A filtering algorithm for global sequencing constraints

Jean-Charles RÉGIN and Jean-François PUGET

ILOG S.A.
9, rue de Verdun, BP 85
94253 Gentilly Cedex, FRANCE
e-mail : regin@ilog.fr

Abstract. Sequencing constraints have proved very useful in many real-life problems such as rostering or car sequencing problems. They are used to express constraints such as: every sequence of 7 days of work must contain at least 2 days off. More precisely, a global sequencing constraint (gsc) C is specified in terms of an ordered set of variables $X(C) = \{x_1, ..., x_p\}$ which take their values in $D(C) = \{v_1, ..., v_d\}$, some integers q, min and max and a given subset V of $D(C)$. On one hand, a gsc constrains the number of variables in $X(C)$ instantiated to a value $v_i \in D(C)$ be in an interval $[l_i, u_i]$. On the other hand, a gsc constrains for each sequence S_i of q consecutive variables of $X(C)$, that at least min and at most max variables of S_i are instantiated to a value of V. In this paper, we propose an automatic reformulation of a gsc in terms of global cardinality constraints. This is equivalent to defining a powerful filtering algorithm for a gsc which deals with a part of the globality of the constraint. We illustrate the power of our approach on a set of difficult car sequencing problems.

1 Introduction

1.1 Constraint Network and filtering algorithm

A constraint network (CN) consists of a set of variables, each of them associated with a domain of possible values, and a set of constraints which link up the variables and define the set of combination of values that are allowed. The search for an instantiation of all variables that satisfy all the constraints is called a Constraint Satisfaction Problem (CSP), and such an instantiation is called a solution of a CSP.

A lot of problems can be easily coded in term of CSP. For instance, CSP has already been used to solve problem of scene analysis, placement, resource allocation, crew scheduling, time tabling, scheduling, frequency allocation, car sequencing ... A recent paper of Simonis [Sim96] presents a survey on industrial studies and applications developed over the last ten years.

Unfortunately, CSP is an NP-Complete problem. Thus, many works have been carried out in order to try to reduce the time needed to solve a CSP. Some of the proposed methods modify the original CSP for a new CSP that has the same set of solutions, but which is easier to solve. The modifications are done by filtering algorithms, that remove values of variables that cannot belong to any solution of the current CSP. So, if the cost of such an algorithm is less than the time required by the backtrack algorithm to discover many times the same inconsistency, then the resolution will be speeded up.

The most widely used filtering algorithm is based on arc consistency[Mac77]. Some efficient algorithms achieving arc consistency for binary or n-ary constraints have been developed for twenty years. However, by taking into account the semantics of the constraints, more efficient filtering algorithm especially dedicated to each kind of constraint can be devised. For instance, [VDT92] have presented AC-5 a general framework to exploit efficiently some knowledge about binary constraints.

Moreover, the use of the semantics is often the only way to avoid memory consumption problems when dealing with non binary constraints. In fact, this approach prevents you from explicitly represent all the combinations of values allowed by the constraint. Last, specific filtering algorithms make it possible to use Operations Research techniques or graph theory. This is the case for the diff-n and cumulative constraints of CHIP [BC94], for ILOG Scheduler based on the edge-finder algorithm [Nui94], or for the IlcAllDiff and IlcDistribute constraints of ILOG Solver [Rég94, Rég96].

In this paper we are interested in global sequencing constraints and we propose such a filtering algorithm for them.

A global sequencing constraint (gsc) C is specified in terms of a ordered set of variables $X(C) = \{x_1, ..., x_p\}$ which take their values in $D(C) = \{v_1, ..., v_d\}$, some integers q, min and max and a given subset V of $D(C)$. On one hand, a gsc constrains the number of variable in $X(C)$ instantiated to a value $v_i \in D(C)$ be in an interval $[l_i, u_i]$. On the other hand, a gsc constrains for each sequence S_i of q consecutive variables of $X(C)$, that at least min and at most max variables of S_i are instantiated to a value of V

These constraints arise in many real-life problems such as car sequencing and rostering problems where a lot of min/max constraints have to be verified for each period of q consecutive time units. In a rostering problem, one has to chose a type of work for each day, satisfying various constraints. Sequencing constraints are useful for expressing regulations such as:

- each sequence of 7 days must contain at least 2 days off.
- A worker cannot work more than 3 night shifts every 8 days.

1.2 The car sequencing problem

Car sequencing problems arise on assembly lines in factories in the automotive industry.

There, an assembly line makes it possible to build many different types of cars, where the types correspond to a basic model with added options. In that context, one type of vehicle can be seen as a particular configuration of options.

Without loss of generality, we can assume that it is possible to put multiple options on the same vehicle while it is on the line. In that way, virtually any configuration (taken as an isolated case) could be produced on the assembly line. In contrast, for practical reasons (such as the amount of time needed to do so), a given option really cannot be installed on every vehicle on the line. This constraint is defined by what we call the "capacity" of an option. The capacity of an option is usually represented as a ratio p/q where for any sequence of q cars on the line, at most p of them will have that option.

The problem in car sequencing then consists of determining in which order corresponding cars should be assembled, while keeping in mind that we must build a certain number of cars per configuration.

For instance, consider the following version of the problem [DSV88]:

- 10 cars to build;
- 5 options available for installation;
- 6 configurations required.

The following chart indicates which options belong to which configuration: • indicates that configuration j requires option i; a blank means configuration j does not require option i. The chart also shows the capacity of each option as well as the number of cars to build for each configuration.

option	capacity	configurations					
		0	1	2	3	4	5
0	1/2	•				•	•
1	2/3			•	•		•
2	1/3	•				•	
3	2/5	•	•		•		
4	1/5			•			
number of cars		1	1	2	2	2	2

For example, the chart indicates that option 1 can be put on at most two cars for any sequence of three cars. Option 1 is required by configurations 2, 3, and 5. We define the demand for an option to be the sum of the demand for the configurations requiring that option. For instance, the demand for option 1 is equal to 6.

With gsc we can easily formulate this problem by:

- Creating a variable for each car that should be assembled on the line. The domain of these variables are the set of possible configurations. The variable array represents the sequence of the cars on the assembly line and is denoted by X.
- Creating for each option j with a capacity p_j/q_j a gsc defined on X that constrains on one hand, the number of times a configuration is assigned to a variable in X to be equal to its demand. On the other hand, this gsc constrains for each sequence of q_j consecutive variables of X that at least 0

and at most p_j variables of the sequence are instantiated to a configuration which requires the option j.

Some systems provide an efficient way to define constraints similar to gsc, see [BC94] for instance. However, an easy way to define a global constraint does not necessarily mean that the system handles efficiently the constraint as a global constraint. For instance, consider the model consisting of ([Smi96]):

- adding a variable o_{ij} with domain $\{0, 1\}$ for each variable x_i of X.
- adding a constraint between x_i and o_{ij} such that $o_{ij} = 1$ iff x_i is instantiated to a configuration that requires option j.
- adding several constraints in order to ensure that the capacity constraint of the option is satisfied. That is if the capacity is p_j/q_j and for each sequence $x_{i_1}, ..., x_{i_{q_j}}$ of q_j consecutive variables, a constraint ensuring that the sum of the variables $o_{i_1 j}, ..., o_{i_{q_j} j}$ is at most p_j.

These constraints are equivalent to the set of gsc, i.e. they accept the same set of solutions. However the filtering is less powerful as each filtering is done independently from the others. Thus, in practice, a lot of inconsistencies are detected by these systems only when a lot of variables are instantiated.

A possible solution to this problem is to add ***implied constraints***. An implied constraint for a given CSP is a constraint that can be deduced from the other constraints of the CSP, but which introduces a filtering algorithm that can reveal inconsistencies which are not discovered by the combination of the filtering algorithms of the other constraints. So the introduction of implied constraints can lead to a reduction of the number of backtracks needed to find one solution or to prove that there is none.

The introduction of implied constraints is quite important for the resolution. For instance, it is said in [DSV88]: "[the introduction of implied constraints] will improve drastically the efficiency of the basic program by detecting failures as soon as possible".

In this paper, we propose to reformulate the CSP using the powerful global cardinality constraint [Rég96], in order to obtain more powerful filtering algorithm.

First, we give some preliminaries and we formally define the global sequencing constraints. After, we propose a filtering algorithm for them. Then, for the car sequencing we introduce new variable and value orderings. At last, we present a few experiments and we conclude.

2 Preliminaries

A finite ***constraint network*** $\mathcal{P}$ is defined as a set of n ***variables*** $X = \{x_1, ..., x_n\}$, a set of current finite ***domains*** $\mathcal{D} = \{D(x_1), ..., D(x_n)\}$ where $D(x_i)$ is the set of possible ***values*** for variable x_i, and a set $\mathcal{C}$ of ***constraints*** between variables.

Then, a constraint C on the ordered set of variables $X(C) = (x_{i_1}, ..., x_{i_k})$ is a subset of the Cartesian product $D(x_{i_1}) \times ... \times D(x_{i_k})$ that specifies the allowed combinations of values for the variables $x_{i_1} \times ... \times x_{i_k}$. An element of $D(x_{i_1}) \times ... \times D(x_{i_k})$ is called a tuple of $X(C)$. $T(C)$ denotes the set of tuples of $D(x_{i_1}) \times ... \times D(x_{i_k})$ that are allowed by C.

Let $\mathcal{P} = (X, \mathcal{D}, \mathcal{C})$ be a constraint network, $C \in \mathcal{C}$ be a constraint:

• C is ***consistent*** iff $T(C) \neq \varnothing$.

• (x, v) is ***consistent with*** C iff $x \notin X(C)$ or $\exists t \in T(C)$ such that v appears in t at the same position as x in $X(C)$; otherwise (x, v) is ***inconsistent*** with C.

• C is ***arc consistent*** iff $\forall x_i \in X(C), \forall v \in D(x_i)$, v is consistent with C

For convenience, we will denote by:

• $D(C)$ the union of domains of variables of $X(C)$;

• $\#(v, t)$ the number of occurrences of the value v in the tuple t;

• $\#(v, Y)$ the number of times the value v can be assigned to a variable of Y;

• $\#(v, t, S)$ the number of times the value v is assigned to a variable of S in the tuple t.

In the following we will use global cardinality constraints [Rég96]. A ***global cardinality constraints*** (gcc) is defined by the minimal and the maximal number of occurrences of each value of $D(C)$ in each tuple of the constraints. The minimal and the maximal number of occurrences of each value can be different from the others.

Definition 1 *A* ***global cardinality constraint*** *is a constraint C in which each value $v_i \in D(C)$ is associated with two positive integers l_i and u_i and*
$T(C) = \{$ *t such that t is a tuple of $X(C)$*
and $\forall v_i \in D(C) : l_i \leq \#(v_i, t) \leq u_i\}$
It is noted $gcc(X(C), l, u)$, where $l = \{l_i\}$ and $u = \{u_i\}$.

A global sequencing constraint is a gcc for which for each sequence S_i of q consecutive variables of X, the number of variables of S_i instantiated to any value $v_i \in V \subseteq D(C)$ must be in an interval $[min, max]$.

Definition 2 *A* ***global sequencing constraint*** *is a constraint C associated with three positive integers min, max, q and a subset of values $V \subseteq D(C)$ in which each value $v_i \in D(C)$ is associated with two positive integers l_i and u_i and*
$T(C) = \{$ *t such that t is a tuple of $X(C)$*
and $\forall v_i \in D(C) : l_i \leq \#(v_i, t) \leq u_i$
and for each sequence S of q consecutive
variables: $min \leq \sum_{v_i \in V} \#(v_i, t, S) \leq max\}$
It is noted $gsc(X(C), V, min, max, q, l, u)$, where $l = \{l_i\}$ and $u = \{u_i\}$.

3 Filtering algorithm

From the previous definitions, one can see that global sequencing constraints are close to global cardinality constraints, because they are more general but also because for each sequence of consecutive variable we can define a particular gcc. Our approach is to reformulate the gsc in terms of a set of gcc using a transformation of the constraint network.

This reformulation is in fact equivalent to producing a specialized filtering algorithm for the gsc. This filtering algorithm is the combination of the filtering algorithms of the resulting reformulated problem.

For the sake of clarity, we explain the transformation using a single gsc $gsc(X(C), V, min, max, q, l, u)$.

Definition 3 *Let S be a sequence of consecutive variables of $X(C)$ such that $|S| \leq q$. An* ***abstract value*** *of S, denoted by $e(S)$, is a value that does not belong to $D(C)$. An* ***abstract variable*** *of a variable x of S w.r.t $e(S)$ is a variable y with a domain equal to $V \cup \{e(S)\}$ such that*

$$y = x \quad xor \quad y = e(S)$$

$Y(e(S), S)$ denotes the set of the abstract variables of the variables of S w.r.t. $e(S)$.

The constraint "$x = y$ xor $y = e(S)$" is equivalent to the constraint "if $x \in V$ then $x = y$ else $y = e(S)$".

In other words, either x and y are instantiated to the same value belonging to V, or x is instantiated to a value not in V and y is instantiated to $e(S)$. We can then prove the following property.

Property 1 *Let $gsc(X(C), V, min, max, q, l, u)$ be a gsc and S be a sequence of consecutive variables of $X(C)$ s.t. $|S| \leq q$ then*

$$|S| - max \leq \#(e(S), Y(e(S), S)) \leq q - min$$

proof:

Suppose that all the variables of S are instantiated. By construction, the number $\#(e(S), Y(e(S), S))$ is exactly the number of variables x_i in S that are instantiated to a value not belonging to V. Thus, the number of variables x_i in S that are instantiated to a value of V is: $\#(V, X) = |S| - \#(e(S), Y(e(S), S))$.

Assume $|S| = q$.
From the definition of the gsc, we know that $\#(V, X)$ must be in the interval $[min, max]$, i.e. $min \leq |S| - \#(e(S), Y(e(S), S)) \leq max$. After rewriting we obtain $q - max \leq \#(e(S), Y(e(S), S)) \leq q - min$.

Assume now $|S| < q$.
Let R be any sequence of $q - |S|$ consecutive variables such that $S \cup R$

forms a sequence of q consecutive variables and $e(S) = e(R)$. We just proved $min \leq |S| + |R| - \#(e(S), Y(e(S), S) \cup Y(e(R), R)) \leq max$. This equation is equivalent to $|S| + |R| - \#(e(R), Y(e(R), R)) - max \leq \#(e(S), Y(e(S), S)) \leq |S| + |R| - \#(e(R), Y(e(R), R)) - min$. By definition $0 \leq \#(e(R), Y(e(R), R)) \leq |R|$, thus $|S| - max \leq \#(e(S), Y(e(S), S)) \leq |S| + |R| - min$, which is equivalent to $|S| - max \leq \#(e(S), Y(e(S), S)) \leq q - min$. $\odot$

Now, let us see how to express the gsc as a whole by a set of global cardinality constraints. This can be done easily by splitting $X(C)$ into a partition of sequences $\mathcal{P} = \{S_1, ..., S_p\}$ and by choosing the abstract values associated with each sequence pairwise distinct. In this case, each $e(S_i)$ can only be assigned to variables of $Y(e(S_i), S_i)$. Then we can define a gcc on the variables $\cup_{S_i \in \mathcal{S}} Y(e(S_i), S_i)$ in which the values v_k of V are constrained by l_k and u_k, and the abstract values $e(S_i)$ are constrained by $|S_i| - max$ and $q - min$.

More formally we have:

Definition 4 *Consider*

- $\mathcal{P} = \{S_1, ..., S_p\}$ *a partition of* $X(C)$ *into sequences of at most* q *consecutive variables,*
- $E(\mathcal{P}) = \{e(S_i)$ *s.t.* $S_i \in \mathcal{P}$ *and* $S_i \neq S_j \Rightarrow e(S_i) \neq e(S_j)\}$ *a set of pairwise distinct abstract values,*
- $Y(E(\mathcal{P}), \mathcal{P}) = \bigcup_{S_i \in \mathcal{P}} Y(e(S_i), S_i)$

The ***global sequential cardinality constraint*** $gscc(\mathcal{P}, V, E(\mathcal{P}), q, min, max, l, u)$ *is the global cardinality constraint* $gcc(Y(E(\mathcal{P}), \mathcal{P}), l, u)$*, where* $l = \{l_i\} \cup \{|S_i| - max\}$ *and* $u = \{u_i\} \cup \{q - min\}$*.*

This transformation is interesting if the abstract values are really constrained, that is if $|S_i|$ is close to q. Since no condition have been made on the partition $\mathcal{P}$, such a result can be easily obtained. For instance, if $|X(C)| = q.p + r$ with, p and $r < q$ two positive integers, we can define $\mathcal{P} = \{S_1, S_2, ..., S_{p+1}\}$ by grouping into S_1 the variables with index from 1 to r of $X(C)$ and then by grouping into S_i, $1 < i \leq p + 1$, the variables with index from $r + 1 + q(i - 1)$ to $r + qi$.

With such a definition of $\mathcal{P}$ the gscc will take globally into account all the sequences belonging to $Y(E(\mathcal{P}), \mathcal{P})$. So, we will have a powerful filtering algorithm for this group of sequence constraints, because we can achieve the arc consistency for the gscc. However, we do not take into account all the possibles sequences of length q.

In order to obtain this result we propose to define a set of q partitions into sequences, such that any sequence of q consecutive variables of $X(C)$ belongs to exactly one partition, and then to define the gscc for each partition.

Each partition is defined in a way similar to the way we have just presented ($n = |X(C)|$):
The first partition $\mathcal{P}_1$ is defined by the sequence S_{11} that contains only the first variable of $X(C)$ and by the sequences S_{1i}, $1 < i \leq \lfloor 1 + (n-1)/q \rfloor$, that contain the variables from $1 + (i - 1)q$ to iq of $X(C)$, and eventually by the sequence that contains the variables of $X(C)$ not contained in the previous sequences. The

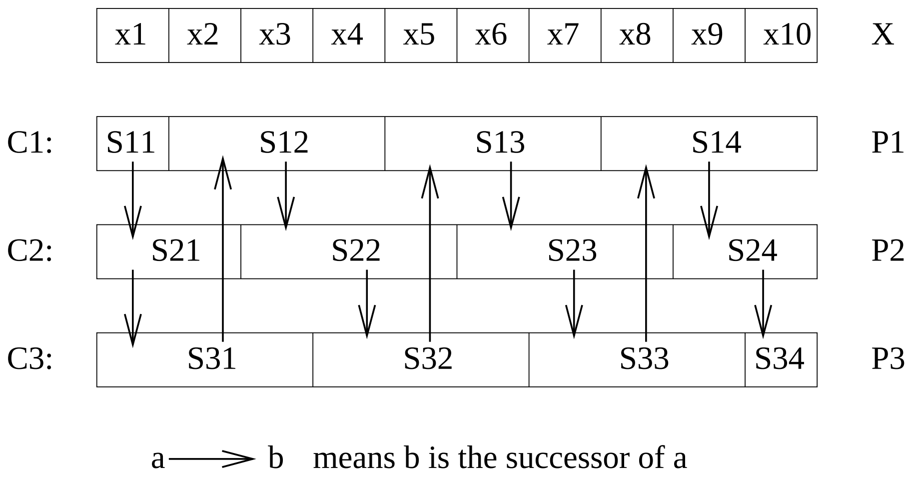

Fig. 1. Construction of $\mathcal{Q}$ for option 2.

second partition $\mathcal{P}_2$ is defined by the sequence S_{21} that contains the first and the second variables of $X(C)$ and by the sequences S_{2i}, $1 < i \leq \lfloor 2 + (n-1)/q \rfloor$, that contain the variables from $2 + (i-1)q$ to $1 + iq$ of $X(C)$, and eventually by the sequence that contains the variables of $X(C)$ not contained in the previous sequences. The other sequences are defined similarly.

Definition 5 *Given a sequence of variables $X(C) = \{x_k\}$, $k = 1..n$, and an integer q, the* ***sequence of order*** *q, i, j of $X(C)$, denoted by $Seq(X(C), q, i, j)$, is the sequence of variables of $X(C)$ with index ranging from $max(1, i+1+q(j-2))$ to $min(n, i+q(j-1))$, where $i = 1..q$, $j \geq 1$ and $i+1+q(j-2) \leq n-q+1$. For a given value of i, the partition of $X(C)$ defined by the sequences $Seq(X(C), q, i, j)$, $j \geq 1$ and $i+1+q(j-2) \leq n-q+1$ is denoted by $Partition(X(C), q, i)$.*

Figure 1 represent all such sequences for $n = 10$ and $q = 3$.

We then define a gscc for each partition, using the abstract variables corresponding to each sequence in the partitions.

Definition 6 *Let C be gsc defined on $X(C)$ and associated with V, min, max, l, u and q. With the notation introduced in definition 4 and 5 we define for each value i from 1 to q:*

- *$\mathcal{P}_i = Partition(X(C), q, i)$;*
- *$E_i = E(\mathcal{P}_i)$ such that $i \neq j \Rightarrow E_i \cap E_j = \emptyset$ (all the abstract values are pairwise different);*
- *$Y_i = Y(E(\mathcal{P}_i), \mathcal{P}_i)$;*
- *K_i is the set of constraints needed to define the abstract variables Y_i of $X(C)$ (see definition 3);*

- $C_i = gscc(\mathcal{P}_i, V, E_i, q, min, max, l, u)$

The constraint network

$$\mathcal{Q} = (X(C) \cup (\bigcup_{i=1..q} Y_i), D(C) \cup (\bigcup_{i=1..q} E_i), \bigcup_{i=1..q} C_i \cup \bigcup_{i=1..q} K_i)$$

is called ***the basic constraint network associated with a gsc.***

We have the following proposition:

Proposition 1 *Let $C = gsc(X(C), V, min, max, q, l, u)$ be a gsc defined on $X(C)$, $\Pi = (X(C), D(C), \{C\})$ be a constraint network and $\mathcal{Q}$ the basic constraint network associated with C then, Π is satisfiable iff $\mathcal{Q}$ is satisfiable.*

proof:

We show first that each solution of Π can be extended into a solution of $\mathcal{Q}$. Consider a variable x_k of $X(C)$ instantiated to a value v in a given solution of Π. For each abstract variable y_{ki} of x_k, if $v \in V$, we instantiate y_{ki} with the abstract value from which y_{ki} has been defined, else (if $v \notin V$), we instantiate y_{ki} with v. These assignments satisfy the constraints in $\mathcal{Q}$.
Conversely, consider a solution of $\mathcal{Q}$. Consider x_k of $X(C)$. If one abstract variable y_{ki} of x_k is such that $y_{ki} = v$ and v is not an abstract value, then because of the constraints between x_k and each of the y_{ki}, all the $y_{ki} = v$, and $x = v$. In the remaining case, we have that for all i, y_{ki} is instantiated to an abstract value and $x = v_k$ s.t. $v_k \notin V$. Then the assignments for the x_k satisfy the gsc.$\odot$

So, a gsc can be filtered by applying the arc consistency to the basic constraint network associated with it. However, we can still improve our model. In fact, we have shown that a gsc can be represented by a set of gscc. But the communication between these gscc is poor, because we do not take into account the fact that several gscc are defined for one gsc. We can add some other constraints that link the different abstract values.

Definition 7 *Let S be any sequence, we will denote by $\#e(S)$ the variable equal to $\#(e(S), Y(e(S), S))$. We also say that the* ***successor*** *of a sequence of abstract variables $Y(e(S), S)$ with $S = Seq(X(C), q, i, j)$ is the sequence of abstract variables $Y(e(R), R)$ with $R = Seq(X(C), q, i + 1, j)$ if $i < q$, and that the successor of $Y(e(S), S)$ with $S = Seq(X(C), q, q, j)$ is $Y(e(R), R)$ with $R = Seq(X(C), q, 1, j + 1)$.*

Intuitively, the successor of a sequence is in fact the same sequence translated by one variable (cf. figure 1).

For convenience, we will respectively denote by $first(O)$ and $last(O)$ the first and the last variable of any ordered set O.

Proposition 2 *Let $\mathcal{Q}$ be the basic constraint network associated with a gsc, F be a sequence $Y(e(S), S)$, L be its successor, then:*

- *if* $|F| = |L| + 1$ *then*
  - $first(F) = e(F) \Leftrightarrow \#e(F) = \#e(L) + 1$
  - $first(F) \neq e(F) \Leftrightarrow \#e(F) = \#e(L)$
- *if* $|F| = |L|$ *then*
  - $first(F) = e(F)$ *and* $last(L) \neq e(L) \Leftrightarrow \#e(F) = \#e(L) + 1$
  - $((first(F) = e(F)$ *and* $last(L) = e(L))$
    or $(first(F) \neq e(F)$ *and* $last(L) \neq e(L))) \Leftrightarrow \#e(F) = \#e(L)$
  - $first(F) \neq e(F)$ *and* $last(L) = e(L) \Leftrightarrow \#e(F) = \#e(L) - 1$
- *if* $|F| = |L| - 1$ *then*
  - $last(L) = e(L) \Leftrightarrow \#e(F) = \#e(L) - 1$
  - $last(L) \neq e(L) \Leftrightarrow \#e(F) = \#e(L)$

sketch of proof:

Let F_X and L_X be the sequences of variables of $X(C)$ respectively from which F and L have been created (if $F = Y(e(S), S)$ then $F_X = S$ for instance). Then $\#e(F) = |\{x \in F_X s.t. D(x) \cap V = \varnothing\}|$ and $\#e(L) = |\{x \in L_X s.t. D(x) \cap V = \varnothing\}|$. Consider, for instance, the case $|F| = |L| + 1$. We have $F_X = \{first(F_X)\} \cup L_X$. Suppose $first(F) = e(F)$ then $D(first(F_X)) \cap V = \varnothing$ so $\#e(F) = 1 + |\{x \in (F_X - \{first(F_X)\}) s.t. D(x) \cap V = \varnothing\}| = 1 + |\{x \in L_X s.t. D(x) \cap V = \varnothing\}| = 1 + \#e(L)$. By a similar reasoning we can prove the other cases. $\odot$

Definition 8 *Let C be a gcc and $\mathcal{Q}$ be the basic network associated with C. The constraint network $\mathcal{R}$ equal to $\mathcal{Q}$ in which the constraints in the proposition 2 have been added is called the* ***enhanced constraint network associated with*** *C.*

Finally we have the property:

Proposition 3 *Let $C = gsc(X(C), V, min, max, q, l, u)$ be a gsc, and $\mathcal{R}$ the enhanced constraint network associated with C then, $\mathcal{R}$ is satisfiable iff $\mathcal{Q}$ is satisfiable.*

The proof of this proposition follows directly from the fact that the constraints in proposition 2 are implied by the gsc.

4 Experiments for the car sequencing problem

Our experiments are done using Ilog Solver, a commercial constraint programming library that supports the definition of new constraint types.

As filtering is incomplete, i.e. it removes some inconsistent values but not every inconsistent values, the backtracking algorithm must still be used. We use a variant of the backtracking algorithm, known as MAC, where filtering is applied at each node in the search tree [PL95]. Although filtering prunes drastically the tree search, the order in which the tree is explored is very important. A poor ordering will lead to poor performance. Some very general variable orderings are quite satisfactory for random problems, such as the first-fail principle (select first the variable with the smallest domain).

For the car sequencing problem, value ordering prove to be quite important. In a previous work, Smith noted [Smi96]: "[The] variables should be assigned consecutively; leaving gaps in the sequence to be assigned later, will cause severe difficulties. Hence, any kind of dynamic variable ordering, which will leave such gaps, will increase the search."

However, her words must not be considered out of their context. In fact, Smith also noted: "there is an interaction between the implied constraints chosen, and the variable ordering: they cannot be chosen independently.". As the transformation we have proposed add implied gcc in a systematical manner, we may not suffer from the same limit. Hence, we may be able to chose different value orderings, hoping for more efficiency.

For the variable ordering we can observe that the variables at the extremities are less constrained than the variables in the middle of the global sequence. Hence, our variable ordering consists in chosing the variable closest to the middle of the sequence.

Less attention has been paid to value ordering. However, our experiments show that without careful value ordering, the difficult problems cannot be solved in a reasonable amount of time.

After some analysis of the problem, it is clear that the most difficult constraints are tied to options. However, in a standard MAC algorithm, the basic decision is to assign a value to a variable, i.e. assign a *configuration* to a car. This is not directly tied to the difficult constraints. For instance, suppose that option 1 is required by configurations 1, 2, and 3. Suppose furthermore that we assign configuration 1 to a given car, and that there will be no solution with this assignment, because of the capacity associated with option 1. Then the algorithm will backtrack, and try another value, say configuration 2. Then for the same reason (capacity of option 1) this will lead to a failure. In other words we may have a trashing behavior. This was observed experimentally on randomly generated problems taken from [Smi96].

Following this analysis, we tried a new type of value ordering: try to assign first the options for which the demand is closer to the capacity. We define the slack of an option with capacity p_j/q_j required to be scheduled k_j times, as follows:

$$n - q_j(k_j/p_j)$$

This is a good measure of the usage of the option. If the slack is negative, then the capacity constraint of the option cannot be satisfied. If the slack is large, then the capacity constraint is easy to satisfy.

Our enumeration algorithm is a combination of these two orderings that departs from the standard backtracking algorithm. Instead of selecting a variable, and then a value for it, we will proceed following a progressive reduction of the domains of the variables. Indeed, we select first the option with the smallest slack, and then we try to assign it to the variables. Assigning an option o to a variable x amounts to removing the configurations that do not use o from the domain of x. Hence this is in general a domain reduction, not an assignment per se.

More precisely, the option o_j with the smallest slack is selected. Assume this option is required k_j times. Then we try to assign this option to k variables. This requires a tree search, where the variables closest to the middle of the global sequence are tried first. After each assignment, the filtering of all constraints are applied, as in MAC. If we succeed to assign o k times, the algorithm is recursively called with a new option. If not, then there is a backtrack to the previous level of the algorithm.

Our algorithm is still a backtracking algorithm were filtering of constraints are applied at each node. The definition of nodes is different from the standard backtrack algorithm, where branches correspond to the assignment of a value to a variable. In our case, a branch corresponds to the assignment of an option to a variable, i.e. a domain reduction for that variable. Moreover a given variable can appear at different levels in the tree, as its domain can be reduceed several times. Consider for instance that configuration c_1, that requires option o_1 and o_2 is assigned to x in a given solution. Then x may appear two times in the branches leading to that solution, a first one when o_2 is assigned to it, and a second time when o_1 is assigned to x.

This algorithm thus tries to consider first the most constrained choices. Experimentally, the algorithm was able to solve some difficult instances of [Smi96] with a very small number of backtracks. Moreover, it appears that our filtering algorithm subsumes all the implied constraints proposed in [Smi96] and in [DSV88].

4.1 Results

Our approach has been applied to data given by a major car manufacturer. Unfortunately, these data cannot be disclosed. Therefore, we present one classical instance from [DSV88] (instance 0) and some random instances which have been found hard to solve (instances 1 to 4). These instances are reproduced under the permission of Barbara Smith. On the one hand, instances 0 and 3 are given as an example in the Ilog Solver 4.0 User's manual. Hence, any user of Solver 4.0 will be able to reproduce our experiments. On the other hand, we hope that these instances will be considered as a benchmark in the future.

For all instances, 100 cars have to be built, and 5 installation options are available. The capacities of the options are the same for all the instances, that is : 1/2 for option 0, 2/3 for option 1, 1/3 for option 2, 2/5 for option 3, and 1/5 for option 4. Each line of a tabular represents an option. The line number i represents the option number i. The last line represents the number of cars that must have the invoked configuration.

instance #0:

configurations																	
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
•	•	•		•	•	•	•						•				
•	•	•	•	•				•	•	•				•			
		•	•				•			•	•	•					•
	•		•			•			•			•				•	
•					•			•			•				•		
5	3	7	1	10	2	11	5	4	6	12	1	1	5	9	5	12	1

instance #1:

configurations																					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
•	•	•						•		•	•		•	•			•	•	•	•	•
	•	•	•		•	•					•	•			•			•	•		•
	•		•			•	•	•	•	•	•									•	•
•				•		•	•	•				•	•						•		•
		•									•		•		•	•	•			•	•
6	10	2	2	8	15	1	5	2	3	2	1	8	3	10	4	4	2	4	6	1	1

instance #2:

configurations																					
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
•			•					•	•	•	•	•		•		•	•		•	•	
		•			•		•		•	•		•	•		•	•			•	•	•
				•		•	•					•		•			•		•		•
	•		•		•	•				•	•		•			•	•		•		•
								•			•		•		•	•		•		•	
13	8	7	1	12	5	5	6	3	12	8	2	2	1	4	4	1	2	1	1	1	1

instance #3:

configurations																								
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
•	•		•		•					•	•				•	•	•	•	•	•	•			
	•	•		•		•			•		•	•		•	•		•	•	•		•	•	•	
		•	•		•		•		•		•	•	•		•			•					•	
•		•			•	•		•	•						•		•			•	•	•		
		•									•		•	•		•			•	•	•	•	•	•
7	11	1	3	15	2	8	5	3	4	5	2	6	2	2	4	3	5	2	4	1	1	1	1	2

instance #4:

configurations																						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
					•	•			•	•		•	•	•	•	•	•				•	
		•		•	•	•		•	•			•		•	•			•		•	•	•
	•	•				•	•				•	•	•		•			•				•
•		•	•	•					•		•	•	•			•					•	
•	•				•	•											•		•	•	•	•
2	2	5	4	4	1	3	4	19	7	10	1	5	2	6	4	8	1	4	2	4	1	1

The time we give are expressed in second and mesured with Solver 4.0 with Windows NT 4.0 on a Pentium Pro 200Mhz machine. The options order column represents the order that has been chosen to select the next variable and the next value to assign. For example, for instance 0, we first consider the variables that can be instantiated to a configuration that requires option 0. If such a variable is found then this variable is instantiated to a configuration which requires option 0. If no such variables exist then we consider the variables that can be instantiated to a configuration that requires option 3 and we repeat the process

If we consider only the basic constraint network associated with each gsc, the performances of our approach dramaticly decrease. The orders we proposed are also important. After a lot of experiments we present them because we think they are the most robust. Moreover, it seems that we can prove the insatisfiability of hard instances only with the kind of order we propose. Such results are difficult to obtain for car sequencing problem and are quite important for rostering problems.

instance	options order	time	number of fails	solution
0	0 1 4 2 3	2.3	0	yes
1	0 3 4 2 1	1.9	0	yes
2	0 2 3 1 4	422	9355	no solution
3	0 4 1 2 3	3.5	41	no solution
4	?	?	?	?

Note that instance #4 remains an open problem.

5 Conclusion

In this paper we have presented a filtering algorithm for global sequencing constraints. This algorithm is powerful because it deals with a part of generality of the constraints and because it exploits the pruning performance of global cardinality constraints. Moreover, for the car sequencing problem, we have proposed new variable and value orderings. The principle of these orderings should easily be used for other problems. The experiments we made for car sequencing problem shows that our method can solve almost all the tested problem with few backtracks and proves that certain difficult instances have no solution with a small number of backtracks.

6 Acknowledgments

We would like to thank very much Barbara Smith for helpful discussions and for some hard instances of the car sequencing problem that she friendly gave us. Also, we would like to thank the reviewer Ω for useful comments on a draft of this paper.

References

[BC94] N. Beldiceanu and E. Contejean. Introducing global constraints in chip. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.

[DSV88] M. Dincbas, H. Simonis, and P. Van Hentenryck. Solving the car-sequencing problem in constraint logic programming. In *ECAI'88, proceedings of the European Conference on Artificial Intelligence*, pages 290–295, 1988.

[Mac77] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[Nui94] W.P. Nuijten. *Time and Resource Constrained Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, 1994.

[PL95] J-F. Puget and M. Leconte. Beyong the glass box: Constraints as objects. In John Lloyd, editor, *Logic Programming, Proceedings of the 1995 International Symposium*, pages 513–527. The MIT Press, Portland, Oregon, 1995.

[Rég94] J-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI-94, proceedings of the Twelth National Conference on Artificial Intelligence*, pages 362–367, Seattle, Washington, 1994.

[Rég96] J-C. Régin. Generalized arc consistency for global cardinality constraint. In *AAAI-96, proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 209–215, Portland, Oregon, 1996.

[Sim96] H. Simonis. Problem classification scheme for finite domain constraint solving. In *CP96, Workshop on Constraint Programming Applications: An Inventory and Taxonomy*, pages 1–26, Cambridge, MA, USA, 1996.

[Smi96] B.M. Smith. Succeed-first or fail-first: A case study in variable and value ordering. In *proceedings ILOG Solver and ILOG Scheduler Second International Users' Conference*, Paris, France, 1996.

[VDT92] P. Van Hentenryck, Y. Deville, and C.M. Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57:291–321, 1992.