
Explications *k-relevantes* pour la programmation par contraintes

Samir Ouis* — Narendra Jussien* — Patrice Boizumault**

* *École des Mines de Nantes*
4 rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3
{Samir.Ouis, Narendra.Jussien}@emn.fr

** *GREYC, CNRS UMR 6072*
Université de Caen, Campus 2,
F-14032 Caen Cedex
Patrice.Boizumault@info.unicaen.fr

RÉSUMÉ. Cet article présente un ensemble d'outils d'aide au développement d'applications en Programmation par Contraintes. Nous proposons des outils de diagnostic (analyse d'état, analyse des contradictions, analyse de l'impact d'une contrainte), ainsi que des outils interactifs (simulation d'ajout/retrait dynamique de contraintes). Ces outils reposent sur la notion d'explication *k-relevante* [Bay 96] qui permet de gérer des explications multiples pouvant ainsi conduire à de meilleurs diagnostics. Un exemple est présenté pour, à la fois illustrer les explications *k-relevantes*, ainsi que les différentes fonctionnalités des outils de diagnostic et des outils interactifs que nous proposons.

ABSTRACT. This paper presents a set of tools based on explanations for Constraint Programming. These tools exploit *k-relevant* explanations [Bay 96] which enable to use several explanations, which can lead to better diagnosis. *k-relevant* explanations are introduced and used to provide : diagnosis tools (state analysis, contradiction analysis, constraint impact analysis), and interaction tools (dynamic constraint addition/retraction simulation). An example is given to illustrate *k-relevant* explanations and to provide concrete situations illustrating the functionalities of our interactive and diagnosis tools.

MOTS-CLÉS : Programmation par Contraintes, CSP, explications, diagnostic, outils interactifs

KEYWORDS: Constraint Programming, CSP, nogoods, explanations, diagnosis, interactive tools

1. Introduction

La Programmation par Contraintes a aujourd'hui prouvé qu'elle était extrêmement bien adaptée pour la modélisation et la résolution de problèmes combinatoires tels que les problèmes d'emploi du temps, les problèmes d'allocation de ressources ou les problèmes de conception. Plusieurs langages et systèmes tels que CHIP, CHOCO [LAB 00], GNUPROLOG [DIA 00], ILOG SOLVER ont été développés. Malheureusement ces systèmes demeurent encore impuissants face à des systèmes de contraintes inconsistants. En effet, la plupart du temps, seul un laconique *pas de solution* s'affiche sur l'écran. L'utilisateur doit, lui même, trouver les causes cette inconsistance : pourquoi le problème n'a-t-il aucune solution ; quelle(s) contrainte(s) relaxer afin de restaurer la cohérence, ...

Ces questions font apparaître deux différents problèmes : *expliquer* l'incohérence et *rétablir* la cohérence. Plusieurs réponses ont déjà été fournies à ces 2 estions : QUICKXPLAIN [JUN 01] calcule des *nogoods*¹ pour la résolution de problèmes de configuration, [BES 91] et [DEB 96] présentent des outils permettant de relaxer dynamiquement des contraintes, PALM [JUS 00a] utilise les *nogoods* pour répondre aux questions précédentes et concevoir de nouveaux algorithmes de recherche, [SQU 96] présente des outils de contraintes spécifiques pour fournir des solutions conviviales, [FRE 00] génère des explications sous forme d'arbres, puis les combine avec des heuristiques/stratégies de choix pour obtenir de meilleures explications selon un critère bien défini, ...

Dans cet article, nous présentons l'apport des explications *k-relevantes*[Bay 96] pour développer des outils de diagnostic (analyse d'état, analyse des contradictions, analyse de l'impact d'une contrainte), ainsi que des outils interactifs (simulation d'ajout ou de retrait dynamique de contraintes) d'aide au développement d'applications en Programmation par Contraintes. Le principe de la *k-relevance* est de conserver un ensemble limité de plusieurs explications (au lieu d'une unique explication [JUS 01]), en se basant sur leur *relevance* (ou pertinence). De manière informelle, la *relevance* d'une explication mesure le fait qu'elle puisse, plus ou moins rapidement, devenir valide dans la suite de la résolution.

Cet article est organisé comme suit : dans la section 2, nous rappelons la définition et les procédés de construction des *nogoods* et des explications en Programmation par Contraintes. Nous présentons les explications *k-relevantes* (section 3) que nous illustrons au travers d'un exemple (section 4). Puis, nous présentons (section 5) la mise en oeuvre, à l'aide des explications *k-relevantes*, de nos différents outils de diagnostic et de développement interactif.

1. Un *nogood* est un ensemble de contraintes qui est inconsistant.

2. Nogoods et explications de retraits

Nous considérons des problèmes de satisfactions de contraintes (CSP) définis par un triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$. $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ est un ensemble de n variables, de domaines finis respectifs $\mathcal{D} = \{d_{v_1}, d_{v_2}, \dots, d_{v_n}\}$ et un ensemble de e contraintes $\mathcal{C} = \{c_1, c_2, \dots, c_e\}$. Une solution d'un CSP est une affectation de l'ensemble des variables, telle que toutes les contraintes de \mathcal{C} soient simultanément satisfaites. Nous notons par $\text{sol}(V, \mathcal{C})$ l'ensemble des solutions du CSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$.

Dans la suite de cet article, nous considérons le domaine d'une variable v_i comme étant une disjonction de contraintes unaires $\{v_i = a, a \in d_{v_i}\}$. En conséquence, la phase d'énumération, qui permet d'explorer l'arbre de recherche, sera traitée comme une série d'ajouts et de retraits de telles contraintes. Ces contraintes particulières seront appelées *contraintes de décision*.

2.1. Définitions

Considérons un CSP C constitué des contraintes initiales du problème et d'un ensemble de contraintes de décision, tel que C soit inconsistant. Un *nogood* [SCH 94] ng est un sous-ensemble de contraintes de C tel que ng soit lui-même inconsistant. Un *nogood* peut être décomposé en deux parties : un sous-ensemble des contraintes initiales ($C' \subset C$ voir équation 1) et un sous-ensemble (dc_1, \dots, dc_k) des contraintes de décision qui ont été ajoutées au système initial de contraintes par l'énumération².

$$\text{sol}(V, (C' \wedge dc_1 \wedge \dots \wedge dc_k)) = \emptyset \quad [1]$$

On peut avoir une vision plus opérationnelle des *nogoods* en réécrivant l'équation 1, faisant ainsi apparaître explicitement la justification d'une décision prise par le solveur, ici dc_j :

$$C' \wedge \left(\bigwedge_{i \in [1..k] \setminus j} dc_i \right) \rightarrow \neg dc_j \quad [2]$$

Soit $s(v)$ la valeur d'une variable v dans une solution s ; considérons la contrainte $dc_j : v_j = a$ et utilisons l'équation 2, on obtient :

$$\forall s \in \text{sol} \left(V, C' \wedge \left(\bigwedge_{i \in [1..k] \setminus j} dc_i \right) \right), s(v_j) \neq a \quad [3]$$

2. Notons que certains cas spéciaux peuvent se présenter. Si $k < 1$, le CSP initial est déjà sur-contraint (au moins une contrainte devra être relaxée). Si $C' = \emptyset$, l'ensemble des décisions prises jusque-là est lui-même contradictoire. Ceci peut se produire uniquement si aucune propagation n'est faite après qu'une décision ait été prise.

La partie gauche de l'implication de l'équation 1 constitue une **explication (de retrait)** de la valeur a du domaine de la variable v_j . Elle sera notée : $\text{expl}(v_j \neq a)$.

De nouvelles explications sont construites lors du filtrage, à partir des explications existantes, et au gré des retraités effectués. Par exemple, un nouveau *nogood* peut être obtenu dès que le domaine d'une variable v_j est devenu vide (par conjonction des explications de chacune des valeurs du domaine de v_j) :

$$\bigwedge_{a \in d_{v_j}} \text{expl}(v_j \neq a) \quad [4]$$

2.2. Quelle durée de vie pour une explication ?

Il existe généralement plusieurs explications différentes pour un retrait donné. Plusieurs approches ont été proposées pour gérer cette multiplicité :

- **DEPENDENCY DIRECTED BACKTRACKING** [STA 77] mémorise et conserve l'ensemble de **toutes les explications rencontrées**. Malheureusement, ce choix comporte un inconvénient majeur : sa complexité spatiale est exponentielle. En effet, le nombre d'explications mémorisées croît de manière monotone durant la résolution.

- Pour remédier à cet inconvénient majeur, plusieurs propositions ont choisi de ne mémoriser qu'**une seule explication par retrait** : **DYNAMIC BACKTRACKING** [GIN 93] et ses améliorations (**MAC-DBT** [JUS 00b], **PARTIAL-ORDER BACKTRACKING** [MCA 93], **CONFLICT-DIRECTED BACKJUMPING** [PRO 95]). L'idée commune à tous ces algorithmes consiste à oublier (donc supprimer) les explications qui ne sont plus valides par rapport à l'instanciation courante des variables. Ainsi, la complexité spatiale reste polynômiale tout en assurant la complétude [GIN 93].

Malheureusement, cette dernière approche n'est pas vraiment compatible avec les besoins du débogage et de l'analyse : comme pour chaque retrait de valeur, une seule explication est mémorisée, beaucoup d'informations concernant l'histoire de la résolution peuvent être irrémédiablement perdues. Un compromis intéressant consiste à sauvegarder les explications **aussi longtemps qu'elles vérifient une propriété** donnée. Différents critères ont été proposés :

- **temps limité** : les explications sont *oubliées* après un temps donné. Ce critère est proche de la gestion d'une liste TABU dans la recherche TABU [GLO 93].

- **taille limitée** : [SCH 94] et [DEC 90] bornent la taille des *nogoods* sauvegardés. La complexité spatiale est ainsi limitée, mais des explications s'avérant cruciales pour établir un bon diagnostic peuvent être détruites.

- **degré de relevance** : une explication est maintenue tant qu'elle est *proche* de l'instanciation courante ; elle est considérée comme étant toujours pertinente (*relevante*). Le paramètre k indique son degré de pertinence ; plus k est faible, plus l'explication est proche d'être valide.

La notion de k -**relevance** a été introduite par [Bay 96]) afin d'améliorer la complexité temporelle de la résolution d'une classe particulière de problèmes structurés. La validité d'une explication n'est plus liée uniquement à l'ensemble des contraintes actives mais tient aussi compte de l'ensemble des contraintes relaxées. Cette approche permet de limiter, elle aussi, la complexité spatiale. Comme notre objectif est la conception d'outils de mise au point, notre préoccupation majeure est de trouver le meilleur compromis entre la taille de l'espace nécessaire pour stocker les explications et la précision/qualité des explications sauvegardées.

3. Les explications k -relevantes

Au cours de la résolution, l'état courant du calcul sera décrit par deux ensembles de contraintes : R l'**ensemble des contraintes relaxées** (les décisions qui ont été défaits pendant la résolution et les contraintes initiales explicitement relaxées par l'utilisateur) et A l'**ensemble courant des contraintes actives** (le store). $\langle A, R \rangle$ est appelée *configuration*.

Définition 1 *Explication k -relevante [Bay 96]*

Soit $\langle A, R \rangle$ une configuration ; une explication e est k -**relevante** ssi elle contient au plus $(k - 1)$ contraintes relaxées, i.e. $|e \cap R| < k$.

Ce nombre de contraintes relaxées, $|e \cap R|$, est appelé **degré de l'explication** e .

Toutes les explications k -relevantes rencontrées durant la résolution sont conservées. Par conséquent, un retrait pourra avoir plusieurs explications. Ainsi $\text{expl}(v_i \neq a)$ ne contiendra plus une explication unique, mais l'ensemble de toutes les explications k -relevantes du retrait de la valeur a du domaine d_{v_i} .

Rq : toute explication k -relevante de degré $k = 1$ est une explication valide, au regard de la configuration courante.

3.1. Calcul des explications k -relevantes

Les explications k -relevantes peuvent être calculées lors du filtrage comme le montre l'exemple suivant :

Exemple 1 (Exemple du calcul d'explications) :

Considérons deux variables v_1 et v_2 ; supposons que la valeur a de v_1 ait comme unique support la valeur b de v_2 pour la contrainte c . Supposons de plus que, par filtrage, b soit retirée du domaine de v_2 et que ce retrait possède 3 explications valides : $\{c_1, c_2\}$, $\{c_1, c_3\}$, et $\{c_4, c_5\}$. Mais, a doit être retirée du domaine de v_1 . Quelle(s) explication(s) choisir pour calculer l'explication de ce retrait ? Faut-il considérer les 3 explications possibles $\{c, c_1, c_2\}$, $\{c, c_1, c_3\}$ ou $\{c, c_4, c_5\}$? ou nous contenter d'une seule ?

Étant donné qu'une valeur ne peut être retirée qu'une seule fois, nous avons choisi de ne retenir qu'une seule explication, la première produite, *i.e.* celle qui correspond au retrait (*physique*) de la valeur du domaine. Cette explication est appelée **explication principale** et sera la seule utilisée pour calculer les futures explications. Bien entendu, nous ne provoquons jamais le calcul exhaustif de l'ensemble de (toutes) les explications k -relevantes pour un retrait de valeur ; nous nous contentons de garder trace uniquement des explications produites par le filtrage.

N.B. L'explication principale est exactement celle qui aurait été calculée par une approche classique (voir la Section 2.1). C'est toujours le retrait (*physique*) d'une valeur d'un domaine qui produit la première explication.

Exemple 1 (suite) :

Si l'explication *principale* du retrait de b du domaine de v_2 est $\{c_1, c_2\}$, alors le retrait $v_1 \neq a$ sera justifié par $\{c, c_1, c_2\}$.

À chaque ajout/relaxation d'une contrainte c , il nous faut mettre à jour la validité de nos explications k -relevantes. Lors de l'ajout d'une contrainte c , toute explication où figure c voit son degré de relevance diminuer de 1 (plus son degré est faible, plus une explication est proche d'être valide). Toute explication de degré 1, où c était la seule contrainte relaxée, devient désormais valide (degré nul). Lors de la relaxation d'une contrainte c , toute explication où figure c voit son degré de relevance augmenter de 1 (plus le degré d'une explication augmente, moins elle a de chances de devenir valide). Dès que le degré d'une explication devient égal à k , cette explication est oubliée (détruite).

3.2. Calcul des nogoods

Le dilemme rencontré lors du calcul des explications k -relevantes apparaît aussi pour le calcul des *nogoods*. La section 2.1 décrit comment l'approche classique permet de calculer un *nogood* unique dès que le domaine d'une variable est devenu vide (par conjonction des explications de retrait de chacune des valeurs du domaine). Mais, avec la k -relevance, chaque retrait de valeur peut être justifié par plusieurs explications. En procédant de manière analogue, (conjonction des explications de retrait de chacune des valeurs du domaine), nous pouvons obtenir plusieurs *nogoods* ; ce nombre peut parfois devenir conséquent.

Soit $expl(v \neq a, i)$ l'ensemble des explications i -relevantes ($1 \leq i \leq k$) pour le retrait ($v \neq a$) ; alors $expl(v \neq a, 1)$ représente l'ensemble des explications valides pour le retrait ($v \neq a$). Le nombre potentiel de *nogoods* est :

$$\prod_{a \in d(v)} |expl(v \neq a, 1)| \quad \text{nogoods} \quad [5]$$

3.3. Complexités spatiales

Comme une approche classique conserve une seule explication par retrait de valeur, on aura au plus $n \times d$ explications de taille maximale $(e + n)$ *i.e.* les e contraintes du CSP initial ainsi que les n contraintes de décision. La complexité spatiale (pire cas) de cette approche est $O((e + n) \times n \times d)$.

Pour la k -relevance, une explication peut contenir jusqu'à $k - 1$ contraintes relaxées ; la taille maximale d'une explication peut donc atteindre $n + e + k - 1$. Le nombre maximum d'explications pour un retrait de valeur donné est limité par le nombre maximum de partitions d'un ensemble en sous-ensembles non-inclus l'un dans l'autre ; *i.e.* dans le pire cas : $\binom{e + n + k - 1}{(e + n + k - 1)/2}$ sous-ensembles de taille $(e + n + k - 1)/2$.

La complexité spatiale (pire cas) de stockage des explications k -relevantes est :

$$O\left(n \times d \times \binom{e + n + k - 1}{(e + n + k - 1)/2} \times (e + n + k - 1)/2\right) \quad [6]$$

3.4. Discussion

– 1-relevance vs. les approches classiques

Les approches classiques (comme DYNAMIC BACKTRACKING ou MAC-DBT) suppriment une explication dès qu'elle devient invalide. La 1-relevance diffère de ces approches par le nombre d'explications enregistrées par retrait. En effet, pendant la résolution, on peut trouver de nouvelles explications pour un retrait déjà effectué. La 1-relevance garde ces explications secondaires³. En outre, toutes les approches classiques travaillent à partir d'un seul et unique *nogood*. Ce *nogood* est calculé selon l'équation 4. La 1-relevance permet de traiter un *ensemble* de *nogoods* (voir équation 5).

– Comment obtenir le meilleur *nogood* ?

Les *nogoods* les plus intéressants sont ceux qui sont minimaux au sens de l'inclusion. Un *nogood* minimal peut être obtenu en calculant le recouvrement de toutes les explications valides. Malheureusement, le calcul d'un tel ensemble est de complexité exponentielle. À l'opposé, le *nogood* le plus simple à calculer est l'union de toutes les explications valides ; mais un tel choix entraîne une perte conséquente de précision.

Nous avons constaté qu'un bon compromis entre les deux critères (précision et simplicité de calcul) est de choisir une explication par retrait de valeur (*i.e.* la prin-

3. Elles seront utilisées pour calculer d'un côté le meilleur *nogood* et d'un autre côté améliorer la qualité des explications ; toutefois, l'explication principale sera toujours la seule utilisée pour calculer de nouvelles explications.

cipale) et de faire l'union de ces explications principales. On peut retenir la première explication obtenue comme étant cette explication principale ou bien la déterminer à l'aide d'un comparateur⁴, qui tiendra compte des préférences exprimées par l'utilisateur.

4. Un exemple : le problème de la conférence

Pour illustrer l'utilisation des explications k -relevantes, nous présentons la résolution du *problème de la conférence* [JUS 96].

4.1. Présentation du problème

Michel, Pierre et Alain se rencontrent inopinément dans un couloir. Michel interpelle les deux autres et leur rappelle leur promesse de présentation de leurs travaux respectifs. Pierre et Alain doivent présenter leurs travaux à Michel qui lui-même doit leur présenter ses travaux. Il y a donc 4 exposés : de Pierre à Michel, d'Alain à Michel, de Michel à Pierre et enfin de Michel à Alain. Ils conviennent de bloquer 4 demi-journées pour faire ces exposés. Chaque exposé prend une demi-journée. Dès le début, Michel signale qu'il lui paraît important *d'avoir écouté Pierre et Alain avant de faire sa présentation*. Puis, Michel indique qu'il *aimerait présenter ses travaux à Pierre lors de la deuxième demi-journée*. Enfin, Michel signale qu'il *ne veut pas présenter ses travaux à Pierre et Alain en même temps*.

4.2. Modélisation du problème

Ce problème se modélise sous forme d'un CSP ayant 4 variables $\{P_m, A_m, M_p, M_a\}$, où chaque variable représente un exposé. P_m désigne l'exposé que doit faire Pierre à Michel, A_m , l'exposé d'Alain à Michel, ... Les exposés devant avoir lieu dans 4 demi-journées, les domaines des variables sont identiques : $\{1, 2, 3, 4\}$. Les contraintes sont de 2 sortes : les contraintes d'intégrité concernant l'organisation des présentations, et les contraintes de préférence exprimées par Michel.

– **Contraintes d'intégrité** : Un orateur ne peut être auditeur dans la même demi-journée : $c_1 : M_a \neq A_m$, $c_2 : M_p \neq P_m$, $c_3 : M_a \neq P_m$ et $c_4 : M_p \neq A_m$. Une personne ne peut assister à deux présentations en même temps : $c_5 : A_m \neq P_m$.

– **Contraintes d'antériorité** : Michel souhaite présenter ses travaux après les présentations de Pierre et Alain : $c_6 : M_a > A_m$, $c_7 : M_a > P_m$, $c_8 : M_p > P_m$ et $c_9 : M_p > A_m$.

4. Un comparateur permet de "comparer" deux configurations et de choisir la plus intéressante selon certains critères de préférence [BOR 89].

– **Contraintes de simultan  it  ** : Michel aimerait pr  senter ses travaux    Pierre lors de la 2^{eme} demi-journ  e : $c_{10} : M_p = 2$. Michel ne veut pas pr  senter ses travaux    Pierre et Alain en m  me temps : $c_{11} : M_a \neq M_p$.

4.3. R  solution avec les explications 1-relevantes

Le tableau 1 d  crit l'  tat des domaines, apr  s filtrage par arc-consistance. Pour chaque valeur a retir  e du domaine (initial) d'une variable v_j , figurent les explications de retrait suivantes : la colonne *Explication* indique l'explication de retrait classique ; la colonne *1-relevance* indique la/les explication(s) 1-relevante obtenue(s) pour chaque retrait⁵.

Gr  ce aux explications 1-relevantes, nous obtenons 2 *nogoods* du fait que le domaine de P_m soit vide :

$ng_1 = \{c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$ et $ng_2 = \{c_4, c_5, c_6, c_8, c_9, c_{10}\}$. ng_2   tant inclus dans ng_1 est le plus pr  cis. Enfin, ng_1 et ng_2 sont eux-m  mes tous les 2 plus pr  cis que le *nogood* obtenu par l'approche classique,    savoir :

$ng = \{c_2, c_4, c_5, c_6, c_7, c_8, c_9, c_{10}\}$. Pour la conception et la mise en   uvre de nos outils de mise au point, plus les *nogoods* seront de petite taille, meilleurs seront les diagnostics que nous serons amen  s    faire.

Variable	Valeur	Explication	1-relevance	Pr��sente ?
P_m	1	$\{c_4, c_5, c_6, c_8, c_9, c_{10}\}$	$\{c_4, c_5, c_6, c_8, c_9, c_{10}\}$	non
P_m	2	$\{c_2, c_8, c_{10}\}$	$\{c_8, c_{10}\}$	non
P_m	3	$\{c_8, c_{10}\}$	$\{c_8, c_{10}\}$	non
P_m	4	$\{c_7\}$	$\{c_7\}, \{c_8\}$	non
A_m	1	\emptyset	\emptyset	oui
A_m	2	$\{c_4, c_8, c_{10}\}$	$\{c_4, c_8, c_{10}\}, \{c_9, c_{10}\}$	non
A_m	3	$\{c_9, c_{10}\}$	$\{c_9, c_{10}\}$	non
A_m	4	$\{c_6\}$	$\{c_6\}, \{c_9\}$	non
M_p	1	$\{c_8\}$	$\{c_8\}, \{c_9\}, \{c_{10}\}$	non
M_p	2	\emptyset	\emptyset	oui
M_p	3	$\{c_{10}\}$	$\{c_{10}\}$	non
M_p	4	$\{c_{10}\}$	$\{c_{10}\}$	non
M_a	1	$\{c_6\}$	$\{c_6\}, \{c_7\}$	non
M_a	2	\emptyset	\emptyset	oui
M_a	3	\emptyset	\emptyset	oui
M_a	4	\emptyset	\emptyset	oui

Tableau 1. Etat des domaines apr  s filtrage par AC

5. apr  s suppression des explications redondantes $\{c_2, c_8, c_{10}\}$ pour le retrait $P_m \neq 2$.

5. Outils de développement basés sur les explications k -relevantes

Comme la k -relevance peut fournir de multiples explications, elle va, de manière générale, permettre d'effectuer de meilleures analyses et d'obtenir de meilleurs diagnostics. Dans cette section, nous présentons plusieurs situations concrètes auxquelles l'utilisateur est fréquemment confronté, notamment dans le cas d'absence de solutions. Nous montrons comment les explications k -relevantes permettent de construire des outils de diagnostic ainsi que des outils interactifs d'aide à la mise au point de programmes avec contraintes.

5.1. Outils de diagnostic

5.1.1. Fournir des explications précises

Si nous avons deux explications e_1 et e_2 justifiant le même retrait et telles que $e_1 \subsetneq e_2$, alors nous pouvons en déduire que les contraintes appartenant à $e_2 \setminus e_1$ ne sont pas responsables du retrait. L'explication e_1 est **plus précise** que e_2 . Par conséquent, les contraintes appartenant à $e_2 \setminus e_1$ ne sont pas responsables de l'incohérence (à condition qu'elles n'interviennent pas dans d'autres retraits). La k -relevance n'est bien sûr pas la solution universelle. Mais, les explications multiples conduisent à des explications et des *nogoods* plus précis.

5.1.2. Analyse de l'impact d'une contrainte

Une situation fréquente dans le débogage est de savoir si une contrainte donnée appartient à un *nogood*. Les explications k -relevantes permettent d'apporter une réponse plus précise à cette question. Supposons qu'un échec soit causé par la variable A_m , dont le domaine est devenu vide (voir tableau 2); imaginons que l'utilisateur veuille savoir si la contrainte c_{11} appartient à un *nogood*. L'approche classique fournit le *nogood* $\{c_4, c_6, c_8, c_9, c_{10}\}$ auquel c_{11} n'appartient pas. La 1-relevance fournit 8 *nogoods* et indique à l'utilisateur que la contrainte c_{11} appartient à au moins un d'entre eux.

Var	Value	Explication	1-relevance	présente
A_m	1	$\{c_8\}$	$\{c_8\}, \{c_{11}\}$	no
A_m	2	$\{c_4, c_8, c_{10}\}$	$\{c_4, c_8, c_{10}\}, \{c_9, c_{10}\}$	no
A_m	3	$\{c_9, c_{10}\}$	$\{c_9, c_{10}\}$	no
A_m	4	$\{c_6\}$	$\{c_6\}$	no

Tableau 2. Nouvel état de la variable A_m

5.1.3. Fournir un diagnostic d'erreur

Après plusieurs interactions, l'utilisateur veut désormais savoir pourquoi la variable M_p ne peut pas prendre la valeur 1 ? L'approche classique fournit l'explication

$\{c_8\}$, alors que la 1-relevance fournit 3 explications possibles : $\{c_8\}, \{c_9\}, \{c_{10}\}$ (cf. tableau 1). Là encore, la 1-relevance s'avère être plus précise.

5.1.4. Reformuler les explications pour l'utilisateur

Comme on peut le constater sur notre exemple, les explications (classiques ou 1-relevantes) sont définies à partir d'un ensemble de contraintes de bas niveau. Seul un spécialiste peut comprendre et correctement interpréter ces explications ; en effet, elles peuvent être bien éloignées de la formulation qu'il a lui-même donnée en termes de primitives de plus haut niveau. Nous avons présenté dans [OUI 02] un outil permettant de fournir des explications compréhensibles par l'utilisateur en utilisant une représentation hiérarchique des différentes entités manipulées.

5.2. Outils interactifs

La k -relevance permet de simuler l'ajout et le retrait de contraintes avec un surcoût négligeable en temps de calcul.

5.2.1. Simuler la relaxation d'une contrainte

Déterminer l'appartenance d'une contrainte à un *nogood* peut être prohibitif en temps si l'on choisit de relaxer tour à tour chaque contrainte suspectée. Pour cette raison, nous proposons un outil qui permet de *simuler la relaxation d'une contrainte*, sans faire appel à la propagation, mais seulement en mettant à jour les explications k -relevantes.

Par exemple, supposons que l'utilisateur suspecte la contrainte c_8 d'appartenir à un *nogood*. L'outil de vérification (voir section 5.1.2) le confirme. La relaxation de cette contrainte remet, dans le domaine de A_m , toutes les valeurs a telles que $c_8 \in \text{expl}(A_m \neq a)$. Selon le tableau 2, la contrainte c_8 est partiellement responsable du retrait $A_m \neq 1$. L'approche classique remettrait la valeur 1 dans le domaine de A_m puis lancerait la phase de propagation. Mais, le problème demeure sur-contraint puisque le retrait ($A_m \neq 1$) est justifié par la contrainte c_{11} ; malgré cette relaxation, le domaine de A_m demeure vide.

Les explications 1-relevantes permettent immédiatement de s'apercevoir que, malgré la relaxation de la contrainte c_8 , l'inconsistance va demeurer (le retrait ($A_m \neq 1$) sera justifié par l'explication : $\{c_{11}\}$). Ainsi, notre outil peut informer l'utilisateur si la relaxation d'une contrainte suspectée va le conduire tout droit vers un autre échec *immédiat*. Bien évidemment, la k -relevance ne permet pas de détecter, de manière immédiate, tous les échecs. Malgré tout, cette possibilité est d'un intérêt pratique important puisqu'elle permet d'écarter très rapidement certaines suppositions.

5.2.2. Simuler l'ajout d'une contrainte

[BES 91] et [DEB 96] ont proposé des algorithmes de maintien d'arc consistance dynamique. Grâce à la k -relevance, nous pouvons tirer profit des informations accu-

mulées au cours de la résolution pour détecter des échecs immédiats. Pour cette raison, nous proposons un outil permettant de simuler la réintroduction d'une contrainte (précédemment relaxée). Cet outil informe l'utilisateur, si l'ajout d'une contrainte déjà relaxée, conduit de nouveau à un échec immédiat. Comme dans le cas du retrait de contrainte, cette simulation s'effectue avec un surcoût négligeable en temps de calcul.

Var	Value	Explication	1-relevance	2-relevance	présente
A_m	1	\emptyset	\emptyset	$\{c_8\}, \{c_{11}\}$	oui
A_m	2	\emptyset	$\{c_9, c_{10}\}$	$\{c_4, c_8, c_{10}\}$	no
A_m	3	$\{c_9, c_{10}\}$	$\{c_9, c_{10}\}$	\emptyset	no
A_m	4	$\{c_6\}$	$\{c_6\}$	\emptyset	no

Tableau 3. *Nouvel état de la variable A_m suite au retrait des contraintes c_8 et c_{11}*

Supposons que l'utilisateur ait précédemment relaxé les contraintes $\{c_8, c_{11}\}$ pour remettre la valeur 1 dans le domaine de A_m . Le tableau 3 décrit les domaines, ainsi que les explications correspondantes pour $k = 1$ et $k = 2$. Supposons, de plus, que l'utilisateur souhaite ré-introduire la contrainte c_8 . Une première approche consiste à activer le maintien de l'arc-consistance (qui conduit à un échec). Mais, à l'aide des explications 2-relevantes, l'ajout de la contrainte c_8 peut être simulé en mettant simplement à jour le degré de relevance des explications concernées. Ici, le retour de la contrainte c_8 fait que l'explication $\{c_8\}$ redevient valide et provoque le retrait de la valeur 1 du domaine de A_m , unique valeur restante. La détection de l'inconsistance relative à la ré-introduction de c_8 est immédiate. Sans aucune propagation, et en utilisant seulement les explications k -relevantes, nous pouvons là aussi détecter certaines inconsistances avec un surcoût négligeable en temps de calcul.

5.3. Amélioration des techniques basées sur les explications

Nous avons expérimenté les apports de la k -relevance pour des algorithmes de recherche basés sur les explications tels que MAC-DBT [JUS 00b]. Nos premiers résultats montrent que si k augmente, les performances diminuent. Cependant, pour $k = 1$ et $k = 2$, nous avons obtenu les mêmes performances temporelles que MAC-DBT. Pour $k = 1$ et $k = 2$, le temps nécessaire pour gérer les explications k -relevantes semble être compensé par les échecs évités. Pour $k \geq 3$, le temps nécessaire pour gérer les explications devient trop important par rapport au bénéfice obtenu. En particulier, beaucoup trop de temps est perdu dans la gestion d'explications qui ne deviendront jamais valides, et qui ne nous seront donc d'aucune utilité. Le bon compromis semble donc d'utiliser $k = 1$ ou $k = 2$, avec toutefois une petite pénalité d'espace mémoire. D'autres expérimentations doivent être faites afin d'analyser plus profondément l'intérêt des explications k -relevantes pour améliorer de manière générale les algorithmes de recherche.

6. Conclusion

Dans cet article, nous avons présenté plusieurs outils interactifs d'aide au développement d'applications en Programmation par Contraintes. Nous avons montré l'apport et l'efficacité de nos outils basés sur la k -relevance. Comparée aux approches classiques, la k -relevance fournit des explications plus précises, procure des informations précieuses qui étaient inaccessibles jusque-là, et de par ses multiples explications, permet des diagnostics plus riches.

[AMI 02] s'intéressent aussi à la conception et à la mise en oeuvre d'outils de diagnostic et de développement interactif d'applications dans le domaine de la conception. Il y a une différence majeure entre leurs travaux et notre approche : le CSP représentant le problème initial est considéré comme figé et, en conséquence, est compilé sous la forme d'un automate. L'utilisateur peut uniquement interagir au travers de l'ajout ou du retrait de contraintes de décision, ce qui n'est pas le cas dans notre proposition, où la dynamicité s'applique aux deux sortes de contraintes.

Nous travaillons actuellement sur la conception d'algorithmes permettant le calcul du meilleur *nogood* (voir Section 3.4). Pour cela, nous étudions l'ajout de comparateurs pour *comparer* automatiquement différentes solutions. D'un point de vue implantation, nous essayons de réduire la complexité spatiale nécessaire à la gestion des explications k -relevantes.

7. Bibliographie

- [AMI 02] AMILHASTRE J., FARGIER H., MARQUIS P., « Consistency restoration and explanations in dynamic CSPs - Application to configuration », *Artificial Intelligence*, vol. 135, n° 2002, 2002, p. 199-234.
- [Bay 96] BAYARDO JR. R. J., MIRANKER D. P., « A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem », *AAAI'96*, 1996.
- [BES 91] BESSIÈRE C., « Arc consistency in dynamic constraint satisfaction problems », *Proceedings AAAI'91*, 1991.
- [BOR 89] BORNING A., MAHER M., MARTINDALE A., WILSON M., « Constraint Hierarchies and Logic Programming », LEVI G., MARTELLI M., Eds., *ICLP'89 : Proceedings 6th International Conference on Logic Programming*, Lisbon, Portugal, juin 1989, MIT Press, p. 149-164.
- [DEB 96] DEBRUYNE R., « Arc-consistency in Dynamic CSPs is no more prohibitive », *8th Conference on Tools with Artificial Intelligence (TAI'96)*, Toulouse, France, 1996, p. 299-306.
- [DEC 90] DECHTER R., « Enhancement Schemes for Constraint Processing : Backjumping, Learning, and Cutset Decomposition », *Artificial Intelligence*, vol. 41, n° 3, 1990, p. 273-312.
- [DIA 00] DIAZ D., CODOGNET P., « The GNU Prolog system and its implementation », *ACM Symposium on Applied Computing*, Villa Olmo, Como, Italy, 2000.

- [FRE 00] FREUDER E. C., LIKITVIVATANAVONG C., WALLACE R. J., « A Case Study in Explanation and Implication », *In CP2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers*, 2000.
- [GIN 93] GINSBERG M., « Dynamic Backtracking », *Journal of Artificial Intelligence Research*, vol. 1, 1993, p. 25–46.
- [GLO 93] GLOVER F., LAGUNA M., *Modern heuristic Techniques for Combinatorial Problems, chapter Tabu Search*, C. Reeves, Blackwell Scientific Publishing, 1993.
- [JUN 01] JUNKER U., « QUICKXPLAIN : Conflict Detection for Arbitrary Constraint Propagation Algorithms », *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, Seattle, WA, USA, août 2001.
- [JUS 96] JUSSIEN N., BOIZUMAULT P., « Implementing Constraint Relaxation over Finite Domains using ATMS », JAMPEL M., FREUDER E., MAHER M., Eds., *Over-Constrained Systems*, n° 1106 Lecture Notes in Computer Science, Springer-Verlag, 1996, p. 265–280.
- [JUS 00a] JUSSIEN N., BARICHARD V., « The PaLM system : explanation-based constraint programming », *Proceedings of TRICS : Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, Singapore, septembre 2000, p. 118–133.
- [JUS 00b] JUSSIEN N., DEBRUYNE R., BOIZUMAULT P., « Maintaining Arc-Consistency within Dynamic Backtracking », *Principles and Practice of Constraint Programming (CP 2000)*, n° 1894 Lecture Notes in Computer Science, Singapore, septembre 2000, Springer-Verlag, p. 249–261.
- [JUS 01] JUSSIEN N., « e-constraints : explanation-based Constraint Programming », *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Paphos, Cyprus, 1 décembre 2001.
- [LAB 00] LABURTHE F., « CHOCO : implementing a CP kernel », *CP00 Post Conference Workshop on Techniques for Implementing Constraint programming Systems (TRICS)*, Singapore, septembre 2000.
- [MCA 93] MCALLESTER D. A., « Partial Order Backtracking », *Tech. Report, Artificial Intelligence Laboratory, MIT*, 1993.
- [OUI 02] OUIS S., JUSSIEN N., « Explications conviviales pour la programmation par contraintes », *Journées Francophones de Progamation en Logique et avec Contraintes (JFPLC'02)*, Nice, France, mai 2002, Hermès, p. 105–118.
- [PRO 95] PROSSER P., « MAC-CBJ : maintaining arc-consistency with conflict-directed backjumping », Research Report n° 95/177, 1995, Department of Computer Science – University of Strathclyde.
- [SCH 94] SCHIEX T., VERFAILLIE G., « Nogood Recording fot Static and Dynamic Constraint Satisfaction Problems », *International Journal of Artificial Intelligence Tools*, vol. 3, n° 2, 1994, p. 187–207.
- [SQU 96] SQUALI M. H., FREUDER E. C., « Inference-Based Constraint Satisfaction Supports Explanation », *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI '96)*, vol. 1, August 1996, p. 318–324, Portland, Oregon.
- [STA 77] STALLMAN R. M., SUSSMAN G. J., « Forward Reasoning and Dependency Directed Backtracking in a System for Computer-Aided Circuit Analysis », *Artificial Intelligence*, vol. 9, 1977, p. 135–196.