République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université A.Mira de Béjaïa

Faculté des Sciences Exactes

Département d'Informatique

Mémoire de fin d'études

En vue de l'obtention du diplôme de Master Recherche

En Informatique

Option :"Réseaux et Systèmes Distribués"

Thème

? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?

Réalisé par :
Zidoune Nadjim

Devant le jury composé de :
Président : M.

Promotion : 2010

Remerciements

Dédicaces

Table des matières

1

Introduction générale

De nombreux problèmes réels peuvent être représentés sous la forme d'un problème de satisfaction de contraintes (CSP). En particulier, le formalisme CSP permet d'exprimer des problèmes de coloration de graphes, d'ordonnancement, de vision, de conception, de configuration, etc. Il vise à représenter sous forme de contraintes les propriétés et les relations qui existent entre les objets manipulés. Ces contraintes peuvent être décrites de multiples façons (par une équation, une inéquation, un prédicat, une fonction booléenne, une énumération des combinaisons de valeurs autorisées, etc.). Elles traduisent l'autorisation ou l'interdiction d'une combinaison de valeurs. Dans le formalisme CSP, la recherche d'une solution requiert de satisfaire toutes les contraintes. Dans la mesure où ces contraintes doivent être obligatoirement satisfaites, on les qualifie généralement de contraintes "dures". Cependant, pour certains problèmes réels, certaines contraintes (dites "molles") ne traduisent, dans la réalité, qu'une préférence, une possibilité, . . . Leur satisfaction n'est donc pas forcément nécessaire. Représenter ces contraintes par des contraintes dures rend souvent les CSP correspondants inconsistants. Aussi, afin de pouvoir exprimer de telles contraintes, plusieurs extensions du formalisme CSP ont été proposées. Le formalisme VCSP augmente le pouvoir d'expression du formalisme CSP en introduisant une graduation dans la violation des contraintes. Une valeur (appelée valuation) est associée a chaque contrainte. La valuation

d'une contrainte traduit l'importance de la violation de cette contrainte. Autrement dit, le cadre VCSP autorise la violation de certaines contraintes, les contraintes étant soit dures, soit molles. L'objectif est alors de trouver une affectation de toutes les variables qui optimise un critère donné portant sur la satisfaction des contraintes. En d'autres termes, une solution du problème est une affectation qui peut éventuellement violer certaines contraintes et dont l'importance des violations est minimale suivant un critère et un ordre donnés. Le formalisme VCSP permet donc l'expression de problèmes d'optimisation.

2

Etat de l'art

La notion de contraintes donne un cadre de formalisation très expressif de plus en plus largement utilisé dans de nombreux travaux en intelligence artificielle, notamment ceux concernant les problèmes de satisfaction de contraintes (Contraint Satisfaction Problem : CSP). Le formalisme CSP, tout en permettant de représenter très simplement des problèmes, est extraordinairement puissant dans la mesure où il est possible de représenter un éventail très large de problèmes.

En outre, l'utilité de ce formalisme, en dehors de son expressivité, réside dans l'efficacité des méthodes de résolution qui y ont été développées durant les 30 dernières années. Une contrainte est une propriété sur différents objets qui donnent des restrictions sur les valeurs simultanées de ces objets. On peut ainsi représenter des problèmes académiques purement combinatoires, des problèmes d'analyse de scènes, de contraintes temporelles, de recherche opérationnelle, d'ordonnancement, de planification, de configuration,. . ., grâce a des contraintes algébriques, temporelles, géométriques. . .

Les problèmes de satisfaction de contrainte sont des problèmes mathématiques où l'on cherche des états ou des objets satisfaisant un certain nombre de contraintes ou de critères.

Les CSP font l'objet de recherches intenses à la fois en intelligence artificielle et en

recherche opérationnelle. De nombreux CSP nécessitent la combinaison d'heuristiques et de méthodes d'optimisation combinatoire pour être résolu en un temps raisonnable.

2.1 Le formalisme CSP[1]

Un problème de satisfaction de contraintes est défini par un ensemble de variables, associée chacune a un domaine discret de valeurs de taille finie et par un ensemble de contraintes qui mettent en relation les variables et définissent des combinaisons de valeurs compatibles.

Définition 2.1.1 Un problème de satisfaction de contraintes (CSP) est un quadruplet (X, D, C, R), où :

- $X = \{X_1, ..., X_n\}$ est un ensemble de n variables.
- $D = \{D_1, ..., D_n\}$ est un ensemble de n domaines D_i finis, Chaque domaine est associé à une variable X_i.
- $C = \{C_1, ..., C_m\}$ est un ensemble de m contraintes. Chaque contrainte C_i est définie par un ensemble de n_i variables$X = \{X_{i_1}, ..., X_{i_{n_i}}\} \in X$.
- $R = \{R_1, \ldots., R_m\}$ est un ensemble de m relations. Chaque relation R_i définit l'ensemble des n_i-uplets sur $D_{i_1} \times ...D_{i_{ni}}$ Dini autorisés par la contrainte C_i.

Définition 2.1.2 Une contrainte est une relation logique (une propriété qui doit être vérifiée) entre différentes variables, chacune prenant ses valeurs dans un ensemble donné, appelé domaine. Une contrainte restreint les valeurs que peuvent prendre simultanément les variables, elle peut être :

- Implicite, définie par une expression logique. Par exemple : $X + Y < 4$.
- Explicite, définie par une relation composée de toute les tuples autorisés.Par exemple : pour un domaine $D = \{1, 2, 3\}$ de X et Y , la contrainte précédente est R = {(1, 1), (1, 2), (2, 1)}. Les variables sur lesquelles porte la contrainte C_i sont appelées Portée de la contrainte,noté $S(C_i)$.

L'arité d'une contrainte $C_i = X_{i_1}, \ldots, X_{i_{n_i}}$ est le nombre ini de variables sur lesquelles porte C_i. Un CSP est dit binaire si pour toute contrainte C_i inclut C, l'arité de C_i est au plus de 2, sinon il est n-aire.

Définition 2.1.3 Le réseau de contraintes associé à un CSP est un graphe (pour les CSP binaire) ou un hypergraphe (pour les CSP n-aire) dont les noeuds sont les variables et les arêtes sont les contraintes.

Définition 2.1.4 Etant donné un CSP, $P = (X, D, C, R)$ et soit $Y X$ un sous ensemble de variables de$X$. On appelle instanciation de Yl'association de chaque variable$y$ de Y a une valeur de $D(y)$ (avec $D(y)$ est le domaine de la variable y).

Définition 2.1.5 Etant donné un CSP, $P = (X, D, C, R)$ et soit $Y \subset X$ un sous ensemble de variables de X. Une instanciation des variables de Y sur D est dite consistante si et seulement si elle satisfait toutes les contraintes portant sur ses variables.

Définition 2.1.6 On appelle une solution d'un CSP $P = (X, D, C, R)$, l'instanciation consistante des variables de X sur$D$. Une instance CSP est dite consistante si elle possède au moins une solution.

2.2 Graphes, hypergraphes et décompositions

Dans cette section, nous allons rappeler des notions incontournables dans la suite de la théorie des graphes. La logique voudrait sans doute que les notions de graphe et hypergraphe soient présentées en même temps sachant qu'un graphe est un cas particulier d'hypergraphes. Cependant pour des raisons pédagogiques, il est préférable de commencer par la définition de graphe.

Définition 2.2.1 : Un graphe orienté G est une paire (X, C) avec X l'ensemble des sommets du graphe et C l'ensemble de ses arcs.

Les arcs du graphe sont des paires de sommets orientées (x_i, x_j) (arc de x_i vers x_j), $1 \leq i, j \leq n$.

Définition 2.2.2 : Un graphe non orienté G est une paire (X, C) avec X l'ensemble des sommets du graphe et C l'ensemble de ses arêtes.

Une arête est un ensemble de deux sommets $\{x_i; x_j\}, 1 \leq i \neq j \leq n$. Nous pouvons constater qu'il existe une orientation dans un arc alors qu'il en existe aucune dans une arête.

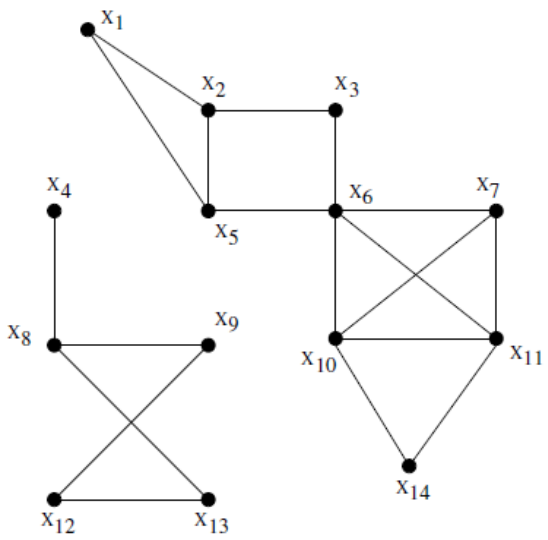

FIGURE 2.1 – Un graphe non orienté.

Définition 2.2.3 : Le sous-graphe de G induit par V (noté $G[V]$) est le graphe (V, C') tel que $C' = \{\{x_i, x_j\} \in C | x_i \in V, x_j \in V$.

Définition 2.2.14 : Deux sommets x_i et x_j sont voisins dans G si $\{x_i, x_j\} \in C$.

Définition 2.2.5 : On appelle voisinage d'un sommet x_i dans G, l'ensemble $N(x_i) = \{x_i \in X\ x_j$ et x_j sont voisins dans $G\}$.

On appelle voisinage fermé de x_i, l'ensemble $N[x_i] = N(x_i) \cup \{x_i\}$. Pour un sous-ensemble de sommets V , le voisinage de V est l'ensemble $N(V) = \bigcup_{x \in V} N(x) - V$ et son voisinage fermé est l'ensemble $N[V] = \bigcup_{x \in V} N(x)$

Définition 2.2.6 : Une chaine entre deux sommets $x_i et x_j$ dans G est une séquence $(x_{u_1}, x_{u_2}, ..., x_{u_p})$ de sommets de G telle que :

- $x_{u_1} = x_i, x_{u_p} = x_j$
- $\forall v, 1 \leq v \leq p-1, x_{u_v}$ et $x_{u_{v=1}}$ sont voisins

Deux sommets sont donc mutuellement accessibles s'il existe une chaine qui les relie. La définition suivante va introduire une notion centrale : *la connexité.*

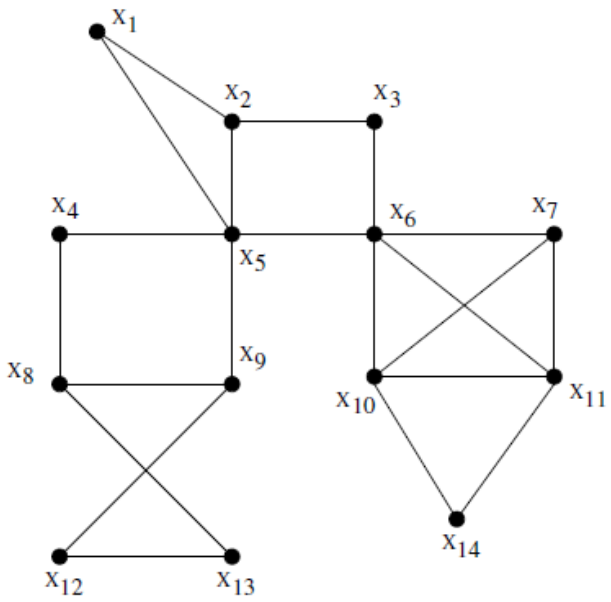

FIGURE 2.2 – Un graphe connexe.

Définition 2.2.7 :G est connexe ssi tous ses sommets sont mutuellement accessibles.

Le graphe de la figure 2.1 n'est pas connexe contrairement a celui de la figure 2.2.

Un graphe non connexe peut étre décomposé en plusieurs sous-graphes connexes donnés par les composantes connexes.

Définition 2.2.8 :$V \subset X$ est une composante connexe de G ssi $G[V]$ est un graphe connexe et il n'existe aucun ensemble V' tel que $V \subsetneq V'$ et $G[V']$ est un graphe connexe.

La relation d'accessibilité mutuelle étant une relation d'équivalence, les composantes connexes représentent donc les composantes maximales de cette relation.

Les ensembles $V = \{x_4, x_8, x_9, x_{12}, x_{13}\}$ et $V' = \{x_1, x_2, x_3, x_5, x_6, x_7, x_{10}, x11; x14\}$ sont les composantes connexes du graphe non connexe de la figure 2.1.

Définition 2.2.9 : Soient $G = (X, C)$ un graphe, x_i et x_j deux sommets de G. $S \subset X$ est un $\{x_i, x_j\}$-séparateur ssi x_i et x_j sont dans deux composantes connexes différentes de $G[X - S]$.

S est un $\{x_i, x_j\}$-séparateur minimal ssi il n'existe aucun ensemble S' tel que $S' \subsetneq S$ et S' est également un $\{x_i, x_j\}$-séparateur.

On appelle séparateur de G tout $\{x_i, x_j\}$-séparateur .

On appelle séparateur minimal de G tout $\{x_i, x_j\}$-séparateur minimal.

Donc S est séparateur de G ssi $G[X - S]$ admet au moins deux composantes connexes.

Les composantes connexes de $G[X-S]$ sont dites induites par S. Un séparateur minimal de G peut contenir un autre séparateur de G. En effet, S est un séparateur minimal de G si S est un séparateur de G qui n'en contient pas un autre dont les composantes connexes induites contiennent celles de S. Cette observation donne une caractérisation différente de la minimalité des séparateurs : S est un séparateur minimal de G s'il induit au moins deux composantes connexes CC_1 et CC_2 telles que :$N(CC1) = N(CC2) = S$.

L'ensemble de sommets $\{x_5, x_6\}$ du graphe de la *figure 2.2*, est un séparateur minimal qui induit trois composantes connexes $\{x_1, x_2, x_3\}$, $\{x_4, x_8, x_9, x_{10}, x_{13}\}$ et $\{x_7, x_{10}, x_{11}, x_{14}\}$.

$\{x_4, x_5, x_6\}$ est également un séparateur, mais il n'est pas minimal. Ses composantes connexes induites $\{x_1, x_2, x_3\}$, $\{x_8 x_9 x_{12} x_{13}\}$ et $\{x_7, x_{10}, x_{11}, x_{14}\}$ sont contenues dans celles de $\{x_5, x_6\}$. Par contre $\{x_7, x_{10}\}$ n'est pas un séparateur car son retrait du graphe le laisse connexe.

Définition 2.2.10 : Un cycle dans G est une chaine $(x_{u_1}, x_{u_2}, ..., x_{u_p})$, $p \geq 4$ qui contient au moins trois sommets distincts, et tel que $x_{u_1} = x_{u_p}$.

Définition 2.2.11 : Un graphe G est acyclique ssi il ne contient pas de cycle.

Un arbre est un graphe connexe et acyclique. Si on retire la propriété de connexité a un arbre, on se retrouve avec la définition d'une forêt.

2.2.1 hypergraphe

soit $X = \{x_1, x_2, ..., x_n\}$ un ensemble fini de sommets . Unehyper arête est un sous-ensemble non vide de X.La différence entre une arête et une hyper arête réside dans l'arité, c'est-a-dire le nombre de sommets dans l'ensemble. Une arête est d'arité 2, alors que celle d'une hyper arête est quelconque. Une arête est donc un cas particulier d'hyper arête.

Définition 2.2.12 : Un hypergraphe Hest une paire (X, C) avec X l'ensemble des sommets du graphe et C l'ensemble de ses hyper arêtes.

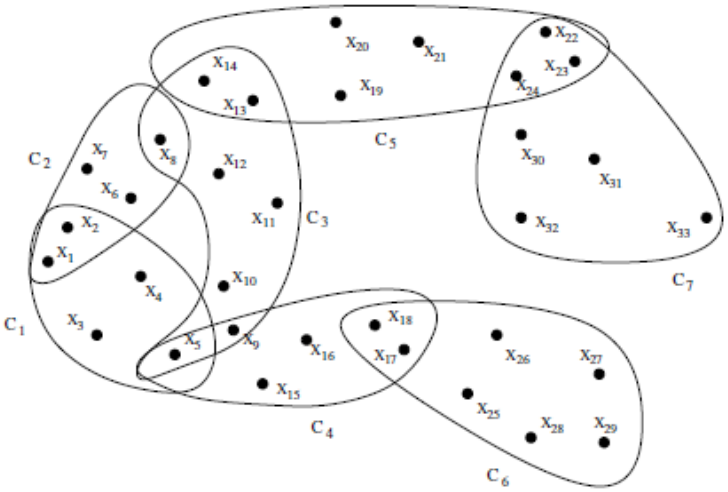

FIGURE 2.3 – Un hypergraphe.

Un hypergraphe qui ne contient que des hyper arêtes d'arité 2 est un graphe. La notion de graphe est donc un cas particulier d'hypergraphe. Dans certains cas, il peut être beaucoup plus facile de travailler sur une représentation d'un hypergraphe en graphe.

Définition 2.2.13 : Soit$V$ un sous-ensemble de X. L'ensemble des hyper arêtes partielles induites par V dans H est $C' = \{c', c' = c \cap V, c \in C\} - \emptyset$

On dit de cet ensemble qu'il est généré par des sommets. Comme dans le cas des graphes, il est possible de définir la notion de sous-hypergraphe induit.

Définition 2.2.14 : Soit $V \subset X$. Le sous-hypergraphe de H induit par V est l'hypergraphe $H' = (V, C')$ avec C', l'ensemble des hyperarêtes partielles induites par V dans H.

Définition 2.2.15 : Un hypergraphe H est connexe ssi toutes ses hyperarêtes sont mutuellement accessibles.

Les composantes connexes sont les composantes maximales de la relation d'équivalence d'accessibilité mutuelle

Définition 2.2.16 : Soient C un ensemble connexe, réduit, d'hyperarêtes partielles, c_1

et c_2 deux éléments de C et $q = c_1 \cap c_2$. q est une articulation de C si le retrait de q de toutes les hyperarêtes de C décompose C en au moins deux composantes connexes.

Dans l'hypergraphe $H = (X, C)$ de la figure 2.3, $q = c_4 \cap c_6 = \{x_{17}, x_{18}\}$ est une articulation de C. En effet, son retrait de H induit deux composantes connexes : $\{\{x_1, x_2, x_3, x_4, x_5\}$, $\{x_1, x_2, x_6, x_7, x_8\}$, $\{x_5, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}\}$, $\{x_5, x_9, x_{15}, x_{16}\}$, $\{x_{13}, x_{14}, x_{19}, x_{20}, x_{21}, x_{22}, x_{23}, x_{24}\}$ $\{x_{22}, x_{23}, x_{24}, x_{30}, x_{31}, x_{32}, x_{33}\}\}$ et $\{\{x_{25}, x_{26}, x_{27}, x_{28}, x_{29}\}\}$.

La notion d'acyclicité définie de manière très naturelle a partir de la notion de cycle dans les graphes, admet de manière surprenante des généralisations multiples au niveau des hypergraphes.

Toutes ces définitions, qui ont été introduites essentiellement dans le cadre de travaux sur les Bases de Données Relationnelles cherchent à capturer des propriétés sur ces dernières. La plus connue et sans doute la plus utilisée est celle d'α-acyclicité.

Définition 2.2.16 : Un hypergraphe H est α-acyclique si tout ensemble d'hyperarêtes partielles, qui est connexe, réduit, induit par un ensemble de sommets et qui n'admet pas d'articulation, est trivial (ne contient qu'un unique élément).

L'hypergraphe H de la figure 2.3 n'est pas acyclique car l'ensemble d'hyperarêtes partielles $\{\{x_1, x_2, x_3, x_4, x_5\}, \{x_1, x_2, x_6, x_7, x_8\}, \{x_5, x_8, x_9, x_{10}, x_{11}, x_{12}\}\}$ est connexe, réduit, non trivial et n'admet pas d'articulation. Par contre celui de la figure 2.4 est acyclique.

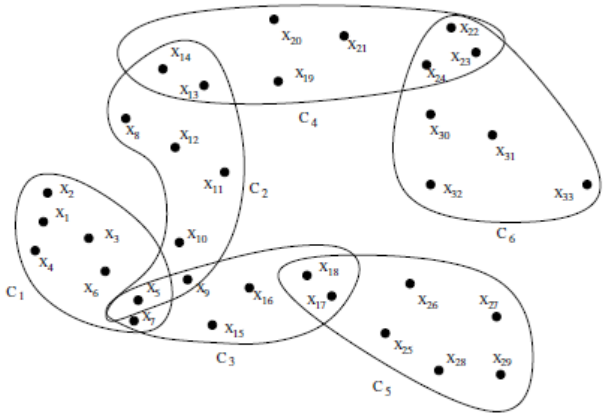

FIGURE 2.4 – Un hypergraphe acyclique.

Définition 2.2.17 : Un arbre des jointures d'un hypergraphe H est un arbre connexe T dont les noeuds sont les hyperarêtes de H tels que si un sommet x de H est contenu dans

deux hyperarêtes c_i et c_j de H alors il est contenu dans l'ensemble des noeuds de l'unique chaine reliant c_i et c_j dans T.

En d'autres termes, l'ensemble des noeuds contenant x induit un sous-arbre connexe de T.

La *figure 2.5* présente un arbre de jointure de l'hypergraphe de *la figure 2.4*. L'hypergraphe de la figure 2.4 vérifie la running intersection. Avec l'ordre $\theta = (c_2, c_3, c_1, c_5, c_4, c_6)$ sur ses hyperarêtes, nous avons bien l'intersection d'une hyperarête avec ces prédécesseurs dans l'ordre qui est contenue dans l'un d'entre eux.

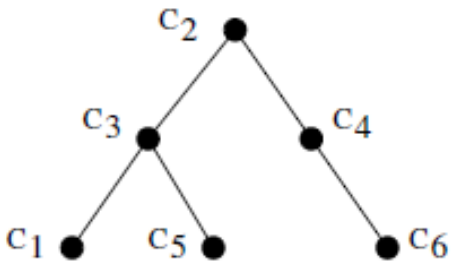

FIGURE 2.5 – Un hypergraphe acyclique.

2.3 Extension au probléme de satisfaction de contraintes valué

Si le formalisme des problèmes de satisfaction de contraintes offre un cadre de modélisation a de nombreux problèmes, en revanche, il ne semble pas apporter une réponse satisfaisante a la modélisation des problèmes dans lesquels entre en jeu la notion de cout ou de préférence. Ainsi, alors que dans les CSP classiques, la notion de contrainte est définie au sens strict du terme (on emploie alors la terminologie de contrainte dure), dans laquelle une combinaison de valeurs peut être autorisée ou interdite, pour de nombreux problèmes réels, le concept de contrainte ne traduit en fait qu'une préférence pour certains tuples de valeurs, ou encore un cout de violation (notion de contrainte molle).

Devant le constat de l'inadéquation de la modélisation de ce genre de relations de préférence par des contraintes dures, plusieurs extensions au cadre CSP ont été proposées. Nous nous intéressons ici au cadre des problèmes de satisfaction de contraintes valués.

En outre, ce cadre permet aussi la résolution de problèmes trop contraints, pour lesquels il n'existe pas d'affectation cohérente, mais dans lesquels on veut essayer de violer le moins de contraintes possibles.

Contrairement au CSP classique, le problème de satisfaction de contraintes valué, que nous noterons VCSP (pour *Valued Constraints Satisfaction Problem*) est un probléme d'optimisation. Le cadre VCSP propose une modélisation des contraintes **molles** passant par le biais d'une structure de valuation qui sert de support a une fonction de valuation, fonction de valuation qui, pour chaque contrainte, associe un cout (ou de manière duale une récompense) a chaque tuple de valeurs.

Définition 2.3.1 (Structure de valuation) :Une structure de valuation est un triplet $(E, \oplus, \succeq)$ qui vérifie :

- E est un ensemble totalement ordonné par $\succeq$, muni d'un élément minimum noté $\bot$ et d'un élément maximum noté $\top$.
- E est muni d'une loi de composition interne commutative et associative notée $\oplus$, dite de combinaison, qui vérifie :
  - monotonie :$\forall a, b, c \in E$ tels que $a \succeq c$, on a $(a \oplus b) \succeq (c \oplus b)$.
  - élément neutre : $\forall a \in E$, $a \oplus \bot = a$;
  - élément absorbant :$\forall a \in E$,$a \oplus \top = \top$.

Définition 2.3.2 (Instance VCSP) Une instance VCSP est définie par la donnée d'une instance CSP (X, D, C, R), d'une structure de valuation $S = (E, \oplus, \succeq)$, et d'une application $\varphi : C \mapsto E$ associant une valuation a chaque contrainte du réseau. La notion classique de satisfaction / violation est remplacée par une notion graduelle de valuation d'une instanciation, obtenue en combinant les valuations des contraintes violées par l'instanciation.

Définition 2.3.3 (Valuation d'une affectation) Soient un VCSP $P = (X, D, C, R, S, \varphi)$ et A une affectation définie sur $V \subseteq X$. La valuation de A dans P se définit par : $V_A = \bigoplus_{c \in C_{Aviolec}} \varphi(c)$

Définition 2.3.4 (Problème de satisfaction de contraintes valué) Le problème de satisfaction de contraintes valué se définit par la donnée d'une instance $P = (X, D, C, R, S, \varphi)$ et d'une valuation k, par la fonction qui a chaque instanciation globale associe sa valuation, et par la question :éxiste-t-il une affectation A telle que $V_A < k$

Le problème d'optimisation associé s'intéresse a la valuation minimale de l'ensemble des valuations des affectations possibles.

Extension de l'arc-cohérence pour contraintes molles

Si l'idée d'étendre la notion de cohérence (et plus particuliérement d'arc-cohérence) a des contraintes valuées semble étre une d´emarche logique au vu de ce qui précéde, en revanche, sa mise en oeuvre est assez d´elicate, et nécessite un point de vue un peu différent de l'arc-cohérence des CSP classiques. Une extension efficace de l'arc-cohérence aux VCSP est due a Cooper et Schiex [6]. Cette extension nécessite toutefois de bonnes propriétés de la structure de valuation, comme nous allons le voir.

Particularisation des structures de valuation

La procédure de filtrage par arc-cohérence sur les CSP classiques peut etre percue comme une opération en deux temps. Lors d'une opération classiquement appelée projection, de l'information est retirée des contraintes pour etre projetée dans les domaines des variables (les valeurs sans support sont retirées des domaines). Dans un deuxiéme temps, une opération d'extension est réalisée : celle-ci a pour but d'effacer les tuples dont une valeur a été effacée, donc qui ne peuvent plus correspondre a un tuple valide. Ces opérations sont répétées successivement jusqu'a atteindre la fermeture arc-cohérente du probléme initial. La procédure est assurée de terminer du fait du nombre fini de tuples possibles dans le probléme initial, mais le point fixe (i.e. la fermeture arc-cohérente) n'est pas unique. L'extension proposée de l'arc-cohérence aux VCSP est fondée sur l'idée de transfert d'information entre contraintes et domaines, ce qui sous-entend d'avoir la possibilité de retirer de la valuation pour la transférer ailleurs. La notion de "différence" n'étant pas induite par la structure monoidale de la structure de valuation, il nous faut donc limiter l'arc-cohérence aux problèmes pour lesquels elle existe :

Définition 2.3.5 Une structure de valuation $(E, \oplus, \succeq)$ est juste si pour toute paire de valuations $u, v \in E$, $v \preceq u$, il existe une valuation $w \in E$ telle que $v \oplus w = u.w$ sera appelée une différence de u et v. L'unique différence maximum de u et v sera notée $u \ominus v$.

Dans le cas général, l'opérateur de différence peut ne pas exister, ou bien exister mais ne pas produire de différence maximum pour toutes les valuations. Cependant, toute structure de valuation possédant un opérateur $\oplus$ strictement monotone, si elle n'ést pas juste, peut

etre étendue en une structure de valuation juste . Ceci est notamment le cas pour les CSP lexicographiques.

Une définition molle de l'arc-cohérence

Afin d'étendre la définition d'arc-cohérence aux contraintes valuées, nous avons besoin de définitions préliminaires :

Définition 2.5.6 Soient $v, w \in E$, deux valuations. Si $v \neq \bot$ et $v \ominus v = v$, nous dirons que v est absorbante. Si $v \oplus w \neq w$ nous dirons que v affecte w.

La définition d'arc-cohérence molle reprend l'idée d'un transfert de valuations des contraintes aux domaines, et des domaines dans les tuples des contraintes, cependant, pour garantir la terminaison de la procédure de filtrage, seules les valuations absorbantes sont étendues dans les tuples des contraintes, et seules les valuations affectant les valeurs des domaines sont projetées. Un autre raffinement consiste a créer (ou modifier) des contraintes unaires (notées c_i) des variables pour modéliser le transfert de couts sur les domaines. Le concept d'arc-cohérence est illustré sur la figure , qui montre la projection d'une valuation issue d'une contrainte binaire sur une contrainte unaire. Nous pouvons désormais introduire l'arc-cohérence de maniére plus formelle :

Définition 2.5.6 (Arc-cohérence molle) Un VCSP juste est arc-cohérent s'il est noeud-cohérent et si pour toute contrainte $c \in C$, pour toute variable $i \in X_c$, pour toute valeur $a \in d_i$ on a :

- la valuation minimum des supports de (i, a) sur c n'affecte pas $c_i(a)$.
- si $c_i(a)$ est absorbante, alors elle n'affecte aucune des valuations des supports de (i, a)sur c.

Arc-cohérence directionnelle

L'une des principales limitations de l'arc-cohérence molle telle qu'elle a été définie précédemment est qu'elle ne peut pas étendre les éléments non absorbants, sous peine de ne pas terminer. De cette limitation peut d´ecouler un filtrage insuffisant, qui peut aboutir a un algorithme peu performant pour certains problémes. Afin d'augmenter la puissance de filtrage de l'algorithme, on peut introduire la notion d'arccohérence directionnelle :

Définition 2.5.7 (Arc-cohérence directionnelle) Un VCSP binaire est arc-cohérent directionnel selon un ordre $<$ sur les variables si $\forall c_{ij} \in C$ telle que $i < j$, $\forall a \in d_i : c_i(a) = min_{b \in d_j} : (c_i(a) \oplus c_{ij}(a, b) \oplus c_j(b))$.

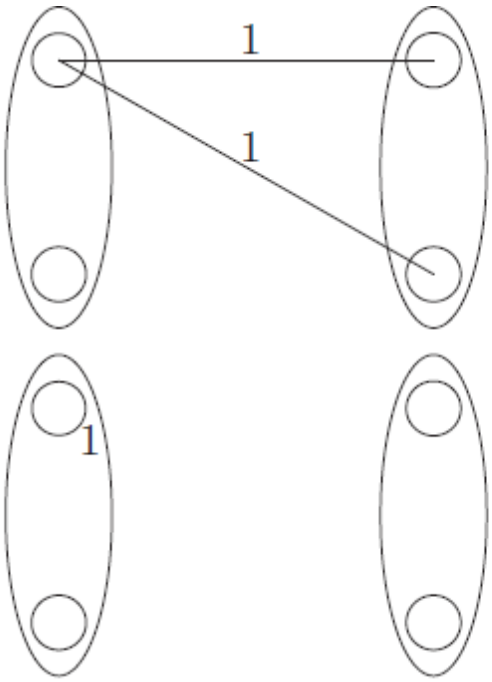

FIGURE 2.6 – Projection de couts sur unevaleur.

L'arc-cohérence directionnelle peut etre établie en $O(ed_2)$ ([]) pour peu que la structure de valuation associée au VCSP soit juste. Elle correspond a un transfert des éléments non absorbants par extension des variables situées en fin d'ordre aux variables situées en début d'ordre.

2.4 Les différentes approches de résolution des problèmes CSP

Pour les problèmes CSP, il n'existe pas de méthode universelle pour une résolution efficace. Au cours des vingt dernières années, de nombreux algorithmes et systèmes ont été mis au point pour résoudre ce type de problèmes, notamment en utilisant les techniques de consistance et de recherche avancé, ainsi que des combinaisons de ces techniques pour obtenir des algorithmes plus performants.

Classiquement, on identifie deux grandes familles au sein de ces techniques de résolution. D'une part, les méthodes complètes (ou exactes), d'autre part, les méthodes incomplètes

(ou approchées).

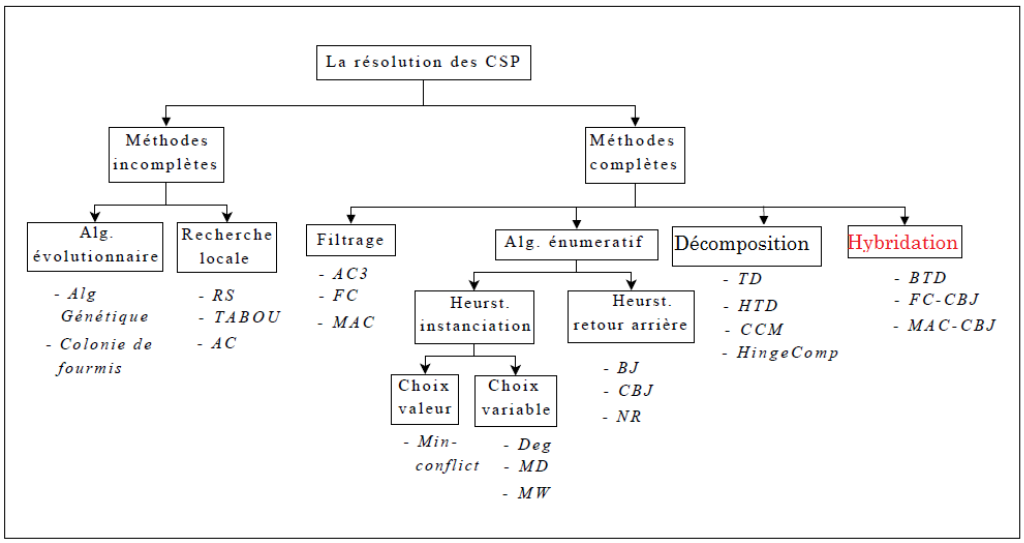

FIGURE 2.7 – les différentes approches de résolution des CSP.

2.4.1 Les méthodes incomplètes

Les méthodes incomplètes considèrent l'espace de recherche dans sa totalité mais elles ne l'explorent qu'en partie en se dotant des combinaisons d'heuristiques pour choisir les zones d'exploration, ces combinaisons sont appelées méta heuristique. Ces méthodes visent à donner un résultat acceptable en un temps raisonnable mais elles ne peuvent prouver l'inexistence de solution d'un problème sur contraint. Autrement dit, elles abordent la résolution d'un CSP comme un problème d'optimisation combinatoire pour lequel il s'agit de calculer une affectation satisfaisant le plus grand nombre de contraintes, l'objectif final étant de les satisfaire toutes.

Les différentes approches possibles d'une résolution incomplètes appartiennent a deux grandes familles de métaheuristques, celle de la recherche locale avec ses algorithmes : recuit simulé (RS), la recherche Tabou, colonies de fourmis (AC,Ant Columns), et celle dites évolutionniste avec les algorithmes génétiques.

2.4.2 Les méthodes complètes

Les méthodes complètes explorent totalement l'espace de recherche et elles sont toujours capables de répondre par vrai ou faux concernant l'existence d'une solution. En effet, elles peuvent prouver la satisfiabilité d'un problème, tout comme déterminer l'ensemble des solutions de ce problème.
Les nombreux travaux qui ont été réalisés dans cet axe et qui tentent d'améliorer ces méthodes de résolution peuvent être classés en trois grandes classes :

1. **Les algorithmes de recherche énumérative :**
   Ces algorithmes consistent à visiter toutes les configurations possibles de l'espace de recherche (l'ensemble des affectations possibles des variables). L'algorithme de type Backtracking constitue la méthode de base. Cette algorithme, beaucoup trop coûteux, a conduit à la recherche d'algorithmes intélligents les plus efficaces.
2. **Les algorithmes de consistance (filtrage par consistance) :**
   Ces algorithmes visent à réduire l'espace de recherche à explorer pour simplifier les instances avant ou pendant la recherche d'une solution. Ils sont utilisés comme des algorithmes de prétraitement pour améliorer le travail des algorithmes de recherche en utilisant des techniques de consistance d'arc, de chemin,...etc.
3. **Résolution par décomposition des CSP :**
   Ces algorithmes exploitent le fait que la traitabilité d'un CSP est reliée à ses propriétés structurelles. Etant donné une instance CSP, ces algorithmes transforment cette instance en une instance acyclique constituée d'un ensemble de sous problèmes dont la complexité est inférieure à celle de l'instance originale.
4. **Résolution par des algorithmes hybrides**
   La combinaison des trois approches de résolution (énumérative, par filtrage et par décomposition) induit des algorithmes de résolution dites hybrides. Plusieurs algorithmes hybrides ont été proposés :BTD pour Backtracking et Tree Decomposition, un algorithme qui fait l'hybridation entre un algorithme énumératif (BT) et la résolution par décomposition (Tree Decomposition), proposé par Cyril Terrioux. L'hybridation entre les algorithmes énumératif et ceux de filtrage se voit dans le FC et le MAC. FC-MRV est un algorithme qui hybride l'algorithme FC et l(heuristique MRV. Il est avéré meilleur que MAC en pratique.

2.5 conclusion :

Nous avons fait un état de l'art sur les méthodes de décomposition de graphes et hypergraphes, les cadres CSP et VCSP.

Beaucoup de méthodes de résolution de CSP ont été développées. Les méthodes énumératives sont relativement efficaces en pratique, mais leurs bornes de complexité théorique en temps, exponentielles en n (le nombre de variables du CSP), sont trés médiocres.

Les méthodes structurelles proposent de meilleures garanties théoriques. Elles utilisent pour la plupart un recouvrement acyclique du probléme. La résolution des différentes parties ainsi construites, donne un CSP acyclique qui peut etre résolu en un temps polynomial. Les excellentes bornes de complexités de ces méthodes sont obtenues généralement au détriment de l'effécacité en pratique. L'espace mémoire requis est souvent inaccessible et rend inopérant la majeure partie de ces techniques. Cependant, certaines d'entre elles, comme BTD, associent ces garanties théoriques avec la souplesse de l'énumération. Elles arrivent a résoudre des problémes inaccessibles aux meilleures méthodes énumératives telles FC et MAC, moyennant une bonne décomposition du probléme en terme de taille (réduite) des différentes parties définies.

Dans le prochaine chapitre nous allons détaillé ces méthodes de décomposition structurelles

3

Les méthodes de décomposition structurelles

Les méthodes de décomposition structurelle fonctionnent selon le même principe que ce soit pour des CSP binaires ou n-aires. Notons que dans le cas ou il existe des contraintes d'arité supérieure strictement à deux, il existe des méthodes spécifiques fondées sur la décomposition de l'hypergraphe de contraintes, et qui sont plus efficaces que les méthodes fondées sur le graphe primal.

3.1 principe

L'objectif d'une méthode de décomposition est de transformer une instance CSP cyclique en une autre instance équivalente (ayant les mêmes solutions) acyclique, qui peut être résolue d'une manière plus efficace. sachant que la classe des CSP dont le graphe de contraintes est un arbre traitable. Informellement, celle-ci est assurée par la décomposition du réseau de contraintes du problème CSP donné en un ensemble de sous problèmes, en formant des clusters de variables ou de contraintes, dont l'interaction a une structure d'arbre. Si la taille de chacun des sous problèmes est plus petite que celle du CSP original, la résolution des sous problèmes (clusters) peut être réalisé d'une manière plus efficace que la

résolution du problème original. Une solution du CSP original peut être dérivée de l'arbre de sous problèmes, d'une manière aussi plus efficace.

Chaque méthode de décomposition définit un concept de largeur (width) qui est le critère de la mesure de cyclicité du graphe ou de l'hypergraphe de contraintes, telle que pour chaque constante k fixée, une méthode de décomposition peut décider, en un temps polynomial, si un hypergraphe donné peut avoir une décomposition de largeur inférieur ou égale à k. Après la décomposition, la complexité de la résolution du CSP est en fonction de la largeur de la décomposition. L'objectif principal de toutes les méthodes de décomposition est donc de minimiser cette largeur.

3.2 Méthode Biconnected components

La méthode Biconnected components repose sur le concept de la composante biconnexe définie ainsi :

Definition 3.2.1 (composante biconnexe) : Un graphe est biconnexe s'il est connexe et n'est pas la réunion de deux sous-graphes connexes qui n'ont en commun qu'un sommet et ont chacun au moins une aréte.

Une composante biconnexe est un sous-graphe biconnexe maximal. Une boucle et un isthme sont des composantes biconnexes. Les composantes biconnexes d'un graphe G définissent une partition de son ensemble d'arétes. Si G est connexe, on peut définir un arbre dont les noeuds correspondent a ses composantes biconnexes et dont toute aréte $A - B$ correspond a un sommet commun aux composantes A et B. On dira que cet arbre est un arbre des composantes biconnexes de G. Cet arbre n'est pas d´efini de manière unique. Par exemple, si G est la réunion de 3 composantes biconnexes A,B,C qui ont un sommet en commun, on peut prendre comme arbre : $A-B-C$ou $B-A-C$ ou $A-C-B$.

La méthode Biconnected components est une méthode développée pour les CSP binaires. Ainsi, la décomposition Biconnected d'un CSP n-aires est la décomposition Biconnected de son graphe primal.

3.3 Méthode Cycle Cutsest

Le principe de cette méthode est de générer un arbre ; et le résultat un ensemble de variables (du cutset) et un arbre (formé des variables qui ne sont pas dans le cuset) en se

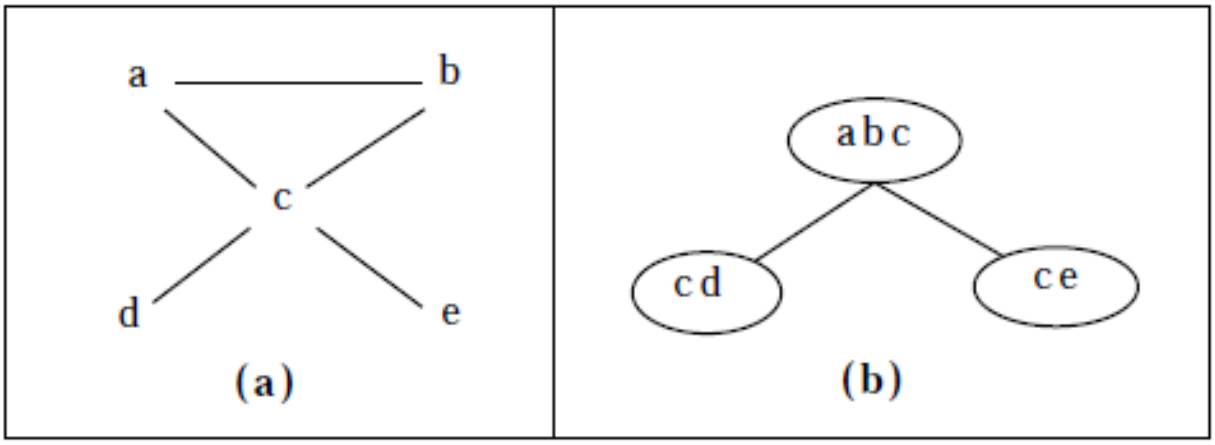

FIGURE 3.1 – Un graphe (a) et sa Biconnected décomposition (b)

basant sur la définition suivante :

Définition 3.3.1 : (Cycle Cutset) Etant donné un graphe de contraintes, l'élimination de certaines variables après les avoir instanciées change la connectivite du graphe. Cette ensemble de variables est appelé coupe cycle ou CycleCutset.

Le principe de la méthode cyclecutset repose sur cette notion et peut être résumé ainsi et initialement, on définit un ensemble cutset pour le réseau de contraintes. On instancie les variables de cette ensemble par une affectation consistante en utilisant un algorithme quelconque. Le résultat après élimintion des variables instanciées, est un réseau de contrainte acyclique qui sera résolu par propagation de consistance en utilisant un algorithme polynomial selon le théorème de Freuder[3]. Si cette instanciation de l'ensemble cutset n'aboutit pas à une solution on fait un backtrack sur les variables de cet ensemble .

La largeur d'une décomposition CycleCutset est égale à la taille de l'ensemble couple cycle, et la largeur (width) d'une instance CSP est égale à la largeur minimale des décompositions CycleCutset de cette instance. Cette méthodes a été développée pour des CSP binaires. Pour les CSP n-aires le même principe s'applique au graphe primal du CSP.

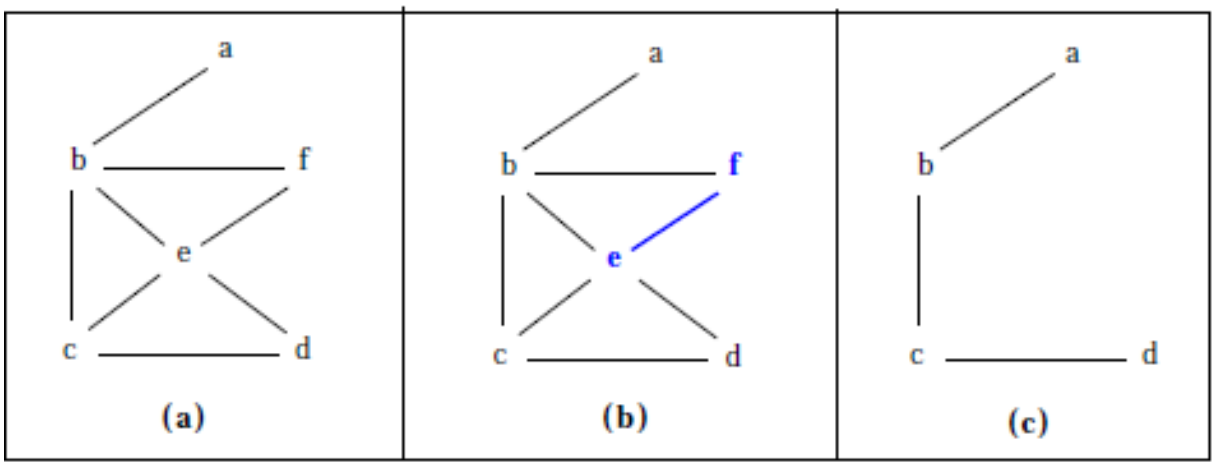

FIGURE 3.2 – Un graphe et sa décomposition Cycle Cutset

3.4 Méthode Hypercutset

La méthode Hypercutset est une variante de la méthode Cycle Cutset qui s'applique aux CSP n-aires en utilisant la définition de Cutset pour un hypergraphe.

Définition 3.3.1 : (Cycle HyperCutset) Un ensemble Hypercutset d'un hypergraphe est un ensemble d'hyperarêtes (au lieu de sommets) dont l'extraction des variables impliquées par ces hyperarêtes induit un graphe acyclique.

La difficulté de cet méthode réside dans la recherche du plus petit ensemble hypercutset qui est, en général, NP-Complet. Cependant, la recherche du plus petit ensemble hypercutset dont la taille est inférieur à k, s'il existe, est traitable.

La largeur Hyper Cutset width d'une instance CSP est égale à la cardinalité minimale de l'ensemble hypercutset des décompositions Hyper Cutset de cette instance.

3.5 Méthode Hinge decomposition

La méthode de Hinge decomposition propose une autre caractérisation de l'acyclicité en se basant sur le concept des Hinges. Elle definit des Hypergraphes acycliques en exploitant, directement, des hypergraphes sans passer par le graphe primal.

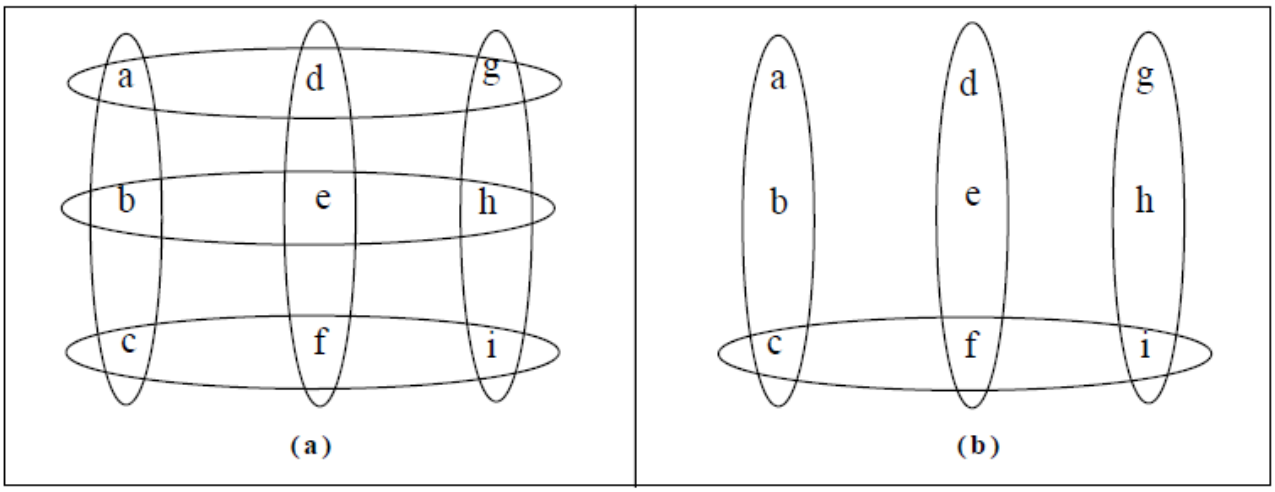

FIGURE 3.3 – Un graphe et sa décomposition Cycle HyperCutset

Définition 3.4.1(Composante connectée) Soit G = (V, E) un hypergraphe, et soit$H \subseteq E$ et $F \subseteq (E - H)$. F est dit connecté par rapport à H si pour chaque paire d'arêtes $\{e, f\}$ de F, il existe une séquence d'arêtes $e_1, ..., e_k$ dans F, tel que $e = e_1$ et $f = e_n$ et pour$i = 1$ à $n - 1 : [(e_i \bigcap e_{i+1}) - (\bigcup H)] \neq \{\}$

Définition 3.4.2 (Hinge) Soit G = (V, E) un hypergraphe réduit et connexe et soit $H \subseteq E$ contenant au moins deux arêtes. Soient $H_1, ..., H_m$ les composantes connexes de $(E - H)$ par rapport à H. H est dit un Hinge si, pour $i = 1$ à m, il existe une arête h_i telle que : $(\bigcup H_i) \bigcap (\bigcup H) \subseteq h_i$.

Un hinge est dit minimal, s'il ne contient aucun autre hinge.

Une Hinge decomposition d'un hypergraphe $G = (V, E)$ est un arbre vérifiant les quatres conditions suivantes :

1. Les noeuds de l'arbre sont les hinges minimaux de G.
2. Chaque arête dans E est contenue dans au moins un noeud de l'arbre.
3. Deux noeuds adjacents$A$ et B de l'arbre partagent précisément une arête L dans De plus, L consiste en l'ensemble des variables partagées par A et B.
4. Les sommets de V partagés par deux noeuds sont entièrement contenus dans tout noeud sur le chemin les liant.

3.6 Méthode de décomposition arborescente (Tree decomoposition)

Définition 3.5.1 (Arbre (Tree)) [4] Soit G = (V, E) un hypergraphe. Un arbre (Tree) de l'hypergraph G est une paire $\langle T, \chi \rangle$, ou $T = (V(T), E(T))$ est un arbre enraciné, et χ est une fonction qui associe à chaque noeud $p \in V(T)$ l'ensemble de variables $\chi(p) \subseteq V$.

Définition 3.5.2 (Décomposition arborescente [5]) : Soit$G = (V, E)$ un graphe. Une décomposition arborescente est un couple (T, C), tel que $T = (I, F)$ est un arbre, et$C = \{Ci | i \in I\}$ une famille de sous-ensembles V , chaque C_i correspondant a un noeud de T, qui vérifie :

- $\bigcup_{i \in I} Ci = V$,
- quelque soit $\{u, v\}$ tel que $\{u, v\} \in E$,$\exists i \in I$ avec $v \in C_i$ et $w \in C_i$,et
- quelque soit$(i, j, k) \in I^3$ si j est sur le chemin de i a k dans T, alors $C_i \cap C_k \subseteq C_j$.

On appelle largeur d'une décomposition arborescente le paramétre $max_{i \in I}(|Ci| - 1)$. La largeur d'arbre (ou treewidth) d'un graphe est la largeur minimale sur toutes ses décompositions arborescentes.

Les éléments C_i sont géneralement appelés **regroupements** ou **clusters**. Notons qu'une décomposition arborescente triviale d'un graphe quelconque est un arbre d'un seul sommet dont le cluster associé contient tous les noeuds du graphe initial.

3.7 Méthode Tree clustering

Cette méthode est proposée pour des problème CSP naires. Elle consiste a former des clusters de variables du problème original pour avoir des sous problèmes structurés en arbre. Elle ne manipule pas l'hypergraphe de l'instance à traiter mais, plutot, son graphe primal en se basant sut la concept suivant :

Définition 3.6.1 (Graphe triangulé) : Un CSP est acyclique si et seulement si son graphe primal est Chordale (triangulé) et conforme.

Définition 3.6.2(Graphe conforme) : Un graphe est dit conforme (triangulé) si tout cycle de longueur supérieure à trois admet une corde. i.e une arête joignant deux sommets non consécutifs le long du cycle.

Définition 3.6.3 (Graphe chordal) : Un graphe primal est dit conforme si chacune de ses cliques maximales correspond à une contrainte dans le CSP d'origine.

Cette méthode est basée sur un algorithme efficace de triangulation, qui transforme n'importe quel graphe en un graphe triangulé en lui ajoutant des arêtes. Les cliques maximales du graphe triangulé obtenues correspondent aux clusters nécessaires pour former un CSP acyclique. Cet algorithme de triangulation consiste en deux étapes :

1. Calculer un ordre en utilisant l'algorithme maximum cardinality search (MCS) qui numérote les sommets de 1 à n dans l'ordre croissant, en assignant, toujours, le prochain numéro au sommet ayant le plus de voisins numérotés.
2. Relier les noeuds non adjacents connectés par des noeuds dont le numéro d'ordre est inférieur à ceux des deux sommets, et ceci d'une manière récursive.

   Si aucune arête n'est ajoutée à l'étape 2, le graphe est triangulé.

3.8 Méthode hypertree decomposition

Définition 3.7.1 (Hyperarbre (Hypertree)) : Soit $H = (V, E)$ un hypergraphe. Un Hyperarbre (Hypertree) de l'hypergraph H est un triplet$\langle T, \chi, \lambda\rangle$ ou $T = (V(T), E(T))$ est un arbre anraciné et χ et λ sont deux fonctions qui associent à chaque noeud $p \in V(T)$ deux ensembles $\chi(p) \subseteq Vet\lambda(p) \subseteq E$.

Définition 3.7.2 (Generalised Hypertree decomposition) : Une décomposition hypertree généralisée d'un hypergraphe $H = \langle V, E\rangle$ est un hyperarbre (hypertree) GHD $=\langle T, \chi, \lambda\rangle$ de H qui satifait les conditions suivantes :

1. quelque soit $t \in T : \chi(t) \subseteq (\cup\lambda(t))$,
2. quelque soit $h \in E$,il existe $t \in T$ tel que $h \subseteq \chi(t)$ et
3. quelque soit $x \in V$,l'ensemble $\{t \in T/x \in \chi(t)\}$induit un sous arbre connecté de T.

La première et la deuxième condition de la décomposition hypertree généralisée sont identiques à celles de tree decomposition, donc une décomposition hypertree généralisée d'un hypergraphe H est, en même temps, une tree decomposition de H. La troisième condition assure que pour chaque noeud de la décomposition hypertree généralisée, chacune des variables de l'ensemble χ doit être contenue par au moins une hyperearête de l'ensemble λ de ce noeud.

Définition 3.7.3 (Hypertree decomposition) : Une décomposition hypertree d'un hypergraphe $H = (V, E)$ est une décomposition hypertree généralisée $HD = \langle T, \chi, \lambda \rangle$ de H qui vérifie la condition suivante :

quelque soit $t \in T : \chi(T_t) \cap (\cup\lambda(t)) \subseteq \chi(t)$ tel que T_t est le sous arbre de T enraciné en t.

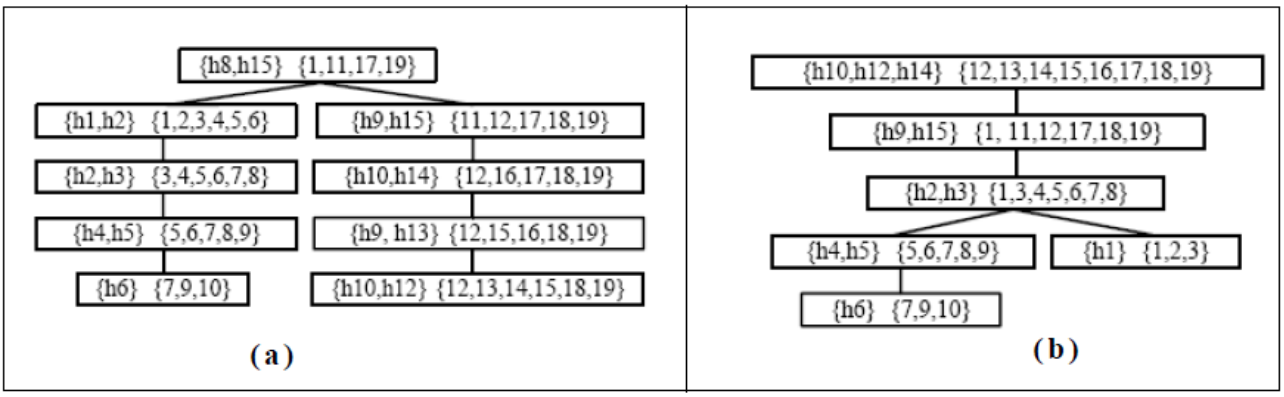

FIGURE 3.4 – (a) hypertree décomposition généralisée (b) hypertree décomposition

3.9 hiérarchie des méthodes structurelles[24]

Dans cette section nous présentons la hiérarchie des méthodes structurelles de décomposition en tenant compte des récents développements dans le domaine

3.9.1 Spread Cut [8] SCD

Un **guarded block** d'un hypergraphe $H = (X, C)$ est une paire (χ, λ) où λ est un ensemble d'hyper-arêtes et χ est un sous ensemble de sommets de l'union des variables de λ.

Un guarded block (χ, λ) d'un hypergraphe H couvre une hyper-arête e de H si $e \subseteq \lambda$

Un ensemble de guarded blocks $\sum$ est appelé guarded cover de H si chaque hyper-arête de H est couverte par un guarded block de $\sum$

Un guarded block (χ, λ) d'un hypergraphe H a des composantes non cassées `unbroken components` si chaque - composante de H a une intersection non vide avec au plus une χ-composante de H et $\{e_1 \cap e_2 / e_1, e_2 \in \lambda\} \subseteq \chi$.

Pour un certain $\alpha \subseteq X$ et une variable $v \in C$, le label $L_\alpha(v) = \{V/Vestune\alpha$ - composante qui contient une hyper-arête contenant la variable v .On dit qu'un guarded block (χ, λ) est **canonique** si pour chaque hyper-arête e de les variables de$e$ qui n'appartiennent pas à χ sont celles qui ont un label particulier. C'est-à-dire $quelquesoitv \in e - \chi, quelquesoitw \in e, w \notin \chi \leftrightarrow L_\lambda(v) = L_\lambda(w)$.

Une spread cut decomposition est un ensemble acyclic guarded cover dans lequel tous les guarded blocks ont des composantes non cassées et ils sont tous canoniques.

3.9.2 Spread Cut New [7] SCD New

Soit λ un ensemble d'hyper-arêtes d'un hypergraphe $H = (X, C)$.

On définit le label d'une variable $v \in (\cup\lambda)$ comme une paire tel que la première composante de la paire est l'ensemble des $\cup\lambda$-composantes ayant une intersection non vide avec une hyper-arête de E contenant v. La deuxième composante des hyper-arêtes de λ contenant la variable v.

On dit qu'un guarded block respecte les labels si quelque soit $v, w \in \cup\lambda, (v \in \chi et L_\lambda(v) = L_\lambda(w)) \Rightarrow w \in \chi$.

Une spread cut decomposition est une hypertree decomposition généralisée où chaque nœud est un guarded block qui n'a pas de composantes cassées `unbroken components` et il respecte les labels.

3.9.3 Component hypertree decomposition [6] CHD

Cette méthode est une `subedge defined method` (basée sur la notion de sous hyper-arête d'une hyper-arête). On note par vertices (edges(C)) l'ensemble des variables des hyper-arêtes ayant une intersection non vide avec la composante C.

Soit M un ensemble d'hyper-arêtes d'un hypergraphe $H = (X, C)$. On définit :
- **prop**(e, M) $= e \cup e' eM, e \neq e' e'$. C'est la part de l'hyper-arête e à M toute seule.
- **internal** (e, M) c'est l'ensemble des variables v de e qui n'appartiennent à aucune var (M)-composante ayant une hyperarete contenant la variable v dans X.

– **La fonction elim**(M, C, e) associe 3 sous hyper-arêtes au triplet (M, C, e) :

1. $e \cap$ vertices $(edges(C))$
2. prop$(e, M) \cap$ vertices $(edges(C))$
3. internal (e, M)

On définit maintenant la fonction subedge comme suit :

$f^c(H, k) = \{e \backslash e'$ M est un ensemble d'hyper-arêtes de l'hypergraphe H,$e \in M$, D est une [vertices (M)-composante] et e'$\in$elim $(M, D, e)\}$

Pour calculer une component hypertree decomposition, on calcule l'hypertree decomposition de l'hypergraphe $H' = H \cup f^c(H, k)$. la largeur de H' sera nécessairement inférieur à l'hypertree width de H [6]. De cette façon, on trouve une largeur inférieur à la largeur optimale hypertree. Mais qui n'est toujours pas une largeur hypertree généralisée optimale.

$CHD \leq HD$ et $CHD \leq SCD$[6]

Avec CHD : component hypertree decomposition, HD : hypertree decomposition et SCD : spread cut decomposition.

De plus décider si pour une constante k un hypergraphe H a une component hypertree decomposition de largeur au plus égale à k est faisable en un temps polynomial.

3.9.4 Extend Component hypertree decomposition [6] ECHD

Cette méthode est une généralisation de la CHD

Soit M un ensemble d'hyper-arêtes d'un hypergraphe H. Soit N un sous ensemble d'hyper-arêtes de M, $N \subseteq M$, on définit :

– **Slice(N,M)**= $\{v/quelquesoite \in N, v \in vertices(e)$et quelque soit$e' \in (MN), v$ n'appartienne pas a$vertices(e')\}$.

Soit M un ensemble d'hyper-arêtes d'un hypergraphe H. Soit N un sous ensemble d'hyper-arêtes de M,$N \subseteq M$, on définit :

– **Internal (N,M)** =$\{v \in slice(N, M)$ et il n'existe pas de $[vertices(M)] - composanteC$ tel que $v \in vertices(edges(C))\}$

Soit M un ensemble d'hyperarêtes, et soit C une vertices (M) -composante. Soient $N_1, \ldots, N_r / \ 1 \leq r \leq 2^{[M]-1}$.

– La fonction **elim**$(M, C, N_1, \ldots, N_r)$ associe un ensemble contenant les sous hyper-arêtes suivantes au tuple $(M, C, N_1, \ldots, N_r)$.

1. $\cup_1 \leq i \leq r(N_i, M) \cap vertices(edges(C))$
2. $\cup_1 \leq i \leq r$ internal(N_i, M)

– **La fonction subedge** f^E est définie comme suit : $F^E(H,k)$=$\{e \backslash e'$/M est un ensemble de $\leq k$ hyper-arêtes de l'hypergraphe H, $e \in M$,C est une vertices (M) - composante ,$1 \leq r \leq 2^{k-1}$, $N_1, \ldots N_r$ sont des ensembles d' hyper-arêtes,Pour chaque $i(1 \leq i \leq r), Ni \subseteq M, ete' \in elim(M, C, N1, \ldots Nr)\}$

$ECHD \leq CHD$ et$ECHD \leq SCD_{New}$. [6]

Dans la version paramétrée, la fonction elim définie précédemment est redéfinie comme suit : Soit H un hypergraphe, soit M un ensemble d'hyper-arêtes de H. Soient $N_1, \ldots, N_r/1 \leq r \leq 2^{M-1}$des sous ensembles de M. Aussi considérons une constante $d(1 \leq d \leq |\ E(H)\ |)$et soient$C_1, \ldots, C_d$ des $[vertices(M)]$-composantes. La fonction $elim(M, C_1, \ldots, C_d, N_1, \ldots, N_r)$ associe un ensemble de sous hyper-arêtes au tuple $(M, C_1, \ldots, C_d, N_1, \ldots, N_r)$:

1. $\cup_1 \leq i \leq r(N_i, M) \cap \cup_1 \leq j \leq d$ vertices$(edges(C_j))$
2. $\cup_1 \leq i \leq r$ internal(N_i, M).

La fonction $subedgef^E$ est redéfinie comme suit :

$F^E(H,k)$=$\{e \backslash (f_0 \cup f_1)$/M est un ensemble de $\leq k$ hyper-arêtes de l'hypergraphe H, $e \in M, C_1, ...C_d$ des $[vertices(M)]$ -composantes ,$1 \leq r \leq 2^{k-1}$, $N_1, \ldots N_r$ sont des ensembles d' hyper-arêtes,Pour chaque $i(1 \leq i \leq r), Ni \subseteq M$, ,$f_0 \in elim(M, C, N_1, \ldots, N_r)$ et $f_1 \in elim(M, C, N_1, \ldots, N_r)\}$

textbfremarque : $ECHD \equiv ECHD(1)$

si $d_1 \leq d_2$ alors$ECHD(d_2) \leq ECHD(d_1)$ [1]

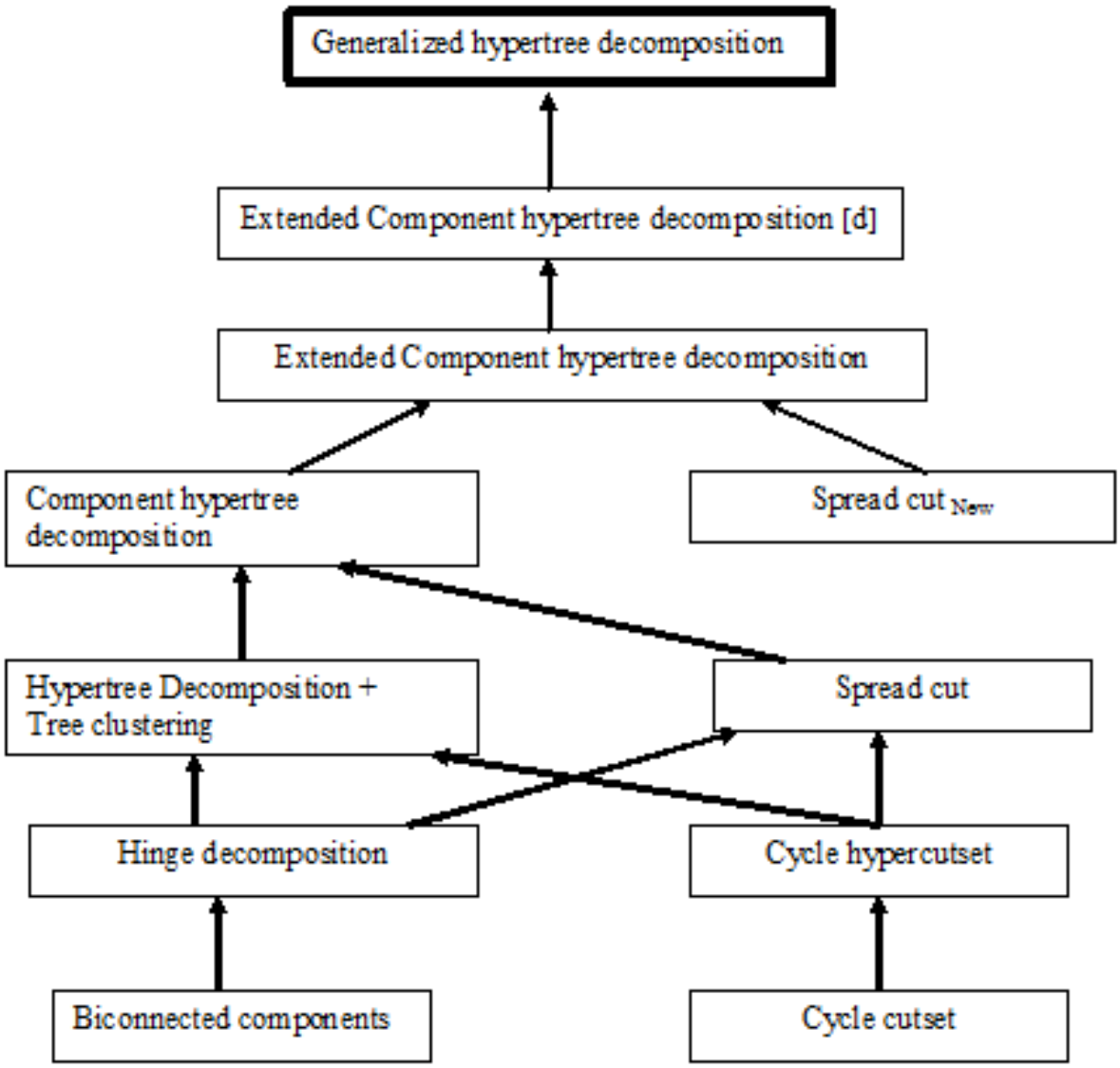

FIGURE 3.5 – Hiérarchie des méthodes de décomposition structurelles

3.10 conclusion :

Dans ce chapitre, nous avons étudié les méthodes de décompositions structurelles qui constituent l'une de très importantes procédures de prétraitement exploitées avant ou pendant la résolution des problèmes CSP. Le gain espéré par la complexité théorique de ces méthodes est indéniable dans de nombreux cas, cela a ouvert un nouveau axe de recherche consistant à :

1. Trouver les meilleures décompositions.
2. Combiner ces méthodes pour obtenir des algorithmes plus éfficaces.
3. Développer des heuristiques pour le calcul des décompositions.
4. Optimiser les mises en oeuvre de ces méthodes et proposer les algorithmes qui les calculent.

dans le chapitre suivant nous allons voir le Comptage de solutions en exploitant la structure du graphe de contraintes.

4

résolution des problèmes de satisfaction de contraintes

Dans ce chapitre, nous proposons une approche pour la résolution des problèmes CSP après une décomposition structurelle. Cette approche est une amélioration de l'approche HD-BT [25] dont l'avantage est une recherche moins couteuse des cluster , ce qui nous permet une performance en terme de temps.

Tout d'abord, nous présentons les différents algorithmes séquentiels de résolution des CSP acycliques binaires et n-aire proposés dans ce cadre, puis nous présentons de façons détaillée notre approche.

4.1 Les différentes classes de méthodes de résolution

4.1.1 Les algorithmes d'approximation avec garanties

Ces algorithmes garantissent une approximation de l'optimum à un facteur prés : la largeur de la triangulation calculée est inférieure à ce facteur fois l'optimum. Pour une

approximation a un facteur constant prés, il n'existe a ce jour que des algorithmes de complexité exponentielle.

Une approximation en un temps polynomial à une constante prés reste un problème ouvert. Robertson et Seymour ont défini une méthode d'approximation basée sur la notion de branchwidth [20] Robiertson a prouvé que la branchwidth donne une approximation supérieure d'au plus 4 fois a la treewidth, avec une complexité en $O(w^2 3^{3w} n^2)$.il a proposé aussi , une approximation d'un facteur 5 constant, avec une complexité en $O(w^2 3^{4w} nlogn)$.

La technique de Becker et Geiger utilise la programmation linéaire pour approcher l'optimum à un facteur constant de 3,66. Elle a une complexité en $O(24; 66wn : poly(n)), poly(n)$ étant la complexité de l'algorithme de programmation linéaire.

Amir a donné deux algorithmes qui améliorent les complexités en temps des algorithmes de Robertson et Seymour en $O(2^{4,38w} n^2.w)$ et de Becker et Geiger en $O(2^{3,6982w} n^3.w^3.log^4 n)$. Il définit pour cela la notion de α-separateur. L'idée est de choisir un séparateur qui induit des composantes connexes contenant au plus un certain pourcentage du nombre de sommets du graphe de départ. Ce pourcentage est défini par la valeur de α. Ensuite, cette opération est réitérée sur les différentes composantes connexes. On s'assure ainsi que les séparateurs que nous choisissons décomposent le graphe de maniére équilibrée.

4.1.2 Les algorithmes heuristiques

Ces approches construisent en général un ordre (dynamiquement) et ajoutent des arêtes dans le graphe, de sorte qu'en fin de traitement, l'ordre obtenu soit un ordre d'élimination parfait pour le graphe résultant G'. La complexité de ces méthodes est en général polynomiale (souvent méme linéaire) mais elles n'offrent, en contrepartie, aucune garantie d'optimalité. cette approche est justifiée en pratique. En effét, Kjaerulff a observé entre autres, qu'au contraire des résultats attendus, ces heuristiques produisent des triangulations raisonnablement proches de l'optimum. De plus, elles sont en général très faciles a implémenter.

4.1.3 Les algorithmes minimaux

A l'image des algorithmes heuristiques, les algorithmes minimaux calculent des triangulations à un cout assez faible. Mais contrairement aux heuristiques, ils nous garantissent que le retrait d'une arête rajoutée pour la triangulation d'un graphe donne un nouveau

graphe qui n'est pas triangulé. En effet, une triangulation minimale ne contient pas d'arête redondante.

4.1.4 Les algorithmes exacts

Les algorithmes exacts calculent une triangulation optimale. Ce problème étant NP-difficile, il n'existe pas d'algorithme connu pour le résoudre en un temps polynomial. Les méthodes proposées ont donc une complexité exponentielle. En outre, elles ont souvent un intérêt pratique limité car pour la plupart, elles ne permettent pas de calculer une triangulation dans un temps raisonnable.

Le *QuickTree* est le premier algorithme qui a pu calculer une triangulation optimale dans un temps raisonnable. Il est basé sur la notion de séparateurs minimaux. Il se dote d'une valeur k qui est un minorant de la treewidth du graphe et calcule l'ensemble des séparateurs minimaux $S(k)$ dont la taille est au plus k, dit ensemble k-régulier. A chaque séparateur est associé un ensemble de fragments qui sont les unions entre les composantes connexes induites par le séparateur et le séparateur complété (transformé en clique). Si pour un séparateur de $S(k)$, les fragments associés sont minimalement k-triangulés (cliques maximales de taille au plus k) alors la composition de ces derniers donne une triangulation minimale du graphe de départ de largeur au plus k. Sinon, les fragments sont ordonnés de maniére croissante suivant leur taille. Pour chaque fragment non triangulé, QuickTree calcule si possible, un ensemble k-régulier de séparateurs minimaux de cet ensemble tel que chaque fragment est soit une clique, soit minimalement k-triangulé. La composition de ces différents fragments triangulés donne une triangulation minimale de treewidth au plus k et de ce fait une triangulation optimale. Si un ensemble vérifiant ces conditions n'existe pas, cela veut dire que la treewidth de G est supérieure a k.

4.2 Résolution des CSPs par la méthode hypertree généralisée

Cette approche nécessite d'abord le calcul d'une décomposition par un algorithme exact ou une heuristique. (En pratique, les algorithmes exacts ne sont pas efficaces). Une fois que la décomposition est obtenue, on la complète de telle sorte que toutes les contraintes figurent dans au moins un nœud de l'hypertree. Ensuite l'approche proposée par Gottlob et al [26] pour résoudre le CSP obtenu est donnée par l'algorithme suivant

Input: Une hypertree decomposition
$\mathcal{HD} =< T, \chi, \lambda >$ associée à un CSP donné.
Output: Une solution $\mathcal{A}$
begin
  $\sigma = \{n_1, n_2, \ldots, n_m\}$ un ordre sur les noeuds de l'hypertree decomposition où n_1 est la racine et chaque noeud précède ses fils dans l'ordre. ;
  foreach *p nœud de l'hypertree* **do**
    $R_p = join(\lambda(p)[\chi(p)]$;
  end
  for $i = m$ *to* 2 **do**
    begin
      Soit v_j le père de v_i dans l'ordre ;
      $R_j = semijoin(R_j, R_i)$;
    end
  end
  for $i = 2$ *to* m **do**
    Construire une solution $\mathcal{A}$ en sélectionnant un tuple dans R_i compatible avec tous ceux qui le précèdent.
  end
  return $\mathcal{A}$;
end

FIGURE 4.1 – algorithme :Méthode de résolution proposée par Gottlob

En pratique, cette méthode est trés coùteuse aussi bien en temps qu'en espace mémoire. En effet, cette approche est basée sur deux principales opérations qui sont des jointures au niveau des noeuds de l'hypertree et des semi-jointures entre les différents noeuds de l'hypertree. Malgré les différentes heuristiques introduites dans le rapport de recherche [26], cette approche souffre toujours du probléme de l'espace mémoire.

Pour remédier à ce problème d'explosion mémoire et exploiter les atouts des décompositions arborescentes, P. Jégou et C. Terrioux ont proposé dans [21] une technique intéressante nommée BTD (Résolution d'un CSP par une méthode de type BT guidée par un ordre statique induit par une tree decomposition). BTD hérite à la fois des avantages des méthodes énumératives pour ce qui concerne l'occupation mémoire et des propriétés structurelles du CSP.

4.2.1 BTD et ces variantes

Dorénavant, il est possible d'exploiter une décomposition hyperarborescente avec des méthodes de type TC ou BTD. Dans [21], une nouvelle extension de TC (notée TC-2009) plus appropriée aux CSPs avec des contraintes n-aires a été définis . Etant donnés un CSP et une décomposition arborescente $TD = (E, T)$, le sous problème associée au cluster E_i est défini, a l'image de TC-1989, par le même ensemble de variables E_i. Mais maintenant, l'ensemble des contraintes d'un cluster E_i est $C_{E_i} = c_j \in C : c_j \cap E_i \neq \emptyset$. Les relations associées a ces contraintes sont $R_{E_i} = r_j[cj\ Ei] : c_j \in C_{E_i}$. TC-2009 comporte également 3 étapes.

La première calcule une décomposition arborescente du réseau de contraintes, en utilisant un algorithme de décomposition de (hyper)graphes alors que les deux étapes suivantes restent identiques. Ainsi, cette première étape est paramétrée par une décomposition graphique quelconque DEC.il considère une décomposition hyperarborescente optimale, il définisse TC-2009HD, qui exécute TC sur une décomposition arborescente TD(HD) induite par HD, considérant comme sous-problèmes, les clusters de variables et les contraintes dont l'ensemble des variables intersecté les clusters.

De même, [21] défini une large collection de méthodes a l'instar de $TC - 2009TD$ (TC basée sur une TD optimale), $TC - 2009MCS(TD)$ (TC basée sur une TD calculée par MCS), BTD_{HD} (BTD basée sur une HD optimale), BTD_{TD} (BTD basée sur une TD optimale), $BTD_{HMIN(HD)}$ (BTD basée sur une HD calculée grâce a une heuristique), $BTD_{HMIN(TD)}$(BTD basée sur une TD calculée grâce a une heuristique), $BTD_{HMIN(TD)+HMAX(HD)}$

(BTD basée sur une TD optimale dont le nombre de contraintes dans les clusters a été maximisé grâce a une heuristique), etc.

On supposons que la largeur de décomposition hyperarborescente soit h dans le cadre de l'analyse de la complexité de $TC-2009HD$. Chaque sous-problème (cluster) est résolu indépendamment en utilisant un algorithme de type $nFC2$. Grace aux résultats présentés dans [22], le cout de la résolution d'un cluster E_i est maintenant en $O(S_i, r^{k_i})$, ou S_i est la taille du sous-problème associé a E_i, et $k_i = k_{(E_i, C_{E_i})}$ (i.e. le paramètre associé au recouvrement minimum de E_i). Il faut noter que la taille de l'ensemble de solutions dans E_i est bornée par $O(r^{k_i})$. De ce fait, le cout total pour la résolution du CSP décomposé dans sa totalité est $O(S.r^k)$ ou $k = max k_i : i \in I$. En plus, nous avons $k \leqslant h$. La complexité en temps de $TC-2009HD$ et de BTD_{HD} est en $O(S.r^h)$.

Ce résultat donne une présentation plus précise de la Hiérarchie des Contraintes Traitables puisque $TC-2009HD$ et BT_{DHD} sont au même niveau (sommet) dans la hiérarchie. Cela démontre que $TC-2009HD$ est au moins aussi performante que $MHD-1999$. Plus précisément, la complexité temporelle de $TC-2009HD$ et donc celle de BTD_{HD} sont identiques a celle de $MHD-1999$. Une autre conséquence de ce résultat est que nous disposons d'une nouvelle implémentation de MHD avec BTDHD qui hérite de la même complexité en temps que MHD tout en limitant drastiquement la complexité en espace.

Les implémentations existantes de TC et MHD ne permettent pas de résoudre ces instances a cause de l'espace mémoire trop important qu'elles requièrent ou du temps beaucoup trop long qu'elles mettent pour résoudre séparément les sous-problèmes d'une décomposition.

Par ailleurs, calculer une décomposition (hyper)arborescente optimale est un problème NP-difficile. La durée d'exécution des techniques exactes est trop importante. En plus, il n'existe aucune garantie sur l'efficacité pratique de l'utilisation de ces décompositions. De ce fait, ils préfèrent des heuristiques avec une meilleure complexité temporelle pour calculer nos décompositions. Donc, ils ne considérèrent ni BTD_{HD}, ni BTD_{TD}. Dans un premier temps, ils ont testé $BTD_{HMIN(HD)}$ et $BTD_{HMIN(TD)}$.

[21] a défini $BTD_{HMIN(HD)}$ comme une extension de BTD qui gère les contraintes de manière analogue a MHD. Ainsi, lors de la résolution d'un cluster, seules les contraintes données par la HD sont prises en compte. En outre, pour calculer les HD,[21] utilise les heuristiques `Bucket Elimination for Hypertree[23]` et `det-k-decomp[23]` .

$BTD_{HMIN(HD)}$ a des performances très pauvres. Elle échoue dans la résolution de beaucoup d'instances (TO ou l'espace mémoire requis dépasse 1GB, ceci étant symbolisé

par MO). Sa durée d'exécution moyenne dépasse 248s. En effet, les sous-problèmes dans une HD sont très difficiles a résoudre a cause du nombre restreint de contraintes considérées. Ce petit nombre de contraintes affaiblit la puissance des techniques de filtrage qui contribuent grandement à l'efficacité de l'énumération dans ces sous-problèmes.

$BTD_{HMIN(TD)}$ se comporte largement mieux grâce a un nombre beaucoup plus grand de contraintes dans les clusters qui deviennent plus faciles a résoudre. Elle réussit a résoudre toutes les instances avec une durée moyenne de 6,67s.

$BTD_{HMIN(TD)+HMIN(HD)}$ et $BTD_{HMIN(TD)}$ obtiennent les memes résultats rassemblés dans la même colonne. $BTD_{HMIN(TD)+HMAX(HD)}$ donne les meilleurs résultats puisque les clusters trés contraints sont plus facile a résoudre. Sa durée de résolution moyenne est 5,42s. En outre, les performances de BTDHMIN(HD) sont drastiquement améliorées si on prend en compte toutes les contraintes possibles (dans la colonne temps2) a l'image de $BTD_{HMIN(TD)}$, tandis que les bornes de complexité théorique sont préservées. Néanmoins, ces résultats restent en dessous de ceux de la méthode $BTD_{HMIN(TD)} + HMAX(HD)$. En effet, la durée moyenne de résolution de cette approche est de 6,05s.

Il faut préciser que FC, tout seul, échoue dans la résolution de prés de la totalité des instances de la classe modifiedRenault, qui ont de bonnes propriétés topologiques (w est en moyenne inférieur a n=10). Alors que, ces résultats dans la classe geom sont meilleurs par rapport a ceux des méthodes de résolution basées décomposition car la taille de ces problèmes est plus petite et la qualité de leur propriétés topologiques est assez faible (w est en moyenne très proche de n=2).

$TC - 2009HD$ et $BTD - 2009HD$, exploitent la décomposition hyperarborescente et TC ou BTD pour résoudre des réseaux de contraintes. Cette approche permet d'avoir de meilleures bornes de complexité tout en héritant de l'efficacité pratique des méthodes énumératives telles que $nFC2$, une des techniques les plus performantes dans la résolution de CSP. Mais la complexité de ces derniers est équivalent a celle de MHD. En pratique, si la largeur est très grande ces méthodes sont très couteuses aussi bien en temps qu'en espace mémoire.

4.2.2 méthde BT-DBT [25]

Une autre approche est proposée dans [25] qui est la combinaison entre l'approche de base de Gottlob et l'approche de résolution de type retour arrière pour le choix d'un tuple pour un nœud donné. Cette approche est appelée HD_DBT (Hypertree Décomposition

versus Dual BackTracking). La procédure HD_DBT considère en entrée une hypertree décomposition généralisée et complétée conformément a la définition de l'hypertree décomposition complète et retourne une solution si elle existe.

Définition Une hypertree décomposition $< T, \chi, \lambda >$ d'un hypergraphe $H =< V, E >$ est complète si chaque hyperarête h de $H =< V, E >$ est fortement couverte dans $HD =< T, \chi, \lambda >$. L'hypertree décomposition est complétée pour s'assurer que pour chaque contrainte c du CSP, il existe un nœud $n =< \lambda_n, \chi_n >$ de l'hypertree $HD =< T, \chi, \lambda >$ tel que les variables de la contrainte c sont contenues dans χ_n et $c \in \lambda_n$. Si ce nœud n'existe pas, on cherche un nœud n' de l'hypertree tel que χ couvre les variables de c puis on crée un nœud n'' fils de n' dont l'ensemble χ est l'ensemble des variables de c et dont le terme λ contient uniquement la contrainte c. Le nœud n' existe forcément parce que c'est l'une des conditions de l'hypertree décomposition généralisée La complexité théorique de cet algorithme est en $O(\mid r \mid^{w \times Nb_{noeuds}})$ ou $\mid r \mid$ est la taille de la plus grande relation, w est l'hypertree-width et Nb_{Noeuds} est le nombre de nœuds de l'hypertree décomposition. Pour résoudre le problème représenté sous forme d'hypertree décomposition [25] a proposé un algorithme de type backtrack. A la différence des algorithmes énumératifs classiques celui-ci instancie un ensemble de variables en une seule étape plutôt que d'instancier variable par variable. Le problème crucial dans la méthode de base de Gottlob est le côut des jointures effectuées aux différents nœuds de l'hypertree décomposition, aussi bien en espace qu'en temps d'exécution. C'est pour cela q'il proposé une approche qui ne calcule pas toutes les solutions au niveau d'un nœud mais ne calcule qu'une seule solution. Si cette solution est consistante avec celle des nœuds déjà résolus on continue sinon on effectue un retour arrière chronologique.

La procédure principale HD_DBT considère l'hypertree décomposition obtenue précédemment en entrée et se compose des différentes étapes décrites par l'algorithme précédent.

la comparaison de cette approche avec celle de Gottlob en terme de temps d'exécution. Les résultats obtenus par cette approche en temps d'exécution sont meilleurs pour 11 benchmarks sur 12.

4.3 Notre proposition :

Le principal inconvénient des algorithmes basés sur le retour-arrière, c'est que si une mauvaise hypothèse est faite au départ, il faudra explorer un sous-arbre de taille potentiellement très importante sans succès. cela augmente la complexité de cette approche ; pour

remédier a ce probléme dans l'approche HD2-BTD nous proposant une amélioration de cette aproche $HD2_DBT$.

la procédure de base de cettte approche est la suivante :

Procedure HD2-DBT(hypertree)

- Pré-traitement(hypertree) ;
- Résolution(hypertree) ;

Cette procédure HD2-DBT est la procédure principale de notre approche de résolution. Dans ce qui suit, nous expliquons les deux procédures.

Procedure Pré-traitement(hypertree)

```
{
 Pour tout nœud node de l'hypertree
 {
      Trouver-sequence( node) // elle n'appartient pas à une autre séquence
déjà traitée
      Nœud=deb-sequence ;
           Tant que nœud ≠ fin-seq Faire
                Filtrer(nœud,nœud-fils) ;
                Nœud=nœud-fils ;
           Fait ;
 Nœud=fin-seq ;
      Tant que nœud ≠ deb-seq Faire
           Filtrer(nœud,nœud-pere) ;
           Nœud=nœud-pere ;
      Fait ;
 }
}
```

La procédure trouver-séquence a pour rôle de trouver toutes les séquences (de longueurs maximales) de nœuds ayant $|\lambda| = 1$ ayant ce nœud comme début de séquence ; cette

séquence ne doit pas être incluse dans une autre séquence déjà traitée.

Vérifier si ce n'est pas un problème NP-complet : Si c'est le cas, prendre deux à deux mais remonter dans la vérification jusqu'au nœud ayant $\lambda > 2$. Et vérifier les autres fils aussi, car c'est possible de supprimer des tuples par lesquels les nœuds précédents étaient liés (Tous les fils d'abord). Les variables deb_seq et fin_seq indiquent respectivement le début et la fin de la séquence de nœuds de $|\lambda| = 1$.

Si ce n'est pas un problème NP complet, ce pré-traitement a un avantage : Toutes les contraintes de la séquence sont deux à deux arc-consistantes, ceci grace aux proprietés de l'hypertree décomposition.

La procédure *filtrer* a pour rôle de supprimer les tuples de son deuxième arguments qui n'ont pas de correspondants dans le premier argument.

Procédure Résolution(hypertree)
nœud=racine de l'hypertree ;
Tant que nœud != NULL Faire
Si (lambda(nœud)¿=2)
Alors traiter (nœud)
Sinon filtrer toutes les relations des fils en fonction des tuples de ce nœud
Si une des relations des fils devient vide
Alors backtrack jusqu'au nœud de *lambda* >= 2 ;
Fsi ;
Nœud=nœud suivant ;
Fait ;

Cette procédure traite les nœuds de l'hypertree ; Si le nombre de contrainte du nœud est supérieur ou égal à 2 on appelle la procédure *traiter (nœud)* pour calculer un tuple de jointure à ce nœud. Sinon on réalise juste le filtrage des relations des contraintes des nœuds fils. On réalise un Bachtrack dans trois situations :

- Soit aucun tuple n'est possible
- Soit aucun tuple n'est compatible avec un tuple du père
- Soit aucun tuple calculé ne permet de laisser une des relations des fils non vide.

Procédure traiter(nœud)
Itération : Calculer un tuple de jointure t (non déjà exploré) pour ce noeud ;
Si aucun tuple n'est possible
Alors remonter les nœuds de l'hypertree jusqu'au nœud de —lambda—¿=2 ;

Si t est compatible avec un des tuples du nœud père Alors
Filtrer toutes les relations des contraintes des nœuds fils de ce nœud courant.
Si une des relations devient vide
alors aller à itération.

La procédure traiter (nœud) ne traite que les nœuds ayant plus de deux contraintes dans le λ. Le traitement pour cette procédure est similaire à HD-DBT sauf pour les retour arrière. Dans cette version, en cas d'échec on retourne vers le nœud ayant plus d'une contrainte dans son λ.

Une fois que tous les nœuds sont traités, on réalise une semi-jointure de façon Bachtrack free pour le calcul d'une solution au problème.

Les avantages de cette approche par rapport à HD-DBT :

- On retourne directement vers le nœud de $|\lambda| > 1$ évitant ainsi beaucoup de retour-arrière entre les nœuds de la séquence. Avec cette approche le nombre de nœuds de l'hypertree concernés par le Bachtrack est égal au nombre de nœuds ayant un nombre de contraintes supérieur à 1. Dans les problèmes réels, comme Renault ou Renault modifiés, le nombre de ces nœuds est très important. Dépassant la moitié pour la famille Renault. Ceci nous permet de réduire la complexité de notre approche.

- Le prétraitement permet de détecter des inconsistances plus rapidement

4.4 conclusion

dans ce chapitre nous avons présentés les déférentes approche de résolution des CSP et pour remédier au problème de retour arrière de l'approche HD-DBT nous avons proposé une nouvelles approche qu' est une amélioration de cette dernière qui retourne directement vers le nœud de $|\lambda| > 1$ évitant ainsi beaucoup de retour-arrière entre les nœuds de la séquence. Avec cette approche le nombre de nœuds de l'hypertree concernés par le Bachtrack est égal au nombre de nœuds ayant un nombre de contraintes supérieur à 1. Dans les problèmes réels, comme Renault ou Renault modifiés, le nombre de ces nœuds est très important. Dépassant la moitié pour la famille Renault. Ceci nous permet de réduire la complexité de notre approche.

5

Conclusion générale et perspectives

Plusieurs techniques de la résolutions des problèmes de satisfaction de contraintes ont été développé durant les quarante dérniéres années. Ces techniques peuvent être divisées grossièrement en deux catégories, ceux basés sur la recherche avec backtrack et ceux basés sur la propagation de contraintes. Pour les deux approches, la complexité des algorithmes est exponentielle en la taille du problème. Afin de réduire cette complexité, on a essayé d'extraire des classes de CSP dites tractable dont celle des CSP acyclique fait partie.

Ainsi, La recherche est dirigée vers un autre axe qui consiste à décomposer la structure d'un CSP qui est un graphe ou plus généralement un hypergraphe en une structure arborescente de largeur bornée.

De nombreuses méthodes de décomposition structurelle ont été developpées, nous nous sommes intéressés à la méthode de décomposition hypetree qui généralise toutes les autres méthodes.

Il a été prouvé qu'un CSP dont la structure est acyclique peut être résolu en un temps polynomial ce qui met en accent l'importance de l'étape de décomposition. Cependant ce résultat théorique est confronté à de nombreux problèmes lors de sa mise en oeuvre parmi lesquelles le coût du travail réalisé au niveau de chaque noeud de l'arbre qui est parfois

prohibitif en terme de temps et d'espace.

Dans ce mémoire nous avons proposé une solution à ces problèmes. Nous avons présenté une nouvelle méthode de résolution des problèmes CSP après une décomposition hypertree .

nous avons proposé l'algorithme HD2-DB. Cet algorithme est constitué de deux procédures principales :

- Une procédure Pré-traitement(hypertree) dans le rôle de trouver toutes les séquences (de longueurs maximales) de nœuds ayant —lambda— égal à 1 ayant ce nœud comme début de séquence ; cette séquence ne doit pas être incluse dans une autre séquence déjà traitée.

- Une procédure Résolution (hypertree) qui traite les nœuds de l'hypertree . Cet algorithme est basé sur la notion backtrack .

Nous proposons comme perspectives les points suivants :
- Mise en pratique de cette approche.
- Utiliser des heuristiques pour le choix des clusters à évaluer en priorité et eventuellement à évaluer en parallàle.
- Chercher une décomposition hypertree qui ne soit pas nécessairement optimale mais qui ne demande pas un temps important.

6

Bibliographie

[1] U.Montanari, Networks of constraintes : Fondamental properties and application to picture, Proc. of Information Sciences (1974).

[2] k.Mackworth, Consistency in networks of relations, Artificial intelligence (1977).

[3] E.C Freuder, A sufficient condition for backtrack free search, Journal of the ACM Vol29 N°1 (1982).

[4] Hypertree decompositions and tractable queries, J. Comput. Syst. Sci.64(3), 579-627 (2002).

[5] Constraint processing, Morgan Kaufmann Publisher (2003).

[6] Zoltan Miklos : Understanding tractable decompositions for constraint satisfaction :Thèse de Doctorat 2008

[7] Philippe Jégou, Samba Ndoye NDiaye et Cyril Terrioux : Complexité du Forward Cheching et hiérarchie des décvompositions revisitées . Actes JFPC 2008.

[8] David Cohen, Peter Jeavons and Marc Gyssens : A Unified theory of structural tractability for constraint satisfaction problems. In Journal of Computer and System Sciences

74 721-743. Elsevier 2007

[9] David Cohen, Peter Jeavons and Marc Gyssens : A Unified theory pf structural tractability for constraint satisfaction and Spread Cut Decomposition. Proceedings of IJCAI '05.2005

[10] Martin Grohe and Daniel Marx. Constraint solving via fractional edge cover. In SODA 2006 pages 289-298

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

[12] T.K Satish Kumar. A model counting characterization of diagnoses. In Proc. of the 13th International Workshop on Principles of Diagnosis, 2002.

[13] A. Darwiche. On the tractable counting of theory models and its applications to truth maintenance and belief revision. Journal of Applied Non-classical Logic, 11 :11-34, 2001.

[14] R. Burton and J. Steif. Nonuniqueness of measures of maximal entropy for subshifts of finite type. In Ergodic theory and dynamical system, 1994

[15] M. Mann, G. Tack, and S.Will. Decomposition during search for propagation-based constraint. In CoRR abs/0712.2389 :2007.

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

[17] D. Sabin and E. C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming, number 874 in LNCS, Rosario, Orcas Island (WA),May 1994. Springer-Verlag.

[18] A. Darwiche. New advances in compiling cnf to decomposable negation normal form. In Proc. of ECAI, pages 328-332,2004.

[19] S. Golumb and L. Baumert, Backtrack programming, Journal of the ACM, pages 516-524 (1965).

[21] P. Jégou,S.N Ndiaye et C Terrioux. Stratégies hybrides pour des décompositions optimales et efficaces. Actes JFPC 2009

[22] P. Jégou, S.N. Ndiaye, and C. Terrioux. A New Evaluation of Forward Checking and its Consequences on Eficiency of Tools for Decomposition of CSPs. In ICTAI 2008.

[23] G. Gottlob and M. Samer. A backtracking-based algorithm for hypertree decomposition. ACM Journal of Experimental Algorithmics, 13, 2008.

[24] K.Amroune Comparaison des méthodes de décomposition structurelles ; Août 2008

[25] K.Amroune Z.Habbas Hypertree Décomposition pour la résolution des problémes de satisfaction de contraintes basée sur un Dual Backtracking,Actes JFPC 2010.

[26] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions : A survey. In Proceedings of MFCS 2001.