Learning from the past to dynamically improve search: a case study on the MOSP problem

Hadrien Cambazard[1] and Narendra Jussien[2]

[1] Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland
h.cambazard@4c.ucc.ie
[2] École des Mines de Nantes – LINA CNRS UMR 6241
4 rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3, France
narendra.jussien@emn.fr

Abstract. This paper presents a study conducted on the minimum number of open stacks problem (MOSP) which occurs in various production environments where an efficient simultaneous utilization of resources (stacks) is needed to achieve a set of tasks. We investigate through this problem how classical look-back reasonings based on explanations could be used to prune the search space and design a new solving technique. *Explanations* have often been used to design intelligent backtracking mechanisms in Constraint Programming whereas their use in nogood recording schemes has been less investigated. In this paper, we introduce a generalized nogood (embedding explanation mechanisms) for the MOSP that leads to a new solving technique and can provide explanations.

1 The Minimum number of Open Stacks Problem

The Minimum number of Open Stacks Problem (MOSP) has been recently used to support the IJCAI 2005 constraint modeling challenge [14]. This scheduling problem involves a set of products and a set of customer's orders. Each order requires a specific subset of the products to be completed and sent to the customer. Once an order is started (*i.e.* its first product is being made) a *stack* is created for that order. At that time, the order is said to be *open*. When all products that an order requires have been produced, the stack/order is closed. Because of limited space in the production area, the maximum number of stacks that are used simultaneously, *i.e.* the number of customer orders that are in simultaneous production, should be minimized.

Therefore, a solution for the MOSP is a total ordering of the products describing the production sequence that minimizes the set of simultaneously opened stacks. This problem is known to be NP-hard and is related to well known graph problems such as the minimum path-width or the vertex separation problems.

Moreover, it arises in many real life problems as packing, cutting or VLSI design. Table 1 gives an example instance of this problem.

The explanations provided in this paper for the MOSP are related to the *removal explanations* of Ginsberg which are the basis of dynamic backracking [4]. It does not provide natural language explanations for a user but is intended to improve the search by avoiding redundant computation. Indeed by carefully *explaining* why a failure occur, one can avoid a number of other failures that would occur for the same reason. It provides at the same time a justification of optimality for an expert. In the case of the MOSP we propose to define an explanation as a subset of products that would account for the minimal number of open orders. As long as those products are processed, a planner knows that there is no way to reduce the minimum number of simultaneous stacks needed. In other words, it provides insights about the bottleneck in the optimal planning.

The following notations will be used throughout the paper:

Table 1. A 6×5 instance of MOSP with an optimal solution of value 3 – no more than 3 ones can be seen at the same time on the Stacks representation. The *instance* and *optimal ordering* parts of the table are to be read as for example, product P_3 has been ordered by customers c_1 and c_2. The *stacks* part shows that for example the order of custumer c_3 is open from the production of product P_1 to the production of product P_3 (in this part of the table, all 1s are consecutive representing the open nature of the stack representing the order). Another example is to consider the order of customer c_2 which remains open only during the production of product P_2 as this order is composed only with this product.

	Instance						Optimal ordering						Stacks					
	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

- P : the set of m available products and C, the set of n orders.
- $P(c)$ is the set of products required by order c. $C(p)$ is the set of orders requiring product p. A natural extension of this last notation is used for sets of products ($C(s_P)$ is the set of orders that requires at least one product in s_P).
- $O^K(S)$ denotes the set of open orders implied by a subset $S \subseteq K$ of products: $O^K(S) = |C(S) \cap C(K - S)|$. $O(S)$ is a short notation for $O^P(S)$. The open orders therefore refer to a certain point in time where a set of products have already been processed.
- $f(S)$ is the minimum number of stacks needed to complete a set S and $f^A(S)$ is the number of stacks needed to complete set S assuming a set A of initially active (opened) orders.

– p_j denotes the product assigned to position j in the production sequence and $open_j$ expresses the number of open orders at time j. Those variables take their value in a set called their original domain ($D^{orig}(x)$ for variable x). This set is reduced to a singleton thanks to the solving process (using propagation). The current domain is denoted $D(x)$.

Using those notations, one can provide a mathematical model of the problem:

$$\begin{array}{ll} \min(\max_{j<m} open_j) & \text{s.t.} \\ \quad \forall j,\ 0 < j \leq m, & p_j \in [1..m] \\ \quad \forall j,\ 0 < j \leq m, & open_j \in [1..n] \\ \quad \text{alldifferent}(\{p_1, \ldots, p_m\}) & \\ \quad open_j = |C(\{p_1, \ldots, p_j\}) \cap C(\{p_j, \ldots, p_m\})| & \end{array}$$

2 First insights in solving the MOSP

We give a rapid review of the main results obtained for solving the MOSP during the IJCAI 2005 challenge.

2.1 Search techniques

A wide variety of approaches were proposed for solving this problem during the IJCAI 2005 constraint modelling challenge [14]. One of the most efficient has been identified by [3] and [2]. It is based on dynamic programming (DP): consider a set S of products that have been placed chronologically up to time t from the beginning of the sequence ($|S| = t-1$ and products from S are set from slot 1 to slot $t-1$). Then, one can notice that $f^{O(S)}(P-S)$ remains the same for any permutation of S. Indeed, problem $P-S$ is only related to problem P by the active set of orders at time t: $O(S)$ which does not depend on any particular order of S (an order c is indeed open if $P(c) \cap S \neq \varnothing$ and $P(c) \cap (P-S) \neq \varnothing$). This fact gives a natural formulation of the problem in DP and the objective function can be recursively[3] written as: $f(P) = min_{j \in P}(max(f(P-\{j\}), |O(P-\{j\})|)$. The strong advantage of this approach is to switch from a search space of size $m!$ to one of size 2^m because one only need to explore the subsets of P. From a constraint programming point of view, if S is a **nogood**, *i.e.* a set of products that has been proven as infeasible (according to the current upper bound), any permutation of products of S will lead to an infeasible subproblem $P-S$. Storing such nogoods during a chronological enumeration of the production sequence leads to the same search space size of 2^m.

2.2 Preprocessing and lower bounds

A useful preprocessing step can be applied by removing any product p such that $\exists p', C(p) \subseteq C(p')$. This can even be done during search: if S is the current set

[3] We consider that if $|P| = 1$ with $P = \{p\}$, $f(\{p\}) = |C(p)|$.

Algorithm 1 : NogoodRecMOSP($\{p_1, \ldots, p_{t-1}\}$)

```
1.If t − 1 ≠ m do
2.  For each i ∈ D(p_t) then
3.    p_t ← i; ∀k > t, remove i from D(p_k);
4.    S ← {p_1, ..., p_t};
5.    Try
6.      filter(S,p_{t+1});
7.      NogoodRecMOSP(S);
8.    Catch (Contradiction c)
9.      add nogood {p_1, ..., p_t};
10.   EndTryCatch;
11. EndFor
12.Else store new best solution, update ub;
13.throw new Contradiction();
```

of already chronologically assigned products up to time t, then one can assign immediately after S the products p such that $C(p) \subseteq O(S)$.

Lower bounds are often based on the *co-demand graph* G which has been defined in the literature in [1]. The nodes of G are associated to orders and an edge (i, j) is added if and only if orders i and j share at least one product. Several lower bounds can be defined from this graph, and we use the size of a clique obtained as a minor of G [1] by edge contraction as it appeared the most effective in our experimentation. These bounds may also be used during search on the problem restricted to $P - S$ by taking into account the current open orders. We used the ones given in [3].

2.3 Our solution

None of the proposed approaches during the IJCAI 05 Challenge involved look-back techniques (intelligent backtrackers or explanation-based techniques). We intend to show in this paper that the MOSP is a good candidate for these approaches because it is a structured problem, and the optimal number of stacks is often related to small kernels of products.

We will first introduce the simple nogood recording scheme and the main ideas of look-back reasonings for the MOSP. Second, we will define formally the generalized nogood and the two related backjumping algorithms. Experimental results finally show that the proposed look-back techniques perform well on the challenge instances.

3 Simple nogood recording

The nogood recording approach (see [11, 7] for a general description of the nogood recording technique) is simply based on the chronological enumeration of

Algorithm 2 : filter($S = \{p_1, \ldots, p_t\}$, p_{t+1})

```
1. For each i ∈ dom(p_{t+1}) then
2.   If |O^P(S) ∪ C(i)| ≥ ub or S ∪ {i} is a nogood
3.     remove i from D(p_{t+1});
4. EndFor
```

p_i from p_1 to p_m. Algorithm 1 takes as input a partial sequence of assigned products and tries to extend it to a complete sequence. If the sequence has not been completed yet (line 1), all remaining products for slot p_t will be tried (line 2). A filtering step is then performed and if no contradiction occurs, the algorithm goes on (recursive call line 7). The filtering applied in line 6 is minimal and only prunes the next time-slot according to the current upper bound ub (*i.e* the value of the best solution found so far) and the known nogoods (Algorithm 2). Once a sequence, $p_1, \ldots, p_k$ is proved as infeasible (line 9), it is stored so that all permutations will be forbidden in future search. [12] outlines this fact while finally choosing another enumeration scheme. Line 13 is called to backtrack when the domain of p_t has been emptied by search or when a new solution is found (line 12) to prove its optimality. Computation of lower bounds and the use of dominance rules on including products should be included in line 6 and a heuristic to order the products in line 2. This basic algorithm will be improved step by step in the following.

4 Learning from failures

Nogoods and explanations have long been used in various paradigms for improving search [4, 11, 10, 6]. In the MOSP, from a given nogood S, we can try to generalize it and design a whole class of equivalent nogoods to speed up search.

4.1 Computing smaller nogoods

The idea is to answer the following question: once the fact that the minimum number of stacks needed to complete the set of remaining products $P - S$ to schedule considering the open orders $O(S)$ due to the sequence of products S is greater that the current upper bound (in other words that $f^{O(S)}(P - S) \geq ub$ – line 9 of Algorithm 1), what are the conditions on S under which this proof remains valid? In other words, what are the conditions (subsets) in the already scheduled orders that makes this combination a not optimal one considering the current upper bound?

As the optimal value $f^{O(S)}(P - S)$ depends on $P - S$ and $O(S)$, removing a product from S that does not decrease $O(S)$ provides another valid nogood. Indeed adding the corresponding product to $P-S$ can only increase $f^{O(S)}(P-S)$. We can therefore compute some minimal subsets of S that keep $O(S)$ by applying the Xplain algorithm [13, 5]. Table 2 gives an example.

Table 2. Example of nogood reduction: we consider here a new instance. We only consider here the first 5 products and give the open stacks ($O(S)$ column (1 if the order is still open, 0 otherwise) when considering the sequence $S = \{P_1, P_2, P_3, P_4, P_5\}$. Suppose that S is a nogood. It is quite obvious that we can remove production (they will be produced later) the production of products P_1, P_2 and P_3 without modifying the open stacks at the end of sequence of the remaining products ($\{P_4, P_5\}$). Indeed, for example for customer c_1 the order will remain open as at least one of its products has been produced, for customer c_2 as the whole set of products is removed, the stack will remain closed, etc. Therefore, here, $\{P_4, P_5\}$ is also a nogood.

	P_1	P_2	P_3	P_4	P_5	O(S)	...
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	

4.2 Computing equivalent nogoods

The main question now becomes: once $f^{O(S)}(P - S) \geq ub$ has been proven, what are the conditions on $P - S$ under which this proof remains valid? Can we build from those conditions larger sets of nogoods? In other words, what are the conditions (subsets) on the products remaining to schedule that makes the current situation on not optimal one considering the current upper bound.

This problem relies on explanations. Instead of computing some conditions S_1 on S which can be seen as the decisions made so far, we compute some conditions S_2 on $P - S$ which can be seen as original constraints of the problem. A contradiction on S is therefore logically justified by $S_1 \cup S_2$. Only S_2 needs really to be stored within the explanation because S_1 can be computed from scratch at each failure and is resolved by search.

Definition 1 *Let $S = p_1, \ldots, p_{j-1}$ be a sequence of products and $S' = p_1, \ldots, p_j$ a sequence that extends S with $p_j = i$. An explanation for the removal of a value i from p_j, $expl(p_j \neq i)$ is defined by a set E, $E \subseteq P - S$ such that $|O^{S \cup E}(S) \cup C(i)| \geq ub$ or $f^{O(S')}(E - \{i\}) \geq ub$ (in other words, the remaining problem reduced to E is infeasible).*

All filtering mechanisms must now be explained. In the simple case of Algorithm 2, a value i can be removed from $D(p_j)$ if $open_j$ is incompatible with the current upper bound ub. An explanation is therefore only a subset of the remaining products that keep open the open orders at time j. If $S = \{p_0, .., p_{j-1}\}$, $E = expl(p_j \neq i)$ is in this case defined as :

$$|O^{(S \cup E)}(S) \cup C(i)| \geq ub$$

As $open_{j-1}$ is compatible with ub, once S is proved infeasible (both by search and pruning), $expl(p_{j-1} \neq k) = \bigcup_{v \in D^{orig}(p_j)} expl(p_j \neq v)$.

Example: Consider the first example in table 3. $S = \{P_1, P_2\}$, $P - S = \{P_3, P_4, P_5, P_6, P_7\}$, $O(S) = \{c_2, c_3, c_4\}$. On step 1, the upper bound is currently 4 and $p_2 \neq P_2$ because $f^{O(S)}(\{P_3, P_4, P_5, P_6, P_7\}) \geq 4$. This is however also true as long as $O(S)$ is unchanged. It is the case with $\{P_3, P_5, P_6\}$ or $\{P_4, P_5, P_6\}$. All values v of p_3 are removed by filtering and $\{P_4, P_5, P_6\}$ is recorded for each $expl(p_3 \neq v)$ so that $expl(p_2 \neq P_2) = \{P_4, P_5, P_6\}$. Going a step further, the search tries $p_2 = P_3$ and an explanation such as $\{P_4, P_5, P_6\}$ or $\{P_2, P_5, P_7\}$ is computed. The first set leads to $expl(p_2 \neq \{P_2, P_3\}) = \{P_4, P_5, P_6\}$ and the process goes on.

Table 3. Example of explanation computation

	Example 1														Example 2				
	Step 1								Step 2										
	P_1	P_2	$O(S)$	P_3	P_4	P_5	P_6	P_7	P_1	P_3	$O(S)$	P_2	P_4	P_5	P_6	P_7		P_1	P_2	P_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

For a filtering due to a nogood N, an explanation $expl(N)$ has already been recorded. A contradiction raised by the lower bound needs also to be *explained*. Xplain can be again applied on the products of $P - S$ to find a subset that respects the needed property.

For each infeasible sequence S, by explaining the proof made on $P - S$, one may first incriminate only a subset $O^P(S)$ that could be used to derive a more accurate subset of S leading to the same contradiction (and a more relevant point to backtrack). Second, it can be useful to generalize the nogood based on the products that are *not involved* in the explanation. In the second example of Table 3, P_4 can be exchanged with $\{P_1, P_3\}$ if P_4 is not needed to prove that $\{P_1, P_2, P_3\}$ is a nogood. Therefore, $\{P_4, P_2\}$ is also a nogood. Equivalent sets to S provided by explanations as well as subsets could therefore allow the pruning of future paths of the search.

Explanations rely on the idea that independency and redundancy among P can lead to small subsets of P having the same optimal value. Explanations provide a way to take advantage of these structures.

5 Generalized nogoods for the MOSP

A *classical* nogood, defined as a partial assignment that can not be extended to a solution, becomes useless as soon as one of its subset becomes a nogood. This is however not true for the nogoods presented above for the MOSP. The nogood $\{P_1, P_3, P_4\}$ is a subset of $\{P_1, P_2, P_3, P_4\}$ but does not forbid the sequence $\{P_1, P_3, P_2, P_4\}$. A MOSP nogood is indeed a sequence of products that forbids to *start* the production sequence by any of its *permutations*. With the subsets of a set S denoted by $\mathcal{P}(S)$, the nogoods considered for our problem are defined as follows:

Definition 2 *A generalized nogood N is defined by a pair of sets (R, T) (root and tail) which forbids to start the production sequence by any permutation of a set belonging to $\{R \cup T_i, T_i \in \mathcal{P}(T)\}$.*

This definition provides a way of factorizing information regarding a set of nogoods (into the *tail* part). It is meant to capture a large number of identified nogoods. The following proposition is used to characterize generalized nogoods when confronted to infeasible sequences of products.

Proposition 1 *If S is an infeasible sequence of products and $expl(S) \subseteq P - S$ an explanation of this fact then a pair (R, T) such that,*

- $(R \cup T) \cap expl(S) = \emptyset$,
- $O^{S \cup expl(S)}(S) \subseteq O(R)$,

is a valid generalized nogood.

Proof: As S is a nogood, $f^{O(S)}(P - S) \geq ub$. Moreover, $expl(S)$ is a subset of $P - S$ such that, after assigning chronologically S, the remaining problem restricted to $expl(S)$ is infeasible. This leads to $f^{O^{S \cup expl(S)}(S)}(expl(S)) \geq ub$. Due to $O^{S \cup expl(S)}(S) \subseteq O(R)$ and $R \cap expl(S) = \emptyset$ the previous inequality becomes $f^{O(R)}(expl(S)) \geq ub$. This inequality shows that $(R, S - R)$ is a valid generalized nogood even if $P - S$ is restricted to $expl(S)$. Moreover, adding products not included in $expl(S)$ to the tail of $(R, S - R)$ cannot decrease $O(R)$. Each order of $O(R)$ is indeed active because of at least one of the product of $expl(S)$. So (R, T) remains a valid nogood as long as $T \cap expl(S) = \emptyset$. □

In practice, such nogoods are obtained by applying to S the reasonings presented in the previous section. This leads to the algorithms detailed in the following.

5.1 Generalized nogood recording

To implement the above idea, lines 8-10 of Algorithm 1 are modified to introduce the computation of the generalized nogood and the backjumping feature. The following pseudo-code of algorithm 3 assumes that a contradiction c is labeled by the *level* where it occurs (c.level in line 9a), in other words, infeasibility

Algorithm 3 extends lines 8-10 of algorithm 1.

```
8.  Catch (Contradiction c)
9a.   If t < c.level
9b.     R^1 ← minimize(S, O^P(S));
9c.     R^2 ← minimize({p_t, p_{t-1}, ..., p_1}, O^P(S));
9d.     ∀j ∈ {1,2} ad nogood (R^j, S − R^j);
9e.     newLevel ← argmax_k(p_k ∈ R_1)
9f.     If newLevel < t
9g.       throw new Contradiction(newLevel);
9h.   Else if (t > c.level)
9i.     throw new Contradiction(c.level);
10. EndTryCatch;
```

Algorithm 4 : filter($S = \{p_1, \ldots, p_t\}$, p_{t+1})

```
1. For each i ∈ D(p_{t+1}) do
2.   If |O^P(S) ∪ C(i)| ≥ ub
4.     remove i from p_{t+1};
5.     expl(p_{t+1} ≠ i) ← E s.t |O^{S∪E}(S) ∪ C(i)| ≥ ub;
6.   Else If S ∪ {i} is a nogood N;
7.     remove i from p_{t+1};
8.     expl(p_{t+1} ≠ i) ← expl(N);
9. EndFor
```

of $p_1, \ldots p_k$ proved by the empty domain of p_{k+1} would raise a contradiction labelled by $k+1$ (throw new Contradiction(k+1)).

The function $minimize(S, O_p)$ computes $S^{'}$, a subset of S, such that $O_p \subseteq O(S')$ based on the Xplain technique. Moreover the order of S is used to guide the generation of the subset of S. If $S = p_1, \ldots p_i$, it ensures that $argmax_k(p_k \in S^{'})$ is minimal[4]. Two nogoods, based on the roots R^1 and R^2, are recorded at each contradiction. The purpose of R^1 is to provide the best backjumping point (as the latest product within R_1 will be as early as possible) whereas R^2 is the one with the best chance of being minimal (as the contradiction may only involve recent products, S is reversed to focus the subset on the last added products). Backjumping is ensured in line 9h by raising immediately a contradiction if the guilty level is not reached.

5.2 Explanation-based generalized nogood recording

Let us go a step further to develop the above ideas. First, Algorithm 4 replaces Algorithm 2 to explain the pruning due to ub and already known nogoods (whose explanations are computed line 9j of Algorithm 5). We also assume that a contradiction c is labeled by its explanation (c.exp).

[4] There is no subsequence of S with a product p_j s.t $j < k$.

Algorithm 1 is extended again when getting a contradiction to deal with explanations leading to algorithm 5. A contradiction explanation is computed in line 9b from the empty domain of p_{t+1}. This explanation will be recorded to explain the removal of the value i that has been tried for p_t (refer to Algorithm 1 for i, p_t and S) when the corresponding level is reached (lines 9n, 9o). Then, at most 4 nogoods are recorded. R^1, R^2 are the same as previously except that $S \cup E$ can be more precise than P for $O^{S \cup E}(S)$ (lines 9c, 9d). However, this should be very rare without more advanced filtering[5] and the main improvements of explanations are to further generalize the nogoods. This generalization occurs in lines 9f, 9g and 9i when using $\overline{E}$ to build the roots R^3,R^4 and $\{\overline{E} \cup S\} - R^i$ to build the tail.

Algorithm 5 extends lines 8-10 of algorithm 1.

```
8.  Catch (Contradiction c)
9a.   If t < c.level
9b.     E ← ⋃_{j∈D^orig(p_{t+1})} expl(p_{t+1} ≠ j);
9c.     R^1 ← minimize(S, O^{S∪E}(S));
9d.     R^2 ← minimize({p_t, p_{t−1}, ..., p_1}, O^{S∪E}(S));
9e.     Ē ← P − S − E;
9f.     R^3 ← minimize(R^1 ∪ Ē, O^{S∪E}(S));
9g.     R^4 ← minimize(R^2 ∪ Ē, O^{S∪E}(S));
9h.     for each j ∈ {1, 2, 3, 4}
9i.       add nogood N_j = (R^j, {Ē ∪ S} − R^j);
9j.       expl(N_j) ← E;
9k.     newLevel ← argmax_k(p_k ∈ R^1)
9l.     If newLevel < t
9m.       throw new Contradiction(newLevel, E);
9n.     Else expl(p_t ≠ i) ← E
9o.   Else if (t = c.level) expl(p_t ≠ i) ← c.exp;
9p.   Else throw new Contradiction(c.level, c.exp);
10. EndTryCatch;
```

The generalized nogood of Definition 2 corresponds to an exponential number of simple nogoods used in DP and it is impossible to store them all individually. We use a simple form of a finite automaton called a TRIE [8] to store them and perform the pruning. Storing and efficiently managing nogoods is always a challenging problem. SAT solvers [9] provides interesting results in that matter that are however difficult to apply in our case.

[5] This may occur due to the lower bound for example.

Table 4. Average and maximum results (time and backtracks) on hard benchmarks from the challenge for NR, GNR and EXP. OptAvg represents the average of the optimum.

Inst	OptAvg	TAvg (s)	Tmax	BkAvg	BkMax
		NR			
simonis_30_30	28.32	1.5	14.5	67617.3	708845
wbo_30_30	22.56	2.1	16.2	99586.9	760602
wbop_30_30	23.84	2.3	18.9	109465.9	829886
wbp_30_30	24.46	2.7	35.5	125704.7	1618700
		GNR			
simonis_30_30	28.32	1.6	14.8	23246.3	196229
wbo_30_30	22.56	2.1	14.2	44181.0	283491
wbop_30_30	23.84	2.8	22.6	59874.1	402049
wbp_30_30	24.46	3.0	35.5	51621.3	504488
		EXP			
simonis_30_30	28.32	8.4	64.3	18126.0	167322
wbo_30_30	22.56	13.0	88.0	35818.9	238380
wbop_30_30	23.84	28.0	177.1	52267.2	361704
wbp_30_30	24.46	25.5	256.9	43836.2	431215

6 Experimental results

Our experiments are performed on the challenge instances [6] on a laptop MacBook, 2Ghz processor with 2Gb of RAM. The algorithms have been implemented in Java.

We first analyze the accuracy of explanations by looking at the explanation of optimality. It gives a subset of P such that the problem reduced to this set will have the same optimal value as the original problem. For comparison reasons, the problem was also iteratively solved within an Xplain scheme and both approaches were tried. Xplain was able to derive shorter explanations with 35.1% of products removed on average against 21.1% for our explanation based approach but is unpractical on larger instances. These rates can reach 51.8% and 40% on simonis20_20 benchmark for example demonstrating that some problems can be very structured.

Secondly, results are given for the three approaches: NR (the existing simple nogood recording which is closely related to DP), GNR (Algorithm 3) and EXP (Algorithm 5). All instances except the last $SP2$, $SP3$ and $SP4$ instances are solved optimally, and smaller instances than 15_30 such as 20_20 ones are all solved in less than one second in the worst case. Average and maximum measures of time (in seconds) and search effort (backtracks) are indicated for the hardest instances of the challenge in Table 4. The search space reduction implied by the backjumping and the generalized nogood recording (GNR) is huge (on average by $55,5\%$ and at most by 64%) with however similar time results. This clearly

[6] http://www.dcs.st-and.ac.uk/~ipg/challenge/

shows the accuracy of the technique and time improvements could be obtained by improving the nogoods computation and management which remain currently quite naive. The use of explanations (EXP) is clearly more costly. The search space is nevertheless reduced again confirming the accuracy of explanations analyzed on smaller problems. But this technique remains competitive with the best pure constraint programming approaches while computing an explanation at the same time. It is therefore able to highlight hard subsets of products responsible for the minimum number of open stacks.

7 Conclusion: beyond the MOSP

The main assumptions for the results given here on the MOSP are related to the chronological enumeration and the nature of the objective function. We believe that side constraints could be naturally added (*e.g.* precedences among orders) and any propagation scheme could be performed as long as it is *explained* on $P - S$.

We investigated on the MOSP, how classical look-back reasonings based on explanations could be used to prune the search space. We focused our attention on deriving a set of conditions which generalize a given failure to a whole class of failures. The experimental results demonstrate the interest of such an approach for the MOSP even if no time improvements have been obtained yet. We believe that the dynamic programming-based approaches could therefore be improved by the ideas presented in the paper. There exist many ways to speed up GNR and EXP which could eventually lead to a time gain. The current data structure storing the nogoods is a critical component which could be vastly improved by allowing incremental propagation. This remains to be done as we first investigate the accuracy of explanations for this problem. Moreover, many nogood generation schemes could be designed.

References

1. 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. *C&OR*, 31:2315–2332, July 2004.
2. T. Benoist. A dynamic programming approach. In *IJCAI'05 Fifth Workshop on Modelling and Solving Problems with Constraints*, 2005.
3. M. Garcia de la Banda and P. J. Stuckey. Using dynamic programming to minimize the maximum number of open stacks. In *INFORMS Journal of Computing*, 2007.
4. M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
5. Ulrich Junker. QuickXplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, 2001.
6. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.
7. G. Katsirelos and F. Bacchus. Generalized nogoods in csps. In *National Conference on Artificial Intelligence (AAAI-2005)*, pages 390–396, 2005.
8. Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*, pages 492–512. Addison-Wesley, 1997.
9. 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.
10. P. Prosser. MAC-CBJ: maintaining arc consistency with conflict-directed backjumping. Technical Report Research Report/95/177, Dept. of Computer Science, University of Strathclyde, 1995.
11. T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problem. *IJAIT*, 3(2):187–207, 1994.
12. 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*, 2005.
13. J.L. De Siqueira and J.F. Puget. Explanation-based generalisation of failures. In *(ECAI'88)*, pages 339–344, 1988.
14. B. Smith and I. Gent. Constraint modelling challenge 2005. In *IJCAI'05 Fifth Workshop on Modelling and Solving Problems with Constraints*, 2005.