Réseaux de contraintes

Thomas Schiex

Avril 2000

Table des matières

Avant-propos

Durant ces dix dernières années, j'ai effectué mes recherches dans trois environnements différents :

- L'IRIT (institut de recherches en informatique de Toulouse) est un centre de recherches universitaire en informatique. J'y ai effectué ma thèse.
- Le CERT-ONERA (centre d'études et de recherches de Toulouse de l'office national d'études et de recherches aérospatiales) est un centre de recherches public, pluri-disciplinaire, et essentiellement financé par des contrats. J'y ai été ingénieur de recherches durant quatre ans, dans le département d'automatique, dans l'équipe de productique.
- Enfin, l'INRA est, comme le CNRS, un établissement public à caractère scientifique et technique, essentiellement concerné par l'agronomie et la biologie. J'y suis chargé de recherches depuis cinq ans, dans la chaleureuse unité de biométrie et intelligence artificielle.

Cette variété d'environnements a naturellement induit une certaine variété dans les thèmes de recherches qui m'ont successivement attiré : sémantique, interprétation et compilation des langages fonctionnels paresseux, problèmes de satisfaction et d'optimisation sous contraintes, bioinformatique. Plutôt que de présenter une (ou plutôt trois) synthèse exhaustive de mes travaux, j'ai préféré privilégier le domaine de recherche qui a, pour mon plus grand plaisir, occupé la majeure partie de mon activité de recherche : le problème de satisfaction de contraintes. C'est par le biais d'un contrat CNES, lors de mon arrivée dans l'équipe de productique du CERT en 1991, que j'ai commencé à m'intéresser à ce domaine de recherche. Si j'ai pu m'y épanouir, c'est grâce aux chercheurs avec lesquels j'ai pu travailler et qui m'ont fait découvrir que la recherche n'est pas, essentiellement, une activité solitaire. Je voudrais les remercier tous à cette occasion en réservant une mention spéciale à Michel Cayrol pour son enthousiasme, sa rigueur et pour avoir accepté d'être pour la deuxième fois mon directeur de recherches et enfin à Gérard Verfaillie qui possède ces qualités humaines et intellectuelles, bien difficiles à énumérer, qui font que j'attends toujours avec fébrilité notre prochaine discussion.

Dans la suite, afin de faciliter l'identification des travaux auxquels j'ai participé, chaque citation d'une publication m'impliquant sera mise en évidence par la présence d'un symbole dans la marge comme ceci. ⋆

1 Introduction

Le formalisme des réseaux de contraintes offre un outil de modélisation simple et général, utilisé dans de nombreux domaines : configuration, gestion et allocation de ressources, ordonnancement, emploi du temps, traitement du langage naturel... Durant les 30 dernières années, un ensemble d'algorithmes permettant de manipuler ces réseaux ont vu le jour, permettant de répondre, au moins en théorie, à une grande variété de questions. Une partie de ces algorithmes est disponible dans des logiciels commerciaux tels que CHIC ou Ilog Solver par exemple.

Un réseau de contraintes est défini par un ensemble de variables, ayant chacune un domaine de valeurs possibles, et par un ensemble de contraintes. Chaque contrainte limite les combinaisons de valeurs que peuvent prendre les variables sur lesquelles elle pèse. Le problème central, dit « problème de satisfaction de contraintes » (ou CSP pour *constraint satisfaction problem*), consiste à affecter une valeur de son domaine à chacune des variables du réseau de façon à ce qu'aucune combinaison de valeur non autorisée par les contraintes n'apparaisse. Ce problème est NP-difficile.

Introduit dans les années 1970, les réseaux de contraintes et leurs algorithmes ont fait l'objet d'un intérêt croissant, en particulier à la suite de leur introduction dans les langages de programmation logique (van

Hentenryck 1989). La grande majorité des travaux sont liés au caractère NP-difficile du problème de satisfaction et ont pour but d'exhiber une méthode de résolution la moins inefficace possible. Les progrès qui ont été réalisés sont énormes et ont aussi contribué à l'intérêt commercial porté à ces techniques.

L'utilisation de ce type de modèle dans le cadre d'applications réelles met en évidence un ensemble de limitations qui ne sont pas nécessairement liées à l'inefficacité des algorithmes mais à la simplicité du formalisme, qui ne permet pas de construire une modélisation réaliste de certains problèmes. Ainsi, depuis le début des années 1990, nombre de travaux se sont intéressés à l'extension du formalisme des réseaux de contraintes. Le caractère NP-difficile du problème central étendu n'est bien sûr pas remis en cause par ces extensions et l'efficacité algorithmique reste donc cruciale. Par contre, les propriétés et algorithmes usuels des CSP deviennent généralement obsolètes et doivent être adaptés, étendus ou reconstruits. Une grande partie de mes travaux se situent dans cet axe de recherche.

Après une rapide introduction, je détaillerai mes recherches selon trois axes :

- l'amélioration de l'algorithmique des CSP classiques,
- la prise en compte de modifications du réseau de contraintes,
- la prise en compte de préférences ou d'incertitudes.

Une grande partie de ces travaux a été motivée par (et réalisée pendant) les contrats que j'avais pris en charge durant mon emploi au CERT-ONERA. Malgré la relative liberté d'action que me laissait l'exécution de certains de ces contrats, j'ai souvent regretté leur durée trop courte, presque toujours inférieure à un an, souvent à 6 mois. Ces durées ne m'ont pas toujours permis d'épuiser, malgré des résultats intéressants, certains des axes de recherche que j'ai pu explorer. J'évoquerai, au fil du texte, les suites que l'on peut donner à certains d'entre eux.

2 Réseaux de contraintes et algorithmes

DÉFINITION 1 *Un réseau de contraintes est défini par un triplet* $\langle X, D, C \rangle$.

- *l'ensemble* $X = \{1, \ldots, n\}$ *est un ensemble de* n variables.
- *chaque variable* $i \in X$ *est associée à un* domaine *de valeurs* $d_i \in D$ *et peut donc recevoir toute valeur* $a \in d_i$ *(cette valeur sera aussi notée* (i, a)*).*
- *l'ensemble* C *est un ensemble de* contraintes*. Chaque contrainte* $c \in C$ *est définie sur un ensemble de variables* $X_c \subset X$ *par une relation, également notée* c*, sous-ensemble du produit cartésien* $\prod_{i \in X_c} d_i$ *qui spécifie les combinaisons de valeurs (ou* tuples*) autorisées pour les variables de* X_c.

La définition de chaque contrainte c peut se faire en extension, par la donnée de l'ensemble des tuples autorisés (ou interdits), ou en intension, par la fonction caractéristique de cet ensemble. On appelle test de contrainte la vérification de la présence d'un tuple t dans une contrainte c. L'hypergraphe $(X, \{X_c\}_{c \in C})$ est appelé l'*hypergraphe des contraintes*. La cardinalité $|X_c|$ est appelée *arité* de la contrainte c.

Une *instanciation* t d'un ensemble de variables $V \subset X$, également notée $t[V]$, associe une valeur de son domaine à chaque variable de V. Elle est dite *partielle* si $V \neq X$ et *complète* sinon. Les variables de V sont dites instanciées dans $t[V]$. Nous nous permettrons, dans la suite de ce document, de confondre tuples de valeurs et instanciations d'un ensemble de variables et de manipuler ces objets comme des ensembles.

Étant donné une instanciation $t[V]$, la projection de t sur $W \subset V$ est notée $t[V]_{\downarrow W}$. Cette notation est étendue aux contraintes (relations) : la projection $c_{\downarrow W}$ d'une contrainte c sur un ensemble de variables $W \subset X_c$ est la contrainte définie sur W et égale à l'ensemble des projections sur W des tuples de c. Étant donné une contrainte $c \in C$, on appelle support de la valeur (i, a) sur c tout tuple $t \in c$ tel que $t_{\downarrow \{i\}} = ((i, a))$. Un tuple $t[V]$ satisfait la contrainte c si $t_{\downarrow X_c} \in c$, il la viole si $X_c \subset V$ et si $t_{\downarrow X_c} \notin c$.

Un tuple $t[V]$ est localement cohérent s'il ne viole aucune contrainte de C. Un tuple $t[X]$ localement cohérent est une solution du réseau de contraintes. On dira que deux réseaux de contraintes $\langle X, D, C\rangle$ et $\langle X, D', C'\rangle$ sont équivalents s'ils possèdent le même ensemble de solutions. Le problème de satisfaction d'un réseau de contraintes consiste à exhiber, quand il en existe, une solution de ce réseau.

Une grande partie de travaux sur les CSP se limitent au cas des réseaux dits *binaires* pour lesquels les contraintes sont toutes d'arité 2. Dans ce cas, des notations spécifiques sont utilisées : la contrainte c telle que $X_c = \{i, j\}$ est notée c_{ij} et l'hypergraphe des contraintes devient alors le *graphe des contraintes*.

3 Algorithmes pour les réseaux de contraintes

Un des premiers algorithmes conçus pour résoudre le problème de satisfaction de contraintes est l'algorithme de *backtrack chronologique* (BT), aussi appelé algorithme de *test and generate* (voir algorithme 1). Étant donné un réseau de contraintes, l'appel à BT($\varnothing$) retourne *vrai* si le réseau a une solution et *faux* sinon. L'algorithme explore un arbre dont chaque nœud correspond à une instanciation. La racine correspond à l'instanciation vide. Les fils d'un nœud sont obtenus en étendant l'instanciation courante de toutes les façons possibles sur une nouvelle variable i. Dans la suite, nous dirons d'une variable qui n'apparaît pas dans l'instanciation courante que c'est une variable *future*. Une variable qui apparaît dans l'instanciation courante t est une variable *passée*. Elle est plus *basse* qu'une autre si elle a été instanciée à une profondeur plus grande dans la branche courante. Cet algorithme fondamental est extrêmement inefficace en pratique et a fait l'objet de nombreuses améliorations. Notre propos n'est pas de les énumérer toutes et nous nous focaliserons sur une classe d'améliorations : l'introduction de mécanismes de *filtrage* pendant la recherche.

Algorithme 1: Backtrack chronologique

```
Fonction BT(t : instanciation) : booléen
    si (|t| = n) alors retourner vrai;
    soit i une variable non instanciée;
    pour chaque a ∈ d_i faire
        t' ← t ∪ {(i, a)};
        si t' est localement cohérent alors
            si BT(t') alors retourner vrai;
    retourner faux;
```

3.1 Filtrage systématique

Une procédure de *filtrage* d'un réseau de contraintes, aussi appelée *propagation de contraintes*, a pour fonction de mettre en évidence le fait que certaines combinaisons de valeurs localement cohérentes ne peuvent mener à une solution. Utilisée après chaque instanciation pendant la recherche, elle permet de détecter les échecs plus rapidement que par le simple test de cohérence locale de BT. Toute procédure de filtrage a un coût qui peut être plus ou moins contrebalancé par la réduction de l'arbre de recherche qu'elle induit. Pendant longtemps, à la suite de l'article de (Haralick & Elliot 1980), on a pensé que le niveau idéal de filtrage était défini par la procédure de *forward-checking*. Dans le cas binaire, où il est défini sans ambiguïté, ce filtrage consiste à supprimer du domaine des variables non instanciées toute valeur qui, avec une variable instanciée, forme une paire non autorisée par une contrainte. L'algorithme de recherche de solution correspondant, aussi appelé *forward-checking*, est dérivé de BT en remplaçant le test de cohérence

locale de t' par un appel à `FC-propagation(`i, a`)` (cf. algorithme 2). Les domaines étant modifiés par cette procédure, il est nécessaire de mettre en place une mécanique de sauvegarde/restauration du contexte. La vérification de la cohérence locale n'est pas nécessaire car après le filtrage les domaines ne contiennent plus que des valeurs compatibles avec l'instanciation courante.

Algorithme 2: Filtrage par *forward checking*

Fonction `FC-propagation(`i : variable, a : valeur`)` : booléen
  pour chaque j *non instanciée tq.* $c_{ij} \in C$ **faire**
    pour chaque $b \in d_j$ *tq.* $((i,a),(j,b)) \notin c_{ij}$ **faire**
      Effacer (j,b) de d_j;
    si $d_j = \varnothing$ **alors retourner** *faux*
  retourner *vrai*

Si l'on néglige le *partial look ahead* (Nadel 1988), maintenant un peu oublié, le niveau de filtrage suivant est le filtrage par cohérence d'arc (cf. définition 2). Il s'agit du filtrage le plus connu et maintenant le plus utilisé.

DÉFINITION 2 *Une contrainte $c \in C$ vérifie la cohérence d'arc ssi $\forall i \in X_c, \forall a \in d_i, (i,a)$ possède un support sur c. Un réseau de contraintes vérifie la cohérence d'arc ssi toutes ses contraintes la vérifient.*

Établir la cohérence d'arc dans un réseau de contraintes $\langle X, D, C\rangle$ consiste à produire un réseau de contraintes $\langle X, D', C\rangle$, équivalent au réseau d'origine et vérifiant la propriété de cohérence d'arc. Ce réseau équivalent et arc cohérent, appelé fermeture maximale arc cohérente, est unique et peut être obtenu par suppression de toutes les valeurs n'ayant pas de support sur une contrainte, jusqu'à stabilisation. L'ensemble de domaines D' est appelé sous-domaine maximal arc cohérent. Comme pour toute procédure de filtrage, si le réseau obtenu a un domaine vide (il est arc incohérent), on a prouvé que le problème original n'avait pas de solution.

Le filtrage par cohérence d'arc a longtemps été utilisé en prétraitement des problèmes et son utilisation dans ce cas reste répandue (en particulier dans le cas d'applications interactives où il fournit un retour de taille raisonnable). Mais tout comme le *forward checking*, on peut l'utiliser pendant la recherche, en remplaçant le test de cohérence locale de BT par un test de non arc incohérence (algorithme *CS2* (Gaschnig 1974) ou *really full look ahead* (Nadel 1988)). On a longtemps jugé que cet algorithme effectuait un travail de filtrage trop lourd et que le jeu n'en valait pas la chandelle. Entre temps, l'augmentation de la puissance des machines a permis de s'attaquer à des problèmes de taille croissante et la sophistication des algorithmes de filtrage par cohérence d'arc ont considérablement accru leur efficacité et leur incrémentalité et l'on considère maintenant qu'une variante de l'algorithme *really full look ahead* (algorithme MAC, introduit dans (Sabin & Freuder 1994; Bessière, Freuder, & Régin 1995)) est un des algorithmes généraux de résolution de CSP les plus efficaces actuellement.

Dans un tel algorithme, et en première approximation, la seule information véritablement utilisée par l'algorithme d'exploration arborescente pour décider s'il effectue ou non un retour-arrière est l'arc incohérence du problème. Dans tous les cas, l'algorithme de filtrage va identifier le sous-domaine maximal arc cohérent. Dans le cas où le problème n'est pas arc-incohérent, il suffit de produire un sous-domaine arc cohérent non vide quelconque. Il s'avère que ce dernier problème peut être sensiblement plus simple à résoudre, en particulier lorsque le réseau de contraintes (ou un sous-réseau) est faiblement contraint.

L'exploitation de cette idée nous a amené à la définition de l'algorithme de filtrage par arc cohérence
⋆ paresseux LAC7 (Schiex *et al.* 1996), une sophistication de l'algorithme AC7 (Bessière, Freuder, & Régin

1995). L'algorithme conserve les complexités temporelles et spatiales dans le pire des cas de l'algorithme AC7 mais son application en prétraitement sur des problèmes aléatoires, académiques ou réels montre que la simplification du problème résolu peut induire des gains très importants, tant en temps qu'en mémoire. Dans le pire des cas, le surcoût est très faible. De plus, l'algorithme LAC7 offre une incrémentalité extrême puisqu'il est possible après son exécution d'étendre incrémentalement le sous-domaine arc cohérent identifié. Si on le désire, le sous-domaine maximal arc cohérent peut *in fine* être produit. Cette utilisation permet d'augmenter la rapidité des interactions dans le cas d'un système de résolution interactif.

En pratique, l'efficacité de l'algorithme MAC dépend fondamentalement de l'ordre utilisé pour instancier les variables. Par exemple, une heuristique d'ordonnancement de variables bien connue sélectionne une variable ayant un nombre de valeurs minimum dans le sous-domaine maximal arc cohérent. Or, l'algorithme LAC7 ne fournit plus cette information. Nous avons donc défini une extension de l'algorithme (Schiex *et al.* 1996) qui fournit cette information en plus d'une éventuelle preuve de non arc ⋆ incohérence (toute autre heuristique dépendant de façon monotone du cardinal du domaine peut être utilisée). On observe encore une fois que la restriction à ces deux services peut induire des gains importants, en particulier sur les problèmes ayant une certaine variabilité dans la taille des domaines, ce qui est le cas pour une grande partie des problèmes réels. Une des suites de ce travail consiste à définir et à évaluer expérimentalement une extension de l'algorithme MAC exploitant cet algorithme LAC7.

3.2 Mémorisation de contraintes

Au lieu d'effectuer un filtrage systématique à chaque nœud de l'arbre de recherche, il est possible de profiter de la mécanique de la recherche arborescente de BT pour effectuer un filtrage « opportuniste ». C'est le principe des algorithmes de mémorisation de contraintes appelés aussi algorithmes d'apprentissage (Dechter 1986; Dechter 1990). Comme les algorithmes de filtrage systématique, ces algorithmes identifient et mémorisent, pendant la recherche, des tuples localement cohérents mais qui ne peuvent s'étendre à une solution. De tels tuples seront appelés par la suite *nogoods*[1].

Durant l'exécution de l'algorithme BT, on appelle situation de cul-de-sac (ou *dead-end*) une situation dans laquelle le tuple t courant n'a pu être étendu en un tuple localement cohérent sur la variable suivante i. On peut alors conclure que le tuple t, est un nogood et explicitement l'interdire.

Dans le cas de l'algorithme BT, le nogood t n'est pas vraiment intéressant car, du fait du parcours ordonné de l'espace de recherche, il ne sera plus jamais rencontré par la suite. On peut toutefois extraire un sous-tuple de t qui reste un nogood. Considérons l'ensemble C_e des contraintes dont la violation a été détectée lors de l'extension du tuple de toutes les façons possibles sur i. Cet ensemble contient une contrainte par valeur de i. La projection du tuple t sur l'ensemble de variables $X_e = (\cup_{c \in C_e} X_c) \setminus \{i\}$ définit un sous-tuple de t qui est nécessairement localement cohérent et qui, comme t, ne peut être étendu en un tuple localement cohérent sur i. En effet, toute extension violera une contrainte de C_e au moins. Ce nogood $t_{\downarrow X_e}$ va pouvoir être explicitement interdit, pendant la recherche. On notera par ailleurs que le tuple courant t violant cette nouvelle contrainte, on va effectuer un retour arrière non chronologique jusqu'à un point où $t_{\downarrow X_e}$ n'est plus inclus dans le tuple courant *i.e.*, jusqu'à ce qu'une variable de X_e (la plus basse) change de valeur. Dans cette description, le retour arrière intelligent apparaît alors comme un apprentissage « jetable » : un nogood inclus dans l'instanciation courante est identifié, on effectue un retour arrière jusqu'à un point où ce nogood n'est plus violé et l'on oublie ce nogood.

L'algorithme que je viens décrire est l'algorithme de *shallow learning* (Dechter 1990). Le retour arrière effectué correspond à celui de l'algorithme BJ de (Gaschnig 1979).

[1] Contrairement aux nogoods des ATMS, aucune condition de minimalité n'est imposée.

Dans le pire des cas, le nombre de nogoods que peut produire cet algorithme croît exponentiellement avec la taille du problème et il est donc judicieux de limiter la mémorisation à un sous-ensemble raisonnable de nogoods. Un des choix qui est fait habituellement est de se limiter à des nogoods de taille inférieure à un entier k fixé. On parle alors d'apprentissage d'ordre k (Dechter 1990). Ce choix se justifie par le fait qu'un nogood a un pouvoir de coupe qui est d'autant plus important qu'il est de petite taille et par le fait que le nombre de ces nogoods est en $O(d^k)$. Les autres nogoods sont simplement utilisés pour effectuer un retour arrière intelligent puis oubliés. D'autres choix ont été considérés : limitation par la topologie du réseau de contraintes (on ne mémorise un nogood que si les variables qu'il implique sont déjà impliquées dans une même contrainte), limitation par la pertinence (*relevance*) utilisée par exemple dans l'algorithme *dynamic backtracking* (Ginsberg 1993) que j'évoquerai rapidement par la suite.

Un des moyens possibles pour améliorer cet algorithme consiste à augmenter le nombre de nogoods identifiés par l'algorithme. On peut remarquer que la situation de *cul-de-sac* n'est pas la seule situation où l'on peut identifier un nogood. Il suffit en effet qu'aucun descendant du nœud courant ne soit une solution pour que l'on conclut que le tuple courant t est un nogood. Dès lors que l'on collecte l'ensemble des contraintes C_e qui ont été la cause des échecs sur toutes les feuilles du sous-arbre, on va pouvoir en extraire, comme précédemment, le nogood $t_{\downarrow X_e}$. Nous avons ainsi défini l'algorithme NR ou *nogood* ⋆ *recording* (Schiex & Verfaillie 1993; Schiex & Verfaillie 1994a). Le retour arrière intelligent effectué par cet algorithme correspond à celui effectué par l'algorithme CBJ (*conflict directed backjumping*) introduit dans (Prosser 1993b; Prosser 1993a).

Il est possible de mettre en relation l'ensemble des nogoods identifiés et mémorisés par cet algorithme et ceux qui seraient identifiés par un algorithme de filtrage systématique. J'ai montré que l'algorithme NR_k, effectuant un apprentissage d'ordre k, mémorise au niveau i de l'arbre de recherche un sous-ensemble des nogoods qui auraient été identifiés par un filtrage par $(k, n-i)$-cohérence forte (Freuder 1985) et peut mémoriser des nogoods qui n'auraient pas été identifiés par un filtrage par $(k, n-i-1)$-cohérence forte. En fait, il s'agit essentiellement d'une forme orientée de (i, j)-cohérence forte.

L'algorithme NR améliore considérablement l'efficacité de l'algorithme de *shallow learning*. Son efficacité peut encore être augmentée en l'hybridant avec un mécanisme de filtrage pendant la recherche. Nous nous sommes limités à un mécanisme de filtrage par *forward checking* (Haralick & Elliot 1980), mais il est possible d'utiliser n'importe quel mécanisme de filtrage pourvu qu'il fournisse, pour chaque valeur (ou tuple) effacée, un ensemble de contraintes $C_e \subset C$ suffisant pour justifier l'effacement. Ce service est ⋆ simple à fournir dans le cas du *forward checking*. Nous avons ainsi défini ainsi l'algorithme NR-FC (Schiex & Verfaillie 1993). Cet algorithme est sensiblement plus efficace que l'algorithme FC et, comme tous les algorithmes effectuant des retours arrières intelligents, plus résistant à de mauvais choix de variables. Il mémorise cependant sensiblement moins de nogoods que l'algorithme NR pour un ordre d'apprentissage donné. Cela n'est pas étonnant : le filtrage effectué à chaque nœud par FC limite les possibilités d'échecs et donc d'apprentissage.

De façon générale, la quantité de filtrage effectuée par l'apprentissage est imposée par la mécanique de l'algorithme d'exploration arborescente qui le supporte car ce sont les échecs de l'algorithme d'explo- ⋆ ration qui sont la source du filtrage réalisé. Nous avons donc tenté dans (Schiex & Verfaillie 1994b) de modifier la mécanique de l'algorithme BT en faisant croître artificiellement le nombre d'échecs réalisés par l'algorithme afin d'augmenter la qualité du filtrage. Notre motivation principale était de tenter de produire un nombre plus important de nogoods car nous pensions que cela pourrait éventuellement améliorer l'efficacité globale de l'algorithme. L'idée de base est très simple : lorsque dans l'algorithme BT, on vérifie la cohérence locale du tuple courant, on cesse cette vérification dès qu'une contrainte est violée. C'est cette contrainte que l'on retrouve ensuite dans l'ensemble C_e. Nous avons modifié cette fonction de vérification de la cohérence locale afin qu'elle ne s'arrête plus à la première contrainte violée mais à la

deuxième contrainte violée (si elle existe). Chaque échec retourne ainsi un ensemble de 1 ou 2 contraintes. Moyennant une sélection drastique des ensembles C_e que l'on peut alors construire, il devient possible, dans la situation où NR pouvait identifier un nogood, d'en identifier jusqu'à deux. En ce qui concerne le retour arrière intelligent, il est possible de l'effectuer sur la plus haute des plus basses variables des deux nogoods. L'algorithme de *stubborn nogood recording* ainsi défini et utilisant un apprentissage d'ordre 2 présente des performances qui, sans aucun mécanisme de filtrage systématique, dépassent les performances de l'algorithme NR-FC.

Ce résultat surprenant demanderait une analyse plus poussée, en particulier en effectuant des expérimentations plus larges que celles réalisées à l'époque. Ces expérimentations utilisaient en effet une méthodologie introduite par P. Prosser (1993a) et consistant à résoudre le fameux, mais maintenant ridicule, problème du Zèbre en utilisant un grand nombre d'ordres aléatoires et statiques de variables. En dehors de la petite taille du problème, ces tests favorisent sensiblement les algorithmes effectuant un retour arrière intelligent qui sont moins sensibles que les autres à de mauvais ordres d'instanciation des variables. En effet, parmi tous les ordres aléatoires considérés, beaucoup définissent de mauvais choix.

Idéalement, une telle expérimentation devrait s'effectuer à la fois sur des problèmes aléatoires (Hubbe & Freuder 1992), ayant le mérite d'être faciles à construire mais malheureusement trop uniformes et peu structurés et sur des problèmes réels, plus structurés, mais malheureusement difficiles à obtenir, souvent très complexes et très spécifiques. La seule façon de contrebalancer cette spécificité serait de construire une bibliothèque suffisamment large de problèmes réels. Ce problème d'expérimentation avait été le sujet de vives discussions lors du workshop que j'avais co-organisé à l'ECAI en 1994. Bien qu'on observe une amélioration sensible des « standards » d'expérimentation, on est encore loin de la définition d'une bibliothèque de problèmes qui permettraient de mettre en évidence les forces et faiblesses des algorithmes existants. C'est pourquoi j'ai participé à la mise en place de la *Benchmark column* du *Constraints journal*, dirigé par Claude Lepape et qui a permis de récupérer un certain nombre de jeux de données de problèmes « réels ». En collaboration avec d'autres initiatives, telles que celle de la CSPLib (initiée par T. Walsh), nous devrions pouvoir aboutir à une amélioration sensible de cet état de chose dans les années à venir.

Un autre algorithme effectue également une forme d'apprentissage pendant la recherche. Il s'agit de l'algorithme de *dynamic backtracking* ou DBT (Ginsberg 1993). L'algorithme DBT peut être vu comme une variante de l'algorithme BT. Il est caractérisé par le fait qu'il effectue un retour-arrière dit « chirurgical ». Lors de la détermination d'un nogood, dans la même situation que NR, il désinstancie la variable i la plus basse du nogood et elle seule. Les variables qui suivent i conservent leur valeur. Le nogood devient alors une justification de l'effacement de la valeur de cette variable (ou *elimination explanation* dans la terminologie de (Ginsberg 1993)) et est conservé tant que les variables du nogood autres que i ne changent pas de valeur. L'apprentissage effectué par cet algorithme n'est donc pas limité par la taille des nogoods mais par leur pertinence (*relevance*). Le nombre de nogoods mémorisés à chaque instant est en $O(nd)$ où d est la taille du plus grand domaine. L'algorithme est très proche de l'algorithme d'*intelligent backtracking* défini dans (Bruynooghe 1981).

De la même façon que nous avions équipé l'algorithme NR d'un mécanisme de filtrage par *forward checking*, nous avons équipé l'algorithme DBT du même type de filtrage (Verfaillie & Schiex 1994b). L'algorithme DBT-FC ainsi défini a montré des performances très intéressantes, que ce soit pour la résolution de réseaux de contraintes classiques ou pour résoudre des problèmes dynamiques, que j'aborderai plus tard. ⋆

Il resterait à équiper les algorithmes NR et DBT d'un filtrage par cohérence d'arc au lieu d'un simple filtrage par *forward checking*. Le calcul des ensembles C_e est un peu plus délicat dans ce cas mais il est déjà effectué par la grande majorité des algorithmes de filtrage par arc cohérence dynamiques (Bessière 1991;

Debruyne 1995). Il est probable que le nombre de nogoods construits diminue encore et les résultat obtenus pour l'instant avec des algorithmes effectuant un maintien de cohérence d'arc et des retours arrières intelligents de type `CBJ` sont plutôt décevants (Bessière & Régin 1996). Malgré cela, Jussien, Debruyne, & Boizumault (2000) semblent obtenir des résultant intéressants en équipant l'algorithme `DBT` d'un mécanisme de maintien de la cohérence d'arc.

4 Problèmes dynamiques

Dans cette partie, nous commençons à aborder les travaux motivés par les limitations de la modélisation par réseaux de contraintes. Un grand nombre de problèmes ne sont pas définis une fois pour toutes puis résolus mais évoluent au cours du temps du fait de l'environnement, de l'utilisateur : changement d'objectif, présence d'aléas (climat, pannes), système interactif...

Il ne s'agit donc pas tant de faire évoluer le modèle des réseaux de contraintes que de fournir des algorithmes incrémentaux, capables en cas de changement du problème de produire aussi rapidement que possible un nouveau résultat. Dans le cas du problème de satisfaction, une solution définit généralement les caractéristiques d'un objet ayant une réalité physique : conception mécanique, ordonnancement, emploi du temps... et cet objet a peut-être déjà été mis en œuvre. Il est donc important que la nouvelle solution produite ne soit pas trop distante, selon une mesure qui est essentiellement dépendante du problème, de la solution précédente. Nous dirons d'un algorithme qui produit des solutions peu éloignées quand elles existent qu'il est « stable ».

En termes de réseaux de contraintes, les changements peuvent être de différents types : ajout ou suppression de variables, de valeurs dans les domaines ou de contraintes. Mais toutes les difficultés sont capturées par l'ajout ou la suppression de contraintes. C'est ainsi qu'ont été définis les réseaux de contraintes dynamiques (Dechter & Dechter 1988) :

DÉFINITION 3 *Un problème de satisfaction de contraintes dynamique est défini par une séquence $P_0, \ldots, P_k$ de réseaux de contraintes. Chaque réseau P_{i+1} est le résultat de l'application d'une modification élémentaire au problème P_i. Cette modification peut être une restriction (une contrainte est ajoutée au réseau) ou une relaxation (une contrainte est enlevée du réseau).*

La suppression d'une contrainte préserve les solutions mais remet en cause toutes les déductions et en particulier tout filtrage ayant pu avoir lieu. L'ajout d'une contrainte préserve les déductions et donc les effets du filtrage mais remet en cause les solutions.

Les premiers algorithmes qui ont été étendus pour prendre en compte ce type de modifications sont les algorithmes de filtrage par arc cohérence. L'ajout de contraintes ne pose pas de problème : si certaines valeurs n'ont pas de support sur la contrainte ajoutée, il va falloir les supprimer et continuer le filtrage. La suppression est plus délicate et peut être traitée via la mémorisation de justifications (Doyle 1979) des effacements en termes d'ensembles de contraintes. De nombreux algorithmes de filtrage par arc cohérence ont ainsi vu le jour (Bessière 1991; Prosser, Conway, & Muller 1992; Berlandier & Neveu 1994; Debruyne 1995). Comme nous l'avons déjà mentionné, certains de ces algorithmes sont intéressants en dehors du cadre dynamique parce qu'ils fournissent le service de justification des effacements indispensable au fonctionnement des algorithmes d'apprentissage ou de retour arrière intelligent.

Le problème devient sensiblement plus difficile quand on souhaite construire un algorithme de satisfaction incrémental. Il va encore une fois falloir mémoriser de l'information, mais le caractère exponentiel de l'espace exploré montre qu'il va falloir rigoureusement limiter la quantité d'information mémorisée en

la sélectionnant selon l'utilité qu'elle pourra avoir dans le futur. Deux approches distinctes apparaissent suivant que les objets mémorisés et réutilisés sont des solutions ou des nogoods.

4.1 Mémorisation de solutions

On sait que la suppression d'une contrainte préserve les solutions. Plus généralement, on peut informellement supposer que si deux réseaux de contraintes partagent l'essentiel de leurs contraintes, une partie des solutions d'un réseau doit pouvoir s'adapter avec peu de modifications en une solution de l'autre réseau (voir aussi les super-modèles de (Ginsberg, Parkes, & Roy 1998)). Il semble donc judicieux de mémoriser une ou plusieurs solutions déjà calculées afin de pouvoir les adapter par la suite.

Une première méthode, très simple à mettre en œuvre, consiste à utiliser, quand elle existe, la solution du problème précédent comme heuristique dans le cadre de la recherche courante. Une approche similaire a été proposée dans (Minton *et al.* 1992). Elle s'appuie sur une heuristique de minimisation du nombre de contraintes insatisfaites. Dans (Verfaillie & Schiex 1994a), nous avons proposé une méthode, appelée *local changes* qui s'appuie sur un mécanisme de retour arrière original et peu destructif : suite à l'affectation de la variable v, seules les variables dont la valeur est incompatible avec celle de v sont retirées du tuple courant, indépendamment de l'ordre d'instanciation des variables. Les expérimentations de ces deux dernières méthodes dans (Verfaillie & Schiex 1995) montrent de très bon résultats sur les problèmes sous-contraints, à la fois en termes d'efficacité et en termes de stabilité (nombre de variables ayant des valeurs modifiées). ⋆ ⋆

4.2 Mémorisation de nogoods

Cette approche n'est pas très éloignée de celle utilisée dans les algorithmes de filtrage par arc cohérence dynamiques. L'ajout de contrainte préservant les nogoods construits par apprentissage, ceux-ci peuvent être conservés dans ce cas. Lors d'une suppression de contraintes, il suffit de disposer d'une justification de chaque nogood en terme d'ensemble de contraintes pour pouvoir les remettre en cause individuellement.

Dans tous les algorithmes d'apprentissage que nous avons détaillés dans la partie précédente, la modification est élémentaire : au lieu de mémoriser le nogood seul, on mémorise avec lui l'ensemble de contraintes C_e qui a permis de le construire et qui forme une justification du nogood. Cette justification n'est pas forcément minimale. En cas de suppression de contraintes, on va oublier tous les nogoods dont l'ensemble C_e associé n'est pas contenu dans l'ensemble de contraintes C courant. On peut aussi simplement invalider temporairement ces nogoods si les mêmes contraintes peuvent apparaître et disparaître fréquemment. Pour résoudre une nouvelle fois le problème de satisfaction, il suffit de relancer l'exécution. Les évaluations expérimentales des différents algorithmes d'apprentissage déjà décrits (Schiex & Verfaillie 1993; Schiex & Verfaillie 1994a; Verfaillie & Schiex 1994b; Verfaillie & Schiex 1995) sur des problèmes aléatoires dynamiques montrent que des gains importants en efficacité sont obtenus pour des problèmes situés à la frontière cohérence-incohérence (les plus difficiles) ou pour des problèmes incohérents. ⋆

Parmi tous les algorithmes que nous avons développés, la version dynamique de l'algorithme DBT-FC mérite une attention particulière. Lors de son exécution, l'algorithme construit et mémorise un ensemble d'*elimination explanations* qui définissent la frontière de l'espace déjà exploré et dont on sait qu'il ne contient pas de solutions. Lors de l'ajout d'une contrainte, toutes ces *elimination explanations* restent actives. On peut donc reprendre la recherche au point où elle a été interrompue lors de la résolution précédente en garantissant que l'espace déjà exploré ne sera pas ré-exploré. On obtient donc naturellement un comportement similaire à celui des *oracles* (van Hentenryck 1990) mais avec une plus grande souplesse sur l'ordonnancement des variables. De plus, le mécanisme de retour arrière « chirurgical » de DBT permet souvent d'aboutir, quand elle existe, à une solution proche de la solution précédente (en nombre

de variables ayant des valeurs modifiées). Les expérimentations confirment l'intuition : les résultats de cette version dynamique de DBT-FC sont excellents sur toute la palette des problèmes dynamiques aléatoires : sous-contraints, sur-contraints ou intermédiaires. En termes d'efficacité seule toutefois, la version dynamique de l'algorithme NR-FC est légèrement supérieure sur les problèmes intermédiaires.

Toutes ces approches traitent le problème de la stabilité de façon purement heuristique. Il est possible, comme (Bellicha 1996), de se focaliser sur ce problème en recherchant explicitement une solution qui minimise, selon un critère qui dépend de l'application, sa distance à la solution précédente. Le problème devient alors un problème d'optimisation dans un réseau de contraintes, problématique que nous allons maintenant aborder.

5 Réseaux de contraintes et optimisation

La nécessité d'ajouter un critère à optimiser au formalisme des réseaux de contraintes est évidente même si elle a été longtemps ignorée sous le prétexte qu'il est inutile de chercher une solution optimale puisqu'il suffit de chercher une solution qui convienne *i.e.* qui respecte l'ensemble des propriétés que l'on souhaite voir satisfaites et qui peuvent être exprimées sous formes de contraintes.

Il existe pourtant une différence de nature fondamentale entre une contrainte qui résulte des propriétés physiques de l'objet modélisé (ressource non partageable en ordonnancement, capacité limitée d'un bus de communication ou d'un entrepôt. . .) et une propriété que l'on souhaite voir satisfaite (dates de livraisons en ordonnancement, absence de cours après 18 heures. . .). La modélisation sous forme de contraintes de ces deux types de propriétés peut mener à un réseau sans solution. On parle de problème sur-contraint. On peut se limiter à résoudre le sous-réseau défini uniquement par les propriétés physiques. S'il est incohérent alors il faut se résigner : le problème est insoluble. Sinon, les solutions trouvées ont de bonnes chances d'être peu satisfaisantes : le problème est généralement sous-contraint.

Dans ces cas, il est envisageable d'utiliser un outil interactif pour rechercher un réseau de contraintes qui soit cohérent et qui contienne les contraintes les plus importantes, et les techniques de CSP dynamiques que nous avons considérées dans la section précédente peuvent alors être utilisées. Mais le nombre des sous-réseaux possibles et la complexité du simple problème de satisfaction rendent cette approche assez lourde et peu fiable. L'approche interactive reste toutefois intéressante lorsque le critère à optimiser est mal défini ou inexprimable formellement pour des raisons de complexité, de confidentialité, *etc*. Dans les autres cas, il est souhaitable de pouvoir exprimer formellement le critère que l'on souhaite optimiser.

Ce sont Rosenfeld, Hummel, & Zucker (1976) qui, à ma connaissance, ont introduit pour la première fois un critère explicite dans la formulation d'un réseau de contraintes. Il s'agissait de la première formulation des réseaux de contraintes floues. Dans ce formalisme, chaque tuple t apparaissant dans une contrainte c est étiqueté par un réel noté $\mu_c(t)$ entre 0 et 1. $\mu_c(t)$ indique à quel point l'utilisation de t est possible dans une solution et est appelé degré d'appartenance du tuple à la contrainte. La qualité d'une instanciation complète est définie par $\min_{c \in C} \mu_c(t_{\downarrow X_c})$ et le problème considéré est celui de la recherche d'une instanciation de qualité maximum. Il s'agit donc d'un problème d'optimisation max-min. L'approche est intéressante car elle est homogène dans le sens où le même type d'objet est utilisé pour exprimer à la fois les contraintes et le critère tout comme en programmation linéaire.

Pour traiter ce type de réseau de contraintes, Rosenfeld, Hummel, & Zucker définissent un algorithme de filtrage par cohérence d'arc étendu. Cet algorithme a été amélioré (en utilisant des principes qui rappellent ceux de l'algorithme α-β) dans (Snow & Freuder 1990). Le formalisme des réseaux de contraintes
⋆ floues ou max-min a été reformulé et étudié par la suite dans de nombreux travaux (Schiex 1992; Martin-
⋆ Clouaire 1992; Ruttkay 1994). Dans (Schiex 1992), j'ai introduit les réseaux de contraintes possibilistes,

dual min-max des réseaux de contraintes floues max-min ainsi qu'un algorithme de filtrage par arc cohérence et un algorithme de type séparation et évaluation en profondeur d'abord (*depth first branch and bound*) exploitant la cohérence d'arc comme minorant. Cet algorithme définit une extension aux réseaux min-max de l'algorithme *really full look ahead*. Le formalisme des CSP flous a été ensuite analysé en détail par H. Fargier (1994) et l'essentiel des propriétés et algorithmes des CSP classiques étendus à ce cadre.

Il existe un lien très fort entre réseaux de contraintes floues et classiques qui explique le fait que l'extension des algorithmes et propriétés classiques se fasse sans trop de difficultés. Si l'on considère un réseau de contraintes floues P, il utilise un nombre de degrés d'appartenance fini k. Pour un degré d'appartenance α donné, on peut construire un réseau de contraintes classiques, noté P_α, ayant la même structure que le réseau flou original et dont chaque contrainte est obtenue à partir d'une contrainte floue c, en sélectionnant les tuples tels que $\mu_c(t) \geq \alpha$. Toute question sur le réseau flou P peut alors se ramener à un nombre limité de questions sur les CSP classiques P_α. Ainsi, pour déterminer une solution optimale de P, on peut chercher une solution de chacun des problèmes P_α. Il suffit de prendre la solution du problème P_α qui maximise α (et qui est cohérent). Une approche dichotomique permet de résoudre le problème en $O(log_2(k))$ appels à un algorithme de satisfaction classique. Cette technique peut être utilisée quasi-systématiquement pour étendre propriétés et algorithmes classiques.

Cependant, le critère exprimé par ce type de réseau est finalement assez peu expressif car il suffit qu'un tuple de degré d'appartenance α soit utilisé dans une instanciation pour que tous les tuples de degrés supérieurs à α soient considérés comme équivalents. On a parlé d'« effet de noyade ». Nous avons proposé dans (Fargier, Lang, & Schiex 1993) une sophistication du critère max-min permettant d'éviter ce com- ⋆
portement. Il consiste à évaluer la qualité d'une solution potentielle non pas sur le degré d'appartenance $\mu_c(t_{\downarrow X_c})$ le plus faible mais sur le vecteur ordonné des degrés d'appartenance $[\mu_c(t_{\downarrow X_c})]_{c \in C}$. L'ensemble de ces vecteurs est ordonné par un ordre lexicographique. Malheureusement, aucun algorithme classique de résolution en dehors des algorithmes BT et FC n'a pu être étendu à ce critère. En particulier, il semble impossible d'étendre la propriété de cohérence d'arc ou les algorithmes de filtrage associés, même les plus simples.

Un autre type de critère a été introduit assez tôt (Shapiro & Haralick 1981). Il s'agit d'un critère additif définissant un problème maintenant désigné, par analogie avec le problème MAX-SAT, sous le nom de MAX-CSP. Il s'agit simplement d'identifier une instanciation complète qui maximise le nombre de contraintes satisfaites. Il faut attendre (Freuder & Wallace 1992) pour que le problème d'optimisation soit explicitement considéré et que des extensions des algorithmes tels que BT ou FC voient le jour pour traiter ce problème. Ce sont encore une fois des algorithmes de type séparation-évaluation en profondeur d'abord utilisant la meilleure solution courante comme majorant de l'optimum et une évaluation (ou minorant) dérivée des mécanismes de filtrage. L'efficacité de ces algorithmes est exécrable et seuls des problèmes de tailles très réduites peuvent être résolus dans des temps raisonnables. Le diagnostic de cette inefficacité est simple : les minorants utilisés sont de qualité trop médiocre. La voie d'amélioration évidente passe par un filtrage accru. Malheureusement, et comme pour (Fargier, Lang, & Schiex 1993), la propriété ⋆
de cohérence d'arc et les algorithmes de filtrage associés ne semblent pas pouvoir s'étendre simplement.

Il y a malgré tout une classe de techniques qui s'étend systématiquement aussi bien aux réseaux de contraintes floues qu'au MAX-CSP : il s'agit des techniques dérivées de la programmation dynamique non sérielle (Bertelé & Brioshi 1972) telles que le regroupement en hyperarbre (Dechter & Pearl 1988) ou la cohérence adaptative (Dechter & Pearl 1989). Ce résultat a été popularisé par Dechter, Dechter, & Pearl (1990) et découle en particulier des travaux sur l'axiomatique de la programmation dynamique de Shafer & Shenoy (1988) plus particulièrement étudiée dans le cadre de l'optimisation discrète dans (Shenoy 1991). Ces techniques définissent une des rares classes polynomiales pour le MAX-CSP, celle des problèmes structurés en arbre ou plus généralement en k-arbre partiel (k borné).

En dehors de ce résultat général, la situation à l'époque était quelque peu confuse : un nombre important d'extensions du cadre CSP classique avaient vu le jour et dans chacune de ces extensions, un sous-ensemble des algorithmes CSP classiques étaient adaptés. Toutes ces adaptations semblaient fortement redondantes. Le formalisme des réseaux de contraintes floues semblait très particulier dans le sens où il était le seul pour lequel l'algorithmique de filtrage par arc cohérence avait pu être étendue.

Cette situation nous a conduit à définir un formalisme général permettant d'unifier ces travaux et, peut-être, de mieux comprendre les raisons de la singularité du formalisme des réseaux de contraintes floues. Ce formalisme devait être un compromis entre généralité et spécificité. La généralité devait être suffisante pour recouvrir l'essentiel des travaux existants, et la spécificité devait permettre de disposer de suffisamment de propriétés pour construire des algorithmes et des théorèmes génériques non triviaux. Il existait bien un cadre général qui recouvrait la majorité des extensions existantes, celui des *Partial CSP* introduit dans (Freuder 1989; Freuder & Wallace 1992). Mais sa généralité extrême rendait difficile toute analyse. C'est pourquoi j'ai décidé de définir un cadre axiomatique mieux adapté à mes besoins : les réseaux de contraintes valuées.

5.1 Réseaux de contraintes valuées

Le formalisme résulte d'une généralisation des travaux existants : réseaux de contraintes floues, lexicographiques, possibilistes, probabilistes, additifs... Dans chacun de ces formalismes une étiquette, généralement numérique, est associée à chaque contrainte (resp. tuple à l'intérieur des contraintes). L'évaluation de la qualité d'une instanciation complète est alors simplement faite en combinant, avec un opérateur dépendant du cadre, les étiquettes de toutes les contraintes violées par l'instanciation (resp. les étiquettes de tous les tuples de contraintes utilisés dans l'instanciation). Le problème habituellement considéré est la recherche d'une instanciation complète optimale (de qualité maximum ou minimum suivant l'ordre utilisé).

La définition des réseaux de contraintes valuées que nous allons donner suppose que ce sont les contraintes qui sont étiquetées. La version plus fine consistant à étiqueter les tuples à l'intérieur des
⋆ contraintes n'est pas, comme nous l'avons rappelé dans (Bistarelli *et al.* 1996; Bistarelli *et al.* 1999), fon-
⋆ damentalement différente et est utilisée par exemple dans (Schiex 2000a). Les étiquettes seront appelées *valuations* et l'on suppose que ces valuations vérifient un certain nombre de propriétés, regroupées dans ce que nous appelons une structure de valuation :

DÉFINITION 4 *Une structure de valuation* $(E, \oplus, \succeq)$ *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*[2] *:* $\forall a \in E, a \oplus \top = \top$.

Cette structure de monoïde commutatif totalement ordonné dont l'opérateur est monotone est assez classique dans la communauté de la représentation de l'incertain (pour $E = [0, 1]$, $\oplus$ est une co-norme triangulaire (Dubois & Prade 1982)). Les éléments de l'ensemble E sont utilisés pour étiqueter les contraintes,

[2] En fait, cette propriété découle des autres axiomes : $\bot$ étant élément neutre on a $(\bot \oplus \top) = \top$; $\bot$ étant minimum, on sait que $\forall a \in E, (a \oplus \top) \succeq (\bot \oplus \top) = \top$; $\top$ étant maximum, $\forall a \in E, (a \oplus \top) \preceq \top$ et donc $(a \oplus \top) = \top$.

l'opérateur $\oplus$ pour combiner les valuations de deux contraintes et l'ordre $\preceq$ pour comparer les valuations entre elles. On définit alors un réseau de contraintes valuées :

DÉFINITION 5 *Un réseau de contraintes valuées (ou VCSP) est défini par un réseau de contraintes classique $\langle X, D, C\rangle$, une structure de valuation $S = (E, \oplus, \succeq)$ et une application φ de C dans E associant une valuation à chaque contrainte du réseau. On le notera comme un quintuplet $\langle X, D, C, S, \varphi\rangle$.*

La notion habituelle de satisfaction est remplacée par une notion graduelle de valuation d'une instanciation, obtenue en combinant les valuations des contraintes violées par l'instanciation.

DÉFINITION 6 *Étant donné un réseau de contraintes valuées P et un tuple t défini sur $V \subset X$, la valuation de t dans P est définie par :*

$$v_P(t) = \bigoplus_{\substack{c \in C \\ t \text{ viole } c}} \varphi(c)$$

Les axiomes d'une structure de valuation sont assez naturels et se justifient simplement. La commutativité et l'associativité de $\oplus$ garantissent que la valuation d'une instanciation ne dépend pas de l'ordre dans lequel les combinaisons sont effectuées et la monotonie de $\oplus$ garantit que la violation d'une contrainte supplémentaire ne peut pas diminuer la valuation d'une instanciation. Elle montre également que la valuation d'une instanciation est un minorant directement utilisable dans la construction d'algorithme de type séparation-évaluation. L'élément minimum $\bot$ permet d'exprimer la satisfaction totale et l'élément $\top$ la nature impérative de certaines contraintes et l'insatisfaction totale.

On définit une notion de relaxation qui permet de faire un lien entre réseaux de contraintes valuées et réseaux de contraintes classiques.

DÉFINITION 7 *Étant donné un réseau de contraintes valuées $P = \langle X, D, C, S, \varphi\rangle$, on dira qu'un réseau de contraintes classique $P' = \langle X, D, C'\rangle$ est une relaxation de P si et seulement si $C' \subset C$.*

Une relaxation rejette donc un sous-ensemble des contraintes du problème. Cette notion est le pendant de la notion de sous-base en logique propositionnelle non-monotone (ayant choisi de focaliser cette synthèse sur les réseaux de contraintes, je ne détaillerai pas les quelques travaux auxquels j'ai participé dans ce domaine (Cayrol, Lagasquie, & Schiex 1998; Dupin de Saint Cyr, Lang, & Schiex 1994b; Dupin de Saint Cyr, Lang, & Schiex 1994a)). ⋆

On peut définir la valuation d'une relaxation par :

DÉFINITION 8 *Étant donné un réseau de contraintes valuées $P = \langle X, D, C, S, \varphi\rangle$ et une relaxation $\langle X, D, C'\rangle$ de P, la valuation de cette relaxation est définie par :*

$$V_P(\langle X, D, C'\rangle) = \bigoplus_{c \in C \setminus C'} \varphi(c)$$

La valuation du sommet du treillis des relaxations, le CSP $\langle X, D, C\rangle$, est évidemment $\bot$. Les valuations des autres relaxations peuvent être interprétées, dans l'esprit des *Partial CSP* de (Freuder 1989), comme une « distance » à ce problème idéal (mais éventuellement incohérent).

Le problème central des réseaux de contraintes valuées, extension du problème de satisfaction, est un problème d'optimisation : produire une instanciation complète des variables de valuation minimum dite instanciation optimale. La propriété suivante montre que ce problème d'optimisation dans l'espace des instanciations complètes peut se ramener à un problème d'optimisation dans l'espace des relaxations cohérentes.

PROPRIÉTÉ 1 *La valuation d'une relaxation cohérente de valuation minimum est égale à la valuation d'une instanciation optimale.*

La généralité du formalisme est effectivement suffisante pour capturer l'essentiel des propositions existantes comme le montre le tableau 1[3].

Classe	E	$\oplus$	$\bot$	$\top$	$\succ$
Classiques	$\{vrai, faux\}$	$\wedge = \max$	*vrai*	*faux*	*faux* $\succ$ *vrai*
Possibilistes	$[0,1]$	$\max$	0	1	$>$
Max-CSP	$\overline{\mathbb{N}}$	$+$	0	$+\infty$	$>$
Probabilistes	$[0,1]$	$x+y-xy$	0	1	$>$
Lexico.	$[0,1]^* \cup \{\top\}$	$\cup$	$\varnothing$	$\top$	lex.

TAB. 1 – Différentes classes de réseaux de contraintes valuées

Un des premiers buts des VCSP est de permettre la comparaison de ces différentes classes. Dans ce but, nous avons introduit une notion de *raffinement polynomial*, inspirée de la notion classique de transformation polynomiale en complexité. Cette notion, proche de la notion de *metric reduction* de (Krentel 1988; ⋆ Krentel 1992), est décrite dans (Schiex 1994; Schiex, Fargier, & Verfaillie 1995) et a permis de mettre en évidence une partition entre CSP classiques et flous d'un coté et Max-CSP, CSP lexicographiques et probabilistes d'autre part. Les premiers vérifient une propriété d'idempotence de $\oplus$, les seconds une propriété de monotonie stricte. On montre que ces deux propriétés sont incompatibles si E contient plus de deux éléments.

Les raffinements polynomiaux exhibés entre classes sont suffisamment simples pour être d'intérêt pratique et la hiérarchie de classes construite au moyen de cette notion permet d'identifier le problème MAX-CSP comme une cible d'intérêt particulier du fait de sa simplicité de formulation et de sa généralité. Il est clair également que le développement d'algorithmes spécifiques pour les CSP probabilistes ou lexicographiques a peu d'intérêt car ces problèmes se traduisent simplement en MAX-CSP. Enfin, on identifie clairement la raison de la spécificité des CSP possibilistes/flous : l'idempotence de $\oplus$, cruciale pour l'applicabilité des algorithmes de filtrage.

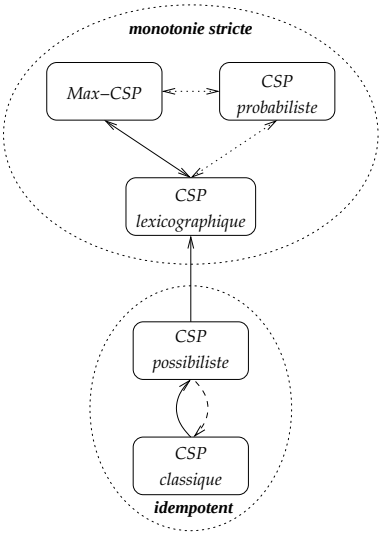

FIG. 1 – Raffinements polynomiaux

Un autre intérêt des réseaux de contraintes valuées est de permettre la définition d'algorithmes génériques, s'appliquant à toutes les structures. Ces algorithmes (ainsi que d'autres développés par la suite) font maintenant partie de la librairie CommonLisp LVCSP, principalement développée par M. Lemaître et L. Lobjois (CERT-ONERA) et dédiée à la résolution des réseaux de contraintes valuées.

Simultanément à la publication du formalisme des réseaux de contraintes valuées, Bistarelli, Montanari, & Rossi (1995) ont défini une extension similaire utilisant un demi-anneau au lieu d'un demi-groupe, et s'appuyant donc sur deux opérateurs. Le premier est un opérateur de combinaison, le second

[3] $[0,1]^*$ dénote l'ensemble des multi-ensembles sur $[0,1]$ et lex. l'ordre lexicographique sur les multi-ensembles

est un opérateur de maximisation. L'utilisation d'une structure de demi-anneau est traditionnelle en programmation dynamique et a été, en particulier, utilisée pour définir les algèbres de chemins dans (Minoux 1976). La différence essentielle entre les réseaux de contraintes valuées et les *semiring-CSP* tient dans la possibilité qu'ont ces derniers de capturer un ordre partiel (un treillis) sur l'équivalent de nos valuations. Nous avons montré dans (Bistarelli *et al.* 1996; Bistarelli *et al.* 1999) qu'une hypothèse d'ordre total suffit ⋆ à confondre les deux formalismes.

5.2 Recherche d'une solution optimale

Tous les algorithmes classiques d'optimisation combinatoire, des méthodes heuristiques de recherche locale aux méthodes complètes de type programmation dynamique ou recherche arborescente (séparation-évaluation) sont *a priori* utilisables pour attaquer le problème de recherche d'une instanciation optimale. Nous ne nous sommes intéressés qu'aux méthodes avec garantie d'optimalité. La programmation dynamique reste, même dans les CSP classiques, d'usage assez rare car elle n'est raisonnablement applicable que sur des réseaux bien structurés. Nous nous sommes donc essentiellement intéressés par la suite à la construction d'algorithmes de recherche arborescente de type séparation-évaluation en utilisant une stratégie en profondeur d'abord. Par rapport à une classique stratégie en meilleur d'abord, elle a l'avantage de conserver une complexité spatiale raisonnable. La mécanique de l'algorithme est très proche de celle de l'algorithme BT.

Nous rappelons brièvement son fonctionnement (cf. algorithme 3) : l'algorithme recherche une solution dont la valuation est strictement inférieure à ub (*upper bound*), initialement fixée à $\top$. Cette valuation ub est remise à jour à chaque fois qu'une nouvelle solution est trouvée. À chaque nœud de l'arbre de recherche, l'instanciation courante t est « évaluée ». Cette évaluation consiste à calculer un minorant de la valuation de la meilleure solution qui contient t. On note lb (*lower bound*) cette fonction d'évaluation. Si $lb(t) \succeq ub$, on cesse l'extension de l'instanciation courante car elle ne peut mener à une solution de meilleure qualité que la meilleure solution connue. Étant donné un réseau de contraintes valuées et une fonction lb de calcul de minorant, l'appel DFBB($\varnothing, \top$) retourne la valuation d'une solution optimale du réseau. L'algorithme suppose que lb est exacte sur les instanciations complètes.

Algorithme 3: Séparation-évaluation

```
Fonction DFBB( t : instanciation, ub : valuation ) : valuation
    v ← lb(t);
    si v ≺ ub alors
        si (|t| = n) alors retourner v;
        soit i une variable non instanciée;
        pour chaque a ∈ d_i faire
            ub ← min(ub, DFBB( t ∪ {(i, a)}, ub ));
        retourner ub;
    retourner ⊤;
```

L'efficacité de cet algorithme, comme celle de BT, dépend :

– de l'ordre utilisé pour instancier les variables : il faut limiter la croissance de l'arbre de recherche en utilisant les domaines de petite taille en premier et en faisant grimper le minorant le plus vite possible (*first fail principle*).

– de l'ordre sur les valeurs : il affecte l'évolution de ub. Idéalement, il serait intéressant de commencer la recherche par l'obtention d'une solution optimale ou d'une bonne solution (qui peut éventuellement être obtenue par recherche locale). Cependant, la preuve d'optimalité prenant généralement sensiblement plus de temps que la découverte d'une solution optimale, cet ordre n'est pas crucial.
– de la qualité du minorant lb : ce terme de qualité regroupe la facilité de calcul du minorant et son pouvoir de coupure. C'est le pendant du filtrage systématique effectué dans les algorithmes classiques. C'est ce dernier point qui, de loin, est le plus important.

C'est donc sur la définition de minorants de bonne qualité qu'a porté l'essentiel de mes efforts. Le premier minorant qui vient à l'esprit est fourni par la valuation $v_P(t)$ de l'instanciation courante t. Le minorant ne prend en compte que les contraintes déjà violées par l'instanciation courante et l'algorithme correspondant est malheureusement d'une inefficacité redoutable.

En s'inspirant des extensions de l'algorithme FC proposées dans (Freuder & Wallace 1992) sur les Max-CSP binaires, on peut construire un minorant exploitant un principe de type *forward checking* pour les réseaux de contraintes valuées binaires. On associe à chaque valeur a de chaque variable i non instanciée une valuation[4], notée fc_{ia}, qui contiendra la combinaison des valuations des contraintes qui relient i à une variable instanciée et dont la valeur est incompatible avec (i, a) (voir algorithme 4). Si cette valeur (i, a) est utilisée dans le futur, ces contraintes seront nécessairement violées. Chaque variable devant prendre une valeur, et en prenant en compte les contraintes déjà violées par t, on obtient un minorant égal à :

$$lb_{fc}(t) = v_P(t) \oplus (\bigoplus_{i \text{ future}} \min_{a \in d_i} fc_{ia})$$

Algorithme 4: Mise à jour des fc_{ia}

Fonction `FC-propagation-val(`i : variable, a : valeur`)`
  pour chaque j *non instanciée tq.* $c_{ij} \in C$ **faire**
    pour chaque $b \in d_j$ *tq.* $((i, a), (j, b)) \notin c_{ij}$ **faire**
      $fc_{jb} \leftarrow fc_{jb} \oplus \varphi(c_{ij})$;

Pour une valeur b de la variable future j, on peut construire un minorant de la valuation optimale d'un tuple qui étend t et qui utilise (j, b) :

$$lb^{jb}_{fc}(t) = v_P(t) \oplus (\bigoplus_{\substack{i \text{ future} \\ i \neq j}} \min_{a \in d_i} fc_{ia}) \oplus fc_{jb}$$

Si ce minorant dépasse ub, on peut, dans l'esprit de l'algorithme FC, effacer la valeur (j, b). Ces effacements n'affectent pas le minorant global $lb_{fc}(t)$ et ne changent donc pas la taille de l'arbre exploré. Comme dans l'algorithme FC, il est nécessaire de mettre en place une mécanique de sauvegarde/restauration du contexte (domaines et compteurs fc_{ia}). Le minorant $lb_{fc}(t)$ exploite les contraintes violées par l'instanciation courante mais aussi celles qui relient les variables instanciées à des variables futures. L'efficacité s'améliore sensiblement, mais reste très décevante. La voie de l'amélioration passe clairement par la prise en compte des contraintes entre variables futures.

C'est une des propriétés de l'algorithme proposé dans (Wallace 1994) pour les Max-CSP binaires. On suppose qu'un ordre fixe d'instanciation des variables est utilisé. Le minorant utilise des compteurs notés

[4] Il faut noter que les compteurs fc_{ia} ne définissent pas des contraintes valuées unaires que l'on ajoute au problème car, dès lors que $\oplus$ est strictement monotone, le problème ne serait plus équivalent au problème d'origine. Il s'agit de valuations externes au problème.

dac_{ia} et appelés compteurs d'arc cohérence orientée (*directed arc consistency counts* ou *DAC*). Le compteur dac_{ia} est égal à la combinaison des valuations des contraintes c_{ij} qui relient i à une variable j située après i dans l'ordre utilisé et telles qu'aucune valeur de d_j n'est un support de (i, a). Ces compteurs sont calculés en prétraitement.. Le même raisonnement que précédemment permet d'aboutir à un minorant égal à :

$$lb_{dac}(t) = lb_{fc}(t) \oplus (\bigoplus_{i \text{ future}} \min_{a \in d_i} dac_{ia})$$

Il est possible, comme pour lb_{fc}, de définir un minorant lb^{jb}_{dac} associé à chaque valeur et d'effacer les valeurs dont le minorant associé dépasse ub. Les dac_{ia} étant calculés une fois pour toute, cela n'affecte pas le minorant global $lb_{dac}(t)$ et ne change pas la taille de l'arbre exploré. Le minorant $lb_{dac}(t)$ est toujours supérieur au minorant $lb_{fc}(t)$ mais avec un gain des performances qui est d'autant plus limité que l'algorithme impose l'utilisation d'un ordre statique. Cet algorithme a été ensuite amélioré par Larrosa & Meseguer (1996) qui ont noté que les compteurs fc_{ia} et dac_{ia} pouvaient être combinés à l'intérieur du min :

$$lb'_{dac}(t) = v_P(t) \oplus (\bigoplus_{i \text{ future}} \min_{a \in d_i} (fc_{ia} \oplus dac_{ia}))$$

L'algorithme reste contraint à utiliser un ordre statique des variables.

Une nette amélioration du minorant a été obtenue dans l'algorithme *russian doll search* ou RDS (Verfaillie, Lemaître, & Schiex 1996). Cet algorithme, comme les précédents, nécessite l'utilisation d'un ordre statique des variables. Le minorant calculé à la profondeur i résulte de la combinaison du minorant $lb_{fc}(t)$ et de la valuation d'une solution optimale du sous-réseau formé par les variables futures et les contraintes qui les relient. Pour calculer cette valuation, on utilise l'algorithme RDS, récursivement. L'algorithme a un parfum de programmation dynamique et semble effectivement profiter des bonnes propriétés structurelles du réseau traité. Il a été utilisé pour résoudre optimalement, et dans des temps raisonnables (inférieurs à une heure), la quasi totalité des problèmes de programmation journalière du satellite SPOT5, sans contrainte mémoire, fournis par le CNES. Il a aussi permis, après application d'un algorithme de décomposition du graphe de contraintes, de prouver pour la première fois, l'optimalité de la meilleure solution connue[5] de l'instance 6 du problème d'affectation de fréquences du CELAR (de Givry, Verfaillie, & Schiex 1997; Cabon *et al.* 1999). L'algorithme est disponible dans la librairie LVCSP. ⋆ ⋆

Dans (Larrosa, Meseguer, & Schiex 1999; Larrosa *et al.* 1998), nous avons sophistiqué l'algorithme de (Wallace 1994; Larrosa & Meseguer 1996) en faisant, entre autres, disparaître la restriction à un ordre statique des variables. Les compteurs dac ne sont alors plus définis par l'ordre d'instanciation des variables : chaque contrainte c_{ij} peut être orientée vers une de ses deux variables i ou j et les éventuelles absences de support sont comptabilisées sur la variable indiquée par cette orientation. Cette liberté supplémentaire peut être exploitée pour améliorer la force du minorant en optimisant rapidement, et après chaque instanciation, les directions qui le maximisent. Cette optimisation est réalisée par un algorithme glouton. L'optimisation complète de ce minorant ne semble pas intéressante car elle définit un problème NP-difficile (Schiex 1998). D'autres améliorations ont été introduites, en particulier la propagation des effacements de valeurs, comme dans l'algorithme MAC, en utilisant une mécanique de type AC6 (Bessière 1994). Un effacement peut permettre d'augmenter un compteur dac_{ia} qui peut permettre un nouvel effacement, *etc.* Les algorithmes PFC-RDAC et PFC-MRDAC ainsi définis offrent, par rapport à la version précédente de (Larrosa & Meseguer 1996), des gains en performance de plusieurs ordres de grandeurs aussi bien sur des problèmes aléatoires que réels. Il s'agit en l'occurrence des mêmes problèmes d'affectation de fréquence que ceux résolus avec RDS. Les algorithmes PFC-{M}RDAC sont aussi beaucoup ⋆ ⋆

[5] Solution construite par A. Kolen, univ. de Maastricht.

plus efficaces que RDS sur ces problèmes. Ces expérimentations montrent combien il est dangereux de se limiter à des tests sur des problèmes aléatoires : le meilleur algorithme sur ces problèmes est PFC-RDAC, plus simple. Sur les problèmes d'affectation de fréquences, plus complexes, utilisant des poids qui ne sont pas tous égaux à 1 et qui étiquettent les tuples et non les contraintes, PFC-MRDAC tire son épingle du jeu.

Simultanément, Affane & Bennaceur (1998) ont défini un minorant paramétrique pour le MAX-
⋆ CSP. Nous avons montré (Affane, Bennaceur, & Schiex 1999) que ce minorant (s'appuyant sur des *weighted arc consistency counts*) est une généralisation de notre minorant basé sur les *reversible DAC*. Malheureusement, les degrés de libertés supplémentaires offerts par ce minorant ne sont pas évidents à exploiter et les algorithmes PFC-{M}RDAC restent, à notre connaissance, les meilleurs algorithmes de résolution du MAX-CSP dans le cas binaire. Il faut noter que l'algorithme RDS, bien qu'il soit nettement dominé sur nos tests par les algorithmes PFC-{M}RDAC, reste intéressant car il fournit un minorant non trivial sur les problèmes SPOT5 ce qui n'est pas le cas des autres algorithmes.

Dans leurs travaux, Bistarelli, Montanari, & Rossi (1995) ne se sont pas intéressés à l'extension des algorithmes d'exploration arborescente des réseaux de contraintes classiques mais à l'applicabilité de procédures de type programmation dynamique non sérielle (voir aussi (Dechter, Dechter, & Pearl 1990; Shafer & Shenoy 1988)). Un second point concernait l'extension des algorithmes de filtrage que nous allons maintenant aborder.

5.3 Filtrage par cohérence d'arc

Le filtrage par cohérence d'arc est un outil fondamental des CSP classiques et il est assez naturel de vouloir l'étendre aux réseaux de contraintes valuées. Il devrait permettre la construction d'un minorant plus fort que ceux présentés dans la section précédente et qui sont, pour l'essentiel, dérivés de formes affaiblies de filtrage par cohérence d'arc. Comme nous l'indiquions, la qualité d'un minorant résulte d'un compromis entre la complexité de son calcul et son pouvoir de coupure. Il nous semble clair, à l'heure actuelle, que dans ce compromis temps/force, il serait judicieux de passer plus de temps pour construire un minorant plus fort.

Bistarelli, Montanari, & Rossi (1995) ont proposé une extension systématique de l'algorithmique de filtrage par k-cohérence aux *semiring-CSP*. Cette extension s'appuie simplement sur une généralisation des opérations de jointure et de projection. Malheureusement, l'algorithme résultant peut ne pas se terminer ou fournir un résultat non équivalent au problème d'origine. L'idempotence de l'opérateur de combinaison est une condition suffisante pour récupérer terminaison et équivalence. Il faut bien analyser l'importance de cette restriction : jointe avec une hypothèse d'ordre total, elle réduit les *semiring-CSP* aux réseaux de contraintes floues pour lesquels le filtrage par cohérence locale a déjà été défini (Rosenfeld, Hummel, & Zucker 1976; Fargier 1994). La contribution essentielle concerne donc les cas où l'ordre est partiel et l'opérateur de combinaison idempotent : c'est le cas pour le treillis des parties d'un ensemble avec des opérateurs $\cup$ et $\cap$ ou pour le produit de plusieurs mesures floues.

Le cas des opérateurs strictement monotones, comme l'addition utilisée dans les MAX-CSP, reste donc ouvert. Nous avons présenté précédemment la notion de relaxation d'un réseau de contraintes valuées, permettant d'établir un lien avec les réseaux de contraintes classiques. Étant donné un filtrage par cohérence locale quelconque défini dans les réseaux de contraintes classiques, nous avons montré
⋆ dans (Schiex, Fargier, & Verfaillie 1995) qu'il était toujours possible d'en déduire un minorant pour les réseaux de contraintes valuées. L'idée est simple : la valuation d'une solution optimale est égale à la valuation d'une relaxation optimale cohérente (voir propriété 1) et l'ensemble des relaxations cohérentes est

par définition inclus dans l'ensemble des relaxations satisfaisant la propriété de cohérence locale. On peut donc conclure que :

PROPRIÉTÉ 2 *Étant donné une propriété de cohérence locale L, un minorant de la valuation d'une solution optimale est défini par la valuation d'une relaxation optimale parmi celles vérifiant la propriété L.*

De tels minorants sont toujours définis, pour toutes les structures de valuation. Cette classe de minorants vérifie de plus deux propriétés intéressantes :

- elle est cohérente avec le comportement de L dans le cas classique. En effet, un réseau de contraintes classique vu comme un réseau valué n'a qu'une relaxation de valuation inférieure à $\top$: lui-même.
- une propriété de cohérence locale plus forte définit un minorant plus fort (mais plus cher).

Son utilisation avec des propriétés de cohérence locale correspondant par exemple au *forward checking* permet de reconstruire pour les réseaux de contraintes valuées l'extension de l'algorithme FC définie précédemment (algorithme 4). Nous attendions beaucoup du minorant induit par le filtrage par cohérence d'arc. Malheureusement, nous avons montré que le calcul de ce minorant définit un problème NP-difficile lorsque l'opérateur $\oplus$ est strictement monotone (Schiex, Fargier, & Verfaillie 1995). ⋆

Dans (Schiex 2000b; Schiex 2000a), j'ai défini une propriété d'arc cohérence et un algorithme de ⋆ filtrage associé qui satisfont pratiquement tous les critères d'une propriété de cohérence locale classique : le filtrage fournit un problème équivalent, il s'effectue en temps polynomial et retourne un problème qui satisfait la propriété de cohérence d'arc associée. L'algorithme ajoute des contraintes unaires au réseau mais modifie le reste du réseau pour conserver l'équivalence entre les problèmes. Il fonctionne pour une large sous-classe des réseaux de contraintes valuées qui utilisent un opérateur idempotent ou strictement monotone. Il n'est pas limité aux contraintes binaires et s'applique en particulier aux MAX-CSP. Seule différence avec l'arc cohérence classique : lorsque l'opérateur est strictement monotone, il peut y avoir plusieurs fermetures arc cohérentes distinctes.

Chaque fermeture arc cohérente définit un minorant qui peut prendre une valeur deux fois plus importante que celle prise par les minorants s'appuyant sur les *reversible DAC* (Larrosa *et al.* 1998) ou ⋆ sur les *weighted AC counts* (Affane & Bennaceur 1998). Il reste maintenant à injecter ce minorant dans un algorithme de type séparation-évaluation et à évaluer son intérêt. Indépendamment du résultat, ce travail permettra sans doute l'extension d'autres algorithmes de filtrage aux réseaux de contraintes valuées, permettant d'augmenter encore la force des minorants disponibles.

6 Décision dans l'incertain

Les problèmes de décision dans l'incertain ont suscité le développement de nombreux formalismes, dans différentes disciplines. En excluant les approches purement logiques, on citera par exemple les processus de décision de Markov (Puterman 1994), les diagrammes d'influence (Howard & Matheson 1984)...

Le formalisme des réseaux de contraintes probabiliste (Fargier & Lang 1993) permet déjà d'exprimer des problèmes de décision dans l'incertain. Dans ce formalisme, l'existence de chaque contrainte dans le problème réel est incertaine, liée à un aléa. Ces aléas sont supposés indépendants et chaque contrainte c est étiquetée avec sa probabilité d'existence. Une solution optimale est une instanciation qui maximise sa probabilité d'être solution du problème final (qui est un problème classique). Ce formalisme constitue déjà une extension stochastique non triviale des réseaux de contraintes mais il ne permet pas de capturer des dépendances complexes entre aléas et problème de décision, ou simplement des dépendances entre aléas (certaines combinaisons d'aléas sont impossibles).

Pour remédier à ces deux points, il est tentant d'utiliser le même formalisme des réseaux de contraintes pour modéliser d'une part le problème de décision et d'autre part les dépendances entre aléas. Les deux réseaux de contraintes ainsi définis sont de natures différentes. Les variables du réseau représentant les aléas ou états du monde, prennent des valeurs incontrôlables : elles dépendent de l'occurrence d'évènements incertains (il pleut, le ciel est couvert) mais sur lesquels on dispose d'une connaissance qualitative (si le ciel n'est pas couvert, il ne pleut pas). Elles seront appelées variables contingentes ou paramètres. Les variables du réseau représentant le problème de décision sont contrôlables et seront appelées variables de décision (dois-je prendre un parapluie ?). Enfin, des contraintes connectent les deux réseaux, capturant les dépendances entre aléas et décision (quand il pleut, il faut prendre un parapluie).

⋆ On définit ainsi la notion de réseau de contraintes mixte (Fargier, Lang, & Schiex 1996; Fargier *et al.* 1997) :

DÉFINITION 9 *Un réseau de contraintes mixte est un sextuplet $\langle \Lambda, W, X, D, C, K \rangle$ formé par :*
- *un ensemble $\Lambda = \{\lambda_1, \ldots, \lambda_p\}$ de paramètres, ou variables contingentes,*
- *un ensemble de domaines $W = \{w_i\}$. Chaque variable contingente λ_i prend ses valeurs dans le domaine w_i associé,*
- *un ensemble $X = \{x_1, \ldots, x_n\}$ de variables de décisions.*
- *un ensemble de domaines $D = \{d_i\}$. Chaque variable de décision x_i prend ses valeurs dans le domaine d_i associé,*
- *un ensemble de contraintes C. Chaque contrainte $c \in C$ implique un ensemble de variables X_c tel que $X_c \cap X \neq \varnothing$,*
- *un ensemble de contraintes K. Chaque contrainte $k \in K$ implique un ensemble de variables Λ_c tel que $\Lambda_c \subset \Lambda$.*

Les contraintes sont définies de la même façon que dans les réseaux de contraintes classiques, par des relations.

Une instanciation ω des variables de Λ sera appelée un *monde*, une instanciation d des variables de X sera appelée une décision. L'ensemble des solutions du réseau de contraintes $\langle \Lambda, W, K \rangle$ représente l'ensemble des mondes possibles. Une fois les valeurs des paramètres instanciées par un monde possible donné, une décision qui satisfait les contraintes de C est une décision compatible avec ce monde, on dit que la décision couvre ce monde.

Selon qu'il est possible ou non d'observer l'état du monde avant de prendre une décision, on s'intéressera à deux problèmes différents :
- si le monde est observable, on cherchera à construire une solution conditionnelle *i.e.*, une fonction associant une décision à chaque monde possible. Si une décision conditionnelle qui couvre tous les mondes possibles existe, on dit que le réseau de contraintes mixte est cohérent.
- si le monde n'est pas observable, on cherchera à construire une décision universelle *i.e.*, une décision couvrant tous les mondes possibles qui peuvent l'être. Si une telle décision existe, on dit que le réseau de contraintes mixte est fortement cohérent.

⋆ D'autres cas sont possibles. Nous nous sommes intéressés dans (Fargier, Lang, & Schiex 1996) à la complexité du problème de cohérence et de forte cohérence d'un réseau de contraintes mixte. Le problème de cohérence est Π_2^p-complet même sous des restrictions fortes (par exemple $K = \varnothing$). Celui de forte cohérence est dans Σ_2^p mais la restriction $K = \varnothing$ ramène sa complexité à la classe NP. Nous avons également proposé un algorithme *anytime* de construction d'une décision conditionnelle utilisant la représentation d'ensemble de solutions sous forme de produit cartésien (introduite dans (Hubbe & Freuder 1992)). L'algorithme reste limité à des instances de très petite taille.

La connaissance du monde représentée par K est purement qualitative dans ce modèle : un monde est

ou n'est pas possible. On peut remplacer cette connaissance qualitative par une connaissance quantitative en substituant à K une distribution de probabilité sur les mondes (Fargier *et al.* 1995; Fargier *et al.* 1997). ⋆ Le problème de la représentation d'une telle distribution dans un cas général n'a pas été abordé mais des outils tels que les réseaux bayésiens pourraient être utilisés. Dans le cas où il y a indépendance mutuelle entre les paramètres, on pourra utiliser une distribution de probabilité par paramètre.

Comme dans le cas mixte, on fait la différence entre décision universelle et conditionnelle mais la qualité de celles-ci peut maintenant être quantifiée en terme de probabilité. Nous avons proposé dans (Fargier ⋆ *et al.* 1995) un algorithme de type séparation-évaluation utilisant un minorant dérivé du *forward checking* qui permet de calculer une décision universelle optimale. Cet algorithme suppose que l'on sait efficacement calculer la probabilité d'un ensemble de mondes, ce qui est le cas par exemple sous l'hypothèse d'indépendance mutuelle des paramètres. Un autre algorithme, utilisant la représentation d'ensemble de solutions sous forme de produit cartésien proposée par Hubbe & Freuder (1992), a été défini pour la construction de décisions conditionnelles[6].

Ces travaux restent relativement marginaux dans la communauté CSP. La majorité des travaux similaires utilisent la logique propositionnelle et exploitent les algorithmes de production d'impliquants et impliqués premiers ou de résolution de formules booléennes quantifiées pour résoudre les problèmes associés. Boutilier (1994) fait lui aussi apparaître une dichotomie entre variables contrôlables et incontrôlables mais ne s'intéresse qu'à des décisions universelles, avec un critère min-max (minimisation des pires conséquences). Plus récemment, Fargier, Lang, & Marquis (1999) ont introduit une troisième classe de variables : les variables de conséquences permettant de décrire l'état du monde après que la décision choisie ait été appliquée. Il reste un travail important à faire pour traiter les nombreuses facettes des problèmes de décision : observabilité parfois partielle des états du monde, décisions séquentielles, possibilités d'actions d'observations, coûts des actions...

[6] Cet algorithme est en fait antérieur à l'algorithme similaire sur les réseaux de contraintes mixte.

References

[1998] Affane, M. S., et Bennaceur, H. 1998. A weighted arc consistency technique for Max-CSP. Dans *Proc. of the 13th ECAI*, 209–213. 20, 21

[1999] Affane, M. S. ; Bennaceur, H. ; et Schiex, T. 1999. Comparaison de deux minorants paramétriques pour Max-CSP. Dans *Actes de JNPC'99*, 95–102. 20

[1996] Bellicha, A. 1996. *Flexibilité dans les problèmes de satisfaction de contraintes*. Thèse de doctorat, Université de Montpellier II, Sciences et techniques du Languedoc, Montpellier, France. 12

[1994] Berlandier, P., et Neveu, B. 1994. Arc-consistency for dynamic constraint problems : A RMS-free approach. Dans *Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*. 10

[1972] Bertelé, U., et Brioshi, F. 1972. *Nonserial Dynamic Programming*. Academic Press. 13

[1996] Bessière, C., et Régin, J.-C. 1996. Mac and combined heuristics : Two reasons to forsake fc (and cbj ?) on hard problems. Dans *Proc. of the Second International Conference on Principles and Practice of Constraint Programming*, 61–75. 10

[1995] Bessière, C. ; Freuder, E. ; et Régin, J. 1995. Using inference to reduce arc-consistency computation. Dans *Proc. of the 14th IJCAI*. 6, 7

[1991] Bessière, C. 1991. Arc-consistency in dynamic constraint satisfaction problems. Dans *Proc. of AAAI'91*, 221–226. 10

[1994] Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artificial Intelligence* 65 :179–190. 19

[1996] Bistarelli, S. ; Fargier, H. ; Montanari, U. ; Rossi, F. ; Schiex, T. ; et Verfaillie, G. 1996. Semiring-based csps and valued csps : Basic properties and comparison. Dans Jampel, M. ; Freuder, E. ; et Maher, M., eds., *Over-Constrained Systems*, numéro 1106 dans LNCS. Springer Verlag. 111–150. 14, 17

[1999] Bistarelli, S. ; Fargier, H. ; Montanari, U. ; Rossi, F. ; Schiex, T. ; et Verfaillie, G. 1999. Semiring-based CSPs and valued CSPs : Frameworks, properties and comparison. *Constraints* 4 :199–240. 14, 17

[1995] Bistarelli, S. ; Montanari, U. ; et Rossi, F. 1995. Constraint solving over semirings. Dans *Proc. of the 14th IJCAI*. 16, 20

[1994] Boutilier, C. 1994. Towards alogic for qualitative decision theory. Dans *Proc. of KR'94*, 75–86. 23

[1981] Bruynooghe, M. 1981. Solving combinatorial search problems by intelligent backtracking. *Information Processing Letters* 12(1) :36–39. 9

[1999] Cabon, B. ; de Givry, S. ; Lobjois, L. ; Schiex, T. ; et Warners, J. 1999. Radio link frequency assignment. *Constraints Journal* 4 :79–89. 19

[1998] Cayrol, C. ; Lagasquie, M.-C. ; et Schiex, T. 1998. Non-monotonic reasoning : from complexity to algorithms. *Annals of artificial intelligence and mathematics* 22(3-4) :207–236. 15

[1997] de Givry, S. ; Verfaillie, G. ; et Schiex, T. 1997. Bounding the optimum of constraint optimization problems. Dans *Proc. of CP'97*. 19

[1995] Debruyne, R. 1995. Les algorithmes d'arc-consistance dynamiques. *Revue d'intelligence artificielle* 9(3) :239–267. 10

[1988] Dechter, R., et Dechter, A. 1988. Belief maintenance in dynamic constraint networks. Dans *Proc. of AAAI'88*, 37–42. 10

[1988] Dechter, R., et Pearl, J. 1988. Tree clustering schemes for constraint processing. Dans *Proc. of AAAI'88*, 150–154. 13

[1989] Dechter, R., et Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 38 :353–366. 13

[1990] Dechter, R. ; Dechter, A. ; et Pearl, J. 1990. Optimization in constraint networks. Dans Oliver, R., et Smith, J., eds., *Influence Diagrams, Belief Nets and Decision Analysis*. John Wiley & Sons Ltd. chapitre 18, 411–425. 13, 20

[1986] Dechter, R. 1986. Learning while searching in constraint satisfaction problems. Dans *Proc. of AAAI'86*, 178–183. 7

[1990] Dechter, R. 1990. Enhancement schemes for constraint processing : Backjumping, learning and cutset decomposition. *Artificial Intelligence* 41(3) :273–312. 7, 8

[1979] Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence* 12 :231–272. 10

[1982] Dubois, D., et Prade, H. 1982. A class of fuzzy measures based on triangular norms. a general framework for the combination of uncertain information. *Int. Journal of Intelligent Systems* 8(1) :43–61. 14

[1994a] Dupin de Saint Cyr, F. ; Lang, J. ; et Schiex, T. 1994a. Gestion de l'inconsistance dans les bases de connaissances : une approche syntaxique basée sur la logique des pénalités. Dans *Actes de RFIA'94*, 507–518. 15

[1994b] Dupin de Saint Cyr, F. ; Lang, J. ; et Schiex, T. 1994b. Penalty logic and its link with dempster-shafer theory. Dans *Proc. of the 10th International Conference on Uncertainty in Artificial Intelligence*. 15

[1993] Fargier, H., et Lang, J. 1993. Uncertainty in constraint satisfaction problems : a probabilistic approach. Dans *Proc. of ECSQARU '93, LNCS 747*, 97–104. 21

[1995] Fargier, H. ; Lang, J. ; Martin-Clouaire, R. ; et Schiex, T. 1995. A constraint satisfaction framework for decision under uncertainty. Dans *Proc. of the 11th Int. Conf. on Uncertainty in Artificial Intelligence*. 23

[1997] Fargier, H. ; Lang, J. ; Martin-Clouaire, R. ; et Schiex, T. 1997. Traitement de problèmes de décision sous incertitude par des problèmes de satisfaction de contraintes. *Revue d'Intelligence Artificielle* 11(3) :375–398. 22, 23

[1999] Fargier, H. ; Lang, J. ; et Marquis, P. 1999. Décision en environnement partiellement observable et résolution de formules booléennes quantifiées. Dans *Actes de JNPC'99*, 129–138. 23

[1993] Fargier, H. ; Lang, J. ; et Schiex, T. 1993. Selecting preferred solutions in Fuzzy Constraint Satisfaction Problems. Dans *Proc. of the 1st European Congress on Fuzzy and Intelligent Technologies*. 13

[1996] Fargier, H. ; Lang, J. ; et Schiex, T. 1996. Mixed constraint satisfaction : a framework for decision problems under incomplete knowledge. Dans *Proc. of AAAI'96*. Portland, OR : AAAI Press. 22

[1994] Fargier, H. 1994. *Problèmes de satisfaction de contraintes flexibles et application à l'ordonnancement de production*. Thèse de doctorat, Institut de Recherche en Informatique de Toulouse (Université Paul Sabatier), Toulouse, France. 13, 20

[1992] Freuder, E., et Wallace, R. 1992. Partial constraint satisfaction. *Artificial Intelligence* 58 :21–70. 13, 14, 18

[1985] Freuder, E. C. 1985. A sufficient condition for backtrack-bounded search. *Journal of the ACM* 32(14) :755–761. 8

[1989] Freuder, E. C. 1989. Partial constraint satisfaction. Dans *Proc. of the 11th IJCAI*, 278–283. 14, 15

[1974] Gaschnig, J. 1974. A constraint satisfaction method for inference making. Dans *Proc. of the twelfth annual Allerton conference on circuit and system theory*, 866–874. 6

[1979] Gaschnig, J. 1979. *Performance measurement and analysis of certain Search Algorithms*. Ph.D. Dissertation, Carnegie-Mellon University. 7

[1998] Ginsberg, M. ; Parkes, A. ; et Roy, A. 1998. Supermodels and robustness. Dans *Proc. of AAAI'98*, 334–339. 11

[1993] Ginsberg, M. L. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1 :25–46. 8, 9

[1980] Haralick, R. M., et Elliot, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14 :263–313. 5, 8

[1984] Howard, R. A., et Matheson, J. E. 1984. Influence diagrams. Dans *The principles and applications of decision analysis*. 21

[1992] Hubbe, P. D., et Freuder, E. C. 1992. An efficient cross-product representation of the constraint satisfaction problem search space. Dans *Proc. of AAAI'92*, 421–427. 9, 22, 23

[2000] Jussien, N. ; Debruyne, R. ; et Boizumault, P. 2000. Maintaining arc-consistency within dynamic backtracking. Dans *Actes de JNPC'00*. 10

[1988] Krentel, M. 1988. The complexity of optimization problems. *Journal of Computer and System Sciences* 36 :490–509. 16

[1992] Krentel, M. W. 1992. Generalizations of optp to the polynomial hierarchy. *Theoretical Computer Science* 97 :183–198. 16

[1996] Larrosa, J., et Meseguer, P. 1996. Exploiting the use of DAC in max-CSP. Dans *Proc. of CP'96*, 308–322. 19

[1998] Larrosa, J. ; Meseguer, P. ; Schiex, T. ; et Verfaillie, G. 1998. Reversible DAC and other improvements for solving max-CSP. Dans *Proc. of AAAI'98*. 19, 21

[1999] Larrosa, J. ; Meseguer, P. ; et Schiex, T. 1999. Maintaining reversible DAC for Max-CSP. *Artificial Intelligence* 107(1) :149–163. 19

[1992] Martin-Clouaire, R. 1992. Dealing with soft constraints in a constraint satisfaction problem. Dans *Proc. of the International Conference on Information Processing of Uncertainty in Knowledge based Systems*, 37–40. 12

[1976] Minoux, M. 1976. Structures algébriques généralisées des problèmes de cheminement dans les graphes. *Recherche opérationnelle* 10(6) :33–62. 17

[1992] Minton, S. ; Johnston, M. D. ; Philips, A. B. ; et Laird, P. 1992. Minimizing conflicts : a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence* 58 :160–205. 11

[1988] Nadel, B. A. 1988. Tree search and arc consistency in constraint satisfaction algorithms. Dans Kanal, L., et Kumar, V., eds., *Search in Artificial Intelligence*. Springer-Verlag. chapitre 9, 287–342. 6

[1992] Prosser, P. ; Conway, C. ; et Muller, C. 1992. A constraint maintenance system for the distributed allocation problem. *Intelligent Systems Engineering* 1(1) :76–83. 10

[1993a] Prosser, P. 1993a. Domain filtering can degrade intelligent backtracking search. Dans *Proc. of the 13th IJCAI*, 262–267. 8, 9

[1993b] Prosser, P. 1993b. Hybrid algorithms for the Constraint Satisfaction Problem. *Computational Intelligence* 9(3) :268–299. 8

[1994] Puterman, M. L. 1994. *Markov decision processes : discrete stochastic dynamic programming*. New York : Wiley-Interscience. 21

[1976] Rosenfeld, A. ; Hummel, R. ; et Zucker, S. 1976. Scene labeling by relaxation operations. *IEEE Trans. on Systems, Man, and Cybernetics* 6(6) :173–184. 12, 20

[1994] Ruttkay, Z. 1994. Fuzzy constraint satisfaction. Dans *Proc. FUZZ-IEEE'94*. 12

[1994] Sabin, D., et Freuder, E. C. 1994. Contradicting conventional wisdom in constraint satisfaction. Dans *Proceedings of the 2nd International Workshop on Principles and Practice of Constraint Programming*, numéro 874 dans LNCS. Rosario, Orcas Island (WA) : Springer-Verlag. 6

[1993] Schiex, T., et Verfaillie, G. 1993. Nogood recording for static and dynamic CSP. Dans *Proceeding of the 5th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'93)*, 48–55. 8, 11

[1994a] Schiex, T., et Verfaillie, G. 1994a. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools* 3(2) :187–207. 8, 11

[1994b] Schiex, T., et Verfaillie, G. 1994b. Stubborness : an enhancement scheme for backjumping and nogood recording. Dans *Proc. of the 12th ECAI*, 165–169. 8

[1996] Schiex, T. ; Régin, J.-C. ; Gaspin, C. ; et Verfaillie, G. 1996. Lazy arc consistency. Dans *Proc. of AAAI'96*. Portland, OR : AAAI Press. 6, 7

[1995] Schiex, T. ; Fargier, H. ; et Verfaillie, G. 1995. Valued constraint satisfaction problems : hard and easy problems. Dans *Proc. of the 14th IJCAI*, 631–637. 16, 20, 21

[1992] Schiex, T. 1992. Possibilistic constraint satisfaction problems or "How to handle soft constraints ?". Dans *Proc. of the 8th Int. Conf. on Uncertainty in Artificial Intelligence*. 12

[1994] Schiex, T. 1994. Préférences et incertitudes dans les problèmes de satisfaction de contraintes. Rapport technique 2/7899 DERA, CERT. 16

[1998] Schiex, T. 1998. Maximizing the reversible DAC lower bound in Max-CSP is NP-hard. Rapport technique 1998/02, INRA. 19

[2000a] Schiex, T. 2000a. Arc cohérence pour contraintes molles. Dans *Actes de JNPC'00*. 14, 21

[2000b] Schiex, T. 2000b. Arc consistency for soft constraints. Dans *Actes de CP'00*. 21

[1988] Shafer, G., et Shenoy, P. 1988. Local computations in hyper-trees. Working paper 201, School of business, University of Kansas. 13, 20

[1981] Shapiro, L., et Haralick, R. 1981. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3 :504–519. 13

[1991] Shenoy, P. 1991. Valuation-based systems for discrete optimization. Dans Bonissone ; Henrion ; Kanal ; et Lemmer., eds., *Uncertainty in AI*. North-Holland Publishers. 13

[1990] Snow, P., et Freuder, E. 1990. Improved relaxation and search methods for approximate constraint satisfaction with a maximin criterion. Dans *Proc. of the 8th biennal conf. of the canadian society for comput. studies of intelligence*, 227–230. 12

[1989] van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*. Logic Programming Series. Cambridge, MA : MIT Press. 4

[1990] van Hentenryck, P. 1990. Incremental constraint satisfaction in logic programming. Dans *ICLP'90*, 189–202. 11

[1994a] Verfaillie, G., et Schiex, T. 1994a. Solution reuse in dynamic constraint satisfaction problems. Dans *Proc. of AAAI'94*, 307–312. 11

[1994b] Verfaillie, G., et Schiex, T. 1994b. Dynamic backtracking for dynamic CSPs. Dans *Proceedings ECAI'94 Workshop on Constraint Satisfaction Issues raised by Practical Applications*. 9, 11

[1995] Verfaillie, G., et Schiex, T. 1995. Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes : bilan de quelques approches. *Revue d'Intelligence Artificielle* 9(3) :269–309. 11

[1996] Verfaillie, G. ; Lemaître, M. ; et Schiex, T. 1996. Russian doll search. Dans *Proc. of AAAI'96*, 181–187. 19

[1994] Wallace, R. 1994. Directed arc consistency preprocessing. Dans Meyer, M., ed., *Selected papers from the ECAI-94 Workshop on Constraint Processing*, numéro 923 dans LNCS. Berlin : Springer. 121–137. 18, 19

Index