Techniques rétrospectives pour résoudre le Minimum Open Stacks Problem

Hadrien Cambazard, Narendra Jussien

▶ **To cite this version:**

Hadrien Cambazard, Narendra Jussien. Techniques rétrospectives pour résoudre le Minimum Open Stacks Problem. Deuxièmes Journées Francophones de Programmation par Contraintes (JFPC06), 2006, Nîmes - Ecole des Mines d'Alès / France. inria-00085778

HAL Id: inria-00085778

https://inria.hal.science/inria-00085778

Submitted on 14 Jul 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Techniques rétrospectives pour résoudre le Minimum Open Stacks Problem

Hadrien Cambazard **Narendra Jussien**

École des Mines de Nantes – LINA FRE CNRS 2729
4 rue Alfred-Kastler – BP 20722 – F-44307 Nantes – France
Hadrien.Cambazard@emn.fr Narendra.Jussien@emn.fr

Résumé

Nous présentons une étude de cas sur le MOSP (*minimum number of open stacks problem*). Ce problème se rencontre dans des environnements de production où la réalisation d'un ensemble de tâches nécessite une utilisation simultanée de différentes ressources (les piles – *stacks*). À travers ce problème, nous cherchons comment des raisonnements rétrospectifs assez classiques mais basés sur les explications peuvent être utilisés pour élaguer l'espace de recherche. Les *explications* ont, en effet, souvent été utilisées pour mettre en place des mécanismes sophistiqués de retour arrière dans le cadre de la programmation par contraintes mais très peu dans des schémas de type *nogood recording*. Ce n'est pas le cas dans la communauté SAT où la notion de clause apprise joue un grand rôle aussi bien pour l'élagage de l'espace de recherche que pour l'exploration elle-même. Dans cet article, nous introduisons un *nogood* généralisé (basé sur les explications) pour le MOSP qui nous permet d'introduire une nouvelle technique de résolution. Des résultats expérimentaux montrent l'intérêt d'une telle approche pour le MOSP et la capacité des explications à identifier et à exploiter dynamiquement la structure de ces problèmes.

1 Le problème : The Minimum number of Open Stacks

Le *Minimum number of Open Stacks Problem* (MOSP) se pose dans de nombreux environnements de production qui exigent l'utilisation simultanée de ressources (les *stacks*) pour réaliser un ensemble de tâches. Ce problème a récemment servi de support au challenge de modélisation en programmation par contraintes lors d'IJCAI 2005 [14]. Il s'agit d'un problème d'ordonnancement mettant en jeu un ensemble de produits et un ensemble de commandes clients. Une commande est constituée par un sous-ensemble spécifique de produits qui doivent être intégralement traités pour que la commande soit finalisée et envoyée au client.

Au moment où le premier produit d'une commande est mis en production, une ressource (*stack*) est mobilisée pour cette commande. Quand tous les produits requis par cette commande ont été traités, la ressource est libérée. La surface de la zone de production étant limitée, l'objectif est de minimiser le nombre maximum de ressources qui sont simultanément mobilisées (ou le nombre de commandes ouvertes simultanément).

Une solution du MOSP est donc un ordre total sur les produits décrivant la séquence de production qui minimise le nombre maximum de commandes ouvertes simultanément. Une instance est donné en exemple sur la table 1.

Nous introduisons ici l'ensemble des notations qui seront utilisées dans la suite du de l'article :

- P désigne l'ensemble complet des m produits qui peuvent être produits et C, l'ensemble des n commandes à satisfaire.
- $P(c)$ est l'ensemble des produits commandés par le client c. $C(p)$ est l'ensemble des commandes qui utilisent le produit p. Une extension naturelle de cette notation est faite pour des ensembles de produits ($C(s_P)$ est ainsi l'ensemble des commandes qui utilisent au moins un produit de s_P).
- $O^K(S)$ fait référence à l'ensemble des commandes qui sont ouvertes suite à la mise en production d'un sous-ensemble $S \subseteq K$ de produits. Pour simplifier les notations, $O(S)$ désignera par la suite $O^P(S)$.
- $f(S)$ est le nombre minimum de ressources nécessaires pour compléter un ensemble de commandes S et $f^A(S)$ est le nombre minimum de

TAB. 1 – Une instance 6×5 du MOSP avec une solution optimale de valeur 3 – Aucune colonne ne comporte plus de 3 "1" sur la représentation des ressources.

	Instance	Ordre optimal	Ressources
commandes \ produits	$P_1 P_2 P_3 P_4 P_5 P_6$	$P_1 P_2 P_6 P_4 P_3 P_5$	$P_1 P_2 P_6 P_4 P_3 P_5$
c_1	0 0 1 0 1 0	0 0 0 0 1 1	- - - - 1 1
c_2	0 1 0 0 0 0	0 1 0 0 0 0	- 1 - - - -
c_3	1 0 1 1 0 0	1 0 0 1 1 0	1 1 1 1 1 -
c_4	1 1 0 0 0 1	1 1 1 0 0 0	1 1 1 - - -
c_5	0 0 0 1 1 1	0 0 1 1 0 1	- - 1 1 1 1

ressources pour un ensemble S en considérant un ensemble A de commandes initialement actives (ouvertes).
- p_j désigne le produit affecté à la position j de la séquence de production.
- $open_j$ exprime le nombre de commandes ouvertes à l'instant j (*i.e.* à la position j dans la séquence de production).

Sur la base de ces notations, un modèle mathématique du problème s'écrit :

$$\begin{aligned} \min(\max_{j<m} \quad & open_j) \quad \text{s.t.} \\ & \forall j < m, \quad p_j \in [1..m] \\ & \forall j < m, \quad open_j \in [1..n] \\ & \text{alldifferent}(\{p_1, \ldots, p_m\}) \\ & open_j = |\{c, P(c) \cap \{p_0 \ldots p_j\} \neq \varnothing \\ & \qquad \wedge P(c) \cap \{p_j \ldots p_m\} \neq \varnothing\}| \end{aligned} \tag{1}$$

2 Quelques éléments pour la resolution du MOSP

Nous présentons à présent succintement les principaux résultats obtenus au cours du challenge qui semblent des éléments importants pour la résolution efficace du MOSP.

2.1 Techniques de résolution

Un grand nombre d'approches ont été proposées pour résoudre ce problème au cours du challenge de modélisation d'IJCAI 2005 [14]. L'une d'entre elles, identifiée par [7] et [3] est basée sur la programmation dynamique : on considère un ensemble S de produits qui ont été placés chronologiquement jusqu'à l'instant t depuis le début de la séquence ($|S| = t$ et les produits de S sont positionnés depuis l'instant 0 jusqu'à l'instant $t-1$). On peut remarquer que $f^{O(S)}(P-S)$ reste identique quelque soit la permutation de S. En effet, le problème $P - S$ est uniquement relié au problème P par l'ensemble de commandes actives à l'instant t : $O(S)$ qui ne dépend pas d'un ordre particulier de S (une commande c est en effet ouverte si $P(c) \cap S \neq \varnothing$ et $P(c) \cap (P-S) \neq \varnothing$). Cette observation permet d'obtenir une formulation assez naturelle du problème en programmation dynamique et la fonction objectif peut s'écrire récursivement[1] de la manière suivante :

$$f(P) = min_{j \in P}(max(f(P - \{j\}), |O(P - \{j\})|) \tag{2}$$

L'énorme avantage de cette approche est de passer d'un espace de recherche de taille $m!$ à un espace de taille 2^m puisqu'on limite l'exploration aux sous-ensembles de P. Du point de vue de la programmation par contraintes, si S est un **nogood**, *i.e.* un ensemble de produits dont l'infaisabilité a été démontrée (par rapport à la borne supérieure courante), toute permutation de S conduit au même sous-problème infaisable $P - S$. L'enregistrement de ces nogoods pendant une énumération chronologique de la séquence de production conduit à un même espace de recherche en 2^m. Les deux approches sont donc équivalentes dans la mesure où elles travaillent sur le même espace de recherche (mais diffèrent en terme d'implémentation).

L'approche *nogood recording* [12, 9] est simplement basée sur une énumération chronologique des p_i de p_1 à p_m. Une fois que la séquence, $p_1, \ldots, p_k$ est prouvée infaisable, elle est stockée en mémoire de manière à interdire toutes ses permutations dans la recherche future. [13] souligne cette idée tout en optant finalement pour un autre schéma de branchement. Des améliorations significatives de ces approches existent et ont été discutées en détail durant le challenge [14].

2.2 Pré-traitement

Une étape utile de pré-traitement peut être appliquée en retirant tous les produits p pour lesquels $\exists p', C(p) \subseteq C(p')$. Une telle procédure peut également être appliquée dynamiquement pendant la recherche :

[1] On considère que si $|P| = 1$ tel que $P = \{p\}$ alors $f(\{p\}) = |C(p)|$.

si S est l'ensemble courant des produits chronologiquement affectés jusqu'à l'instant t, on peut alors insérer immédiatement après S, les produits p tels que $C(p) \subseteq O(S)$ sans modifier la valeur optimale.

2.3 Bornes inférieures

Les bornes inférieures sont souvent basées sur le graphe de *co-demande* G qui est défini dans la littérature par [2]. Les nœuds de G sont associés à des commandes et un arc (i, j) est présent si et seulement si les commandes i et j partagent au moins un produit. De nombreuses bornes inférieures peuvent être définies sur ce graphe. Nous avons retenu les trois bornes suivantes :

- le degré minimum + 1 de G ;
- la taille de la clique maximum dans G (obtenue avec l'algorithme de Bron et Kerbosh [4]) ;
- la taille de la clique obtenue dans un mineur de G [2] par contraction d'arcs.

La dernière borne apparaît comme la plus puissante dans nos expérimentations, particulièrement avec une bonne heuristique de contraction d'arc sélectionnant les nœuds parmi ceux de degré minimal pour produire une clique au plus tôt dans la procédure de contraction.

Ces bornes peuvent aussi être utilisées dynamiquement pendant la recherche sur le problème restreint à $P - S$ en prenant en compte les commandes ouvertes à l'instant courant. Inspirée par [7], la nouvelle borne inférieure suivante s'est révélée très intéressante : soit G' la version *orientée* du graphe de co-demande (les arcs (c_1, c_2) et (c_2, c_1) existent tous les deux). Soit $d^+_{G'(c)}$ le degré sortant du nœud c dans G'. G' est défini de la manière suivante : $G' = (V, E - \{(c_1, c_2) | c_2 \in C(S)\})$, *i.e.* tous les arcs pointant vers des commandes ouvertes ont été retirés.

Définition 1 *Le nombre minimum de ressources requises pour l'ensemble $P - S$ de produits est au moins de :*

$$lb(P - S) = min_{c \in C(P-S)}(|O(S) \cup c| + d^+_{G'(c)}) \quad (3)$$

Preuve : Si c est la première commande fermée, alors, avant de fermer c, toutes les commandes adjacentes à c (les destinations des arcs sortants de c) seront ouvertes. c étant la première commande fermée, aucune de ces commandes adjacentes ne peut être fermée avant c, pas plus que les commandes de $c \cup O(S)$. Ces deux ensembles étant disjoints puisque aucune commande de $c \cup O(S)$ n'est adjacente à c (par définition de G'), la somme de leur taille est une borne inférieure du nombre de ressources ouvertes simultanément. □

2.4 Notre challenge

Aucune des approches proposées durant le challenge IJCAI 05 n'a mis en œuvre des techniques rétrospectives (backtrack intelligent ou s'appuyant sur un calcul d'explications d'échecs). Nous souhaitons montrer dans ce papier que le MOSP est un bon candidat pour de telles approches parce qu'il s'agit d'un problème structuré. Le nombre minimal de ressources est en effet souvent relié à de petits ensembles de produits. Ce papier présente donc une nouvelle approche pour ce problème s'appuyant sur des techniques rétrospectives en programmation par contraintes et offre de nouvelles perspectives sur les instances du challenge.

L'article est organisé de la manière suivante : nous introduisons dans un premier temps les raisonnements au cœur de notre approche pour la résolution du MOSP avant de définir précisément les *nogoods* généralisés qui vont être utilisés pour améliorer la recherche. Enfin, les résultats expérimentaux montrent que la technique proposée se comporte très bien sur les instances du challenge.

3 Analyser les échecs

Les nogoods et les explications ont été introduits depuis longtemps en programmation par contraintes pour améliorer la recherche. Plus récemment, certains travaux ont montré que l'analyse des échecs peut permettre d'appréhender dynamiquement certaines structures des problèmes [5]. Pour le MOSP, à partir d'un nogood donné S, on peut essayer de le généraliser et concevoir toute une classe de nogoods équivalents en exploitant certaines propriétés du problème. Nous avons exploré deux types de généralisation :

- calculer des sous-ensembles de S qui demeurent des *nogoods*.
- calculer des ensembles de *nogoods* équivalents à S en identifiant des conditions d'échecs plus précises ou en *expliquant* la preuve réalisée par le solveur pendant la recherche.

3.1 Calculer de plus petits nogoods

L'idée est de répondre à la question suivante : dès lors que $f^{O(S)}(P - S) \geq ub$[2] est établi, quelles sont les conditions sur S sous lesquelles cette preuve reste correcte ?

- Comme la valeur optimale $f^{O(S)}(P - S)$ dépend de $P - S$ et $O(S)$, retirer un produit de S qui ne

[2] ub désigne la borne supérieure courante (la valeur de la meilleure solution trouvée jusqu'a présent). On ne pourra donc pas identifier à partir de S de meilleure solution que la solution courante.

fait pas décroitre $O(S)$ fournit un nouveau nogood valable. L'ajout de ce produit à $P - S$ ne peut en effet que faire croître $f^{O(S)}(P - S)$. On peut donc chercher à calculer des sous-ensembles minimaux de S qui conservent $O(S)$ en appliquant par exemple un algorithme comme Xplain[3] [6]. Le tableau 2 en donne un exemple.

TAB. 2 – Exemple de réduction d'un nogood – $S = \{P_1, P_2, P_3, P_4, P_5\}$ est un nogood. P_1, P_2 et P_3 peuvent être retirés sans changer $O(S)$ de telle sorte que $\{P_4, P_5\}$ soit aussi un nogood. Dans ce cas particulier, les commandes déjà fermées induisent des produits inutiles dans le nogood.

	$P_1P_2P_3P_4P_5$	O(5)	...
c_1	1 0 0 1 0	1	
c_2	0 1 1 0 0	0	
c_3	0 0 1 0 1	1	...
c_4	1 1 0 0 0	0	
c_5	0 0 0 1 0	1	
c_6	0 0 0 0 1	1	

– Le calcul, pendant la recherche, de l'ensemble coupable de commandes (un sous-ensemble de $O(S)$) qui soit suffisant pour valider la preuve $f(P - S) \geq ub$ relève d'un calcul d'explications (voir la section suivante). Dans l'exemple précédent (tableau 2), en imaginant que les commandes 1, 3, et 6 soient suffisantes pour obtenir le nogood $\{P_1, P_2, P_3, P_4, P_5\}$, on pourrait en déduire que $\{P_1, P_5\}$ est également un nogood.

Le problème étant symétrique, la séquence peut être inversée sans perturber les raisonnements précédents. Les arguments donnés pour S sont aussi valides pour $P - S$. Par exemple, si S est un nogood alors $P - S$ l'est également ainsi que tous ses sous-ensembles qui conservent $O(S)$ intact (On a ainsi toujours $O^P(S) = O^P(P - S)$).

3.2 Calculer des nogoods équivalents

La question centrale devient à présent : dès lors que $f^{O(S)}(P-S) \geq ub$ est établi, quelles sont les conditions sur $P - S$ sous lesquelles cette preuve reste correcte ? Peut-on à partir de ces conditions, construire des ensembles plus larges de *nogoods* ?

Ce problème est rattaché aux explications. Une explication est un ensemble globalement inconsistant de contraintes (dont les contraintes de décisions, incluant les traditionnelles affectations variables-valeurs). Elle diffère du *nogood* qui est une instantiation partielle localement inconsistante.

3.2.1 Expliquer le MOSP

Ainsi, au lieu de se focaliser sur des conditions sur S qui conservent la validité de la preuve, on peut orienter l'explication sur $P - S$ en cherchant un sous-ensemble suffisant pour reproduire la preuve réalisée par le solveur. C'est en effet un problème symétrique et on peut concevoir l'explication comme un sous-ensemble des décisions qui repousseraient les produits de $P-S$ après l'instant t.

En l'absence de propagation, une valeur i peut être retirée de p_j pour uniquement trois raisons :

– $open_j$ est incompatible avec la borne supérieure courante ub. Une explication pour $p_j \neq i$ est un sous-ensemble des produits restants qui conserve ouvertes les commandes ouvertes à l'instant j. En rappelant que $S = \{p_0, .., p_j\}$, $expl(p_j \neq i)$ est défini par un sous-ensemble S' :
$expl(p_j \neq i) = S' \subseteq P - S$ tel que : $\forall k \in S'$, $|O^{S \cup S'}(S) \cup \{C(k)\}| \geq ub$.
Comme de nombreux ensembles S' existent souvent, on choisit un S' qui contient autant que possible les produits déjà inclus dans les explications existantes pour des instants inférieurs à j. La raison est que tous les produits participeront à l'explication finale sauf en cas de *back-jump* significatif au dessus des instants correspondants.

Exemple : *L'exemple 1 du tableau 3 met en scène un calcul d'explication.* $S = \{P_1, P_2\}$, $P - S = \{P_3, P_4, P_5, P_6, P_7\}$, $O(S) = \{c_2, c_3, c_4\}$. *A l'étape 1, la borne supérieure est de 4 et* $p_2 \neq P_2$ *à cause de* $f^{O(S)}(\{P_3, P_4, P_5, P_6, P_7\}) \geq 4$. *On peut néanmoins obtenir un ensemble plus précis que l'intégralité de* $P - S$. *En effet, la situation est inchangée tant que* $O(S)$ *est intact donc pour* $\{P_3, P_5, P_6\}$ *ou* $\{P_4, P_5, P_6\}$ *par exemple. On peut ainsi enregistrer* $expl(p_2 \neq P_2) = \{P_4, P_5, P_6\}$. *En se projetant à l'étape suivante, la recherche essaye en vain* $p_2 = P_3$ *et l'explication de cet échec pourrait à nouveau être constituée par* $\{P_4, P_5, P_6\}$ *ou* $\{P_2, P_5, P_7\}$. *En retenant la première explication, on obtient :* $expl(p_2 \neq \{P_2, P_3\}) = \{P_4, P_5, P_6\}$. *Une fois que le domaine de* p_2 *est vide, on retire* P_1 *de* p_1 *avec pour explication, l'union de toutes les explications calculées pour chaque valeur de* p_2.

– L'infaisabilité d'une permutation de la séquence $p_0, \ldots, p_j$ a déjà été prouvée. Une explication à donc déjà été calculée et attachée au nogood cor-

[3]Nous avons utilisé Xplain au lieu de QuickXPlain [8] pour des raisons d'implémentation. Comme nous le verrons par la suite, le goulot d'étranglement de notre technique réside davantage dans la gestion des nogoods appris pendant la recherche que dans leur calcul.

TAB. 3 – Exemple de calcul d'explications

	Exemple 1							Exemple 2			
	Etape 1			Etape 2							
	P_1P_2	$O(S)$	$P_3P_4P_5P_6P_7$	P_1P_3	$O(S)$	$P_2P_4P_5P_6P_7$		$P_1P_2P_3$	$O(S)$	P_4...
c_1	0 0	0	1 0 0 1 1	0 1	1	0 0 0 1 1	c_1	1 0 0	1	1
c_2	0 1	1	0 0 0 1 0	0 0	0	1 0 0 1 0	c_2	0 1 1	0	0
c_3	1 0	1	1 1 0 0 0	1 1	1	0 1 0 0 0	c_3	1 0 1	0	0 ...
c_4	1 1	1	0 0 1 0 0	1 0	1	1 0 1 0 0	c_4	0 1 0	1	0
c_5	0 0	0	0 1 1 0 1	0 0	0	0 1 1 0 1	c_5	0 0 1	1	1
							c_6	0 0 0	0	1

respondant.

– La borne inférieure $lb(\{p_{j+1}, \ldots, p_m\})$ est plus grande que ub. Dans ce cas, le graphe de codemande orienté G' est coupable. Une telle situation peut produire une explication plus précise que celle obtenue par la recherche car de nombreux ensembles de produits peuvent être à l'origine du même graphe puisque plusieurs produits sont généralement associés à un arc de G'. On peut donc à nouveau essayer de calculer un sous-ensemble des produits dont le graphe de codemande induit, SG, respecte la propriété qui a permis à la borne $lb(P-S)$ de provoquer un échec. Deux solutions ont été envisagées :
  – calculer une explication gloutonne en ajoutant un à un les produits les plus prometteurs. Les produits prometteurs étant ceux qui permettent de faire augmenter le degré minimal de SG et sont reliés aux commandes actives.
  – appliquer l'algorithme Xplain sur la liste des produits ordonnés sur le même critère que celui utilisé par l'algorithme glouton.

  Il est difficile pour le glouton de trouver un bon compromis entre l'ajout d'un produit qui inclurait une commande active dans SG (et augmenterait ainsi d'une unité l'évaluation de tous les nœuds de SG) et l'ajout d'un produit connecté entre les nœuds d'évaluation minimum de SG. La seconde solution a été retenue en dépit de son coût plus élevé, le but de notre article étant d'analyser le degré avec lequel le problème peut être précisément expliqué. Identifier un algorithme pour expliquer un raisonnement dans le cas d'un problème non pur n'est pas toujours une chose facile. Il manque notamment au glouton la capacité de critiquer son explication et de retirer des produits ajoutés au début et subsumés par d'autres produits nécessaires de l'explication.

De plus, les produits qui n'ont pas été nécessaires à la preuve (qui n'appartiennent pas à l'explication), peuvent être utilisés pour généraliser les conditions d'échecs. Sur le second exemple du tableau 3, P_4 peut être échangé avec $\{P_1, P_3\}$ si P_4 n'est pas utile pour prouver que $\{P_1, P_2, P_3\}$ est un nogood. $\{P_4, P_2\}$ est donc également un nogood. Des ensembles équivalents à S peuvent être obtenus à partir de l'explication et des sous-ensembles de S de manière à élaguer davantage l'espace de recherche pour la recherche future. Les sous-ensembles de S peuvent par ailleurs fournir un point de backtrack plus pertinent.

Les explications sont rattachées à l'idée qu'une instance du MOSP peut parfois présenter des structures très spécifiques. Un exemple d'une telle structure est le cas d'ensembles indépendants de produits. Il peut aussi y avoir de la redondance parmi les produits et de petits sous-ensembles de P peuvent être responsables de la valeur minimale (à l'image des problèmes structurés de Bayardo qui sont construits autour de petits sous-problèmes inconsistants [1]). Les explications fournissent une manière de tirer parti de ces structures (redondance et indépendance parmi P) dynamiquement pendant la recherche puisque ces situations peuvent apparaître suite aux décisions de l'algorithme. Les résultats donnés en dernière section sur la pertinence des explications confirment en partie cette hypothèse.

4 Nogoods généralisés pour le MOSP

Nous introduisons maintenant les *nogoods* spécifiques manipulés par l'algorithme de recherche et calculés sur la base des raisonnements précédents. Un *nogood* classique est défini comme une affectation partielle qui ne peut être étendue à une solution. Un tel nogood devient inutile dès qu'un de ses sous-ensembles est identifié en tant que *nogood* (il est dominé ou subsumé). Ce n'est pas le cas des *nogoods* présentés pour le MOSP car chacun d'entre eux n'est valide que depuis le début de la séquence. Ils seraient donc inutiles si ils n'interdisaient pas toutes leurs permutations.

Un *nogood* S du MOSP n'est donc pas dominé par l'un de ses sous-ensembles. Le nogood $\{1, 3, 4\}$ est un sous-ensemble de $\{1, 2, 3, 4\}$ mais n'interdit pas la

séquence $\{1,3,2,4\}$, ce qui rend utile $\{1,2,3,4\}$ mais en présence de $\{1,3,4\}$. Soit $\mathcal{P}(S)$ l'ensemble des sous-ensembles de S et $\mathcal{P}_k(S)$, les sous-ensembles de taille fixée k. Un nogood S est dit dominé dès lors que $U_{|S|}(S)$ est vrai avec :

$$\begin{cases} U_0(S) = false \\ U_i(S) = \bigwedge_{S_j \in \mathcal{P}_{i-1}(S)} (S_j \text{is a nogood} \vee U_{i-1}(S_j)) \end{cases} \quad (4)$$

Pour construire la séquence S, il est d'abord nécessaire de construire une séquence de taille $|S|-1$. Or toutes ces sous séquences sont soit interdites (elles correspondent à d'autres *nogoods*) soit innatteignables car toutes les permutations de tailles réduites menant à S sont elles-même interdites. S est donc dominé dans ces circonstances. Tout cela suggère que la taille des nogoods n'est probablement pas un critère de leur pertinence. Pour cette raison, les *nogoods* considérés pour notre problème sont définis de la manière suivante :

Définition 1 *Un nogood généralisé N est défini par une paire d'ensembles (R,T) (pour* root *et* tail*) qui interdit de commencer la séquence de production par toute permutation d'un ensemble appartenant à $\{R \cup T_i, T_i \in \mathcal{P}(T)\}$.*

Cette définition fournit une manière de factoriser l'information contenue dans un ensemble de nogoods (à travers la *queue* du nogood). L'objectif est de représenter de manière compacte un large ensemble de *nogoods* identifiés à chaque échec. La propriété suivante est utilisée pour caractériser les *nogoods* généralisés au moment d'un échec :

Propriété 1 *Si S est une séquence inconsistante de produits et $expl(S) \subseteq P - S$ une explication de cette situation alors la paire (R,T) telle que,*
- *$R \cap expl(S) = \emptyset$,*
- *$O(R) = O^{S \cup expl(S)}(S)$,*
- *$T \cap expl(S) = \emptyset$*

est un nogood généralisé valide.

Preuve : S étant un nogood, on a $f^{O(S)}(P-S) \geq ub$. Par ailleurs, $expl(S)$ est un sous-ensemble de $P-S$ de sorte que le problème restreint à $expl(S)$, après avoir affecté chronologicalement S, soit inconsistant. Ce constat nous amène à $f^{O^{S \cup expl(S)}(S)}(expl(S)) \geq ub$. En raison de $O(R) = O^{S \cup expl(S)}(S)$ et de $R \cap expl(S) = \emptyset$, l'inégalité précédente devient $f^{O(R)}(expl(S)) \geq ub$. On peut noter qu'il est ainsi possible d'incriminer les commandes coupables comme un sous-ensemble des commandes ouvertes $O^P(S)$. En effet $O(R) \subseteq O^P(S)$ puisque $S \cup expl(S) \subseteq P$. $f^{O(R)}(expl(S)) \geq ub$ montre que $(R, S-R)$ est un nogood généralisé valide même si $P-S$ est restreint à $expl(S)$. De plus, ajouter un produit non inclus dans $expl(S)$ à la queue de $(R, S-R)$ ne peut pas faire décroitre $O(R)$. Chaque commande de $O(R)$ étant en effet ouverte à cause d'au moins un produit de $expl(S)$. Ainsi (R,T) reste un *nogood* généralisé valide tant que $T \cap expl(S) = \emptyset$ □

En pratique, de tels nogoods sont obtenus en appliquant sur S (qui a été prouvée infaisable avec l'explication $expl(S)$) les raisonnements présentés dans la section précédente. De nombreux schémas de génération de nogoods peuvent être implémentés tant qu'ils respectent la propriété 1. Les procédures de génération de nogoods s'appuyent sur les deux principes suivant :
- Dériver de S plusieurs sous-ensembles $R^1, \ldots R^k$ tels que $\forall i,j\ R^i \not\subseteq R^j$ et $O^{S \cup expl(S)}(S)$ soit inchangé (les produits non inclus dans l'explication $expl(S)$ peuvent être utilisés également à cette fin). Soit $R^1, \ldots R^k$ les racines de S et $T^1, \ldots, T^k$ les queues de chaque R^k tel que $T^i = S - R^i$. Alors (R^k, T^k) est un nogood généralisé valide.
- Ajouter à chaque T^k, tous les produits non inclus dans R^k et inutiles pour la preuve *i.e.*, non inclus dans $expl(S)$. Le nogood final (R^k, T^k) est alors enregistré.

Le *back-jumping* peut être réalisé en utilisant la racine dont la dernière décision est la plus récente (dans la séquence chronologique). C'est le cas quand plusieurs commandes ont été fermées et que le problème demeure infaisable à cause de commandes ouvertes auparavant. Les expérimentations sont effectuées avec le schéma décrit figure 1. La fonction $minimize(S, O_p)$ calcul un sous-ensemble S' de S tel que $O_p \subseteq O(S')$ en s'appuyant l'algorithme Xplain[4]. Au plus, quatre (et au moins un) *nogood* généralisés sont générés à chaque échec. L'intérêt de R^1 est de fournir le meilleur point de *back-jump* alors que R^2 est celui qui a le plus de chance d'être minimal. Finalement R^3 et R^4 sont intéressants parce qu'ils se réfèrent à des produits qui n'ont jamais été utilisés à cet instant. Ainsi, sur l'exemple du tableau 2 le nogood produit serait égal à $(\{P_4, P_5\}, \{P_1, P_2, P_3\})$.

Notre schéma de génération s'efforce de produire des nogoods dont la racine est la plus courte et la queue la plus longue possible.

4.1 Gestion des nogoods généralisés

L'enregistrement et la gestion efficace des nogoods est toujours un problème délicat (les clauses apprises sont ainsi critiques pour l'efficacité des solvers SAT mais leur gestion et propagation est subtile [11]). Les nogoods généralisés définis à la section précédente correspondent à un nombre exponentiel de nogoods

[4] L'ordre de S est utilisé pour guider la génération d'un sous-ensemble parce que chaque élément est itérativement ajouté depuis le premier élément de S. Les premiers éléments de S sont donc favorisés.

procédure `nogoodGeneration`(S, E)
`entrée : S, une séquence infaisable`
`à cause de l'explication E`
`1. trier` S `par temps croissant`
`2.` $R^1 = minimize(S, O^{S \cup expl(S)}(S))$;
`3. trier` S `par temps décroissant`
`4.` $R^2 = minimize(S, O^{S \cup expl(S)}(S))$;
`5.` $\overline{E} = P - S - E$;
`6.` $R^3 = minimize(R^1 \cup \overline{E}, O^{S \cup expl(S)}(S))$;
`7.` $R^4 = minimize(R^2 \cup \overline{E}, O^{S \cup expl(S)}(S))$;
`8.` **forall** R^i, `ajouter le nogood` $(R^i, \{\overline{E} \cup S\} - R^i)$

FIG. 1 – Procédure de génération de nogoods.

simples utilisés par la programmation dynamique et il est impossible de les stocker tous individuellement. On dispose donc d'une collection C de nogoods généralisés, deux points principaux doivent être analysées :

- Comment vérifier efficacement qu'un ensemble S est inclus dans C. Un ensemble S est interdit si et seulement si $\exists (R^k, T^k) \in C$ tel que : $R^k \subseteq S \subseteq T^k \cup R^k$
- Comment ajouter efficacement un nouvel (R^k, T^k).

L'extraction de toutes les racines incluses dans S telles que leurs queues contiennent S est basée sur une forme simple d'automate fini déterministe connu sous le nom de TRIE [10]. Un TRIE est une structure de donnée arborescente ordonnée où un ensemble est associé à chaque nœud et déterminé par le chemin menant de la racine à ce nœud. Ainsi, tous les descendants d'un nœud ont un sous-ensemble commun associé à ce nœud. La figure 2(a) montre un exemple d'un TRIE. Identifier et ajouter un ensemble spécifique dans le TRIE s'effectue en $O(m)$ (m est le nombre de produits). En revanche, obtenir tous les sous-ensembles d'un ensemble donné n'est plus polynomial.

Le TRIE est utilisé de la manière suivante :

- Un nœud du TRIE est associé à une racine d'un nogood R^k et contient la liste de toutes les queues T^i telles que (R^k, T^i) est un nogood.
- Un nœud I est également attaché à un ensemble Un_I correspondant à l'union des racines et des queues de tous ses descendants.

La vérification d'un ensemble S est faite par une recherche en profondeur d'abord récursive dans le TRIE qui consume les éléments de S pour atteindre toutes les racines R^k incluses dans S. Pour chaque racine R^k atteinte, on vérifie simplement pour toutes les queues du nœud si $T^k \cup R^k \supseteq S$. La recherche est par ailleurs interrompue à tout nœud I dès que $Un_I \not\subseteq S$. Par exemple, le TRIE étant ordonné, les éléments $3, 4$ n'apparaîtront jamais dans un descendant de $\{1, 5\}$. Si de tels éléments appartiennent à S, ils doivent donc apparaître dans $Un_{\{1,5\}}$.

En pratique, le TRIE permet d'arrêter la recherche assez tôt et n'est pas coûteux à maintenir pour ajouter un nogood. Vérifier l'ensemble $\{1, 2, 3, 5\}$ ne demande que de parcourir les nœuds $\{1\}, \{1, 3\}, \{1, 5\}$ sur l'exemple (b) de la figure 4.1.

Nous utilisons le TRIE pour élaguer le domaine de l'instant suivant (chaque position étant représentée par les variables p_j). Cette règle de propagation très simple effectuée sur les nogoods appris est cependant très coûteuse par rapport à la programmation dynamique qui peut vérifier en temps constant si S est ou non un nogood. Une limitation très importante de notre approche actuelle est la vérification incrémentale d'un ensemble et l'élimination des nogoods dominés.

5 Résultats expérimentaux

Nos expérimentations sont réalisées sur les instances du challenge[5] sur un portable Pentium M cadencé à 1.7Ghz avec 1Go de RAM. Les algorithmes ont été implémentés en Java sur la base du solveur de contraintes choco (`choco-solver.net`)[6]. Toutes les instances présentées sont résolues optimalement et les petites instances de taille inférieure à 15_30 telles que les séries de 20_20 sont résolues en moins d'une seconde dans le pire des cas. Cependant les trois dernières instances $SP4$ restent ouvertes. On peut aussi noter, pour une bonne interprétation des résultats que les bornes inférieures sont très pertinentes dans la mesure où 4132 bornes inférieures sont égales à la valeur optimale sur 5802 instances. Ce résultat provient de notre heuristique spécifique de contraction des arcs qui sélectionne les arcs dont la somme des degrés de leurs extrémités est minimale de manière à produire un mineur de G égal à une clique le plus tôt possible.

5.1 Pertinence des explications

Nous analysons dans un premier temps la pertinence des explications en cherchant une explication de la valeur optimale du problème. L'explication fournit ainsi un sous-ensemble de produits suffisants pour expliquer cette valeur (le problème réduit à cet ensemble possède donc la même valeur optimale). Á des fins de comparaison, le problème est également résolu de manière répétée dans le cadre d'une approche de type

[5] disponibles sur *http ://www.dcs.st-and.ac.uk/~ipg/ challenge/*

[6] Une implémentation directe permettrait certainement d'avoir de bien meilleures performances mais ce choix nous permet de coupler facilement cette approche avec de la propagation (sujet non abordé dans cet article).

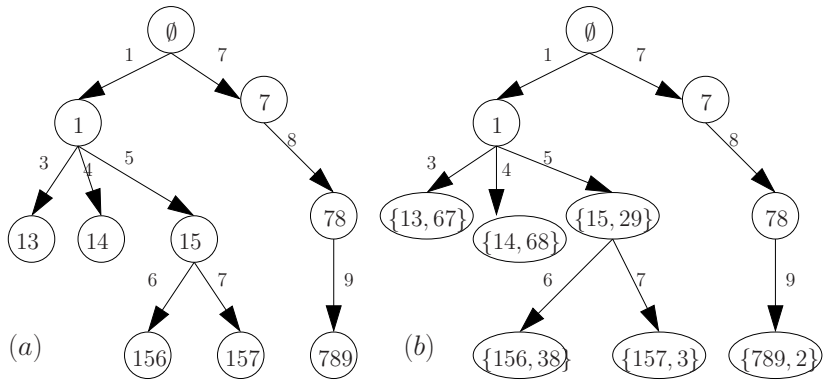

FIG. 2 – (a) Exemple d'un TRIE. (b) Exemples de nogoods stockés à l'aide d'un TRIE (une unique queue est représentée pour chaque racine).

Xplain. Une telle approche fournit au moins un ensemble minimal au sens de l'inclusion. Le pourcentage de réduction est donné au tableau 4 pour une sélection d'instances (2985 instances sur les 5802) de l'ensemble du jeu de tests originel[7]. Alors que Xplain est capable de calculer des explications plus petites (36.4% de produits éliminés au lieu de 22.6 % en moyenne), il est 40 fois plus lent[8] (8.8 secondes contre 0.3 secondes en moyenne) et reste incapable de traiter des instances plus larges telles que les 30_30. Par ailleurs, on peut souvent rapidement améliorer l'explication finale en résolvant une fois de plus le problème réduit à son explication. On peut noter que les instances de tailles n_m avec $m > n$ peuvent être de plus en plus réduites quand m augmente (l'inverse est aussi vrai).

La réduction est assez impressionnante dans certain cas montrant effectivement que les explications sont capables de capturer certaines structures du problème basées sur la redondance et l'indépendance de produits.

TAB. 4 – Pourcentage de réduction obtenu pour l'approche à base d'explications et l'approche Xplain.

		%reduc	
	NumInst	Explication	Xplain
wbo10_10	40	20,8	41,8
wbo10_20	40	29,8	48,4
wbo10_30	40	38,6	53,1
simonis15_30	120	53,7	62,5
wbo20_10	40	2,5	13
wbo30_10	40	0,5	6,3
wbo20_20	90	15,4	39
wbop20_20	90	13,1	28,6
wbp20_20	90	27,8	41,7
simonis20_20	220	39	51,8
testset	2985	22,6	36,4

5.2 Résolution

Les résultats sont donnés pour trois approches :
- NR : Le schéma de nogood recording simple équivalent à la programmation dynamique.
- GNR : Notre schéma d'enregistrement de nogoods généralisés couplé à du *back-jumping*.
- EXP : GNR augmenté avec le calcul complet des explications.

Les mesures moyennes (table 5) et maximales (table 6) du temps (en secondes) et l'espace de recherche (nombre de nœuds/points de choix ainsi que les backtracks) sont indiqués pour les plus dures instances du challenge (à l'exception de la série SP4). Ces résultats montrent clairement que le *back-jumping* couplé à l'enregistrement de nogoods généralisés est une amélioration des approches précédentes de nogood recording ou programmation dynamique. L'espace de recherche est significativement réduit (en moyenne par 61, 5% et jusqu'a 70%) et l'algorithme est 32% plus rapide en moyenne et 39% sur les instances les plus difficiles[9].

Cependant, il semble que l'utilisation des explications ne soit pas rentable. L'espace de recherche est

[7] Les instances plus larges, de tailles supérieures à 30_30 ont été exclues à cause des temps de calculs de Xplain.

[8] De plus les bornes intiales n'ont pas été expliquées (à l'exception de $max_{p \in P}|C(p)|$ qui est expliquée avec le produit correspondant p) donc l'explication est entièrement produite par recherche. Xplain à l'avantage de tirer parti des bornes inférieures.

[9] Nous avons néanmoins eu du mal à comparer nos résultats avec [7]. Le nombre de nœuds de l'approche d'enregistrement de nogoods simples est significativement plus petit et le nombre de backtrack plus grand que la mesure de l'effort de recherche indiqué dans [7] pour la programmation dynamique.

TAB. 5 – Moyennes des temps, nœuds et backtracks sur les instances difficiles du challenge

		NR			GNR			EXP		
Inst	OptM	TAvg (s)	NAvg	BkAvg	TAvg	NAvg	BkAvg	TAvg	NAvg	BkAvg
wbo15_30	11,58	0,1	304	4596	0,1	259	1750	0,2	228	1121
wbo30_30	22,56	4,1	10944	125732	2,6	8351	47467	16,3	7261	35858
wbop15_30	12,15	0,1	249	4114	0,1	241	1884	0,3	231	1439
wbop30_30	23,84	4,3	11319	132295	3	9589	62332	28,4	9043	52751
wbp30_30	24,46	4,6	12375	145366	3	9661	53151	24,8	8730	42694
simonis30_30	28,32	2,7	6400	82885	1,7	4904	25089	9,3	4047	17682
simonis40_20	36,38	0,2	1133	6728	0,1	821	3824	2	764	3534
nwrsLarger4	12,5	0,2	320	5447	0,4	315	2406	1	298	1683
gp50by50	38,75	0	49	20	0	49	19	0	49	15
gp100by100	76,25	2,4	1002	17065	1,6	732	3943	17,8	634	1765

TAB. 6 – Valeurs maximales des temps, nœuds, backtracks sur les instances difficiles du challenge

	NR			GNR			EXP		
Inst	Tmax (s)	NMax	BkMax	Tmax	NMax	BkMax	Tmax	NMax	BkMax
wbo15_30	1,2	2533	42493	1,2	2079	13286	1,2	1705	7737
wbo30_30	31,8	89601	1024219	19,1	64791	314141	103	54028	233466
wbop15_30	1,3	2415	44178	1,4	2317	16202	2,2	2150	10687
wbop30_30	35,9	105617	1053597	23,2	82491	435781	162	76188	374592
wbp30_30	64,1	168259	1994950	37,5	122575	538753	261	103699	420546
simonis30_30	27,5	77593	886794	14,2	50133	208991	70	39138	157517
simonis40_20	1	6196	37385	0,7	3847	16458	8,4	3587	15893
nwrsLarger4	0,9	1230	21788	1,7	1208	9624	4,2	1139	6734
gp50by50	0	101	47	0	101	46	0,1	101	35
gp100by100	8,1	3357	65240	5	2283	14186	56	1897	6143

réduit à nouveau (une réduction supplémentaire de 22%), confirmant la pertinence des explications analysée sur les petits problèmes. Cette technique reste néanmoins compétitive avec les meilleures approches par programmation par contraintes. Ce résultat est assez intéressant dans la mesure où l'algorithme est capable de fournir en plus une explication de la solution optimale du problème qui peut se révéler très intéressante pour un utilisateur final en mettant en lumière les sous-ensemble critiques de produits responsables du nombre minimum de ressources à mobiliser.

6 Conclusion

Les explications ont souvent été utilisées pour concevoir des algorithmes de backtrack intelligent en programmation par contraintes alors que leur utilisation pour élaguer l'espace de recherche a été moins explorée. Ce n'est pas le cas dans la communauté SAT où les clauses apprises jouent un rôle très important autant pour le *back-jump* que pour le filtrage. La limitation principale en programmation par contraintes vient du besoin de structures de données efficaces pour obtenir une représentation compacte des nogoods et limiter leurs besoins exponentiels en mémoire.

Nous avons évalué sur le MOSP comment les raisonnements de type *look-back* pouvaient être mis en œuvre sur un cas réel en soulignant la méthodologie. Des conditions de généralisation des échecs ont notamment été identifiées.

Les résultats expérimentaux démontrent l'intérêt d'une telle approche pour le MOSP : l'approche fondée sur le *back-jumping* est nettement plus performante que les approches basées sur la programmation dynamique. De plus, nous avons montré que les explications sont capables de capturer certaines structures du problème.

Nous pensons qu'il existe de nombreuses voies d'amélioration pour les explications qui pourraient éventuellement mener à un gain de temps. La structure de donnée courante pour les nogoods est un composant critique qui pourrait être énormément amélioré

en travaillant sur l'incrémentalité de la propagation des nogoods ainsi que le retrait des nogoods dominés ou leur résolution. La technique peut s'appliquer sur des variantes du problème dans la mesure où une propagation peut être ajoutée de manière naturelle à l'algorithme sous réserve de conserver l'exploration chronologique et d'expliquer les raisonnements de filtrage par rapport à $P - S$.

Références

[1] Roberto J. Bayardo and Robert Schrag. Using csp look-back techniques to solve exceptionally hard sat instances. In *Proceedings CP 1996*, pages 46–60, 1996.

[2] J.C. Becceneri, H.H. Yannasse, and N.Y. Soma. A method for solving the minimization of the maximum number of open stacks problem within a cutting process. *Computer and Operations Research*, 31 :2315–2332, July 2004.

[3] T. Benoist. A dynamic programming approach. In *IJCAI'05 Fifth Workshop on Modelling and Solving Problems with Constraints*, Edinburgh, Scotland, July 2005.

[4] Coen Bron and Joep Kerbosch. Algorithm 457 : finding all cliques of an undirected graph. *Commun. ACM*, 16(9) :575–577, 1973.

[5] Hadrien Cambazard and Narendra Jussien. Identifying and exploiting problem structures using explanation-based constraint programming. In *Proceedings (CP-AI-OR'05)*, volume 3524 of *LNCS*, pages 94–109, Prague, Czech Republic, 2005.

[6] J.L. de Siqueira and J.F. Puget. Explanation-based generalisation of failures. In *Proceedings (ECAI'88)*, pages 339–344, 1988.

[7] M. Garcia de la Banda and P. J. Stuckey. Dynamic programming to minimize the maximum number of open stacks. In *IJCAI'05 Fifth Workshop on Modelling and Solving Problems with Constraints*, Edinburgh, Scotland, July 2005.

[8] Ulrich Junker. QUICKXPLAIN : Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, Seattle, WA, USA, August 2001.

[9] G. Katsirelos and F. Bacchus. Generalized nogoods in csps. In *National Conference on Artificial Intelligence (AAAI-2005)*, pages 390–396, 2005.

[10] Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*, pages 492–512. Addison-Wesley, Reading, Massachusetts, third edition, 1997.

[11] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[12] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problem. *International Journal of Artificial Intelligence Tools*, 3(2) :187–207, 1994.

[13] P. Shaw and P. Laborie. A constraint programming approach to the min-stack problem. In *IJCAI'05 Fifth Workshop on Modelling and Solving Problems with Constraints*, Edinburgh, Scotland, July 2005.

[14] B. Smith and I. Gent. Constraint modelling challenge 2005. In *IJCAI'05 Fifth Workshop on Modelling and Solving Problems with Constraints*, Edinburgh, Scotland, July 2005.