

Résumé

Nous présentons dans cet article un système de relaxation de contraintes sur les CSP sur-contraints. Ce système utilise un mécanisme de maintien de déductions basé sur un enregistrement de justifications pour chaque action (retrait de valeur, retrait de contrainte, identification de contradiction). Notre système de relaxation de contraintes peut alors non seulement éviter de recommencer la recherche à zéro après chaque modification du système de contraintes mais aussi identifier de manière précise un ensemble de contraintes à relaxer.

1 Introduction

Comme le remarquent G. VERFAILLIE et T. SCHIEX [1995], *de nombreux problèmes de RO ou d'IA s'expriment naturellement sous forme de problèmes de satisfaction de contraintes (CSP)*. Mais, *nombre de ces problèmes sont en réalité dynamiques en ce sens que l'ensemble des variables et des contraintes à considérer évolue du fait d'agents extérieurs*.

Le formalisme des CSP dynamiques permet de tenir compte de telles situations. Un CSP dynamique est constitué d'une séquence de CSP déduits l'un de l'autre par l'ajout ou le retrait d'une contrainte. Notons que l'on sait à chaque étape quelle contrainte ajouter ou supprimer (il s'agit d'un facteur exogène).

Mais, de nombreux problèmes réels sont souvent sur-contraints¹ (emploi du temps, planification, ...). Il faut donc retirer une (ou plusieurs) contrainte(s) afin de permettre l'obtention d'une solution. En général, l'utilisateur n'a pas, *a priori*, de connaissances particulières pour déterminer judicieusement les contraintes à relaxer.

Dans cet article, nous proposons un système de relaxation de contraintes sur les CSP qui permet :

- de déterminer la (ou les) contrainte(s) à supprimer en cas d'échec,
- d'effectuer efficacement les ajouts/suppressions afin d'éviter des traitements inutiles lors de chaque évolution du système. Il s'agit d'une certaine forme de réactivité.

Notre système repose sur un mécanisme de maintien de déductions qui permet à la fois d'identifier les contraintes à relaxer et d'éviter d'explorer certaines parties de l'arbre de recherche devenues inutiles.

1. Le système de contraintes associé ne possède pas de solution.

La résolution d'un CSP passe par une phase d'énumération (recherche de solution). Cette phase est généralement vue comme disjointe du CSP à proprement parler. Nous considérons (section 2) l'énumération comme un processus «dynamique» d'ajouts/retraits de contraintes d'égalité. Cette approche permet de traiter de manière uniforme les contraintes initiales (CSP) et l'énumération.

Dans la section 3, nous passons tout d'abord en revue diverses approches de la relaxation de contraintes sur les CSP puis nous présentons notre proposition.

Nous décrivons ensuite section 4 notre système de maintiens de déductions et montrons son utilisation dans le cadre de la relaxation de contraintes sur les CSP.

Dans la section 5, nous comparons notre système avec les approches utilisant un système de maintiens de déductions.

2 Contraintes explicites et contraintes implicites

Après quelques rappels sur les CSP statiques et dynamiques, nous comment l'énumération peut être vue en terme d'ajout/suppression de contraintes.

2.1 Rappels sur les CSP

Définition 1 (CSP)

Un CSP est défini par un triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ où :

- $\mathcal{V} = \{v_1, \dots, v_n\}$ est un ensemble de n variables,
- $\mathcal{D} = \{d^1, \dots, d^n\}$ est un ensemble de n domaines (ensembles de valeurs possibles) associés aux n variables.
- $\mathcal{C} = \{c_1, \dots, c_m\}$ est un ensemble de m contraintes.

Soit d^i le domaine de la variable v_i , on pose: $d^i = \{d_1^i, \dots, d_\ell^i\}$ On note l'affectation de la valeur d_j^i à la variable v_i par: $\langle v_i \leftarrow d_j^i \rangle$.

On note $\mathbf{vars}(c)$ les variables de la contrainte c . On note $\mathbf{const}(v)$ les contraintes portant sur la variable v .

Proposition 1 (Simplification)

Pour simplifier le discours, on suppose que pour tout ensemble V de variables ($V \subset \mathcal{V}$) il existe au plus une contrainte c telle que $\mathbf{vars}(c) = V$.

De plus, nous supposons, sans perte de généralité, qu'aucune contrainte ne porte sur plus de deux variables.

Une contrainte unaire c avec $\mathbf{vars}(c) = \{x\}$, est notée c_x . Une contrainte binaire c avec $\mathbf{vars}(c) = \{x, y\}$, est notée c_{xy} . Lorsqu'aucune confusion n'est possible, c_{v_i} est notée c_i et c_{v_i, v_j} , c_{ij} .

Définition 2 (Application d'une contrainte)

La fonction booléenne $c_x(\ell)$ est vraie si $\langle x \leftarrow \ell \rangle$ permet de satisfaire la contrainte unaire c_x .

De même, la fonction booléenne $c_{xy}(k, \ell)$ est vraie si le couple $(\langle x \leftarrow k \rangle, \langle y \leftarrow \ell \rangle)$ permet de satisfaire la contrainte binaire c_{xy} .

Définition 3 (Solution d'un CSP)

Une solution \mathcal{S} d'un CSP est un ensemble d'affectations tel que :

- $\forall v_i \in \mathcal{V}, \exists! d_k^i \in d^i, \langle v_i \leftarrow d_k^i \rangle \in \mathcal{S}$. On note $\mathbf{val}_{\mathcal{S}}(v_i)$ la valeur de v_i dans la solution \mathcal{S} .
- $\forall v_i \in \mathcal{V}, \forall c_i \in \mathbf{const}(v_i), c_i(\mathbf{val}_{\mathcal{S}}(v_i))$
- $\forall v_i \in \mathcal{V}, \forall c_{ij} \in \mathbf{const}(v_i), c_{ij}(\mathbf{val}_{\mathcal{S}}(v_i), \mathbf{val}_{\mathcal{S}}(v_j))$

2.2 CSP dynamiques

Afin de représenter des problèmes exigeant un environnement évolutif, le formalisme des CSP a été étendu aux CSP dynamiques (DCSP).

Définition 4

Un DCSP P est une séquence P_1, P_2, \dots, P_n de CSP $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{C}_1), \dots, (\mathcal{V}_n, \mathcal{D}_n, \mathcal{C}_n)$ où :

$$\forall i, \mathcal{C}_i = \mathcal{C}_{i-1} \pm c$$

Nous supposons que les contraintes sont introduites une à une dans le système de contraintes, comme c'est le cas pour la PLC (Programmation Logique avec Contraintes).

Proposition 2 (Démarche de résolution)

La résolution d'un CSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ se ramène à la :

- maintenance d'une certaine propriété (arc consistance par ex.) dans le DCSP $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{C}_1), \dots, (\mathcal{V}_m, \mathcal{D}_m, \mathcal{C}_m)$ où $\mathcal{C}_i = \mathcal{C}_{i-1} + c_i$ pour $i \in [1, m]$.
- résolution du CSP $(\mathcal{V}_m, \mathcal{D}_m, \mathcal{C}_m)$

On a $\mathcal{V}_m = \mathcal{V}, \mathcal{D}_m = \mathcal{D}$ et $\mathcal{C}_m = \mathcal{C}$.

2.3 L'énumération en tant que CSP dynamique

Pour tout CSP, nous considérons deux types de contraintes :

- les contraintes explicites (originales) du problème,
- les contraintes implicites qui n'apparaissent que lors de l'énumération. Il s'agit des contraintes d'égalité entre une variable et une valeur de son domaine.

Nous proposons de ne plus différencier ces deux types de contraintes, en présentant une vision de l'énumération permettant de rendre explicites les contraintes implicites du CSP.

L'énumération consiste en fait à ajouter et retirer des contraintes. Nous le montrons en utilisant la procédure standard de backtrack chronologique² (cf. FIG. 1).

L'algorithme de la figure 1 utilise :

- la procédure **enum**(\mathcal{V}) qui énumère un ensemble \mathcal{V} de variables,
- la procédure **enumVar**(v, d, \mathcal{V}) qui tente d'affecter une valeur à la variable v de domaine d compatible avec les contraintes du système, les variables de \mathcal{V} restant à énumérer,
- la procédure **affect?**(v, ℓ, \mathcal{V}) qui détermine si on peut affecter la valeur ℓ à la variable v .

La ligne 2 de la procédure **affect?** qui correspond à l'affectation de la valeur ℓ à la variable v équivaut implicitement à l'ajout de la contrainte $c : (v = \ell)$. De même, les lignes 4 et 8 de la même procédure qui correspondent à la remise en cause de l'affectation courante de la variable v équivalent au retrait d'une contrainte $c : (v = \ell)$ du système de contraintes.

En remplaçant les lignes 2, 4 et 8 de la procédure **affect?** par les ajouts et les retraits de contrainte implicite correspondants, on montre qu'énumérer sur un CSP statique, c'est résoudre un CSP dynamique.

Cette approche de l'énumération par identification à un CSP dynamique permet ainsi de ne plus dissocier l'énumération de la mise en place des contraintes et surtout de traiter de la même manière contraintes explicites et implicites. Ce traitement identique va faciliter la gestion de la relaxation de contraintes en permettant de ne pas traiter l'énumération de manière indépendante.

2. L'illustration de nos conceptions ne change pas fondamentalement pour des algorithmes de backtrack plus évolués.

```

procédure enum( $\mathcal{V}$ )
(1) début
(2)   si  $\mathcal{V} = \emptyset$  alors
(3)     retourner SUCCÈS
(4)   sinon
(5)     choisir  $v_i \in \mathcal{V}$ 
(6)     retourner enumVar( $v_i, d^i, \mathcal{V}$ )
(7)   fin
(8) fin

procédure enumVar( $v, d, \mathcal{V}$ )
(1) début
(2)   si  $d = \emptyset$  alors
(3)     retourner ÉCHEC
(4)   sinon
(5)     choisir  $\ell \in d$ 
(6)     si affect?( $v, \ell, \mathcal{V}$ ) alors
(7)       retourner SUCCÈS
(8)     sinon
(9)       retourner enumVar( $v, d - \{\ell\}, \mathcal{V}$ )
(10)    fin
(11) fin
(12) fin

procédure affect?( $v, \ell, \mathcal{V}$ )
(1) début
(2)   affecter  $\ell$  à  $v$ 
(3)   si  $\exists c \in \text{const}(v)$  insatisfaite alors
(4)     annuler l'affectation de  $v$ 
(5)     renvoyer ÉCHEC
(6)   sinon
(7)     si enum( $\mathcal{V} - v$ ) = ÉCHEC alors
(8)       annuler l'affectation de  $v$ 
(9)       retourner ÉCHEC
(10)    sinon
(11)      retourner SUCCÈS
(12)    fin
(13) fin
(14) fin

```

FIG. 1 - *Algorithme d'énumération*

3 Relaxation de Contraintes

Un système de relaxation de contraintes, permettant de *résoudre* les problèmes sur-contraints est un système permettant :

- de déterminer la (ou les) contrainte(s) à supprimer pour obtenir une solution,
- une réactivité par rapport aux modifications du système de contraintes (ajouts/suppressions).

3.1 Un formalisme pour les systèmes sur-contraints

L'utilisateur du système est capable de *juger* l'importance des contraintes qu'il manipule. Ce jugement peut être quantitatif (il peut alors s'agir d'un poids) ou qualitatif. Pour prendre en compte ces nouvelles informations, nous introduisons la notion de WCSP. Nous nous inspirons pour cela des travaux de [Bel *et al.*, 1992] et de [Borning *et al.*, 1989]. Le concept de WCSP permet à l'utilisateur de fixer avant la résolution l'importance relative qu'il accorde à la satisfaction de chacune des contraintes du système.

Définition 5 (Ensemble ordonné)

On appelle ensemble ordonné, tout ensemble \mathcal{E} sur lequel il existe une relation d'ordre total notée $\prec_{\mathcal{E}}$ et pour lequel il existe un plus petit élément noté $\perp_{\mathcal{E}}$ et un plus grand élément noté $\top_{\mathcal{E}}$.

Définition 6 (WCSP)

Un WCSP est un quintuplet $(\mathcal{V}, \mathcal{D}, \mathcal{C}, \mathcal{W}, \pi)$ où :

- $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ est un CSP,
- \mathcal{W} est un ensemble ordonné de jugements où $\perp_{\mathcal{W}}$ est noté Λ_{α} et où $\top_{\mathcal{W}}$ est noté Λ_{ω} .
- $\pi : \mathcal{C} \rightarrow \mathcal{W}$ est une fonction associant un jugement à toute contrainte.

Une contrainte dans un WCSP peut être active (faire partie du système courant de contraintes) ou inactive (avoir été retirée du système courant de contraintes).

Définition 7 (Préférence entre contraintes)

$\pi(c_i) = j$ est noté : $c_i@j$. On dira qu'une contrainte $c_i@k$ est préférée à une contrainte $c_j@l$ (on note $c_i \prec c_j$) si $k \prec_{\mathcal{W}} l$. Deux contraintes de même jugement ne sont pas comparables. On définit donc un ordre partiel sur les contraintes.

Dans ce cadre, on se rend compte que les contraintes de *jugement* Λ_α sont les contraintes les plus importantes du système de contraintes. Il s'agit donc des contraintes *requises* du système de contraintes.

Définition 8 (Solution d'un WCSP)

Une solution d'un WCSP $(\mathcal{V}, \mathcal{D}, \mathcal{C}, \mathcal{W}, \pi)$ est un doublet $(\mathcal{S}, \mathcal{C}')$ où :

- \mathcal{C}' est un ensemble de contraintes tel que $\mathcal{C}' \subset \mathcal{C}$ et $\forall c \in \mathcal{C}' : \pi(c) \neq \Lambda_\alpha$. \mathcal{C}' est l'ensemble des contraintes violées par l'ensemble \mathcal{S} d'affectations. Ces contraintes sont inactives.
- \mathcal{S} est une solution du CSP $(\mathcal{V}, \mathcal{D}, \mathcal{C} \setminus \mathcal{C}')$

Lorsqu'on l'on cherche à résoudre un CSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ classique, on peut le transformer en WCSP $(\mathcal{V}, \mathcal{D}, \mathcal{C}, \mathcal{W}, \pi)$ pour lequel on va rechercher une solution $(\mathcal{S}, \mathcal{C}')$. Cette solution ne peut être quelconque. On cherche donc une solution pour laquelle l'ensemble \mathcal{C}' vérifie une certaine propriété.

3.2 Systèmes existants – Approche PLC

La relaxation de contraintes a été approchée de différentes façons :

3.2.1 Systèmes complets de relaxation de contraintes

- Haselböck *et al.* [1993] proposent d'utiliser un algorithme de réparation en partant d'une instance, proche du problème en cours, déjà résolue. Cette approche nécessite une solution déjà existante et de savoir quelles opérations exécuter (ajout, retrait de contraintes). On retombe donc en partie sur le problème des CSP dynamiques.
- Un problème de relaxation de contraintes peut être résolu par des recherches arborescentes où les contraintes pouvant être relaxées sont intégrées à la fonction objectif [Fages *et al.*, 1994]. On utilise alors des techniques connues de Recherche Opérationnelle. Une telle approche repose sur une comparaison *numérique* des contraintes ce qui limite l'approche car l'utilisateur n'a aucun contrôle possible sur la résolution qui n'est guidée que par la fonction objectif.
- Fages *et al.* [1995] proposent un schéma réactif en PLC(FD) permettant d'ajouter et de retirer efficacement des contraintes. Ce schéma est présenté en termes de transformations d'arbres CSLD. Là encore, les contraintes à supprimer ou à ajouter sont des facteurs exogènes et ne sont donc pas automatiquement détectées par le système.
- La relaxation de contraintes peut être vue comme le traitement d'une *tour de contraintes* [Borning *et al.*, 1989]. On peut citer le système HCLP(\mathcal{R}). Celui-ci suppose que le problème original n'a pas de solution. Ainsi, il essaie de trouver une solution respectant tout d'abord l'intégralité des contraintes les plus importantes, puis raffine la solution obtenue en intégrant les contraintes niveau d'importance par niveau d'importance jusqu'à ce qu'un échec se produise. Malheureusement, une telle approche n'est pas réalisable sur les Domaines Finis pour lesquels les techniques de consistance sont locales (savoir si un système de contraintes est satisfaisable est NP-complet).

Aucun de ces systèmes n'est donc totalement adapté à notre problématique. Néanmoins, citons l'approche proposée par [Brewka *et al.*, 1992] qui relie les techniques de relaxation de contraintes aux techniques de raisonnement non monotone. Elle est intéressante car elle propose une nouvelle vision de la notion de contrainte.

3.2.2 Système d'identification de contraintes à relaxer

Le système IHCS [Menezes *et al.*, 1993] propose une identification de contraintes à relaxer en cas de contradiction à l'aide d'un graphe de dépendance entre contraintes défini par la relation de dépendance : *une contrainte c_a dépend d'une contrainte c_b si la contrainte c_b modifie le domaine d'une variable apparaissant dans c_a* . Ce graphe permet d'identifier un ensemble de contraintes qui *pourrait* être responsable d'une contradiction. Cette condition ne permet pas d'assurer la pertinence de l'ensemble de contraintes déterminées. En effet, cet ensemble contient assurément un ensemble contradictoire de contraintes mais n'en est pas un lui-même. Ceci est principalement dû au fait que le système IHCS perd des informations en ne travaillant qu'au niveau des contraintes et non pas au niveau plus fin des valeurs des variables.

3.3 Systèmes existants – Approche CSP

Il existe différentes méthodes de recherche de solution dans les CSP basées sur des systèmes de maintien de déductions (*nogood recording*, *dynamic backtracking*). Ces techniques de backtrack non chronologique (*nogood recording*, *dynamic backtracking*) et/ou non destructif (*dynamic backtracking*) peuvent être vues comme des systèmes de relaxation de contraintes sur les contraintes implicites du problème.

En effet, ces méthodes de *backtrack intelligent* cherchent à déterminer un point de backtrack (affectation à remettre en cause) pertinent lors d'une recherche de solution. Cela revient en fait à identifier un certain nombre de contraintes à relaxer (remise en cause d'une affectation) lorsque le système courant de contraintes (y compris les affectations déjà réalisées) devient contradictoire. D'autre part, ces approches cherchent à réutiliser au mieux les informations collectées avant le backtrack pour faciliter l'exploration après celui-ci. Il s'agit alors véritablement d'un système de relaxation de contraintes sur les contraintes implicites du problème.

3.4 Notre proposition

Nous proposons un système de relaxation capable d'identifier un ensemble de contraintes à relaxer en cas de contradiction dans le système de contraintes. Ce système permet également d'effectuer efficacement les ajouts/suppressions de contraintes en évitant les traitements inutiles.

Pour cela, nous utilisons un système de maintien de déductions. Ce système de maintien de déductions, présenté dans la section suivante, permet de traiter toutes les contraintes, aussi bien explicites qu'implicites et ce, de manière totalement uniforme.

4 Maintien de déductions

Nous proposons dans cette partie, un système de gestion de *justifications* permettant une maintenance de déductions en vue de réaliser un système de relaxation de contraintes correspondant aux objectifs définis en introduction.

4.1 Déductions : Concepts de base

4.1.1 Justifications et Déductions

Lorsqu'on étudie les algorithmes de maintien d'arc-consistance dans les systèmes dynamiques (DNAC4 [Bessière, 1991], DNAC6 [Debruyne, 1995], ...) et les algorithmes de recherche de solution (*nogood recording* [Schiex et Verfaillie, 1994], *dynamic backtracking* [Ginsberg, 1993]) on constate qu'il est fondamental d'être capable de maintenir un certain nombre d'informations permettant d'enregistrer et de justifier les décisions prise en cours de résolution.

Il existe plusieurs types de décisions (ou déductions)

- le retrait d'une valeur dans le domaine d'une variable,

- le retrait d'une contrainte du système de contraintes courant,
- l'identification d'une branche de l'arborescence de recherche conduisant à une contradiction.

Comme il s'agit de déductions, il en existe une preuve que nous appelons *justification*. Les *justifications* sont un élément essentiel de la plupart des algorithmes de maintien d'arc consistence. Pour un algorithme de recherche, il est important, aussi, de conserver les justifications des déductions.

4.1.2 Spécifications formelles

Au cours de la résolution d'un CSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, on construit un ensemble Δ de déductions.

Définition 9 (Justification)

On appelle **justification** d'une déduction $\delta \in \Delta$ tout doublet (C, D) fourni comme preuve pour la déduction. C est un ensemble de contraintes ($C \subset \mathcal{C}$) et D , un ensemble de déductions ($D \subset \Delta$). On note J_δ l'ensemble des justifications fournies pour la déduction δ .

Définition 10 (Environnement)

On peut déterminer, à partir d'une déduction δ , un ensemble de contraintes dont la conjonction conduit à la preuve de la déduction. Cet ensemble de contraintes est appelé **environnement** de la déduction et est noté $C_{(\delta)}$. Il est construit à partir d'une justification (C, D) choisie dans J_δ :

$$C_{(\delta)} = C \cup \bigcup_{\delta_i \in D} C_{(\delta_i)}$$

Il existe donc plusieurs environnements.

Définition 11 (Étiquette)

À une déduction δ est associé l'ensemble E_δ des environnements de δ . E_δ est appelé l'**étiquette** de la déduction δ .

4.1.3 Utilisation d'une étiquette

On associe une étiquette à toute déduction réalisée en cours de résolution. On appelle **nogood** un élément de l'étiquette d'une contradiction. Un **nogood** correspond donc à un ensemble de contraintes dont l'action conjointe provoque une contradiction.

Nous avons vu précédemment que la résolution d'un CSP se ramenait à la résolution d'un DCSP. Le système des contraintes actives évolue donc continuellement au cours de la résolution.

Afin d'éviter d'emprunter continuellement les mêmes branches de l'arborescence de recherche, les étiquettes permettent de savoir quelles déductions (en particulier les contradictions) sont impliquées par le système courant de contraintes actives.

Lors d'une relaxation ces étiquettes permettent de ne pas avoir à recommencer la recherche à zéro, puisqu'elles permettent d'identifier les déductions toujours valides malgré l'évolution du système de contraintes.

Définition 12 (Environnements impliqués)

Supposons une déduction δ d'étiquette E_δ . Soit J_k et J_ℓ deux éléments de E_δ . On peut affirmer que si $J_k \subset J_\ell$ alors J_ℓ peut être retiré de E_δ sans perte d'information.

En effet, si J_k suffit à prouver δ alors les contraintes supplémentaires (i.e. $J_\ell \setminus J_k$) n'apportent aucune information supplémentaire sur la preuve de la déduction.

On dit alors que J_ℓ est impliqué par J_k .

4.1.4 Logique booléenne des étiquettes

La résolution d'un CSP ou d'un DCSP amène de nombreux changements dans le système de contraintes courant. Une contrainte retirée à un instant donné peut être réintroduite à une étape ultérieure³.

Les justifications doivent donc être conservées tout au long de la résolution. Il faut être capable d'identifier une justification valide à un instant donné.

Définition 13 (Validité – Ensembles)

Un environnement est dit valide s'il ne contient pas de contraintes non actives au moment où la validité est demandée.

Une étiquette est dite valide si elle contient au moins un environnement lui-même valide.

Définition 14 (Validité – Booléens)

On associe à toute contrainte c intervenant dans la résolution une valeur booléenne de même nom qui est vraie lorsque la contrainte est active.

On associe à tout environnement $e = \{c_1, \dots, c_n\}$ une variable booléenne de même nom exprimant la validité de celui-ci et ayant pour valeur, la valeur booléenne de l'expression : $c_1 \wedge \dots \wedge c_n$.

On associe à toute étiquette $E = \{e_1, \dots, e_n\}$ une variable booléenne de même nom exprimant la validité de celle-ci et ayant pour valeur la valeur booléenne de l'expression : $e_1 \vee \dots \vee e_n$.

4.2 Maintien de déductions et Relaxation de Contraintes

Un système de gestion de justifications est très utile pour un système de relaxation de contraintes (résolution de WCSP). En effet, il permet de gérer non seulement le maintien dynamique des déductions réalisées mais il permet aussi de fournir des informations facilitant l'identification de contraintes à relaxer (utilisation active des *nogoods*).

4.2.1 Utilisation active des *nogoods*

La proposition suivante fournit une des bases de notre système de relaxation de contraintes.

Proposition 3 (Contradiction)

Pour supprimer une contradiction, il faut invalider tous les *nogoods* la justifiant (contenu de l'étiquette).

Il ne s'agit pas là d'une condition suffisante puisque l'information contenue dans les étiquettes n'est que partielle. En effet, d'une part, tous les *nogoods* n'ont pas été extraits lors de la recherche et, d'autre part, l'arbre de recherche n'a pas encore été totalement exploré. Par contre, la condition précédente est bien nécessaire puisque la validité d'un des *nogoods* implique à coup sûr une contradiction.

Notons que cette proposition est utilisée aussi pour montrer rapidement qu'un CSP classique ne possède pas de solution. En effet, il suffit qu'un *nogood* ne contienne pas de contrainte *implicite* pour affirmer que le problème est sur-contraint. En effet, pour éviter une contradiction, il faut relaxer une contrainte originale du problème.

Proposition 4 (Invalidation de *nogoods*)

Jussien et Boizumault [1995] identifient deux façons différentes de voir l'invalidation d'un ensemble de *nogoods* :

1. Sélection d'un ensemble de contraintes à retirer du système courant de contraintes afin de rendre invalide chacun des *nogoods*. Il s'agit d'une vision « théorie des graphes » du problème. En effet, on retrouve un problème classique de recouvrement dans un hypergraphe dont les sommets sont les *nogoods* et dont les hyperarêtes sont les contraintes [Minoux, 1983].

3. Ceci est particulièrement flagrant pour les contraintes implicites d'affectation.

2. Choix de variables booléennes à rendre fausses pour rendre la formule booléenne exprimant la validation de l'étiquette de la contradiction fausse. Il s'agit là d'une vision « logique » des choses. On retrouve ici un problème de satisfaction de formule booléenne.

Il reste bien sûr à définir un certain critère pour déterminer ces ensembles de contraintes. Il s'agit en fait de la propriété vérifiée par l'ensemble de contraintes inactives de la solution du WCSP (cf. section 3.1).

4.2.2 Fonctionnalités

Notre système de gestion de justifications pour la relaxation de contraintes intègre les fonctions suivantes :

- **ajouteJustification**(δ, C, D) qui ajoute la justification (C, D) pour la déduction δ . La déduction δ étant une information du type :
 - **retrait**(v_i, d_i^i) dans le cas du retrait d'une valeur dans le domaine d'une variable.
 - **retrait**(c) dans le cas du retrait d'une contrainte.
 - **contradiction** dans le cas de la justification d'une contradiction.
- **étiquette**(δ) qui renvoie l'étiquette du nœud δ ,
- **estValide**(δ) qui renvoie **vrai** si la déduction δ est valide et **faux** sinon.
- **déductions**(c) qui renvoie les déductions dont la validité dépend de l'activation de la contrainte c . Il s'agit de l'ensemble $\alpha(c)$ défini ainsi :

$$\alpha(c) = \left\{ \delta_i \mid c \in \bigcap_{e_j \in E_{\delta_i}} e_j \right\}$$

- **dépendance**(δ) qui renvoie les déductions dont la validité d'une justification dépend de la validité de δ . Il s'agit de l'ensemble $\beta(\delta)$ défini ainsi :

$$\beta(\delta) = \{ \delta_i \mid \exists (C, D) \in J_{\delta}, \delta \in D \}$$

4.2.3 Résolution d'un WCSP

Nous présentons ici une démarche de résolution pour les WCSP.

Proposition 5 (Justifier un retrait de valeur)

Prenons comme exemple AC4. Cet algorithme peut être facilement adapté pour les enregistrements demandés par notre système. L'algorithme AC4 est scindé en deux phases : une première phase d'initialisation (calcul des compteurs et listes support) et une deuxième phase de propagation (propagation récursive des retraits de valeurs – on part des retraits effectués lors de la première phase).

Pendant la première phase, AC4 retire des valeurs (soit a une telle valeur retirée du domaine de la variable i) par application directe d'une contrainte (soit c une telle contrainte). On ajoute alors, dans l'algorithme AC4, après le retrait de valeur, la ligne : **ajouteJustification**($\delta_{(i \neq a)}$, $\{c\}$, $\{\}$)⁴.

Par contre, dans la deuxième phase, AC4 retire des valeurs des domaines (soit b de j une telle valeur) pendant la phase de propagation du retrait d'une autre valeur (par exemple, a de i). On ajoute alors la ligne : **ajouteJustification**($\delta_{(j \neq b)}$, $\{c\}$, $\{\delta_{(i \neq a)}\}$).

On peut mettre ici en évidence la différence fondamentale entre notre approche et l'approche retenue dans DNAC4. En effet, DNAC4 n'utilise que la partie C de la justification et ne tient pas compte de la partie D . De plus, C est alors un singleton et il ne peut y avoir qu'une justification pour une déduction (retrait de valeur).

⁴. $\delta_{(i \neq a)}$ est la déduction : retrait de la valeur a du domaine de i

Proposition 6 (Justifier une contradiction)

Une contradiction (un échec) intervient lorsque le domaine d'une variable v devient vide. On note δ_ℓ la déduction associée au retrait de la $\ell^{\text{ème}}$ valeur du domaine de v . On suppose que ce domaine contient k valeurs à l'origine. La justification de la contradiction est alors : $(\{\}, \{\delta_\ell \mid \ell \in [1, k]\})$

Proposition 7 (Retirer une contrainte)

Pour retirer une contrainte c , il faut déterminer quelles sont les déductions dont la validité dépend de c , ce que fournit la fonction `déductions`. Supposons que `déductions(c) = D`. Pour tout $\delta \in D$ correspondant à un retrait de valeur, il faut réintroduire cette valeur⁵.

On peut trouver une autre justification pour δ au cours de ce processus. Il faut alors propager cette information à toutes les étiquettes des éléments de `dépendance`(δ).

Proposition 8 (Justifier un retrait)

Lorsqu'une contrainte c est retirée après identification (on utilise la proposition 4) suite à une contradiction de déduction associée δ il faut fournir une justification pour ce retrait.

La contrainte c permet d'invalider un ou plusieurs nogoods. Appelons N l'ensemble des nogoods invalidés par le retrait de c . Il existe alors plusieurs justifications pour le retrait de c : $(n \setminus \{c\}, \{\})$ une pour tout élément $n \in N$.

Proposition 9 (Gestion des contradictions)

Lorsqu'une contradiction est identifiée :

- une nouvelle déduction δ est définie et sa justification est fournie au système de gestion des justifications à l'aide de la fonction `ajouteJustification`. (Proposition 6)
- L'étiquette E_δ est demandée grâce à la fonction `étiquette` et celle-ci est utilisée pour déterminer un ensemble C_δ de contraintes à relaxer. (Proposition 4)
- Les contraintes de C_δ sont retirées du système courant de contraintes. Pour chaque retrait, un ensemble de justifications est proposé. (Proposition 8)
- Si au cours du retrait des contraintes, une nouvelle contradiction apparaît, elle est traitée de la même manière. Ainsi, dans le pire des cas on sera amené au retrait de l'intégralité des contraintes du système (ce qui garantit l'arrêt d'une telle méthode).

Proposition 10 (Résolution de WCSP)

La résolution d'un WCSP $(\mathcal{V}, \mathcal{D}, \mathcal{C}, \mathcal{W}, \pi)$ se ramène à (modification de la proposition 2) :

- maintenance d'une certaine propriété avec enregistrement des justifications des retraits de variables dans le DWCSPP $(\mathcal{V}_1, \mathcal{D}_1, \mathcal{C}_1, \mathcal{W}, \pi), \dots, (\mathcal{V}_m, \mathcal{D}_m, \mathcal{C}_m, \mathcal{W}, \pi)$ où $\mathcal{C}_i = \mathcal{C}_{i-1} + c_i$. Si une contradiction apparaît à l'étape k , elle est traitée comme nous l'avons vu précédemment et on reprend la résolution à l'étape $k + 1$.
- résolution du WCSP $(\mathcal{V}_m, \mathcal{D}_m, \mathcal{C}_m, \mathcal{W}, \pi)$ par une méthode d'énumération. Nous ne détaillerons pas celle-ci car nous proposons dans la suite une alternative à cette approche.

On a $\mathcal{V}_m = \mathcal{V}, \mathcal{D}_m = \mathcal{D}, \mathcal{C}_m = \mathcal{C}$ et $\mathcal{W}_m = \mathcal{W}$.

4.3 Implémentation

Une première implémentation a été réalisée en particulierisant les concepts ATMS [Jussien et Boizumault, 1995].

Nous avons choisi d'implémenter un système paramétrable. En effet, nous pouvons identifier divers paramètres qui influent sur la qualité des résultats obtenus.

5. Des valeurs peuvent être réintroduites à tort au cours de ce processus, il faut donc vérifier chaque retrait (cf. DNAC*)

4.3.1 Paramètres Généraux

- **Algorithme de filtrage utilisé.** Plus l'algorithme utilisé est évolué, plus les justifications des déductions vont être pertinentes et intéressantes. En effet, les algorithmes évolués travaillent plus vite et exploitent donc au mieux les informations fournies par le problème traité.
- **Propriété vérifiée par la solution (S, C') du WCSP.** Comme nous l'avons signalé dans la section 3.1, l'ensemble C' doit vérifier une certaine propriété. Nous avons choisi : C' est tel que la contrainte la plus importante de C' est la moins importante possible⁶. Ce critère est intéressant car il se prête aisément à une satisfaction incrémentale (lorsqu'on on identifie une contradiction, on n'est pas obligé de remettre en cause des relaxations faites précédemment). Un critère optimisant par exemple la *somme* des *poids* des contraintes relaxées requiert non seulement la résolution d'un problème NP-complet à chaque fois (recouvrement de poids minimal dans un hypergraphe) mais nécessite de remettre en cause des relaxations effectuées précédemment.

4.3.2 Paramétrer le fonctionnement

- **Degré de précision des justifications.** Pour un même algorithme de filtrage, les justifications fournies peuvent être plus ou moins précises. On influe ici sur l'utilisation ou non des deuxième et troisième paramètres de `ajouteJustification`(δ , C , D).
- La **complétude du calcul** des étiquettes et des autres informations est un paramètre important. En effet, doit-on recalculer une étiquette pour un nœud lors de l'appel de la fonction `étiquette`, ou doit-on maintenir les étiquettes en permanence en les faisant évoluer au fur et à mesure que de nouvelles informations arrivent⁷ ?
- **Suppression d'informations invalides.** On sait bien qu'une information invalide a instant donné peut redevenir valide dans la suite de la résolution. C'est pourquoi toutes les informations sont conservées dans notre système. Mais, en pratique, on peut préférer supprimer de telles informations pour économiser du temps et de l'espace même si cela peut se révéler dommageable.

4.4 Alternative pour l'énumération

Nous avons montré (*cf.* section 2.3) comment l'énumération pouvait se ramener à la résolution d'un DCSP. En pratique, la recherche de solution se fait par énumération et reste totalement séparée de la phase de mise en place des contraintes. Nous proposons donc de transformer cette phase en une mise en place dynamique de contraintes. Cette transformation nous permet d'ailleurs d'unifier le comportement de notre système de maintien de déduction face à la résolution d'un CSP. En effet, on manipule alors uniquement des contraintes et non pas des contraintes (explicites) *et* des affectations (contraintes implicites).

4.4.1 Énumération et disjonction

La phase d'énumération associée à toute recherche de solution dans un CSP correspond en fait à la mise en place d'un certain nombre de contraintes disjonctives.

En effet, dire qu'une variable v_i a pour domaine $d^i = \{d_1^i, \dots, d_k^i\}$ correspond en fait à la mise en place de la contrainte disjonctive :

$$(v_i = d_1^i) \vee \dots \vee (v_i = d_k^i)$$

6. C'est le critère du système HCLP [Borning *et al.*, 1989].

7. On peut encore se demander s'il faut les faire évoluer en fonction de nouvelles informations. La réponse dépend de l'intérêt d'une information complète par rapport à une information partielle suivant l'objectif que l'on se fixe. En effet, dès lors que la condition d'invalidation de nogoods (proposition 3) n'est pas suffisante, on peut se contenter d'une information parcellaire puisqu'elle est valide.

Il ne s'agit pas là d'une contrainte disjonctive classique puisqu'un et un seul des membres de la disjonction ne peut être vrai à la fois. Plus précisément, une telle spécification de domaine correspond donc à la mise en place de la contrainte **oneof** ($\{(v_i = d_1^i), \dots, (v_i = d_k^i)\}$)

4.4.2 oneof et Relaxation de contraintes

Nous proposons le traitement suivant de la contrainte **oneof** :

Supposons l'appel à : **oneof** ($\{c_1, \dots, c_\ell\}$).

- Introduire la contrainte $(c_i)@A_\omega$, une contrainte choisie dans $\{c_1, \dots, c_\ell\}$.
- En cas de relaxation de c_i au cours de la résolution, déterminer l'ensemble C_v des contraintes de $\{c_1, \dots, c_\ell\}$ dont la déduction associée à un retrait (si elle existe) ne possède pas d'étiquettes valides⁸. Il s'agit des contraintes dont l'ajout n'est pas sensé provoquer une contradiction.
- Si $C_v \neq \emptyset$ alors choisir $c_k \in C_v$ pour introduction (on introduit alors $(c_k)@A_\omega$)
- Si C_v est vide. Il y a contradiction puisqu'une au moins des contraintes de $\{c_1, \dots, c_\ell\}$ doit être active. La justification de la contradiction est alors : (**oneof**, $\{\delta_1, \dots, \delta_\ell\}$) où δ_i correspond à la déduction : «je dois relaxer⁹ c_i ». Après traitement de la contradiction, on réitère.

4.4.3 Avantages de l'approche

Cette approche utilise les propriétés d'un système de relaxation de contraintes. En effet, un mauvais choix dans l'alternative n'est pas trop pénalisant car tout le travail indépendant de ce choix est conservé en cas de remise en cause.

Un mécanisme de disjonction constructive peut être mis en place en introduisant une à une toutes les contraintes de l'alternative ce qui permet de conserver le travail indépendant de la contrainte introduite par la suite.

La contrainte **oneof** permet de modéliser avantageusement l'énumération sur une variable par une disjonction. Les contraintes implicites du problème sont ainsi totalement explicitées et il n'y a plus lieu de faire la distinction entre les contraintes originales et les contraintes introduites lors de l'énumération.

5 Apports et Comparaisons

5.1 Apports

Nous avons présenté une nouvelle approche du formalisme CSP dans laquelle toutes les contraintes sont explicites. Cette connaissance complète des contraintes permet de ne plus distinguer la recherche de solution de la mise en place des contraintes.

Notre système de maintien de déductions étend l'enregistrement de justifications aux contradictions et aux retraits de contraintes.

Ces deux composantes nous permettent à la fois de déterminer précisément un ensemble de contraintes à supprimer (*cf.* section 4.2.1) et d'effectuer le retrait de ces contraintes. Nous pouvons maintenant comparer notre approche avec les techniques de recherches de solution utilisant la relaxation de contraintes sur les contraintes implicites.

8. Dans le cas d'une énumération, on ne retiendra dans C_v que les contraintes dont la valeur associée fait encore partie du domaine

9. Dans le cas d'une énumération la déduction est : soit je dois relaxer c_i , soit j'ai retiré la valeur correspondante du domaine de la variable

5.2 Recherche de solution

Deux algorithmes efficaces de recherche de solution dans les DCSP (*nogood recording* et *dynamic backtracking*) manipulent ce que Verfaillie et Schiex [1995] appellent des *nogoods*. Un *nogood* étant défini comme une affectation partielle (affectation d'un sous-ensemble des variables du problème) qui ne fait partie d'aucune solution du problème.

Dans notre système, une affectation partielle correspond en fait à un ensemble de contraintes d'affectation. Un *nogood* (dans notre acception) correspond alors tout à fait à la définition de [Verfaillie et Schiex, 1995] si les seules contraintes relaxables sont les contraintes d'affectation. Pour cela, il suffit d'affecter le jugement Λ_α aux contraintes originales du CSP (elles ne peuvent être inactivées (*cf.* Définition 8)).

Les *nogoods* permettent non seulement d'expliquer les causes d'échec mais facilitent aussi le traitement. En effet, dès qu'un *nogood* est valide, il n'est pas nécessaire de continuer la recherche (elle conduit inmanquablement vers un échec).

Comme nous le signalions section 3.3, ces algorithmes évolués de recherche de solution fonctionnent en fait comme des systèmes de relaxation sur les contraintes implicites.

5.2.1 Application à *dynamic backtracking*

L'algorithme *dynamic backtracking* permet l'utilisation la plus fine du travail précédent dans le cas d'une utilisation dans un cadre dynamique (*cf.* [Verfaillie et Schiex, 1995]). Nous allons donc montrer comment son comportement peut être simulé dans notre système.

L'algorithme *dynamic backtracking* utilise une *eliminating explanation* pour chaque valeur retirée du domaine d'une variable. Cette *eliminating explanation* est définie comme étant la première contrainte mise en échec du fait de l'affectation d'une valeur à une variable, cela correspond donc à l'enregistrement `ajouteJustification($\delta, \{c\}, \{\}$)` dans notre système.

Notons que dans la présentation originale de cet algorithme [Ginsberg, 1993] l'*eliminating explanation* proposée consistait en l'ensemble des variables affectées à l'instant de l'affectation d'une valeur à une variable v_i . Cela correspond donc à `ajouteJustification($\delta, \{c_k \mid k < i\}, \{\}$)` où c_k est la contrainte mise en place à l'occasion de l'affectation d'une valeur à la variable v_k . La première proposition est donc plus précise car la deuxième est trop générale dans sa justification.

L'algorithme *dynamic backtracking* ne se charge pas simplement d'identifier des *nogoods*, il les utilise pour permettre une méthode de backtrack efficace lors de la recherche de solution. Pour cela, l'algorithme utilise les *nogoods* afin de déterminer le prochain point de backtrack. Il ne s'agit pas d'un vrai backtrack puisque les déductions effectuées après le point de backtrack ne sont pas défaites. Cela s'apparente beaucoup plus à une relaxation de contrainte. Cette identification de point de backtrack est *sommaire*¹⁰ dans *dynamic backtracking*. Nous, nous pouvons utiliser notre mécanisme d'identification de contrainte à relaxer pour déterminer ce point de backtrack.

5.2.2 Application à *nogood recording*

Nogood recording [Schiex et Verfaillie, 1994] est un algorithme de recherche de solution utilisant des *nogoods* identifiés au cours de la recherche pour couper l'arbre d'exploration des recherches ultérieures.

Un *nogood* est identifié lors d'un échec, il est composé d'une affectation partielle (les variables déjà instanciées) et de la première contrainte (c) mise en échec par l'affectation partielle courante. Cela correspond à l'enregistrement `ajouteJustification($\delta, \{c\} \cup \{c_i \mid v_i \in \mathcal{V}_1\}, \{\}$)` où δ est une contradiction, \mathcal{V}_1 l'ensemble des variables instanciées et c_i la contrainte d'affectation associée à v_i .

En utilisant ce type d'enregistrements, on peut donc simuler le fonctionnement de *nogood recording* dans notre système.

10. Sommaire dans le sens relaxation de contraintes, mais amplement suffisante pour le *backtrack intelligent* !

5.2.3 Constat

Notre système se différencie des algorithmes *nogood recording* et *dynamic backtracking* dans l'utilisation du troisième paramètre de la fonction `ajouteJustification`, la partie «déductions précédentes» de la justification. En effet, aucune des simulations que nous avons effectuées n'utilisait ce troisième paramètre.

C'est d'ailleurs ce troisième paramètre qui permet la collecte du plus grand nombre d'informations utilisées pour la détermination des contraintes à relaxer.

6 Conclusion et travaux futurs

Cet article propose une vision uniforme des CSP (on ne distingue plus contraintes explicites et contraintes implicites) qui nous permet de construire un système de maintien de déductions. Ce système apparaît comme un outil indispensable pour la relaxation de contraintes, puisqu'il permet à la fois d'identifier et de supprimer efficacement une (ou des) contrainte(s) dans un problème sur-contraint.

Nous réalisons actuellement une implémentation en taille réelle de notre système de relaxation de contraintes. Parallèlement, quatre axes de recherche sont envisagés :

- Étudier plus avant les apports de la contraintes `oneof` dans le cas de l'énumération d'un CSP,
- Étudier l'influence des divers paramètres dans notre implémentation pour dégager des principes généraux sur l'utilisation de tels systèmes,
- Proposer une sémantique de la relaxation de contraintes afin d'assurer une compatibilité de notre approche en *programmation en logique*,
- Proposer une représentation ATMS du problème de la relaxation de contraintes à la manière de de Kleer [1989].

Remerciements

Nous remercions Philippe David et Olivier Lhomme pour leurs commentaires fructueux sur des version préliminaires de ce papier.

Références

- [Bel *et al.*, 1992] Gérard Bel, Éric Bensana, Khaled Ghédira, David Lesaint, Thomas Schiex, Gérard Verfaillie, Christine Gaspin, Roger Martin-Clouaire, Jean-Pierre Rellier, Pierre Berlandier, Bertrand Neveu, Brigitte Trousse, Hène Fargier, Jérôme Lang, Philippe David, Philippe Janssen, Tibor Kökény, Marie-Catherine Vilarem, et Philippe Jégou. Représentation et traitement pratique de la flexibilité dans les problèmes sous contraintes. In *Actes des Journées Nationales du PRC GDR Intelligence Artificielle*, Marseille, France, October 1992.
- [Bessière, 1991] Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*, 1991.
- [Borning *et al.*, 1989] Alan Borning, Michael Maher, Amy Martindale, et Molly Wilson. Constraint hierarchies and logic programming. In Giorgio Levi et Maurizio Martelli, editors, *ICLP'89: Proceedings 6th International Conference on Logic Programming*, pages 149–164, Lisbon, Portugal, June 1989. MIT Press.
- [Brewka *et al.*, 1992] Gerhard Brewka, Hans Werner Guesgen, et Joachim Hertzberg. Constraint relaxation and nonmonotonic reasoning. Technical Report TR-92-002, ICSI, Berkeley, 1992.

- [de Kleer, 1989] Johan de Kleer. A comparison of ATMS and CSP techniques. In *IJCAI-89: Proceedings 11th International Joint Conference on Artificial Intelligence*, pages 290–296, Detroit, 1989.
- [Debruyne, 1995] Romuald Debruyne. Les algorithmes d’arc-consistance dans les CSP dynamiques. *Revue d’Intelligence Artificielle*, 9(3), 1995.
- [Fages *et al.*, 1994] François Fages, Julian Fowler, et Thierry Sola. Handling preferences in constraint logic programming with relational optimization. In *PLILP’94*, Madrid, September 1994.
- [Fages *et al.*, 1995] François Fages, Julian Fowler, et Thierry Sola. A reactive constraint logic programming scheme. In *International Conference of Logic Programming, ICLP’95*, Tokyo, 1995.
- [Ginsberg, 1993] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
- [Haselböck *et al.*, 1993] Alois Haselböck, Thomas Havelka, et Markus Stumptner. Revising inconsistent variable assignments in constraint satisfaction problems. In Manfred Meyer, editor, *Constraint Processing: Proceedings of the International Workshop at CSAM’93, St. Petersburg, July 1993*, Research Report RR-93-39, pages 113–122, DFKI Kaiserslautern, August 1993.
- [Jussien et Boizumault, 1995] Narendra Jussien et Patrice Boizumault. Implementing constraint relaxation over finite domains using ATMS. In Michael Jampel, Eugene Freuder, et Michael Maher, editors, *OCS’95: Workshop on Over-Constrained Systems at CP’95*, Cassis, Marseilles, 18 September 1995.
- [Menezes *et al.*, 1993] Francisco Menezes, Pedro Barahona, et Philippe Codognet. An incremental hierarchical constraint solver. In Paris Kanellakis, Jean-Louis Lassez, et Vijay Saraswat, editors, *PPCP’93: First Workshop on Principles and Practice of Constraint Programming*, Providence RI, 1993.
- [Minoux, 1983] Michel Minoux. *Programmation Mathématique, Théorie et Algorithmes*, chapter Annexe 4, pages 231–266. Editions Dunod, 1983.
- [Schiex et Verfaillie, 1994] Thomas Schiex et Gérard Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problems. *International Journal of Artificial Intelligence Tools*, 3(2):187–207, 1994.
- [Verfaillie et Schiex, 1995] Gérard Verfaillie et Thomas Schiex. Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes : bilan de quelques approches. *Revue d’Intelligence Artificielle*, 9(3), 1995.