.2 *L'algorithme PC_BT garantit une coopération entre les agents.*

Preuve 6.2 *La coopération entre les agents est introduite par l'heuristique de moindre utilisation mise en œuvre et par le fait que chaque agent applique la remarque 6.4 à chaque réception d'un message de type* ***Info****, et applique le remarque 6.5 à chaque réception d'un message de type* ***Back****. En effet, les indicateurs de fréquence ainsi que les domaines initiaux constituent une mémoire partagée par les différents processus de recherche. Ces informations partagées permettent aux processus de coopérer c'est-à-dire de bénéficier des résultats des*

autres processus, en particulier lors de l'appel de la fonction MAJ ligne 43 du pseudo code de la section 6.1.3.

Remarque 6.6 *Le parallélisme est mis en œuvre par le lancement de N processus de recherche qui explorent simultanément le même espace de recherche et résolvent le même problème.*

Remarque 6.7 *Lorsqu'un agent a un domaine global de sa variable initiale vide, cela signifie que le problème est inconsistant.*

Remarque 6.8 *Lorsqu'un agent a une CPA complète et consistante, il diffuse à tous les agents un message* ***succes*** *pour arrêter les autres processus de recherche en cours, et informer tous les agents que la solution a été trouvée.*

Remarque 6.9 *La diminution du domaine global initial de la variable d'un agent, assure une diminution de l'arbre de recherche pendant la phase de résolution.*

En effet, enlever une valeur d'un domaine d'un agent, revient à élaguer une branche dans l'arbre de recherche, donc à réduire l'espace de recherche.

Propriété 6.3 *L'algorithme PC_BT est complet.*

Preuve 6.3 *Un algorithme est dit complet si et seulement si il assure une exploration globale de l'espace de recherche. Puisque les processus de recherche qui explorent l'arbre de recherche travaillent de manière coopérative, une branche n'est élaguée qu'à l'initiative d'un agent initiateur d'un processus de recherche lorsque la branche entière a été explorée sans trouver de solution conformément aux remarques 6.2 et 6.3. Donc la réduction de l'arbre de recherche n'est effectuée qu'en supprimant des branches qui ne contiennent pas de solution. Par conséquent, l'algorithme PC_BT est complet.*

Propriété 6.4 *L'algorithme PC_BT termine.*

Preuve 6.4 *Si le problème est inconsistant, il est clair que cet algorithme se termine d'après les remarques 6.5 et 6.7 : un des agents a un domaine vide, et envoie un message de type* ***Fail*** *à tous les agents d'où la fin de l'algorithme PC_BT. Si le problème DisCSP admet une solution, elle est forcément trouvée par l'un des processus de recherche d'après le théorème 6.3 et d'après la remarque 6.1. Donc, l'algorithme termine.*

6.1.6 Complexité

On rappelle que n est le nombre de variables, d est la taille des domaines et m est le nombre de contraintes. On suppose ici que chaque agent connaît sa propre position dans l'anneau. Dans la phase de traitement, à chaque instant, au sein de chaque processus de recherche, un agent reçoit un message (de son prédécesseur ou de son successeur). Complexité temporelle : dans le pire des cas, sur un agent s'exécutent n processus de recherche. Donc, dans le pire des cas un agent peut recevoir n messages et exécuter n opérations locales. Chaque opération locale nécessite de parcourir les d valeurs du domaine courant de la variable locale : pour chaque valeur on doit vérifier la consistance de cette valeur avec l'instanciation partielle reçue ce qui se fait au plus en $O(n)$. Donc dans le pire des cas, un agent exécute à chaque étape $O(d * n^2)$ opérations.

Complexité spatiale : toujours dans le pire des cas, n processus de recherche s'exécutent sur un agent. On a donc n domaines courants différents associés à la variable de l'agent et on a également, 1 domaine global. La complexité spatiale sur chaque agent est donc en $O(d * (n + 1))$.

6.2 Extension de l'approche

Nous présentons ici une extension de l'approche décrite ci-dessus associée à une technique de filtrage. Cette extension utilise d'une part les propriétés de l'architecture générique PC qui est basée sur le couplage de deux notions : la coopération et le parallélisme, et d'autre part permet de prendre en compte les propriétés des algorithmes de filtrage. Elle a donc pour objectif de bénéficier des avantages des deux approches (architecture générique PC et filtrage). Pour ce faire les agents établissent une consistance locale (au niveau de chaque agent) durant la phase de la procédure d'initialisation. Ensuite, à chaque réception d'un message de type **Info** qui contient une instanciation partielle CPA, l'agent tente d'étendre l'instanciation partielle à une solution. Si l'agent réussit à étendre cette instanciation alors avant de retransmettre à l'agent suivant le message d'instanciation partielle comme dans l'approche précédente, cette fois-ci l'agent effectue une tâche en plus qui consiste à filtrer les domaines des variables futures. Le filtrage consécutif à l'affectation de la variable x avec la valeur v consiste à supprimer du domaine de chaque variable y non affectée du voisinage de x les valeurs qui violent la contrainte liant x et y. Ce filtrage est mis en œuvre par un envoi d'un message de type **MAJ** qui contient l'identificateur du processus de recherche initiateur du filtrage. Ce message de type **MAJ** est envoyé aux agents voisins dont les variables ne sont pas encore instanciées. Ensuite, l'agent envoie un message de type **Info** à l'agent suivant dans l'anneau virtuel défini au

départ. Par contre, si un conflit est détecté alors l'agent envoie un message de type **Back** à son prédécesseur dans l'anneau.

Remarque 6.10 *On dit que deux agents sont voisins s'il existe au moins une contrainte qui relie leurs variables.*

De même que l'algorithme de backtrack BT implémenté dans l'architecture PC nous a permis d'obtenir l'algorithme PC_BT, de même on peut implémenter n'importe quelle variante des algorithmes de résolution dédiés aux CSP ou aux DisCSP dans cette architecture PC et définir ainsi de nouvelles stratégies d'hybridation distribuées.

6.2.1 Variante de l'algorithme de filtrage

Famille d'algorithmes PC

Pour tout algorithme de résolution de CSP ou DisCSP nommé NAME, on peut appliquer le mécanisme PC pour produire un nouvel algorithme de résolution de DisCSP nommé PC_*NAME*.

Chaque algorithme de résolution de CSP, c'est-à-dire BT, BJ, FC, ... peut ainsi permettre de générer un nouvel algorithme de DisCSP PC_BT, PC_BJ, PC_FC, ...

Il est clair que tout nouvel algorithme ainsi produit à partir d'un algorithme Name standard en le combinant à PC est complet et termine.

En effet, PC_*Name* est une application constituée de N processus de résolution *Name* qui travaillent de manière coopérative, où N est le nombre de variables dans le problème. Chacun d'eux se termine. Chacun s'exécute de manière indépendante et seule des données en lecture sont partagées. Chaque mémoire partagée n'est accessible en écriture que par un seul processus. Donc aucune section critique n'est possible. La fin d'un processus s'accompagne de l'envoi d'un message de type **Fail** ou **Success** en broadcast à tous les autres processus. Donc PC_*Name* se termine.

6.2.2 Variante de l'heuristique de choix

L'algorithme PC_BT constitue une approche qui permet d'exploiter certaines caractéristiques de CSP, qui bénéficie des avantages de la distribution de l'algorithme, et qui s'appuie sur un réel parallélisme. Ainsi, PC_BT bénéficie aussi de l'heuristique d'ordre d'instanciation des variables et des valeurs. Le premier point important : la notion de diversification dans l'arbre de recherche rend notre algorithme plus efficace, car il bénéficie de la coopération entre les différents processus de recherche. La coopération entre les agents est introduite au

moment de l'application des remarques 6.4 ou 6.5 par la réduction des domaines des variables. Il est clair que chaque processus de recherche s'exécute jusqu'à avoir une instanciation complète. Un agent est capable de diriger un processus de recherche dans l'arbre de recherche selon l'indicateur associé à chaque valeur de sa variable. Mais il n'y a pas d'échanges de données entre les différents processus de recherche.

Nous avons proposé une modification de l'heuristique de choix de la prochaine valeur ce qui nous a conduit à définir deux nouvelles versions d'algorithme PC_BT nommées successivement PC_BT$_2$ et PC_BT$_3$. Leurs principes consistent à modifier la propriété citée dans la remarque 6.4. La première version PC_BT$_2$ est basée sur l'heuristique min-conflict. Dans cette version l'indicateur associé à chaque valeur est incrémenté lors de chaque réception d'un message de type **Back**. L'indicateur de la valeur qui introduit le conflit sera incrémenté. Afin d'évaluer les performances de PC_BT et PC_BT$_2$, nous proposons une autre version nommée PC_BT$_3$, dans cette version l'ordre d'instanciation des valeurs est le même pour tous les agents. Dans le même esprit, d'autres stratégies dynamiques de choix des valeurs pourraient être conçues.

6.3 Résultats expérimentaux

Dans cette section, nous évaluons l'intérêt pratique de l'approche PC. Les travaux ont été implémentés dans le langage de programmation C/C++, et en utilisant la librairie MPI pour la communication entre les processus de recherche.

6.3.1 Validation expérimentale de PC_BT

Les premiers tests ont été réalisés sur le problème de coloriage d'un graphe. Le problème de coloriage d'un graphe consiste à trouver une coloration des n sommets d'un graphe à l'aide de k couleurs, de façon à ce que deux sommets adjacents aient une couleur différente.

Dans un premier temps, nous avons résolu le problème de coloriage sur des graphes de 64 variables et de densité de contraintes 0,5. On a fait varier le nombre de couleurs disponibles (et donc la taille des domaines des variables) entre 2 et 20. En conséquence, certains DisCSP sont sous-contraints, d'autres sont sur-contraints. Dans nos expérimentations, nous avons mesuré les moyennes des valeurs suivantes sur 15 graphes aléatoires différents : le temps CPU (temps), le nombre de messages (#msg) et le nombre de tests de consistance (#ccks). On a utilisé dans cette expérimentation 4 processeurs distincts sur lesquels les agents sont répartis.

k	#msg	#c-ccks	temps (ms)
2	482	786	11
4	13 026	119 571	372
6	365 386	8 502704	11 185
10	867 653	55 172950	30 605
13	1331090	148657644	49 327
14	230132	31532987	10 530
20	3618	620 583	119

TABLE 6.1 – *Résultats d'exécution de l'algorithme PC_BT sur le problème de coloriage d'un graphe.*

On observe que lorsqu'on a des contraintes fortes entre les agents, par exemple lorsque le nombre de couleurs est petit, l'algorithme converge rapidement vers une solution car l'espace de recherche est petit. Ce phénomène est dû à la réduction des domaines mentionné dans la remarque 6.5 qui précise que lorsqu'un agent reçoit un message de type Back dans le processus de recherche qu'il a initié, il peut supprimer cette valeur définitivement du domaine global initial de sa variable. À l'inverse, augmenter le nombre de couleurs revient à avoir des problèmes sous-contraints, c'est-à-dire des problèmes qui admettent beaucoup de solutions. Dans ce cas, c'est la diversification des parcours qui permet une convergence rapide. Ces premiers résultats permettent de valider expérimentalement la terminaison de PC_BT et la validité des résultats.

Dans un second temps, nous avons expérimenté l'algorithme PC_BT sur le coloriage de graphes aléatoires de 90 sommets et de densité 0,5. Ce type de graphes appartient à une classe de graphes qui nécessitent des temps de calcul importants. Nous avons fait varier k de 2 à 22. Nous avons effectué ces tests sur un cluster composé de 28 processeurs 18 Ghz, et 700 Mb de RAM chacun. Nous avons généré 50 instances pour chaque valeur de k. La table 6.2 donne les performances dans ce nouveau contexte. On peut constater que les temps de calcul sont raisonnables et que l'algorithme a un comportement satisfaisant.

TABLE 6.2 – Exécution de l'algorithme PCBT

k	#msg $*10^3$	#c-ccks $*10^3$	#temps (seconds)
2	0.248	0.376	0.26
4	293	546	2.05
6	428	1,947	22.8
8	1,124	4,234	150
10	1,467	7,589	322
12	1,676	12,672	844
14	1,789	15,674	1,123
16	1,860	19,866	1,265
17	2,197	25,765	1,307
18	1,145	6,785	16.9
20	6	73	0.85
22	5	42	0.09

6.3.2 Évaluation des performances parallèles de l'algorithme PC_BT

Classe(n, k, d)	T_1	T_2	T_4	$Acc = T_1/T_4$	Efficacité E
(16, 8, 0.5)	0.015	0.0071	0.0036	4.16	1.04
(32, 8, 0.5)	5.20	2.71	1.262	4.12	1.03
(64, 8, 0.5)	91.973	40.45	18.96	4.85	1.21

TABLE 6.3 – *Évaluation des performances parallèles du PC_BT*

Nous avons effectué des tests sur des graphes aléatoires de la classe (n, k, d) où n est le nombre de sommets, k est le nombre de couleurs utilisées et d est la densité du graphe. T1, T2, et T4, sont les temps d'exécution sur 1, 2, et 4 processeurs respectivement. On a calculé l'accélération $Acc = T_1/T_4$ et l'efficacité sur 4 processeurs $E = Acc/4$. Nous observons sur 4 processeurs des accélérations super-linéaires et une efficacité voisine de 1. Ce qui justifie l'utilisation du parallélisme. Les temps obtenus augmentent bien sûr en fonction de la taille du problème.

6.3.3 Comparaisons de différentes versions de PC_BT

Afin de mieux cerner l'efficacité de notre proposition, nous comparons la version PC_BT de base, avec deux autres versions. On rappelle que la version PC_BT_2 utilise l'heuristique min-conflict pour l'ordre d'instanciation des valeurs, et que la version PC_BT_3 utilise un ordre statique d'instanciation des valeurs, identique pour tous les processus de recherche. Pour cette comparaison, nous utilisons les instances de benchmarks DIMACS [76], et nous essayons

de colorier ces graphes avec le nombre chromatique présenté dans les fichiers DIMACS. Le but est de tester la validité de ces versions.

TABLE 6.4 – Comparaison entre PC_BT, PC_BT$_2$, et PC_BT$_3$ en utilisant des problèmes DIMACS consistants

Problème	PC_BT (ms)	PC_BT$_2$ (ms)	PC_BT$_3$ (ms)
queen5_5 (25, 5)	0.32	0.33	0.51
queen6_6 (36, 7)	23.1	28.1	42.4
queen7_7 (49, 7)	20.1	19.9	42.9
queen8_8 (64, 9))	32.1	31.6	42.4
queen9_9 (81, 10)	59.6	62.7	76.8
queen10_10 (100, 10)	66.8	68.9	90.8

On remarque que l'heuristique de diversification que nous avons proposée est au moins aussi performante que l'heuristique min-conflict. D'autres tests avec d'autres types de problèmes ont confirmé cette observation et l'écart ne permet pas réellement de distinguer ces deux heuristiques.

Le générateur aléatoire utilisé pour générer les instances d'un graphe est disponible sur le site suivant [1]

6.3.4 Tests sur des CSP aléatoires classiques

Afin de comparer PC_BT aux algorithmes classiques comme ABT ou AFC, nous étudions le comportement de différentes variantes proposées de PC_BT. Pour effectuer ces tests, nous utilisons le générateur aléatoire [2] des instances CSP. Un CSP aléatoire (n, m, p_1, p_2) est défini par quatre paramètres : n est le nombre de variables, m est la taille des domaines, p_1 est la densité du graphe de contraintes, et p_2 est la densité de tuples autorisés pour chaque contrainte. Nous avons généré 10 instances pour chaque entrée, et nous fixons $n = 15$, $m = 10$, $p_1 = 0.7$ et nous faisons varier p_2 de 0.1 à 0.9. Nous comparons les méthodes PC_BT et PC_FC présentées dans ce chapitre et l'algorithme ABT. La figure 6.2 illustre le nombre de messages circulant dans le réseau. Les courbes ABT, PC_BT et PC_FC représentent respectivement le nombre de messages échangés lors de l'exécution des algorithmes ABT Asynchronous Backtracking Algorithm, PC_BT et PC_FC. L'axe des abscisses représente la densité de tuples autorisés pour chaque contrainte.

1. `http://www.lita.univ-metz.fr/~saad/RGenerator`
2. `http://www.lirmm.fr/~bessiere/generator.html`

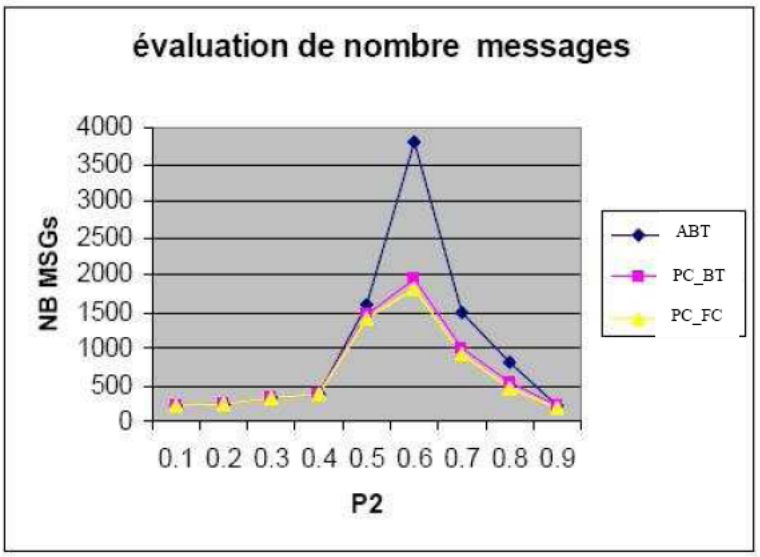


FIGURE 6.2 – Comparaison de PC_BT et PC_FC avec l'algorithme ABT

D'après les résultats expérimentaux, nous constatons que PC_BT et PC_FC induisent un nombre de messages inférieur à ABT. Par ailleurs, on constate également une légère réduction du nombre de messages de PC_FC par rapport à PC_BT grâce au mécanisme de filtrage. Le faible écart entre ces deux méthodes, s'explique par le surcoût important en communication du mécanisme de filtrage.

Un autre point qui découle naturellement de notre méthode est la résistance aux pannes. Dans un réseau de communication la perte des messages est possible, et la solution utilisée est la réplication de messages. Notre méthode ne réclame pas cette mise en œuvre grâce à l'utilisation de plusieurs processus de recherche. En effet, lorsqu'un processus de recherche tombe en panne, les autres peuvent continuer à explorer le reste de l'espace de recherche puisqu'ils sont indépendants. Notons bien qu'il s'agit ici de la panne d'un processus de recherche. Par contre, si c'est un agent qui est défaillant, notre algorithme ne pourra pas s'exécuter correctement. D'autres solutions seraient alors à définir pour pallier ce problème.

6.4 Conclusion

Notre objectif dans ce chapitre était de proposer une nouvelle méthode de résolution de DisCSP. Cette nouvelle méthode, nommée PC_BT (Distributed Parallel and Cooperative Backtracking algorithm), est une méthode énumérative qui exploite la distribution des données sur différents agents et qui génère une heuristique d'ordre d'instanciation des variables et des valeurs dans le but d'obtenir une coopération entre les différents processus de recherche. Ensuite, nous proposons une généralisation de cette méthode sur les différents algorithmes de résolution des CSP. En effet, il est possible de remplacer le principe du backtrack par celui du Forward-Checking ou de MAC, pour obtenir des variantes que l'on peut nommer PC_FC, PC_MAC, etc. De plus, notre algorithme peut également être étendu aux algorithmes DisCSP, c'est-à-dire que l'on peut concevoir des algorithmes de type PC_ABT (Parallel and Cooperative Asynchronous Backtracking) et PC_AFC (Parallel and Cooperative Asychronous Forward Ckecking) basés sur l'hybridation de PC avec les algorithmes ABT et AFC présentés au paragraphe 6.2. Les résultats expérimentaux ont montré une réduction conséquente du nombre de messages échangés par rapport à ABT.

Parmi d'autres extensions possibles de cette méthode, on peut également envisager la généralisation aux CSP n-aires et aux problèmes DisCSP sécurisés. D'ailleurs, puisque la sécurité des données constitue une de nos préoccupations majeures dans le contexte de l'application de ces travaux au sein du CRP Gabriel Lippmann, le chapitre suivant est consacré à la généralisation de la méthode PC_BT aux DisCSP sécurisés.

Chapitre 7

CSP sécurisés distribués

The personal life of every individual is based on secrecy, and perhaps it is partly for that reason that civilized man is so nervously anxious that personal privacy should be respected
Anton Chekhov

L'une des motivations d'apparition des modèles de CSP distribués (DisCSP) est la possibilité de rendre confidentielles les données détenues par chacun des agents [167]. Or, cette propriété de confidentialité est quasiment absente dans les algorithmes standards dédiés aux DisCSP [13, 67, 166]. Ce qui pousse les chercheurs à concevoir de nouveaux algorithmes qui s'appuient sur les techniques cryptographiques. Par conséquent, un nouveau modèle est décliné que l'on appelle *Secure DisCSP*. Les premiers travaux dans cette voie ont été abordés par Meseguer et Jimenez [89]. L'objectif de ce modèle est que le calcul reste confidentiel sans divulgation d'information aux participants. Autrement dit, aucun des participants à la résolution du problème ne connaît les données des autres participants (à l'exception de ses entrées/sorties). Dans Secure DisCSPs, le problème est distribué naturellement entre un ensemble d'agents autonomes où chaque agent souhaite garder ses contraintes et ses valeurs aussi privées que possible. Ainsi, les agents devront collaborer et communiquer pour trouver la solution globale. Dans la littérature, nous distinguons deux aspects pour protéger la confidentialité de données des individus (voir [15]) : (1) protéger les valeurs affectées aux variables de l'agent, qui ne doit pas divulguer les valeurs assignées à ses variables, (2) protéger les contraintes d'un agent.

Dans ce chapitre, nous présentons les travaux principaux relatifs aux DisCSPs sécurisés. En particulier nous détaillons les outils de sécurisation, le formalisme utilisé, les modèles et les algorithmes.

7.1 Définitions et présentation des outils

Le formalisme SDisCSP (Secure DisCSP) introduit la notion de préservation des données de chaque agent. Yao dans l'article [164] a abordé ce concept, en introduisant le problème des deux millionnaires Alice et Bob, qui veulent savoir lequel d'entre eux est le plus riche, sans révéler le montant précis de leur fortune. Pour résoudre ce problème, une stratégie communément utilisée est d'assumer la fiabilité des prestataires de services, ou d'assumer l'existence d'un tiers de confiance. Toutefois, cette solution n'est pas toujours acceptable. Par conséquent, d'autres solutions ont été proposées qui offrent des mécanismes de protection de la confidentialité des informations détenues par les participants. Depuis l'introduction de ce problème, connu sous l'appellation de problème du calcul sécuritaire multi-partie (Secure Multi-party Computation Problem), différentes solutions ont été proposées. Nous pouvons les classifier en deux catégories : (1) les approches cryptographiques et (2) les approches non-cryptographiques. Tout d'abord, nous donnons quelques définitions, puis nous décrivons ces deux approches.

7.1.1 Cryptographie

La cryptologie (du grec Kryptos voulant dire caché) signifie littéralement la science du secret. On peut la définir comme une science mathématique qui comporte deux branches : la cryptographie et la cryptanalyse.

Définition 7.1 *La **cryptographie** traditionnelle est l'étude des méthodes permettant de transmettre des données de manière confidentielle afin de protéger un message. On applique à ce message une transformation qui le rend incompréhensible à un tiers. C'est ce qu'on appelle le chiffrement qui, à partir d'un **texte clair** donne **un texte chiffré** ou **cryptogramme**. Inversement, le **déchiffrement** est l'action qui permet de reconstruire le texte en clair à partir du texte chiffré et des éléments liés au chiffrement.*

Définition 7.2 *La **cryptanalyse**, à l'inverse, est l'étude des procédés cryptographiques dans le but de trouver des faiblesses et en particulier, de pouvoir décrypter des textes chiffrés. Le décryptement est l'action consistant à retrouver le texte clair sans connaître les éléments liés au chiffrement.*

Définition 7.3 *Un **crypto-système** est l'ensemble des deux méthodes de chiffrement et de déchiffrement.*

Le but de la cryptographie est d'offrir un ou plusieurs services de sécurité comme la confidentialité, l'intégrité, l'authentification, ou la non-répudiation.

Pour cela, on utilise quelques mécanismes basés sur des algorithmes cryptographiques.

Il existe deux familles d'algorithmes cryptographiques à base de clefs : les algorithmes à clef symétrique ou à clef secrète, et les algorithmes à clefs asymétriques ou à clefs privée et publique.

Pour communiquer et créer des messages, les participants exploitent un certain nombre de primitives cryptographiques.

Algorithmes symétriques ou à clé secrète

Un algorithme symétrique est une fonction cryptographique dont la clé de chiffrement sert également au déchiffrement. Cette clé doit être partagée entre l'émetteur et le récepteur et doit être tenue secrète pour les autres. Ces algorithmes symétriques (DES [51, 103], AES [28]...) sont beaucoup plus rapides que les algorithmes asymétriques que l'on va présenter dans la section suivante, mais la distribution de la clé doit être confidentielle. Le nombre de clés nécessaires est important : en effet, pour assurer la communication cryptée entre n partenaires, il faut $\frac{n*(n-1)}{2}$ clés.

Algorithmes asymétriques ou à clé publique

Les algorithmes à clé symétrique nécessitent le partage d'une clé entre l'émetteur et le récepteur. Une faiblesse majeure de ces algorithmes est la difficulté pour l'émetteur et le récepteur de partager la clé au départ. Les algorithmes asymétriques permettent de résoudre cette difficulté. Ils sont basés sur différentes propriétés arithmétiques ou algébriques. Par exemple la méthode RSA [115] est basée sur des propriétés arithmétiques simples (congruences, algorithme d'Euclide, ...) pour générer la paire de clés nécessaires au fonctionnement de l'algorithme. La difficulté à calculer la clef privée à partir de la clef publique repose sur la difficulté à factoriser des produits de grands nombres premiers. On peut également citer les algorithmes de chiffrement ou de déchiffrement ElGamal [43] et Schnorr [136] qui reposent sur le problème des logarithmes discrets.

Une étude détaillée d'algorithmes de chiffrement symétriques ou asymétriques peut être trouvée dans [134].

Dans l'article [170], les auteurs utilisent une propriété de certains algorithmes asymétriques : l'homomorphisme. Cette propriété permet de préserver les données privées dans les DisCSP sécurisés comme nous verrons un peu plus loin dans ce chapitre.

Définition 7.4 (Un chiffrement homomorphique) *Un chiffrement est dit homomorphique s'il vérifie la propriété suivante : soient E la fonction de cryptage, k la clé privée, m_1 et m_2 des messages à chiffrer. Si $E(m_1, k)$ et $E(m_2, k)$*

est le résultat de chiffrement des textes m_1 et m_2 avec la clé publique k, alors il existe une fonction $f(k)$ qui vérifie la propriété $E(m_1, k) * E(m_2, k) = E(m_1 * m_2, f(k))$.

7.2 DisCSP sécurisés et résolution

Il existe deux méthodes principales pour préserver la confidentialité des données et des contraintes. La première méthode exclut l'utilisation des techniques de cryptage [139, 17]. La deuxième méthode par contre exploite précisément les outils cryptographiques pour dissimuler les valeurs et les contraintes aux agents coopérant [170, 141, 106].

7.2.1 Méthodes sans utilisation des techniques cryptographiques

Ces méthodes sont introduites pour la première fois par Meseguer et Jimenez [89]. Ils ont présenté deux modèles qui permettent de résoudre en partie la question de protection des données privées dans les CSP distribués.

Le premier modèle est basé sur la notion d'exploitation des contraintes partiellement connues. Il permet de partager les contraintes impliquant des variables détenues par des agents différents, tout en conservant la structure du problème. Par exemple, si on note C_{ij} l'ensemble des contraintes entre deux variables x_i et x_j qui sont possédées par deux agents différents, alors quelques contraintes de l'ensemble C_{ij} sont connues par l'agent qui détient la variable x_i, et les autres contraintes de l'ensemble C_{ij} sont connues par l'agent qui détient x_j. Ces contraintes partiellement connues peuvent être définies de la manière suivante :

- $C_{i(j)}$ contient les contraintes qui sont connues par l'agent qui détient la variable x_i. Dans ce modèle, chaque contrainte de $C_{i(j)}$ est définie par une liste des tuples interdits de $d_{x_i} \times d_{x_j}$.
- $C_{(i)j}$ contient les contraintes qui sont connues par l'agent qui détient la variable x_j. De même, chaque contrainte de $C_{(i)j}$ est définie par une liste des tuples interdits de $d_{x_i} \times d_{x_j}$.
- $C_{i(j)} \bigcup C_{(i)j} = C_{ij}$, l'union des deux sous-ensembles constitue l'ensemble initial de toutes les contraintes qui portent sur les deux variables.

Ce modèle est basé sur la notion de contraintes partiellement connues afin que chaque agent puisse avoir une vue partielle des contraintes qui portent sur ses variables. L'objectif est de pouvoir affecter une valeur à une variable de manière à satisfaire toutes les contraintes, et en révélant le moins possible les contraintes. Pour illustrer le principe des contraintes privées, il suffit de considérer le problème du jeu d'échec, dans lequel un agent est spécialisé dans

le déplacement d'une pièce. Par exemple, on peut considérer deux agents, l'un s'occupant de la reine et un autre s'occupant d'un cavalier. L'agent s'occupant de la reine ne connaît que les déplacements possibles pour cette pièce et ne connaît pas les déplacements possibles pour le cavalier. Les règles de déplacements sont des contraintes privées.

Le second modèle consiste à protéger les valeurs qui peuvent être affectées aux variables de chaque agent. Brito et al. [15, 17], ont proposé que les agents communiquent et s'échangent des affectations partielles. Plus précisément, chaque agent transmet à ses voisins l'ensemble des valeurs interdites à leurs variables. Autrement dit, lors de la vérification de ses contraintes locales, l'agent ne va pas envoyer la valeur ou les valeurs assignée(s) à sa/ses variables, par contre il va transmettre les valeurs interdites aux variables des agents voisins.

De plus, les agents sont ordonnés selon une priorité fixée dans une phase de pré-traitement. Il est clair que cet algorithme n'assure pas un niveau sécurité satisfaisant. En effet, les agents prioritaires vont réduire progressivement les domaines des agents les moins prioritaires. Finalement, l'agent le moins prioritaire aura un domaine fortement réduit dans lequel il sera statistiquement beaucoup plus probable de trouver la valeur instanciée à sa variable. De plus, lorsqu'un agent a un domaine vide, il doit informer les agents les plus prioritaires, afin qu'ils changent leurs valeurs. Ce qui constitue une information sur l'activité des agents prioritaires. On peut en déduire que cet algorithme assure son niveau de sécurité le meilleur lorsque le problème admet beaucoup de solutions, c'est-à-dire lorsqu'il y a peu d'échecs et que toutes les valeurs du domaine sont équiprobables.

7.2.2 Méthodes avec utilisation des techniques cryptographiques

Parmi ces méthodes, on peut citer l'algorithme de Yokoo et al. [170]. Les auteurs présentent, d'une manière très théorique, un modèle générique pour résoudre les DisCSP sécurisés. Ce modèle est basé sur l'utilisation d'un tiers de confiance, et sur la propriété d'homomorphisme. Il consiste à coder les contraintes sous la forme de matrices carrées de taille $m * m$, où m est la taille des domaines des variables. On note ces matrices $C^i_{x_i,x_j}$. La matrice $C^i_{x_i,x_j}$ code la contrainte connue par l'agent A_i concernant les deux variables x_i et x_j. Dans le cas où l'arité des contraintes est n, les matrices à construire sont des matrices dans R^n, donc très coûteuses en temps de calcul. Un élément de la matrice $C^i_{x_i,x_j}[d_i, d_j] = 1$ si le couple (d_i, d_j) est autorisé aux variables $(x_i, x_j$ et $C^i_{x_i,x_j}[d_i, d_j] = 0$ dans le cas contraire. Les élements des matrices sont cryptés

par une fonction de cryptage homomorphique E.

Soient par exemple deux agents A_1 et A_2. Chaque agent détient une variable, respectivement x_1 et x_2, et le domaine de chaque variable est $\{1, 2, 3, 4\}$. On suppose que l'agent A_1 connait la contrainte $x_1 = x_2$. L'agent A_1 construit alors la matrice suivante :

$$C^1_{x_1,x_2} = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & E(1) & E(Z) & E(Z) & E(Z) \\ 2 & E(Z) & E(1) & E(Z) & E(Z) \\ 3 & E(Z) & E(Z) & E(1) & E(Z) \\ 4 & E(Z) & E(Z) & E(Z) & E(1) \end{array}$$

De la même manière, l'agent A_2 génère la matrice de la contrainte qui relie les deux variables x_1 et x_2. On suppose que l'agent A_2 connait la contrainte $x_1 > x_2$. Donc, A_2 génère la matrice suivante :

$$C^2_{x_1,x_2} = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ 1 & E(Z) & E(Z) & E(Z) & E(Z) \\ 2 & E(1) & E(Z) & E(Z) & E(Z) \\ 3 & E(1) & E(1) & E(Z) & E(Z) \\ 4 & E(1) & E(1) & E(1) & E(Z) \end{array}$$

$C^i_{x_i,x_j}$ représente implicitement la contrainte entre les variables x_i et x_j. Elle est connue par l'agent A_i. Ces matrices sont envoyées aux agents intermédiaires appelés agents intelligents. Chaque agent intelligent est responsable d'une paire de variables x_i et x_j, sachant que l'agent A_i connaît une partie des contraintes reliant les variables x_i et x_j et que l'agent A_j en connaît l'autre partie. L'agent intelligent construit la matrice globale qui relie les deux variables en utilisant la propriété de chiffrement homomorphique ($E(a) * E(b) = E(a * b)$).

Ensuite, il effectue un certain nombre de permutations aléatoires sur les lignes et les colonnes de la matrice. La matrice ainsi permutée est envoyée aux serveurs externes (le « search controller »et les « decryptors »). Ces deux serveurs externes recherchent, en coopérant, la solution pour le problème en appliquant l'algorithme de Backtracking.

Cette méthode n'engendre aucune perte d'information privée des agents participant. Par contre, si les variables n'ont pas les mêmes tailles de domaine, les agents devront construire des matrices de taille $(m1+m2)*(m1+m2)$ avec $m1$ la taille du domaine de la variable x_1 et $m2$ la taille du domaine de la variable $x2$. Ce qui augmente significativement les communications entre les agents participants et les agents intelligents, ainsi que les communications entre le search

controller et les decryptors pour la recherche d'une solution. Un autre défaut majeur de cet algorithme est qu'il ne tire pas profit du parallélisme. Les agents participants restent au repos pendant que le calcul s'effectue au niveau des agents intelligents. De plus, à notre connaissance, il n'y a pas d'implémentation effectuée.

Une manière d'améliorer cet algorithme est d'utiliser une notation binaire (par exemple la position des valeurs dans le domaine) afin de n'utiliser que de simples opérateurs logiques (AND, OR, NOT). Il est également possible de s'inspirer de ce formalisme pour transformer des problèmes de satisfaction de contraintes en des problèmes de satisfaction d'expressions booléennes (SAT).

En 2005, un autre article proposé par Zivan et al. [106] améliore les travaux de Yokoo. Il présente deux protocoles de résolution de DisCSP sécurisés : un protocole centralisé et un autre distribué. Dans le protocole centralisé, les agents participants envoient leurs matrices chiffrées directement à un serveur central, qui réalise la recherche. Dans le protocole distribué, plusieurs serveurs coopèrent dans la recherche de solution, en ajoutant des liens de communication. Chaque serveur ne s'occupe que de deux agents participant. Ces travaux n'ont cependant pas été implémentés par les auteurs.

7.3 Conclusion

Nous observons que deux grandes catégories d'approches ont été proposées pour la préservation de la confidentialité des informations détenues par les agents. La première approche consiste en l'utilisation de perturbations du modèle de contraintes partiellement connues. La seconde approche utilise des techniques cryptographiques et en particulier la propriété d'homomorphisme de certains cryptosystèmes asymétriques. Cette seconde approche assure un niveau de sécurité supplémentaire par rapport à la première contre les attaques extérieures. Néanmoins, elle ne garantit pas un niveau plus élevé au niveau de la protection des données privées contre les participants. Au contraire, elle demande plus de calcul pour les opérations de chiffrement et déchiffrement. Nous observons aussi que l'intervention des intrus extérieurs, et les actes internes malveillants sont en général non traités dans ces modèles. Il est donc nécessaire de développer d'autres techniques qui permettent de renforcer la sécurité de ces modèles. Dans le chapitre suivant, nous proposons un compromis entre ces deux approches pour améliorer la confidentialité.

Chapitre 8

Approche parallèle, coopérative, et sécurisée

Privacy is not something that I'm merely entitled to, it's an absolute prerequisite.
Marlon Brando

Le chapitre 6 avait pour but de proposer une technique performante PC qui permet aux différentes entités d'un système distribué de collaborer pour résoudre le problème de manière efficace. Dans ce chapitre, nous nous intéressons plus particulièrement à raffiner cette stratégie afin de préserver les données privées de chaque agent sans utilisation d'un tiers de confiance. En particulier, nous présentons une version sécurisée de PC_BT qui exploite un mécanisme de chiffrement asymétrique. Ensuite, nous examinons les propriétés et les caractéristiques de cette technique, et la comparons avec les méthodes existantes.

8.1 La stratégie proposée

Dans cette section, nous allons décrire la stratégie proposée pour résoudre les Secure DisCSP sans utilisation d'un tiers de confiance. Cette stratégie réduit le coût des communications, garantit le même niveau de confidentialité que les protocoles existants présentés dans le chapitre 7, et utilise un chiffrement à clé publique. Une partie de ce travail a été présentée dans [120, 125].

8.1.1 Principe général

Dans la définition des DisCSP sécurisés, S est un ensemble des éléments du problème, et A est un ensemble d'agents qui détiennent chacun un sous

ensemble des connaissances. Le but est que ces agents travaillent de manière collaborative pour trouver une solution au problème partagé, tout en gardant quelques informations locales aussi confidentielles que possible. Les contraintes partagées entre plusieurs agents seront connues d'un seul agent. Toute contrainte d'un agent sera représentée en extension à partir des domaines des variables impliquées dans la contrainte. Nous notons que nous travaillons ici sur des CSP binaires à domaine fini. Chaque agent génère une paire de clés privée et publique (E, P) du schéma de cryptage asymétrique RSA pour chacune de ses variables. La clé publique est connue par l'ensemble des agents. La clé privée n'est connue que par l'agent qui génère cette clé. Après cette étape de génération de clés, chaque tuple de chaque relation associée à une contrainte est crypté avec les clés publiques des variables de la contrainte. Plus précisément, chaque colonne de la matrice de la relation, qui représente des valeurs qui peuvent être attribuées à une variable x_i, est chiffrée avec la clé publique de cette variable. L'idée de ce chiffrement est de résoudre le problème en ne travaillant qu'avec des données chiffrées. À la fin de l'algorithme, chaque agent peut déduire la valeur réelle de sa propre variable, en appliquant le déchiffrement avec sa clé privée. Dans la suite nous supposons que chaque agent ne détient qu'une seule variable du problème afin de faciliter la description de la stratégie.

8.1.2 Détail de la stratégie

Formellement, étant donné n agents : A_1, A_2,..., A_k, chacun dispose d'une variable x_i et d'un domaine d_i de valeurs propres à cette variable. En outre, les relations en extension sont réparties sur l'ensemble des agents. Plus précisément, les agents A_i et A_j connaissent tous deux l'ensemble des tuples autorisés par la contrainte $c_{i,j}$. De fait, en plus du domaine de sa propre variable, un agent connaît forcément partiellement le domaine des autres variables impliquées dans les contraintes dont il est responsable.

Lors de l'initialisation, chaque agent chiffre avec sa clé publique E_i les tuples associés à chaque contrainte $c_{i,j}$ et les partage avec l'agent A_j.

En définitive la contrainte $c_{i,j}$ est constituée d'un ensemble de tuples chiffrés à la fois avec la clé publique de A_i, et celle de A_j. Ces tuples cryptés sont connus des deux agents. Cette redondance du chiffrement est importante car elle permet de ne faire transiter entre les agents que des valeurs chiffrées. Une CPA est donc constituée des valeurs chiffrées $E_1(v_1), E_2(v_2), \ldots, E_i(v_i)$. L'agent A_j récepteur de cette CPA, choisit dans son domaine une valeur v_j pour laquelle il doit vérifier la consistance. Pour ce faire, il calcule $E_k(v_j)$ pour k de 1 à n. Puis il vérifie pour k de 1 à i si la contrainte $c_{k,j}$ existe et le cas échéant, si le couple $(E_k(v_k), E_k(v_j))$ est présent dans la liste des tuples autorisés de $c_{k,j}$.

Si v_j est consistante, l'agent A_j ajoute $E_j(v_j)$ dans la CPA et transmet toute la CPA ainsi cryptée à son successeur. Par contre, s'il ne trouve pas de valeur consistante, il renvoie la CPA à son prédécesseur avec un message BACK.

Lorsqu'une CPA complète est obtenue, on sait que le problème est consistant. Puis la CPA effectue un tour supplémentaire dans l'anneau pour que chaque agent remplace la valeur chiffrée par la valeur en clair.

Par exemple, soit le problème suivant avec trois variables $\{x_1, x_2, x_3\}$ et les domaines qui leur sont respectivement associés $D = \{d_1, d_2, d_3\}$. Soit C l'ensemble des contraintes $C = \{c_{1,2}, c_{2,3}\}$. Enfin, soient les trois agents $\{A_1, A_2, A_3\}$ responsables respectivement d'une variable x_1, x_2, x_3. Les tuples associés à $c_{1,2}$ sont $\{(1,7),(2,8)\}$ et les tuples associés à $c_{2,3}$ sont $\{(7,3),(2,5)\}$. A_1 et A_2 connaissent tous deux les valeurs cryptées des tuples de $c_{1,2}$ à savoir $(E_1(1), E_1(7)), (E_2(1), E_2(7)), (E_1(2), E_1(8)), (E_2(2), E_2(8))$. De même, A_2 et A_3 connaissent les valeurs des tuples de $c_{2,3}$ cryptées avec leurs clés publiques respectives E_2 et E_3.

Si A_2 reçoit la CPA $x_1 = E_1(2)$, alors A_2 peut choisir dans ses tuples cryptés le couple $(E_1(2), E_1(8))$ parce que le premier élément de ce couple correspond à la valeur de x_1 dans la CPA. Il doit ensuite rechercher dans le domaine de x_2 la valeur en clair dont le chiffrement est égal au deuxième élement du couple en l'occurrence $E_1(8)$ et affecter à x_2 cette valeur (égale à 8 dans notre exemple).

Il ajoute $x_2 = E_2(8)$ à la CPA et la transmet à son successeur.

La différence principale de cette méthode par rapport à la variante parallèle et coopérative PC introduite dans le chapitre 6 réside dans la procédure d'initialisation où chaque agent génère une paire de clefs publique et privée. Une clef publique déposée dans un espace accessible à tous les participants et une clef privée qui permet de déchiffrer le résultat à la fin de l'algorithme. De plus, la recherche de la consistance consiste simplement à vérifier que les relations en extension contiennent les tuples chiffrés avec des valeurs qui permettent de prolonger la CPA.

8.2 Discussion

Nous discutons dans cette section des travaux de sécurisation des problèmes de satisfaction de contraintes distribués, et nous comparons notre stratégie avec les stratégies existantes.

La propriété fondamentale de notre algorithme est de préserver les caractéristiques de coopération et de parallélisme et toutes les propriétés de l'algorithme PC_BT vues au chapitre 6. En effet, hormis l'initialisation, dans le déroulement de l'algorithme, la principale modification introduite par le chiffrement est située dans la recherche de la consistance. Tout le déroulement de PC_BT est

conditionné par ce test de consistance. Si une valeur consistante est trouvée, un message INFO est envoyé au successeur. Sinon, un message BACK est envoyé au prédécesseur.

La complexité reste identique à la constante près induite par le coût du chiffrement lors de l'initialisation. De même le nombre de messages échangés reste identique au cours d'exécution de l'algorithme.

Pour évaluer la robustesse au sens de la sécurité, de ce protocole, il convient de se placer au niveau d'un agent malveillant ou d'un attaquant extérieur à l'ensemble des agents. Par conséquent, nous pouvons identifier dans ce type de problème, deux adversaires : l'adversaire externe et l'adversaire interne.

Définition 8.1 (adversaire externe) *Un adversaire externe ne participe pas à la recherche de la solution du problème et souhaite obtenir une information sur le comportement des agents participant.*

Définition 8.2 (adversaire interne) *Un adversaire interne est un agent qui participe à la résolution du problème. Cet agent détient une partie de la solution et souhaite obtenir la totalité des informations des autres agents.*

Un agent non fiable de manière intentionnelle ou non, et qui perturbe la résolution en transmettant des informations fausses, ou en ayant un comportement contraire à toute la procédure ne peut pas être détecté. Ce problème de fiabilité ne relève pas de la proposition faite ici qui se concentre sur la confidentialité.

Du point de vue de l'adversaire interne, les informations dont il dispose sont ses contraintes internes et les valeurs chiffrées des CPA. Il peut éventuellement collectionner les valeurs chiffrées des CPA qui transitent par lui. Mais il serait alors contraint de déchiffrer ces valeurs, ce qui ne peut se faire qu'en ayant connaissance des clés privées.

Pour ce qui est de l'attaquant externe, il ne connaît aucune contrainte et ne peut que écouter le contenu des messages. Ces messages contiennent des valeurs chiffrées qui sont là encore insuffisantes pour déduire les valeurs claires et les domaines des variables.

En définitive, l'attaquant externe ou interne ne peut connaître que les domaines chiffrés des variables, puisque ce sont les seules informations qui transitent.

En résumé, cet algorithme a la même complexité que la version non sécurisée, il n'utilise pas de tiers de confiance contrairement à [170] et il est sûr d'un point de vue cryptographique puisque sans les clés privées les attaquants ne peuvent pas obtenir d'information sur les valeurs autorisées et sur les domaines.

8.3 Conclusion

Nous avons défini une nouvelle stratégie pour préserver la confidentialité des informations dans un DisCSP. Le problème est réparti sur des agents sur lesquels sont distribuées les variables, leurs domaines, ainsi que les contraintes exprimées en extension. La confidentialité des données est assurée par un schéma de cryptographie asymétrique. Cette stratégie ne nécessite pas de tiers de confiance. Les seules informations échangées entre les agents sont des CPA dont les valeurs sont chiffrées chacune avec la clé publique de l'agent qui en est responsable. Seul cet agent pourra déchiffrer cette valeur grâce à sa clé privée. L'algorithme conserve toutes les propriétés de l'algorithme PC_BT vu au chapitre 6. Enfin, le point fort de cette méthode est de permettre l'adaptation facile des algorithmes de résolution des DisCSP aux DisCSP sécurisés.

Chapitre 9

Conclusion & perspectives

A good begining,
Makes a good ending.

Conclusions et perspectives

Le travail présenté dans cette thèse consiste à utiliser le formalisme CSP pour résoudre efficacement les problèmes à base de connaissances dans un contexte distribué et sécurisé. Nous nous sommes dans un premier temps intéressés à l'hybridation des CSP et des SE à base de connaissances, puis à l'élaboration d'un protocole de communication dans un environnement distribué et enfin à la préservation des données privées de chaque entité participant. Nous avons montré comment l'hybridation peut permettre de résoudre certains problèmes complexes dans la pratique, en donnant trois stratégies d'hybridation possibles. Pour mener à bien cette hybridation, nous avons élaboré un algorithme de transformation des SE vers les CSP, domaine qui a été peu traité jusqu'à présent. Cela a abouti en particulier à un nouveau formalisme qui permet de modéliser à l'aide de contraintes des problèmes initialement exprimés dans un système à base de connaissances. Ce formalisme a été nommé DDCSP (Dynamic Domain Constraint Satisfaction Problem). Nous avons proposé deux approches pour la résolution des DDCSP. La première stratégie consiste à explorer l'espace de recherche jusqu'à la fin puis à mettre à jour les relations qui ont été modifiées. Ensuite, le processus de recherche est réitéré. La seconde est caractérisée par le redémarrage immédiat du processus de recherche à chaque fois qu'une relation est modifiée.

Ensuite, dans la deuxième partie de la thèse nous nous sommes intéressés au développement d'un mécanisme de communication entre plusieurs entités distribuées. Cette contribution se traduit par la définition d'une approche

PC (Parallèle et Coopérative). Un des principaux avantages de l'approche PC est qu'elle bénéficie des avantages du parallélisme et de la coopération entre plusieurs processus de recherche. La première proposition de l'approche PC, appelée PC_BT, s'appuie sur un algorithme de BackTrack et sur la distribution du problème sur plusieurs agents. Cet algorithme utilise une topologie en anneau pour la communication entre les différents agents. On a montré qu'il est complet et qu'il termine. Un des principaux intérêts de cet algorithme est qu'il bénéficie des avantages du parallélisme, qu'il permet de varier l'ordre d'instanciation des variables et des valeurs, et qu'il garantit une diversité de l'exploration de l'arbre de recherche par les processus de recherche. Des variantes de la méthode PC ont été proposées en faisant varier les algorithmes de filtrage ou les heuristiques de choix des valeurs. Enfin, nous avons proposé une extension de l'algorithme PC_BT à un algorithme de résolution de DisCSP sécurisés, basé sur des techniques cryptographiques asymétriques.

En perspective, ces travaux ouvrent trois nouvelles pistes de recherche. La première piste consiste en l'hybridation entre le CSP et le SE. A l'heure actuelle, seule l'hybridation concurrente a été implémentée. Ces résultats doivent être complétés par l'implémentation des autres schémas d'hybridation proposées dans le chapitre 3. Les jeux d'essais testés devront également être étendus à d'autres problèmes plus variés.

La deuxième piste de recherche consiste à valoriser le nouveau formalisme proposé (DDCSP). La méthode de résolution proposée dans ce document consiste simplement à appeler dans un processus itératif un solveur de CSP standard. Par la suite, de nouveaux algorithmes spécifiques plus efficaces devront être conçus.

Enfin, la troisième piste de recherche consiste à évaluer expérimentalement de nouvelles variantes de l'approche PC, notamment en diversifiant les techniques de filtrage, sur différents jeux de tests afin de les comparer.

L'objectif de cette thèse était de proposer un système distribué et sécurisé pour intégrer des contraintes dans un moteur d'inférence. Cet objectif est atteint dans le sens où toutes les briques de l'architecture cible ont été conçues et développées. Mais la phase d'intégration reste encore à effectuer et sera l'objet de nos travaux futurs.

Bibliographie

[1] D. Allouche, S. Givry, T. Schiex. Towards Parallel Non Serial Dynamic Programming for Solving Hard Weighted CSP. CP'2010, pp. 53-60, 2010.

[2] J. Amilphastre. Représentation par automate d'ensemble de solutions de problèmes de satisfaction de contraintes. Thèse de doctorat, Université Montpellier II, Montpellier, France, janvier 1999.

[3] K. R. Apt. Logic programming, in : J. van Leeuven (Ed.), Handbook of Theoretical Computer Science. MIT Press, Cambridge, MA, pp. 493-574, 1990.

[4] P. Berlandier and B. Neveu. Arc-consistency for dynamic constraint problems : A RMS free approach. In Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications, 1994.

[5] C. Bessière. Arc-consistency in dynamic constraint satisfaction problems. In Proceedings of AAAI-91, pp. 221-226, Anaheim, CA, 1991.

[6] C. Bessière. Arc-consistency and arc-consistency again. Artificial Intelligence, volume 65, pp.179-190, 1994.

[7] C. Bessière and J. C. Régin. Arc consistency for general constraint networks : preliminary results. In Proceedings IJCAI'97, pp. 398-404, Nagoya, Japon, 1997.

[8] C.Bessière and J. C. Régin. Refining The Basic Constraint Propagation Algorithm. In Proceedings of the 17th International Joint Conference on Artificial Intelligence, pp. 309-315, 2001.

[9] C. Bessière, A. Maestre and P. Meseguer. Distributed Dynamic Backtracking. In Proc. IJCAI-01 Workshop on Distributed Constraint Reasoning, 2001.

[10] C. Bessière and P. Van Hentenryck. Etre ou ne pas être... une contrainte globale, In Proceedings JNPC'03, pp.67-81, Amiens, France, 2003.

[11] C. Bessiere, A. Chmeiss and L. Sais. Neighborhood-based variable ordering heuristics for the constraint satisfaction problem. In Proceedings CP'01, Paphos, Cyprus, pp. 565-569, 2001.

[12] C. Bessière, I. Brito, A. Maestre, and P. Meseguer. Asynchronous backtracking family. Technical Report 03139, 2003.

[13] C. Bessière, A. Maestre, I. Brito and P. Meseguer. Asynchronous backtracking without adding links : a new member in the ABT family. Artificial Interlligence, volume 161, pp. 7-24, 2005.

[14] L. Brisoux. Coopération consistante de bases de connaissance au premier ordre fini. Dans Actes des Journées jeunes chercheurs Ganymède, pp. 24-28, Lille, France, 28 mai 1998.

[15] I. Brito and P. Meseguer. Distributed Forward Checking. In Proc. 2003, 9th International Conference, Kinsale, Irelande, pp. 801-806, 2003.

[16] I. Brito and P. Meseguer. Synchronous, Asynchronous and hybrid algorithms for DisCSP. In Mark Wallace editor, Principales and Practic of Constraint Programming. 2004.

[17] I. Brito Rodringuez. Distributed Constraint Satisfaction. PhD thesis, University Barcelona, Spain, 2007.

[18] A. Bruin, G. Kindervater and H. Trienekens. Asynchronous Parallel Branch-and- Bound and Anomalies. In A. Ferreira and J. Rolim, editors, Parallel Algorithms for Irregularly Structured Problems, volume 980 of Lecture Notes in Computer Science, Springer-Verlag, 1995.

[19] A. Chmeiss. Sur la consistance de chemin et ses formes partielles. In Actes du Congrès AFCET-RFIA 96, pages 212-219, Rennes, France, 1996.

[20] A. Chmeiss and P. Jégou. Efficient Path-Consistency Propagation. International Journal of Artificial Intelligence Tools,volume 7, pp.121-142, 1998.

[21] S. Clearwater, B. Huberman, and T. Hogg. Cooperative Solution of Constraint Satisfaction Problems. Science, Volume 254, pp. 1181-1183, 1991.

[22] S. Clearwater, T. Hogg, and B. Huberman. Cooperative Problem Solving. In B. Huberman, editor, Computation : The Micro and the Macro View,World Scientific, pp. 33-70, 1992.

[23] J. Conrad and D. Agrawal. Asynchronous Parallel Arc Consistency Algorithms on a Distributed Memory Machine. Journal of Parallel and Distributed Computing, 24(1), pp. 27-40, 1995.

[24] M. Cooper. An Optimal K-Consistency Algorithm. Artificial Intelligence, 41, 1989.

[25] P. Cooper and M. Swain. Arc consistency : parallelism and domain dependence. Artificial Intelligence, 58 :207-235, 1992.

[26] E. Coquery. Logique classique, Notes de cours. `http://liris.cnrs.fr/~ecoquery/dokuwiki/doku.php?id=enseignement:logique:start`.

[27] R. Cowan. Expert systems : aspects of and limitations to the codifiability of knowledge. Research Policy, Volume 30, Issue 9, pp. 1355-1372, December 2001,

[28] J. Daemen and V. Rijmen. The design of Rijndael : AES. the Advanced Encryption Standard. Springer-Verlag, Berlin, Germany/ Heidelberg, Germany/London, UK/etc., 2002.

[29] R. Debruyne. Les algorithmes d'arc-consistance dynamiques. Revue d'intelligence artificielle 9(3) :239-267, 1995.

[30] R. Debruyne and C. Bessière. Domain Filtering Consistencies. Journal of Artificial Intelligence Research, 14 :205-230, 2001.

[31] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. In Proceedings of the tenth International Joint Conference on Artificial Intelligence (IJCAI-89), pp. 271-277, Detroit, Michigan, 1989.

[32] P. Dehornoy. Logique et théorie des ensembles, Notes de cours, FIMFA ENS, version 2006-2007.

[33] H.D. Ebbinghaus, J. Flum, and W. Thomas. Mathematical logic. Second edition, Springer, 1994.

[34] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. Communications of the ACM 28, pp. 637-647, 1985.

[35] E. A. Feigenbaum. Knowledge Engineering for the 1980's. Computer Science Dept., Standford University, California, 1982.

[36] M. Fieschi. Expert systems for medical consultation Health Policy. Volume 6, Issue 2, pp. 159-173, 1986.

[37] E. Freuder. Synthesizing constraint expressions. CACM, 21(11), pp. 958-966, 1978.

[38] E. Freuder. A Sufficient Condition for Backtrack-free Search. JACM, 29 , pp. 24-32, 1982.

[39] E. Freuder. A Sufficient Condition for Backtrack-Bounded Search. JACM, 32 , pp. 755-761, 1985.

[40] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 95), pp. 572-578, Montréal, Canada, 1995.

[41] MPI Forum. MPI : A Message Passing Interface standard. International Journal of super computer Application, 8(3/4), pp. 165-416, MIT press, 1994.

[42] C. Forgy. Rete : A Fast Algorithm for the Many Patterns/Many Objects Match Problem. Artificial Intelligence, 19(1), pp. 17-37, 1982.

[43] T. El Gamal. A Public Key Cryptosystem a signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory, IT 31(4), pp. 469-472, 1985.

[44] J. G. Gasching. Performance Measurement and Analysis of Certain Search Algorithms. Technical Report CMU-CS, pp. 79-124, Carnegie-Mellon University, 1979.

[45] J. Gasching, R. Reboh and J. Reiter. Development of a Knowledge-Based System for Water Resources Problems. Technical Report, SRI Project 1619, SRI International, 1981.

[46] M.R Garey and D.S Johnson. Computers and Intractability - A guide to the theory of NP-Completeness. Freeman and Company, 1979.

[47] E. Gelle and B. Faltings. Solving Mixed and Conditional Constraint Satisfaction Problems. Constraints, pp. 107-141, 2003.

[48] F. Geller and M. Veksler. Assumption-Based Pruning in Conditional CSP. In VAN BEEK, pp. 241-255, 2005.

[49] R. Greenstadt, J. P. Pearce, E. Bowring and M. Tambe. Experimental analysis of privacy loss in dcop algorithms. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1424-1426, ACM, 2006.

[50] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. In Proceedings of the tenth International Joint Conference on Artificial Intelligence (IJCAI-89), pp. 271-277, Detroit, Michigan, 1989.

[51] W. Diffie and M. Hellman. Exhaustive cryptanalysis of the NBS Data Encryption Standard. IEEE Computer, volume 10, pp. 74-84, 1977.

[52] R. Génisson, B. Benhamou and A. Rauzy. Une version concurrente de la procédure de Davis et Putnam. Revue d'intelligence artificielle, volume 10, pp. 499-506, Avril 1996.

[53] I.P. Gent and T. Walsh. CSPLib : a benchmark library for constraints. In : 5th International Conference on Principles and Practice of Constraints Programming, vol. 1, pp. 480-481, Alexandria, Virginia, USA, 1999.

[54] A. Gavilanes-Franco and F. Lucio-Carrasco. A first order logic for partial functions. Theoretical Computer Science, Volume 74, Issue 1, pp. 37-69, 1990.

[55] D. Gale and L. S. Shapley. College Admissions and the Stability of Marriage. American Mathematical Monthly 69, 9-14, 1962.

[56] A. Geist, A. Beguelin, J. Dongarra, R. Manchek and V. Sunderman. PVM : Parallel Virtuel Machine. MIT press, 1994.

[57] E. Gelle. On the generation of locally Consistent solution spaces in mixed dynamic constraint problems. Ph.D. thesis, École Polytechnique Fédérale de Lausanne, 1998.

[58] J. Gaschnig. A general Backtracking algorithm that eliminates most redundant tests. In Proceedings IJCAI'77, 1977.

[59] J. Giarratono and G. Riley. Expert systems : Principles and programming. PWS Publishing Company, Boston, 1998.

[60] M.L Ginsberg. Dynamic backtracking. J.of Artificial Intelligence research, volume 1, pp. 25-46,1993.

[61] F. Glover and M. Laguna. Tabou search. In Modern Heuristic Techniques for Combinatorial Problems, Oxford, England, 1993.

[62] S. Golumb and L. Baumbert. Backtrack programming. In journal of ACM, volume 12, pp. 516-524, 1965.

[63] M. Golumbic. Algorithmic Graph Theory and Perfect Graphs. Academic Press, New York, 1980.

[64] P.M.D. Gray, S.M. Embury, K.-Y. Hui and G.J.L. Kemp. The Evolving Role of Constraints in the Functional Data Model, J. Intelligent Information Systems (12), pp. 113-137, 1999.

[65] R.F. Grove. Design and development of knowledge-based systems on the web. In : Proceedings of ISCA 2000 : Ninth International Conference on Intelligence Systems, Louisville, KY, USA, International Society of Computer Applications (USCA), pp. 147-150, 2000.

[66] Y. Hamadi. Traitement des problèmes de satisfaction de contraintes distribués. PhD thesis, University of Montpellier II, July 1999.

[67] Y. Hamadi. Interleaved backtracking in distributed constraint networks. International Journal on Artificial Intelligence Tools, 11(2), pp. 167-188, 2002.

[68] Y. Hamadi, S. Jabbour and L. Sais. ManySAT : a parallel SAT solver. In Journal on Satisfiability, Boolean Modeling and Computation, JSAT, Volume 6, Special Issue on Parallel SAT, pp. 245-262, IOS Press, 2009.

[69] R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence, 14, pp. 263-313, 1980.

[70] F. Hayes Roth. Rule based systems. Communications of the ACM 28(9), pp. 921-932, 1985.

[71] T. Hogg and C.P. Williams. Expected Gains from Parallelizing Constraint Solving for Hard Problems. In Proceedings of AAAI 94, pp. 331-336, Seattle, WA, 1994.

[72] P.V. Hentenryck. The OLP Optimization Programming Language. The MIT Press, Cambridge, Massachusetts, 1999.

[73] P. Jégou. Decomposition of Domains Based on the Micro-Structure of Finite Constraint Satisfaction Problems. In Proceedings of AAAI 93, pp. 731-736, Washingtown, DC, 1993.

[74] N. Jennings, K. Sycara and M.Wooldridge. A roadmap of agent research and development. Autonomous Agents and Multi-Agent Systems (AAMAS), 1(1), pp. 7-38, 1998.

[75] H. Jiang and J.M. Vidal. Reducing Redundant messages in the asynchronous backtracking algorithm. Sixth International Workshop on Distributed Constraint Reasoning, 2005.

[76] D.S. Johnson and M.A. Trick. Proceedings of the 2nd DIMACS Implementation Challenge. DIMACS Series in Discrete Mathematics and Theoretical Computer Science 26, American Mathematical Society, 1996.

[77] S. Kasif. On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks. Artificial Intelligence, 45(3), pp. 275-286, 1990.

[78] S. Kasif and A. Delcher. Local consistency in parallel constraint satisfaction networks. Artificial Intelligence, 69, pp. 307-327, 1994.

[79] R. M. Karp and Y. Zhang. Randomized Parallel Algorithms for Backtrack Search and Branch and Bound Computation. Journal of the Association for Computing Machinery, 40(3), pp. 765-789, 1993.

[80] A. Kazaz. Application of a Expert System on the Fracture Mechanics of Concrete. Artificial Intelligence, pp. 177-190, 2003.

[81] M. Kifer and E. Lozinskii. RI : a logic for reasonning about inconsistency. TARK IV. Asilomar. CA, pp. 253-262, 1988.

[82] S. Kirkpatrick, C. Gelatt and M. Vecchi. Optimization by simulated annealing. In Science, volume 200, pp. 671-680, 1983.

[83] S.H. Liao. Expert system methodologies and applications-a decade review from 1995 to 2004. Expert Systems with applications, Volume 8, Issue 1, pp. 93-103, 2005.

[84] L. Lobjois and M. Lemaitre. Coopération entre méthodes complètes et incomplètes pour la résolution de (V)CSP : une tentative d'inventaire. In journées Nationales de la Résolution Pratique de Problèmes NP-Complets, pp. 67-73, Rennes, France.

[85] G.F. Luger. Artificial Intelligence : Structures and Strategies for Complex Problem Solving. Addison-Wesley, Fourth Edition, UK, 2002.

[86] H. P. Lundsgaarde. Evaluating medical expert systems. Social Science and Medicine. Volume 24, Issue 10, pp. 805-819, 1987.

[87] A. Mackworth. Consistency in Networks of Relations. Artificial Intelligence, volume 8, pp.99-118, 1977.

[88] K. Marriott, M. Wallace, P. J. Stuckey. Constraint logic programming, Foundations of Artificial Intelligence, Volume 2, pp. 409-452, 2006.

[89] P. Meseguer and M. A. Jimenez. Distributed forward checking. In Proc. CP-2000 Workshop on Distributed Constraint Satisfaction, Singapore, September 2000.

[90] W. Van Melle. MYCIN : a knowledge-based consultation program for infectious disease diagnosis. International Journal of Man-Machine Studies, Volume 10, Issue 3, pp. 313-322, May 1978,

[91] F. Melvin and L.M. Richard. First-Order Modal Logic. Kluwer Academic Publishers, Dordrecht, 1998.

[92] A. Meisels. Distributed Constraints Satisfaction Algorithm. Performance, Communication, 2004.

[93] A. Meisels and R. Zivan. Asynchronous Forward Checking for distributed CSP. In Frontiers in Artificial Intelligence and Applications. IOS Press, 2004.

[94] P. Mérel. Les problèmes de satisfaction de contraintes : recherche n-aire et parallélisme - Application au placement en CAO. Ph.D. thesis, Université de Metz, 1998.

[95] R.H. Michaelson, D. Michie and A. Boulanger. The Technology of Expert Systems. Byte, volume 10, pp. 303-312, 1985.

[96] M. Minsky. A framework for representing Knowledge. The psychology of computer vision, MCGraw-Hill, New York, pp. 211-281, 1975.

[97] S. Minton, M. D. Johnston, A. B. Philips and P. Laird. Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence 58, pp. 160-205, 1992.

[98] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In AAAI, pp. 25-32, Boston, US, 1990.

[99] U. Montanari. Networks of Constraints : Fundamental Properties and Applications to Picture Processing. Artificial Intelligence, Volume 7, pp.95-132, 1974.

[100] S. Montani, and R. Bellazzi. Supporting decisions in medical applications : The knowledge management perspective. International Journal of medical Informatics. Vol 68, pp. 79-90, 2002.

[101] R. Mohr and T. C. Henderson. Arc and Path Consistency Revisited. Artificial Intelligence, 28, 1986.

[102] J. Naganuma and T. Ogura. A Highly or-parallel Inference Machine (multi-ASCA) and its Performance Evaluation : an Architecture and its Load Balancing Algorithms. IEEE Transaction on Computers, 43(9), pp. 1062-1075, 1994.

[103] National Bureau of Standards. FIPS Publication 46-1 : Data Encryption Standard, January 1988.

[104] S. Nason and J. E. Laird. Soar-RL : integrating reinforcement learning with Soar Cognitive Systems Research, Volume 6, Issue 1, pp. 51-59, March 2005,

[105] L. Nedovic and V. Devedzic. Expert systems in finance - a cross section of the field. Expert Systems with Applications, Volume 23, Issue 1, pp. 49-66, July 2002.

[106] K. Nissim and R. Zivan. Secure DisCSP protocols - from centralized towards distributed solutions. In Proc. 6th workshop on Distributed Constraints Reasoning, DCR-05, 2005.

[107] G. Nydahl. Expert systems in education : new opportunities, or threats, to the learning environment ? Education and Computing, Volume 8, Issues 1-2, pp. 107-111, June 1992.

[108] I. Popescu. Combining Constraints and Consistency Techniques. Advances in information system, LNCS 1909, pp. 191-200, Springer-Verlag, 2000.

[109] N. Prcovic, B. Neveu and P. Berlandier. Distribution de l'arbre de recherche des problèmes de satisfaction de contraintes en domaines finis. In 2ème Conférence Nationale sur la Résolution de Problèmes NP-Complets, pp. 275-290. JNPC 96, 1996.

[110] P. Prosser. Hybrid Algorithms for the constraint satisfaction problem. Computational Intelligence, volume 9, pp. 268-299, 1993.

[111] M. Rabin. How to exchange secrets by oblivious transfer. Tech. Report Memo TR-81, Aiken Computation Laboratory, 1981.

[112] N. Rao and V. Kumar. Parallel depth-first search. Part I : Implementation. International Journal of Parallel Programming, 16 :479-499, 1987.

[113] J. M. Richer. Une approche de résolution de problèmes en logique des prédicats fondée sur des techniques de satisfaction de contraintes. Université de Dijon, 1999.

[114] J. M. Richer and J.J. Chabrier. Sacre : a Constraint Satisfaction Based Theorem Prover. Proceedings of AAAI-99, Orlando FL, USA, July 1999.

[115] R.L. Rivest, A. Shamir and L.M. Adleman. A method for obtaining digital signature and public-key cryptosystems. Communications of the ACM, 21 , pp. 120-126, 1978.

[116] C. Roucairol. Exploration parallèle d'espace de recherche en Recherche Opérationnelle et Intelligence Artificielle. Masson, Dans Algorithmique Parallèle de M. Cosnard, M. Nivat et Y. Robert, chap. 14, pp. 201-211, 1992.

[117] S. J. Russell and P. Norvig. Artificial Intelligence. A Modern Approach. Prentice Hall, 2003.

[118] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier : Mixing protocol based on the cooperation and the parallelism for distributed environment. In : 7th IEEE International Conference on Computer and Information Science, Portland, Oregon, USA (2008).

[119] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier. Résolution des problèmes de satisfaction de contraintes distribués à travers le parallélisme, la coopération, et le filtrage. Colloque sur l'Optimisation et les systèmes d'Information COSI'2008, 8-10 juin 2008, Tizi-Ouzou, Algérie.

[120] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier : a privacy preserving technique for distributed constraint satisfaction problems. In : 1st International Symposium on Operational Research ISOR 2008, 2-6 November 2008, Algiers, Algeria.

[121] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier. DDCSP : Constraint Satisfaction Problem Used For Rule-based System. International Conference on Data Storage and Data Engineering (DSDE 2010).

[122] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier. L'hybridation la coopération et le parallélisme pour construire des protocoles robustes pour les CSP distribués. 9ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Clermont Ferrand, France, Février, 2008.

[123] B. Saad, F. Herrmann and T. Tamisier. La préservation de la vie privée des problèmes de satisfaction de contraintes distributes. 9ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Clermont Ferrand, France, Février, 2008.

[124] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier. Efficient solving of distributed constraint satisfaction problems through cooperation and parallelism. The 12th World Systemics, Cybernetics and Informatics : WMSCI 2008. Orlando, Florida, USA, June 29th - July 2nd, 2008.

[125] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier. Secure distributed constraint satisfaction problems without using trusted third-party. The 2nd International Conference on Modelling Computation and Optimization in Information Systems and Management Sciences : MCO 2008. Luxembourg, Metz, September , 2008.

[126] B. Saad, F. Herrmann, Y. Lanuel and T. Tamisier. Toward a Multicore Reasoning System. 3th Workshop, in Network of the Greater Region for Operational Research and Decision Support Systems. Working Group "GreatRoad". Luxembourg, January, 2010.

[127] D. Sabin and E. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In Proceedings of eleventh ECAI, pp. 125-129, 1994.

[128] D. Sabin and E. C. Freuder. Configuration as Composite Constraint Satisfaction. In Proceedings of the First Artificial Intelligence and Manufacturing Research Planning Workshop, 1996.

[129] N. Sadeh and M. Fox. Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. Technical report, CMU-RI-TR-95-39, Robotics Institute, Carnegie Mellon University, 1995.

[130] A. Samal and T. Henderson. Parallel Consistent Labeling Algorithms. International Journal of Parallel Programming, 16(5), pp.341-364, 1987.

[131] P. Sanders. Better algorithms for parallel backtracking. In A. Ferreira and J. Rolim, editors, Parallel Algorithms for Irregularly Structured Problems, volume 980 of Lecture Notes in Computer Science, pp. 333-347, Springer-Verlag, 1995.

[132] A. Shamir. How to share a secret. Communication of the ACM 22, no. 11, pp. 612-613, 1979.

[133] T. Schiex. Réseaux de contraintes. Habilitation à diriger les recherches, 2000.

[134] B. Schneier. Applied cryptography : Protocols, algorithms, and source code in C. John Wiley and Sons Inc., 1996.

[135] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde and B. Wielinga. Knowledge Engineering and Management : The CommonKADS Methodology. The MIT Press, Cambridge, Massachusetts, 2000.

[136] C.P. Schnorr. Efficient Signature Generation for Smart Cards. In proceedings of Advances in Cryptology. CRYPTO'89. pp. 239-252, 1990.

[137] M.C. Silaghi. Asynchronous solving Problems with Privacy Requirements. PhD thesis, Swiss Federal Institute of Technology (EPFL), 2002.

[138] M.C. Silaghi, D. Sam-Haroud and B. Faltings. Hybridizing ABT and AWC into a polynomial space complete protocol with reordering. Technical Report, EPFL, Lausanne, 2001.

[139] M. C. Silaghi. Asynchronously Solving Distributed Problems With Privacy Requirements. PhD thesis, 2002.

[140] M.C. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In Proc. of the Third International Conference on Intelligence Agent Technology, pp. 531-535, 2004.

[141] M.C. Silaghi and V. Rajeshirke. The effect of policies for selecting the solution of a discsp on privacy loss. In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1396-1397, IEEE Computer Society, 2004.

[142] M. Singh. Path Consistency Revisited. In Proceedings of the 5th International Conference on Tools for Artificial Intelligence, pp. 318-325, 1995.

[143] M. Singh. Path Consistency Revisited. IJAIT, 5, pp. 127-141, 1996.

[144] T. Soininen and E. Gelle. Dynamic Constraint Satisfaction in Configuration. In AAAI'99, Workshop on Configuration, pp. 95-100, Orlando, Floride, USA, 1999.

[145] G. Soloterevsky and E. Gudes. Algorithms for Solving Distributed Constraint Satisfaction Problems (DCSPs), in AIPS, pp. 191-198, 1996.

[146] T. Tamisier, F. Feltz, and Y. Didry. Cadral : an automated decision-making framework for business collaboration, 12th World Multiconference on Systemics, Cybernetics and Informatics (WMSCI), Orlando, USA, 2008.

[147] Teradata Corporation : DBC/1012 database computer concepts and facilities. Technical Report teradata Documenr C02-001-05, Teradata Corparation, 1988.

[148] R. Tucho, J. M. Sierra, J. E. Fernández, R. Vijande and G. Morís. Expert tutoring system for teaching mechanical engineering. Expert Systems with Applications, Volume 24, Issue 4, pp. 415-424, May 2003.

[149] É. Vareilles. Conception et approches par propagation de contraintes : contribution à la mise en ouvre d'un outil d'aide interactif. Thèse de doctorat, Institut National Polytechnique de Toulouse, 2005.

[150] G. Verfaillie and T. Schiex. Dynamic backtracking for dynamic CSP. Dans Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications, 1994.

[151] G. Verfaillie and T. Schiex. Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes : bilan de quelques approches. Revue d'Intelligence Artificielle 9(3), pp. 269-309, 1995.

[152] M. Veron et M. Aldanondo. Yet Another Approach to CCSP for configuration problem. In ECAI'00, European Conference on Artificial Intelligence, Workshop on Configuration, pp. 59-62, Berlin, Allemagne, 2000.

[153] P.R.S. Visser and V.A.M. Tamma. An Experiment with Ontology-Based Agent Clustering. IJCAI-99 Workshop on Ontologies and Problem-Solving Methods : Lessons Learned and Future Trends, Stockholm, Sweden, August, 1999.

[154] R.J. Wallace and E.C. Freuder. Constraint-based multi-agent meeting Scheduling : Effects of agents heterogeneity on privacy loss. In M. Yokoo editor, AAMAS, 2002.

[155] R.J. Wallace. Reasoning with possibilities in multiagent graph coloring. In 4th Intern Workshop on Distributed Consraint Reasoning, pp. 122-130, 2003.

[156] R.J. Wallace and E.C.Freuder. Constraint-based reasoning and privacy/ efficiency tradeoffs in multi-agent problem solving. Artificial Intelligence 161, pp. 209-227, 2005.

[157] D. Waltz. Understanding line drawings of scenes with shadows. In the Psychology of Computer Vision, pp. 19-91, P.H. Winston, McGraw Hill, New York, 1975.

[158] D. Weyns, H. V. D. Parunak, F. Michel, T. Holvoet and J. Ferber. Environments for multiagent systems state-of-the-art and research challenges. Lecture Notes in Computer Science, 3374, pp. 1-47. Springer, 2004.

[159] T. Winograd. Frame Representation and the Declarative/Procedural Controversy. In Representation and Understanding : Studies in Cognitive Science, ed. A. Collins, Academic Press, New York, 1975.

[160] M. Wooldridge and N. R. Jennings. Agent theories, architectures, and languages : a survey. In the workshop on agent theories, architectures, and languages on Intelligent agents (ECAI-94), pp. 1-39, New York, NY, USA, Springer-Verlag New York, 1995.

[161] R. M. Wygant. CLIPS - A powerful development and delivery expert system tool. Computers Industrial Engineering, 17, 1-4, pp. 546-549, 1989.

[162] Y. Xia, V. K. Prasanna. Parallel exact inference on the Cell Broadband Engine processor. Parallel and Distributed Computing, Volume 70, Issue 5, pp. 558-572, 2010.

[163] P. Xidonas, E. Ergazakis, K. Ergazakis, K. Metaxiotis, D. Askounis, G. Mavrotas and J. Psarras. On the selection of equity securities : An expert systems methodology and an application on the Athens Stock Exchange. Expert Systems with Applications, 36, 9, pp. 11966-11980, November 2009.

[164] A. Yao. Protocols for secure computations. Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, 1982.

[165] M. Yokoo, E.H. Durfee, T. Ishida and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In 12th International Conference on Distributed Computing Systems (ICDCS-92), pp. 614-621, 1992.

[166] M. Yokoo. Asynchronous weak-commitement search for solving distributed constraint satisfaction problems. In Proceedings the First International Conference on Principale and Practice of Constraint Programming (CP-95), pp. 88-102, 1995.

[167] M. Yokoo, E.H. Durfee, T. Ishida and K. Kuwabara. The distributed constraint satisfaction problem : formalization and algorithms. IEEE Trans. Knowledge Data Engineering, 10(5), pp. 673-685, 1998.

[168] M. Yokoo and T. Ishida. Algorithms for agents. In G. Weiss editor, Multiagent Systems, pp. 165-199. MIT Press, 1999.

[169] M. Yokoo. Algorithms for Distributed Constraint Satisfaction Satisfaction problems : A review. Autonomous Agents & Multi-Agent System, 3, pp. 198-212, 2000.

[170] M. Yokoo, K. Suzuki and K. Hirayama. Secure distributed constraint satisfaction : reaching agreement without revealing private information. Artificial Intelligence, 161, pp. 229-245, 2005.

[171] Y. Zhang and A. Mackworth. Parallel and Distributed Finite Constraint Satisfaction : Complexity, Algorithms and Experiments. Parallel Processing for Artificial Intelligence. Elsevier Sience Publishers B.V., chap. 1, 1993.

[172] R. Zivan and A. Meisels. Synchronous vs asynchronous search on DisCSP. In Proceedings European. Wokshop on Multi Agent system, EUMAS, Oxford, 2003.

[173] R. Zivan and A. Meisels. Concurrent backtrack search for DisCSP. In Proceedings FLAIRS-04, Maiami Florida, 2004.

[174] R. Zivan and A. Meisels. Concurrent dynamic backtracking for distributed CSP. In Proceedings Constraint Programming, pp. 782-787, 2004.

[175] Page Web presentant Prolog. `http://www.swi-prolog`.

[176] Portail du logiciel Soar. `http://sitemaker.umich.edu/soar/home`.

[177] Portail du logiciel Choco. `http://www.emn.fr/z-info/choco-solver/`

[178] Portail du logiciel Clips. `http://clipsrules.sourceforge.net/OnlineDocs.html`