Global constraints for lexicographic orderings[0]

Alan Frisch[1], Brahim Hnich[2], Zeynep Kızıltan[2], Ian Miguel[1], and Toby Walsh[3]

[1] Department of Computer Science, University of York, Heslington, York, United Kingdom. {frisch, ianm}@cs.york.ac.uk

[2] Computer Science Division, Department of Information Science, Uppsala University, Uppsala, Sweden. {Brahim.Hnich, Zeynep.Kiziltan}@dis.uu.se

[3] Cork Constraint Computation Center, University College Cork, Ireland. tw@4c.ucc.ie

Abstract. We propose some global constraints for lexicographic orderings on vectors of variables. These constraints are very useful for breaking a certain kind of symmetry arising in matrices of decision variables. We show that decomposing such constraints carries a penalty either in the amount or the cost of constraint propagation. We therefore present a global consistency algorithm which enforces a lexicographic ordering between two vectors of n variables and domains of size d in $O(nd)$ time. The algorithm can be modified very slightly to enforce a strict lexicographic ordering. Our experimental results on a number of domains (balanced incomplete block design, social golfer, and sports tournament scheduling) confirm the efficiency and value of these new global constraints.

1 Introduction

Global (or non-binary) constraints are one of the most important and powerful aspects of constraint programming. Specialized propagation algorithms for global constraints are vital for efficient and effective constraint solving. A number of consistency algorithms for global constraints have been developed by several researchers (see [1] for examples). To continue this line of research, we propose a new family of efficient global constraints for lexicographic orderings on vectors of variables.

These global constraints are especially useful for breaking a certain kind of symmetry arising in matrices of decision variables. Dealing efficiently and effectively with symmetry is one of the major difficulties in constraint programming. Symmetry occurs in many scheduling, assignment, routing and supply chain problems. These can often be modelled as constraint programs with matrices of decision variables [6] in which the matrices have symmetry along their rows

[0] Submitted in confidence. Only for distribution to those members of the programme committee who have a need to read this. We are very thankful to Warwick Harvery, Nicolas Beldiceanu, the members of the APES research group (especially Ian Gent, Patrick Prosser, and Barbara Smith), and Pierre Flener for valuable discussions on this work. This research was made possible by VR grant 221-99-369, EPSRC grant GR/N16129 and an EPSRC advanced research fellowship.

and/or columns [5]. For instance, a natural model of the sports tournament scheduling problem has a matrix of decision variables, each of which is assigned a value corresponding to the match played in a given week and period [10]. In this problem, weeks and periods are indistinguishable so we can freely permute rows and columns. To break this symmetry, we can add an additional constraint that the rows and columns are lexicographically ordered [5]. These global constraints can also be used in multi-criteria optimization problems where the objective function consists of features which are ranked [4].

2 Formal background

A constraint satisfaction problem (CSP) consists of a set of variables, each with a finite domain of values, and a set of constraints that specify the allowed values for given subsets of variables. The solution to a CSP is an assignment of values to the variables which satisfies all the constraints. To find such solutions, constraint solvers often explore the space of partial assignment enforcing some level of consistency like (generalized) arc-consistency or bounds consistency. A CSP is generalized arc-consistent (GAC) iff, when a variable in a constraint is assigned any of its values, there exist compatible values for all the other variables in the constraint. When the constraints are binary, we talk about arc-consistency (AC). For totally ordered domains, like integers, a weaker level of consistency is bounds consistency. A CSP is bounds consistent (BC) iff, when a variable in a constraint is assigned its maximum or minimum value, there exist compatible values for all the other variables in the constraint. If a constraint c is AC or GAC then we write AC(c) or GAC(c) respectively.

In this paper, we are interested in lexicographic ordering constraints on vectors of distinct variables and having the same length. Throughout, we assume finite integer domains, which are totally ordered. Given two vectors, $\bar{x}$ and $\bar{y}$ of n variables, $\langle x_0, x_1, \ldots, x_{n-1}\rangle$ and $\langle y_0, y_1, \ldots, y_{n-1}\rangle$, we write a lexicographical ordering constraint as $\bar{x} \leq_{\rm lex} \bar{y}$ and a strict lexicographic ordering constraint as $\bar{x} <_{\rm lex} \bar{y}$. Indexing is from left to right with the most significant index for the lexicographic ordering at 0. The lexicographic ordering constraint $\bar{x} \leq_{\rm lex} \bar{y}$ ensures that: $x_0 \leq y_0$; $x_1 \leq y_1$ when $x_0 = y_0$; $x_2 \leq y_2$ when $x_0 = y_0$ and $x_1 = y_1$; $\ldots$; $x_{n-1} \leq y_{n-1}$ when $x_0 = y_0$, $x_1 = y_1$, $\ldots$, and $x_{n-2} = y_{n-2}$. The strict lexicographic ordering constraint $\bar{x} <_{\rm lex} \bar{y}$ ensures that: $\bar{x} \leq_{\rm lex} \bar{y}$; and $x_{n-1} < y_{n-1}$ when $x_0 = y_0$, $x_1 = y_1$, $\ldots$, and $x_{n-2} = y_{n-2}$.

We are also interested in multiple lexicographic ordering constraints. For example, all rows or columns in a matrix of variables might be lexicographically ordered. We denote m vectors of n domain variables by:

$$\begin{array}{rcl}
\bar{x}_0 & = & \langle x_{0,0}, \quad x_{0,1}, \quad \ldots \quad, x_{0,n-1}\rangle \\
\bar{x}_1 & = & \langle x_{1,0}, \quad x_{1,1}, \quad \ldots \quad, x_{1,n-1}\rangle \\
\vdots & & \\
\bar{x}_{m-1} & = & \langle x_{m-1,0}, x_{m-1,1}, \ldots, x_{m-1,n-1}\rangle
\end{array}$$

We write $\bar{x}_0 \leq_{\rm lex} \bar{x}_1 \ldots \leq_{\rm lex} \bar{x}_{m-1}$ for the single global constraint on these $m.n$ variables which ensures that each pair of vectors is in lexicographic order. Also, we write $\bar{x}_0 <_{\text{lex}} \bar{x}_1 \ldots <_{\text{lex}} \bar{x}_{m-1}$ for the single global constraint on the $m.n$ variables which ensures that each pair of vectors is in strict lexicographic order.

We need the following additional notation. The sub-vector of $\bar{x}$ with start index a and last index b inclusive is denoted by $\bar{x}_{a \to b}$. The minimum element in the domain of x_i is denoted by $min(dom(x_i))$, and the maximum by $max(dom(x_i))$. Given a binary constraint c, if any assignment of values to x_i and y_i guarantees that c holds then we write $x_i\ c^* y_i$, whilst if any assignment of values to x_i and y_i fails to satisfy c then we write $\neg(x_i\ c^* y_i)$. Hence, $x_i \leq^* y_i$ is equivalent to $max(dom(x_i)) \leq min(dom(y_i))$, $x_i <^* y_i$ to $max(dom(x_i)) < min(dom(y_i))$, $x_i >^* y_i$ to $min(dom(x_i)) > max(dom(y_i))$, and $\neg(x_i >^* y_i)$ to $min(dom(x_i)) \leq max(dom(y_i))$. If x_i and y_i are ground and equal then we write $x_i \doteq y_i$, otherwise we write $\neg(x_i \doteq y_i)$. Finally, the function $\texttt{floor}(\bar{x})$ assigns all unassigned variables in a vector $\bar{x}$ to their minimum values, whilst $\texttt{ceiling}(\bar{x})$ assigns all to their maximum values.

3 GAC algorithms for lexicographic ordering

We sketch the main features of a family of linear time algorithms for enforcing GAC on a (strict or non-strict) lexicographic ordering constraint. Consider the lexicographic ordering constraint $\bar{x} \leq_{\text{lex}} \bar{y}$ with:

$$\begin{aligned} \bar{x} &= \langle \{1\},\ \ \{0,1\}, \{0,1\}, \{1\} \rangle \\ \bar{y} &= \langle \{0,1\},\ \ \{0\},\ \ \{0,1\}, \{0\} \rangle \end{aligned}$$

The key idea is to have two pointers, α and β which save us from repeatedly traversing the vectors. The pointer α points to the index such that all variables above it are ground and equal. The pointer β points either to the most significant index starting from which the sub-vectors are lexicographically ordered the wrong way whatever the remaining assignments are (that is, $\texttt{floor}(\bar{x}_{\beta \to n-1}) >_{\text{lex}} \texttt{ceiling}(\bar{y}_{\beta \to n-1})$), or (if this is not the case) to infinity. As variables are assigned, α and β are moved inwards, and we terminate when $\beta = \alpha + 1$. The algorithm restricts domain prunings to the index α, and as α is strictly increasing and bounded by n, and as at most d values from any variable are pruned, the algorithm runs in $O(nd)$ time.

In this example, we initialize α to point to the index 0 since y_0 is not ground. We initialize β to the index 3 since the sub-vectors starting at index 3 are ordered the wrong way round (i.e. $\bar{x}_{3 \to 3} >_{\text{lex}} \bar{y}_{3 \to 3}$):

$$\begin{array}{rcccc} \bar{x} = & \langle \{1\}, & \{0,1\}, & \{0,1\}, & \{1\} \rangle \\ \bar{y} = & \langle \{0,1\}, & \{0\}, & \{0,1\}, & \{0\} \rangle \\ & \alpha \uparrow & & & \uparrow \beta \end{array}$$

Since α is at 0, the start of the vectors, there can be no support for any value in the domain of y_α which is less than the minimum value in the domain of x_α. We can therefore remove 0 from the domain of y_α and increment α to 1:

$$\begin{array}{rcccc} \bar{x} = & \langle \{1\}, & \{0,1\}, & \{0,1\}, & \{1\} \rangle \\ \bar{y} = & \langle \{1\}, & \{0\}, & \{0,1\}, & \{0\} \rangle \\ & & \alpha \uparrow & & \uparrow \beta \end{array}$$

As the vectors above α are ground and equal, there can be no support for any value in the domain of x_α which is more than the maximum value in the domain of y_α. We can therefore remove 1 from the domain of x_α and increment α to 2:

$$\begin{array}{l} \bar{x} = \langle \{1\}, \{0\}, \{0,1\}, \{1\} \rangle \\ \bar{y} = \langle \{1\}, \{0\}, \{0,1\}, \{0\} \rangle \\ \qquad\qquad\qquad\quad \alpha \uparrow \quad \uparrow \beta \end{array}$$

Since α has now reached $\beta - 1$, there can be no support for any value in the domain of x_α (resp. y_α) which is greater (resp. less) than or equal to the maximum (resp. minimum) value in the domain of y_α (resp. x_α). We must therefore strictly order the index α. That is, we can assign 0 to x_2 and 1 to y_2:

$$\begin{array}{l} \bar{x} = \langle \{1\}, \{0\}, \{0\}, \{1\} \rangle \\ \bar{y} = \langle \{1\}, \{0\}, \{1\}, \{0\} \rangle \\ \qquad\qquad\qquad\quad \alpha \uparrow \; \uparrow \beta \end{array}$$

We now terminate as $\beta = \alpha + 1$. Enforcing GAC has, in this case, assigned all the variables. Of the 16 possible assignments for the two vectors, this is the unique one which satisfies the lexicographic ordering constraint.

3.1 GAC on $\leq_{\text{lex}}$

We now define more formally the algorithm for enforcing GAC on the constraint $\bar{x} \leq_{\text{lex}} \bar{y}$. As we have seen, the pointers α and β play a central role in the algorithm. The pointer α points to an index of $\bar{x}$ and $\bar{y}$ such that for all $j < \alpha$ we have $x_j \doteq y_j$, and $\neg(x_\alpha \doteq y_\alpha)$. The pointer β points either to the most significant index starting from which the sub-vectors are lexicographically ordered the wrong way whatever the remaining assignments are:

$$\begin{array}{r} \texttt{floor}(\bar{x}_{\beta \to n-1}) >_{\text{lex}} \texttt{ceiling}(\bar{y}_{\beta \to n-1}) \\ \forall i \;\; 0 \leq i < \beta \,.\, \neg(\texttt{floor}(\bar{x}_{i \to n-1}) >_{\text{lex}} \texttt{ceiling}(\bar{y}_{i \to n-1})) \end{array}$$

or (if this is not the case) to ∞. Consider two vectors $\bar{x}$ and $\bar{y}$ of length 1 and x_0 and y_0 both have the domain $\{0, 1\}$. The pointer α is set to 0 and if we set the pointer β to 1, then our rule would assign 0 to x_α and 1 to y_α, which is wrong. To avoid such situations, the pointer β is set to a value bigger than the length of the vectors. It is not hard to show that generalised arc-inconsistent values can exist only in the interval $[\alpha, \beta)$ where $\beta < n$. Indeed, we can restrict pruning to the index α, and show that GAC($\bar{x} \leq_{lex} \bar{y}$) iff: $AC(x_\alpha < y_\alpha)$ when $\beta = \alpha + 1$, and $AC(x_\alpha \leq y_\alpha)$ when $\beta > \alpha + 1$. The algorithm also has a flag, *consistent* which indicates that all possible assignments satisfy the lexicographic ordering constraint. That is, $\texttt{ceiling}(\bar{x}) \leq_{\text{lex}} \texttt{floor}(\bar{y})$. We first give the procedure to initialise the two pointers (α and β) and the flag (*consistent*).

1. **Procedure Initialise()**
2. *consistent* $:= false$

3. $i := 0$
4. **WHILE** $(i < n \ \wedge \ x_i \doteq y_i)$ $i := i + 1$
5. **IF** $(i = n)$ $consistent := true$, **Return**
6. **ELSE** $\alpha := i$
7. **IF** (**Check_Lex**(i)) $consistent := true$, **Return**
8. $\beta := -1$
9. **WHILE** $(i \neq n \wedge \neg(x_i >^* y_i))$
10.   **IF** $(min(dom(x_i)) = max(dom(y_i)))$
11.   **IF**$(\beta = -1)$ $\beta := i$
12.   **ELSE** $\beta := -1$
13.   i:=i+1
14. **IF** $(i = n)$ $\beta := \infty$
15. **ELSE IF** $(\beta = -1)$ $\beta := i$
16. **IF** $(\alpha \geq \beta)$ **FAIL**
17. **GACLexLeq**(α)

Line 4 traverses $\bar{x}$ and $\bar{y}$, starting at index 0, until either it reaches the end of the vectors (all pairs of variables are ground and equal), or it reaches an index where the pair of variables are not ground and equal. In the first case, `ceiling(`$\bar{x}$`)`$\leq_{\text{lex}}$ `floor(`$\bar{y}$`)`. Hence, *consistent* is set to *true* and the algorithm returns (line 5). In the second case, α is set to the most significant index where the pair of variables are not ground and equal (line 6). In line 7, *consistent* is set to *true* if the call to **Check_Lex**(i) succeeds. This call checks whether the lexicographic ordering constraint is satisfied for all possible assignments:

1. **Boolean Check_Lex**(i)
2. **IF** $(i = n - 1)$ **Return**$(x_i \leq^* y_i)$
3. **ELSE Return**$(x_i <^* y_i)$

Line 9 traverses $\bar{x}$ and $\bar{y}$, starting at index α, until either it reaches the end of the vectors (none of the pairs of variables satisfy $x_i >^* y_i$), or it reaches an index i where the pair of variables satisfy $x_i >^* y_i$. In the first case, β is set to ∞ (line 14). In the second case, β is guaranteed to be at most i (line 15). If, however, there exist a pair of sub-vectors, $\bar{x}_{h \to i-1}$ and $\bar{y}_{h \to i-1}$ such that $min(dom(x_j)) = max(dom(y_j))$ for all $h \leq j \leq i - 1$, then β can be revised to h (lines 10-11).

The complexity of the initialisation is $O(n)$ since both vectors are traversed in the worst case. Initialization terminates either with failure (if $\alpha \geq \beta$) or by calling GACLexLeq(α). GACLexLeq(i) is also called by the event handler whenever a value is removed from a variable at index i.

1. **Procedure GACLexLeq**(i)
2. **IF** $(i \geq \beta \vee consistent)$ **Return**
3. **IF** $(i = \alpha \ \wedge \ i + 1 = \beta)$
4.   **AC**$(x_i < y_i)$
5.   **IF** (**Check_Lex**(i)) $consistent := true$, **Return**
6. **IF** $(i = \alpha \ \wedge \ i + 1 < \beta)$

7.   **AC**($x_i \leq y_i$)
8.   **IF** (**Check_Lex**(i)) *consistent* := *true*, **Return**
9.   **IF** ($x_i \doteq y_i$) **UpdateAlpha**($i+1$)
10. **IF** ($\alpha < i < \beta$)
11.   **IF** (($i = \beta - 1 \ \wedge \ min(dom(x_i)) = max(dom(y_i))$) $\vee\ x_i >^* y_i$)
12.   **UpdateBeta**($i-1$)

In the **GACLexLeq** procedure, lines 2, 3-5, 6-9, and 10-12 are mutually exclusive, and will be referred as parts A, B, C, and D respectively.

Part A: Generalised arc-inconsistent values exist only in $[\alpha, \beta)$ where $\beta < n$. Therefore, no inconsistent value can exist at an index greater than or equal to β. Hence, if $i \geq \beta$, the vectors are already GAC and the algorithm returns. If the flag *consistent* is true, `ceiling`$(\bar{x}) \leq_{\text{lex}}$ `floor`$(\bar{y})$ so the algorithm returns. The complexity of this step is constant.

Part B: **AC**$(x_i < y_i)$ stands for maintaining arc-consistency on $x_i < y_i$. This is implemented as follows: If $max(dom(x_i))$ (resp. $min(dom(y_i))$) is supported by $max(dom(y_i))$ (resp. $min(dom(x_i))$) then all the other elements in the domain of x_i (resp. y_i) are supported. Otherwise, the upper (resp. lower) bound of the domain of x_i (resp. y_i) is tightened. The worst-case complexity of maintaining arc-consistency on $x_i < y_i$ is thus $O(d)$, where d is the domain size of the variables. If $\beta = \alpha + 1$ then we need to ensure that $AC(x_\alpha < y_\alpha)$. If a domain wipe-out occurs then the algorithm fails, and the vectors cannot be made GAC. Otherwise, the vectors are now GAC. After the propagation carried out by maintaining arc-consistency, **Check_Lex**(i) is called. This part of the algorithm thus has an $O(d)$ complexity.

Part C: If $\beta > \alpha + 1$ then we need to ensure $AC(x_\alpha \leq y_\alpha)$. If a domain wipe-out occurs then the algorithm fails, and the vectors cannot be made GAC. Otherwise, the vectors are now GAC. After the pruning carried out by maintaining arc-consistency, **Check_Lex**(i) is called and α updated if necessary.

1. **Procedure UpdateAlpha**(i)
2. **IF** ($i = \beta$) **FAIL**
3. **IF** ($i = n$) *consistent* := *true*, **Return**
4. **IF** ($\neg(x_i \doteq y_i)$)
5. $\alpha := i$
6.   **GACLexLeq**(i)
7. **ELSE UpdateAlpha**($i+1$)

In lines 4 and 7 of **UpdateAlpha**(i), the vectors are traversed until the most significant index k where $\neg(x_k \doteq y_k)$ is found. If such an index does not exist, `ceiling`$(\bar{x}) \leq_{\text{lex}}$ `floor`$(\bar{y})$, so *consistent* is set to true (line 3). Otherwise, α is set to k (line 5). **GACLexLeq** is then called with this new value of α. In the worst case, α moves one index at a time, and on each occasion AC is enforced. Hence, this part of the algorithm gives an $O(nd)$ complexity.

Part D: β is updated by calling **UpdateBeta**$(i-1)$ when we set a variable at an index i between α and β, and either x_i now must be greater than y_i (i.e $x_i >^* y_i$), or i is adjacent to β (i.e. $i = \beta - 1$) and y_i can at best only equal x_i.

```
1. Procedure UpdateBeta(i)
2. IF (i + 1 = α) FAIL
3. IF (min(dom(x_i)) < max(dom(y_i)) )
4.         β := i + 1
5.         IF ¬(x_i <* y_i) GACLexLeq(i)
6. ELSE IF (min(dom(x_i)) = max(dom(y_i)) ) UpdateBeta(i − 1)
```

In lines 3 and 6 of **UpdateBeta(i)**, the vectors are traversed until the most significant index k where $min(dom(x_k)) < max(dom(y_k))$ is found. The pointer β is set to $k+1$ (line 4). **GACLexLeq(**$\beta - 1$**)** is then called in case $\beta = \alpha + 1$, and we need to ensure $AC(x_\alpha < y_\alpha)$. If $\beta = \alpha + 1$ and $x_\alpha <^* y_\alpha$ then *consistent* would already have been set to *true*. The algorithm, however, always terminates after one more step. The worst case complexity of the algorithm remains $O(nd)$.

When updating α or β, failure is established if these pointers meet. This situation can only arise if the event queue contains several domain prunings due to other constraints, for the following reasons. After initialisation, the constraint is GAC. Hence, we can find a single consistent value for all variables. If after every assignment we enforce GAC on this constraint, this property persists. Therefore, we can only fail when the queue contains a series of updates which must be dealt with simultaneously.

3.2 GAC on $<_{\text{lex}}$

With very little effort, **GACLexLeq()** can be adapted to give an algorithm, **GACLexLess()** that enforces GAC on a strict lexicographic ordering constraint between two vectors. The reason that the two algorithms are so similar is that, as soon as β is assigned a value other than ∞, **GACLexLeq()** enforces strict inequality in subvectors above β. In the **GACLexLess()** algorithm, β again points either to the most significant index starting from which the sub-vectors must be ordered the wrong way:

$$\texttt{floor}(\bar{x}_{\beta \to n-1}) \geq_{\text{lex}} \texttt{ceiling}(\bar{y}_{\beta \to n-1})$$
$$\forall i \;\; 0 \leq i < \beta \,.\, \neg(\texttt{floor}(\bar{x}_{i \to n-1}) \geq_{\text{lex}} \texttt{ceiling}(\bar{y}_{i \to n-1}))$$

or (if this is not the case) to n (so that equality of the vectors is not allowed). There are only two other changes. When α is initialized, we fail if α gets set to n (as the vectors are ground and equal so cannot be strictly ordered). Finally, for obvious reasons, all calls to **Check_Lex**(i) are replaced by $(x_i <^* y_i)$.

Given two vectors $\bar{x}$ and $\bar{y}$ with non-empty domains, executing the two algorithm **GACLexLeq** and **GACLexLess** maintain generalised arc-consistency on $\bar{x} \leq_{\text{lex}} \bar{y}$ and $\bar{x} <_{\text{lex}} \bar{y}$, respectively.

Theorem 1. *Given a pair of vectors $\bar{x}$ and $\bar{y}$ of domain variables,* **GACLexLeq** *(resp.* **GACLexLess***) either establishes failure if $\bar{x} \leq_{\text{lex}} \bar{y}$ (resp. $\bar{x} <_{\text{lex}} \bar{y}$) is unsatisfiable, or prunes elements from $\bar{x}$ and $\bar{y}$ such that GAC($\bar{x} \leq_{\text{lex}} \bar{y}$) (resp. GAC($\bar{x} <_{\text{lex}} \bar{y}$)) and the set of solutions is preserved.*

Proof. For reasons of space, we are unable to give the correctness proofs of the algorithms. These can, however, be found in an accompanying technical report [8].

4 Alternative approaches

There are at least two other ways of posting global lexicographic ordering constraints: by decomposing them into smaller constraints, or by posting an arithmetic inequality constraint. We show here that such decomposition usually increases the runtime or decreases the amount of constraint propagation. The experimental results in Section 6 support this observation.

A lexicographic ordering constraint $\bar{x} \leq_{\text{lex}} \bar{y}$ can be decomposed it into n non-binary constraints:

$$\begin{aligned}\{&x_0 \leq y_0,\ \ x_0 = y_0 \rightarrow x_1 \leq y_1,\ \ x_0 = y_0\ \wedge\ x_1 = y_1 \rightarrow x_2 \leq y_2, \ldots,\\ &x_0 = y_0\ \wedge\ x_1 = y_1\ \wedge \ldots \wedge\ x_{n-2} = y_{n-2} \rightarrow x_{n-1} \leq y_{n-1}\}\end{aligned}$$

Similarly a strict lexicographic ordering constraint $\bar{x} <_{\text{lex}} \bar{y}$ can be decomposed into n non-binary constraints:

$$\begin{aligned}\{&x_0 \leq y_0,\ \ x_0 = y_0 \rightarrow x_1 \leq y_1,\ \ x_0 = y_0\ \wedge\ x_1 = y_1 \rightarrow x_2 \leq y_2, \ldots,\\ &x_0 = y_0\ \wedge\ x_1 = y_1\ \wedge \ldots \wedge\ x_{n-2} = y_{n-2} \rightarrow x_{n-1} < y_{n-1}\}\end{aligned}$$

As the following theorem shows, such decompositions do not affect the pruning of GAC. However, in many solvers (e.g. Solver 5.0), only nFC0 [1] is enforced on such non-binary decompositions. Our linear time GAC algorithm will clearly do more propagation. In addition, the experiments in Section 6 show that, despite enforcing a weaker consistency like nFC0, a state of the art system like ILOG's Solver 5.0 handles such decompositions inefficiently.

Theorem 2. *GAC($\bar{x} \leq_{\text{lex}} \bar{y}$) is equivalent to GAC on the decomposition, but strictly stronger than nFC0 on the decomposition. Similarly, GAC($\bar{x} <_{\text{lex}} \bar{y}$) is equivalent to GAC the decomposition, but strictly stronger than nFC0 on the decomposition.*

Proof. (Outline) We just consider GAC($\bar{x} \leq_{\text{lex}} \bar{y}$) as the proof for GAC($\bar{x} <_{\text{lex}} \bar{y}$) is entirely analogous. Clearly GAC on $\bar{x} \leq_{\text{lex}} \bar{y}$ is as strong as GAC on the decomposition. To show the reverse, suppose that every constraint in the decomposition is GAC but that the undecomposed constraint is not. There is some index α such that for all $j < \alpha$ we have $x_j \doteq y_j$ and $\neg(x_\alpha \doteq y_\alpha)$, and x_α or y_α has a value that is not GAC. The two cases are dual so we just consider the first. There exists a value a in the domain of x_α that has no support in the domain of y_α. Hence, all values in the domain of y_α are smaller than a. However, this is not possible if the decomposed constraint, $x_0 = y_0\ \wedge\ x_1 = y_1\ \wedge \ldots \wedge\ x_{\alpha-1} = y_{\alpha-1} \rightarrow x_\alpha \leq y_\alpha$ is GAC.

Clearly GAC on the decomposition is as strong as nFC0. To show strictness, consider $\langle\{0,1\},\{1\}\rangle \leq_{\text{lex}} \langle\{0,1\},\{0\}\rangle$. This problem is not GAC. However,

[1] Forward checking (FC) has been generalised to non-binary constraints [2]. nFC0 makes every k-ary constraint with $k-1$ variables instantiated AC. nFC0 usually denotes the search algorithm, but we overload notation here to use nFC0 to describe the level of local consistency that it enforces at each node in its search tree.

nFC0 leaves the problem unchanged as more than one variable is uninstantiated. □

Note that naively enforcing GAC on the decomposition of a global lexicographic ordering constraint will usually be expensive as the decomposition contains non-binary constraints, some of which are large in size. Suppose we use an optimal algorithm like GAC-schema [3]. For e constraints of arity k and variables with domains of size d, GAC-schema takes $O(ed^k)$ time. As the decomposition introduce n constraints with arity up to $2n$, GAC-schema on the decomposition takes $O(nd^{2n})$ time.

A second way of enforcing a lexicographic ordering is via an arithmetic constraint. To ensure that $\bar{x} \leq_{\text{lex}} \bar{y}$ with domains of size d, we post the arithmetic constraint:

$$d^{n-1} * x_0 + d^{n-2} * x_1 + \ldots + d^0 * x_{n-1} \leq d^{n-1} * y_0 + d^{n-2} * y_1 + \ldots + d^0 * y_{n-1}$$

And to ensure that $\bar{x} <_{\text{lex}} \bar{y}$ we post the arithmetic constraint:

$$d^{n-1} * x_0 + d^{n-2} * x_1 + \ldots + d^0 * x_{n-1} < d^{n-1} * y_0 + d^{n-2} * y_1 + \ldots + d^0 * y_{n-1}$$

Maintaining BC on such arithmetic constraints does the same pruning as GAC.

Theorem 3. *GAC($\bar{x} \leq_{\text{lex}} \bar{y}$) and GAC($\bar{x} <_{\text{lex}} \bar{y}$) are equivalent to BC on the corresponding arithmetic constraints.*

Proof. (Outline) We just consider GAC($\bar{x} \leq_{\text{lex}} \bar{y}$) as the proof for GAC($\bar{x} <_{\text{lex}} \bar{y}$) is entirely analogous. Suppose that the arithmetic constraint is BC, but that $\bar{x} \leq_{\text{lex}} \bar{y}$ is not GAC. There is some index α such that for all $j < \alpha$ we have $x_j \doteq y_j$ and $\neg(x_\alpha \doteq y_\alpha)$, and x_α or y_α has a value that is not GAC. The two cases are analogous so we just consider the first. There exits a value a in the domain of x_α that has no support in the domain of y_α. Hence, all the values in the domain of y_α are smaller than a. However, this contradicts the arithmetic constraint being BC. □

Maintaining BC on such arithmetic constraints can be achieved in $O(ndc)$ where n is the length of the vectors, d is the domain size, and c is the time required to check that a particular (upper or lower) bound of a variable is BC. At best, c is a constant time operation. However, when n and d get large, d^{n-1} will be much more than the word size of the computer and computing BC will be significantly more expensive. Hence, this method is only feasible when the vectors and domain sizes are small. For instance, on the BIBD problem in Section 6 with vectors of length 120 and domain size 2, the coefficients in the arithmetic constraint would exceed 2^{31}, the maximum integer size allowed in Solver 5.0.

5 Extensions

We often have multiple lexicographic ordering constraints. For example, all rows or columns in a matrix of decision variables might be lexicographically ordered. We can treat such a problem as a single global ordering constraint over the whole matrix. Alternatively, we can decompose it into lexicographic ordering constraints between all pairs of vectors. We can decompose this further by posting lexicographic ordering constraints just between immediate pairs of vectors (and calling upon the transitivity of the orderings). The following theorems demonstrate that such decompositions hinder constraint propagation in general. However, we identify the special case of a (non-strict) lexicographical ordering on 0/1 variables where it does not.

Theorem 4. *GAC($\bar{x}_0 \leq_{\text{lex}} \bar{x}_1 \ldots \leq_{\text{lex}} \bar{x}_{m-1}$) is strictly stronger than GAC($\bar{x}_i \leq_{\text{lex}} \bar{x}_j$) for all $i < j$. Similarly, GAC($\bar{x}_0 <_{\text{lex}} \bar{x}_1 \ldots <_{\text{lex}} \bar{x}_{m-1}$) is strictly stronger than GAC($\bar{x}_i <_{\text{lex}} \bar{x}_j$) for all $i < j$.*

Proof. Consider the following 3 vectors:

$$\bar{x}_0 = \langle\{0,2\},\{1\}\rangle \quad \bar{x}_1 = \langle\{1,2\},\{0,3\}\rangle \quad \bar{x}_2 = \langle\{2\},\{0,2\}\rangle$$

Although GAC($\bar{x}_i \leq_{\text{lex}} \bar{x}_j$), and GAC($\bar{x}_i <_{\text{lex}} \bar{x}_j$) for all $i < j$, neither GAC($\bar{x}_0 \leq_{\text{lex}} \bar{x}_1 \leq_{\text{lex}} \bar{x}_2$) nor GAC($\bar{x}_0 <_{\text{lex}} \bar{x}_1 <_{\text{lex}} \bar{x}_2$) holds as $x_{0,0} = \{2\}$ cannot be consistently extended. □

A simple corollary is that enforcing GAC on (strict) lexicographic ordering constraints between neighbouring pairs of vectors does less pruning than enforcing GAC on a single global ordering constraint. Indeed, GAC on neighbouring pairs does less pruning than GAC on each pair of vectors.

Theorem 5. *GAC($\bar{x}_i \leq_{\text{lex}} \bar{x}_j$) for all $i < j$ is strictly stronger than GAC($\bar{x}_i \leq_{lex} \bar{x}_{i+1}$) for all i. Similarly, GAC($\bar{x}_i <_{\text{lex}} \bar{x}_j$) for all $i < j$ is strictly stronger than GAC($\bar{x}_i <_{lex} \bar{x}_{i+1}$) for all i.*

Proof. Consider the following 3 vectors:

$$\bar{x}_0 = \langle\{0,1\},\{1\},\{0,1\}\rangle \quad \bar{x}_1 = \langle\{0,1\},\{0,1\},\{0,1\}\rangle \quad \bar{x}_2 = \langle\{0,1\},\{0\},\{0,1\}\rangle$$

Although GAC($\bar{x}_i \leq_{\text{lex}} \bar{x}_{i+1}$), and GAC($\bar{x}_i <_{\text{lex}} \bar{x}_{i+1}$) for all i, neither GAC($\bar{x}_0 \leq_{\text{lex}} \bar{x}_2$) nor GAC($\bar{x}_0 <_{\text{lex}} \bar{x}_2$) holds as $x_{0,0} = \{1\}$ cannot be consistently extended. □

In the special case that the domains of the vectors are 0/1, enforcing GAC on lexicographic ordering constraints between *every* pair of vectors achieves global consistency. This is not true, however, for the strict lexicographic ordering constraint, nor for non-strict lexicographic ordering constraints between neighbouring pairs of vectors.

Theorem 6. *For 0/1 variables, GAC($\bar{x}_i \leq_{\text{lex}} \bar{x}_j$) for all $i < j$ is equivalent to GAC($\bar{x}_0 \leq_{\text{lex}} \bar{x}_1 \ldots \leq_{\text{lex}} \bar{x}_{m-1}$).*

Proof. (Outline) We assume that we have a problem in which GAC($\bar{x}_0 \leq_{\text{lex}} \bar{x}_1 \ldots \leq_{\text{lex}} \bar{x}_{m-1}$) does not hold and show that there exist $i < j$ such that GAC($\bar{x}_i \leq_{\text{lex}} \bar{x}_j$) does not hold. As GAC($\bar{x}_0 \leq_{\text{lex}} \bar{x}_1 \ldots \leq_{\text{lex}} \bar{x}_{m-1}$) does not hold, there is variable with a value which lacks support. With this value of the variable, there exist indices p, q, and k such that $p < q$,$x_{p,k} = 1$, and $x_{q,k} = 0$. Also, for all $k' < k$ any assignment of values to the variables guarantees that either $x_{0,k'} = x_{1,k'} \ldots = x_{m-1,k'}$, or there exist two indices p' and q' such that $p' < q'$, $x_{p',k'} = 1$, and $x_{q',k'} = 0$. Hence, there exist a pair of vectors $\bar{x}_i$ and $\bar{x}_j$ such that $\texttt{floor}(\bar{x}_i) >_{lex} \texttt{ceiling}(\bar{x}_j)$. □

This result, however, does not hold for a strict lexicographical ordering.

Theorem 7. *For 0/1 variables, GAC($\bar{x}_0 <_{\text{lex}} \bar{x}_1 \ldots <_{\text{lex}} \bar{x}_{m-1}$) is strictly stronger than GAC($\bar{x}_i <_{\text{lex}} \bar{x}_j$) for all $i < j$.*

Proof. Consider 5 vectors $\bar{x}_0, \ldots, \bar{x}_4$, where $\bar{x}_i = \langle\{0,1\},\{0,1\}\rangle$ for all $i \in [0,4]$. Although GAC($\bar{x}_i <_{\text{lex}} \bar{x}_j$) for all $i < j$ there is no globally consistent solution as there are only 4 possible distinct vectors. □

6 Experimental results

We tested our global constraints on three problem domains: the balanced incomplete block design (`prob028` in CSPLib: www.csplib.org), the social golfer (`prob010` in CSPLib), and the sports tournament scheduling (`prob026` in CSPLib). Each is naturally modelled by matrices of decision variables which exhibit a high degree of symmetry. The rows and columns of these matrices can therefore be ordered lexicographically to break much of this symmetry. Throughout, we compare the performance of the global constraints developed here with FC on the decomposed form described in Section 4 using ILOG's Solver 5.0 on a 750Mhz PentiumIII, 128Mb RAM, but not with GAC-schema as it is computationally expensive.

When using symmetry-breaking constraints, the variable and value ordering (VVO) is very important. In particular, if the VVO moves right to left along the rows, it will increasingly conflict with the lexicographic ordering constraints. We can then expect to gain from both the lower complexity and increased pruning achieved by our global constraints. Given a VVO that agrees with the lexicographic ordering constraints, we can expect to gain more from the lower complexity of our global constraint than from the increased pruning.

6.1 The balanced incomplete block design problem

Balanced Incomplete Block Design (BIBD) generation is a standard combinatorial problem from design theory with applications in cryptography and experimental design. A BIBD is specified by a binary matrix of b columns and v rows, with exactly r ones per row, k ones per column, and a scalar product of λ between any pair of distinct rows. Our model consists of sum constraints on each row and each column as well as the scalar product constraint between every pair of rows. Trivially, we can exchange any pair of rows, and any pair of columns of a solution to obtain another symmetrical solution. We therefore impose lexicographic ordering constraints on rows and columns.

We used a static variable ordering, tuned by initial experimentation, which gives the best results we have found so far on this problem. This ordering begins

Problem	GACLexLeq (Adjacent Pairs)			GACLexLeq (All Pairs)			Decomposition		
v, b, r, k, λ	Fails	Choice-points	Time	Fails	Choice-points	Time	Fails	Choice-points	Time
6,50,25,3,10	2738	2787	1.7	2738	2787	1.8	2758	2807	10.7
6,60,30,3,12	5924	5982	4.6	5924	5982	4.9	5959	6017	45
6,70,35,3,10	11731	11798	11.4	11731	11798	11.7	11787	11854	137.6
10,90,27,3,6	90610	90827	111	90610	90827	120.4	90610	90827	742.2
9,108,36,3,9	2428	2619	8.4	2428	2619	7.6	2428	2619	73.3
15,70,14,3,2	2798	3080	6.2	2798	3080	8.4	2798	3080	20.7
12,88,22,3,4	139988	140236	249	139988	140236	317	139988	140236	1153.6
9,120,40,3,10	1646	1858	8	1646	1858	7.2	1646	1858	81.5
10,120,36,3,8	577280	577532	1316.3	577280	577532	1132.3	—	—	—
13,104,24,3,4	114666	114999	397.6	114666	114999	448.3	114666	114999	1666.9

Table 1. BIBDs: Time is in seconds and a dash means no result is obtained in 1 hour.

by filling the first row left to right. Our value ordering is to try 0 then 1. Hence, this gives a first row with $b - r$ 0's followed by r 1's (by the row sum constraint). We then fill from top to bottom, the rightmost r columns from right to left. This is the most constrained part of the problem due to the λ constraints. Finally we fill the remaining rows alternately right to left, then left to right from top to bottom. Alternating directions favours the updating of α and β in **GACLexLeq**, and pruning in the decomposition.

Results are presented in Table 1. They indicate a substantial gain in efficiency by using **GACLexLeq** in preference to the decomposition. The similar size of the search trees explored indicates that the variable ordering is highly compatible with lexicographic ordering, and it is rare for the global constraint to be able to prune more than the decomposition. However, we report lower runtimes due to the increased efficiency of our **GACLexLeq** algorithm.

Although we have shown that, in theory, enforcing lexicographic ordering between all pairs of rows or columns can increase pruning, we do not see any evidence of it on these problems. In most cases, the increased overhead results in increased run-times. However, in three cases, enforcing lexicographic ordering between all pairs reduces the run-time. Since the size of the search tree remains the same, we conjecture that this is a result of a complex dead-end being detected more quickly with the extra global constraints.

6.2 The social golfer problem

The social golfer problem is to schedule a golf tournament over w weeks. In each week, the golfers must be divided into g groups of size s. Every golfer must play once in every week, and every pair of players can meet at most once. Our model consists of a 3-dimensional 0/1 matrix of groups $\times$ weeks $\times$ golfers. Assigning 1 to an entry at index $[g, w, p]$ means that golfer p plays in group g in week w. Sum constraints ensure that every golfer plays once in every week and that every group contains s players. A constraint similar in form to the λ constraint described above ensures that every pair of players meet only once.

Each of the weeks, golfers and groups are symmetrical and cannot be equal. In the first two cases, we impose a strict lexicographic ordering constraint between the planes of the matrix that represent weeks and the planes of the matrix that

Problem	GACLexLess (Adjacent Pairs)			GACLexLess (All Pairs)			Decomposition		
w, g, s	Fails	Choice-points	Time	Fails	Choice-points	Time	Fails	Choice-points	Time
11,6,2	166	305	0.7	116	305	0.8	314	453	7
13,7,2	1525	1792	9.7	1525	1792	10.4	16584	16851	547.2
5,6,3	2090342	2090435	2976.1	2090342	2090345	3120.6	—	—	—
4,7,3	248	371	0.7	248	371	0.8	340	463	8.2
5,8,3	625112	625328	2450.9	625112	625328	2655	—	—	—
4,5,4	1410774	1410825	1675	1410774	1410825	1865.3	—	—	—
3,6,4	2777416	2777486	2771.7	2777416	2777486	3073	—	—	—
3,7,4	646124	646235	1147.9	646124	646235	1173.7	—	—	—
9,8,4	27	380	3.3	27	380	3.6	32	385	68.9
2,7,5	32944	33029	53.2	32944	33029	55.7	58512	58597	959.1
2,8,5	42008	42123	93.2	42008	42123	91.2	122271	122386	3462.5
9,8,8	19	373	16	19	373	18.7	—	—	—

Table 2. Golfers: Time is in seconds and a dash means no result is obtained in 1 hour.

represent golfers. However, the contents of a group from one week to the next are independent of each other. Hence, we impose a strict lexicographic ordering between groups within each week. A static variable ordering was again used, filling the matrix a player at a time, assigning each week in turn. The results are presented in Table 2.

The more complex interactions resulting from 3-dimensional symmetry breaking result in more pruning with the global constraint than with the decomposition. Hence, we observe a reduction in the size of the search tree in many cases. As expected, this leads to an even larger reduction in run-times. Given the size of the vectors involved, the machine on which these experiments were run started to run low on memory using the decomposition. Better run times may be possible using a machine with more memory, but as we try to model larger problems this is a clear disadvantage of using decompositions.

Again, we see no evidence that enforcing lexicographic ordering between all pairs of vectors leads to increased pruning. Indeed, given the relatively large size of the vectors compared to those found in the BIBD models, we usually observe a small increase in overhead.

6.3 The sports tournament scheduling problem

In the sports tournament scheduling problem we have n teams playing over $n-1$ weeks. Each week is divided into $n/2$ periods, and each period divided into two slots. The first team in each slot plays at home, whilst the second plays the first team away. Every team must play once a week, every team plays at most twice in the same period over the tournament and every team plays every other team. We use a model consisting of two matrices proposed by Van Hentenryck et al. [10]. The first matrix, *teams* is a 3-dimensional matrix of periods $\times$ (extended) weeks $\times$ slots, each element of *teams* can take a value between 1 and n expressing that a team plays in a particular period in a particular week, in the home or away slot. Weeks are extended to include a dummy week to makes posting some constraints easier. The second matrix, *games* is a 2-dimensional matrix of periods $\times$ weeks, each element of which has a domain in $[1,\ n^2]$, recording a particular unique

	GACLexLess (Adjacent Pairs)			GACLexLess (All Pairs)			Decomposition		
n	Fails	Choice-points	Time	Fails	Choice-points	Time	Fails	Choice-points	Time
6	285	295	0.2	285	295	0.2	285	295	0.2
8	53	81	0.2	53	81	0.2	53	81	0.2
10	4931	4979	0.6	4931	4979	0.6	4933	4982	0.9
12	433282	433357	38.6	433282	433357	48.2	433283	433358	92.7
14	20404472	20404581	2016.7	20404472	20404581	2643.7	20439055	20439164	6516.8

Table 3. Sports: Time is in seconds.

combination of home and away teams. All-different and occurrence constraints are then used to enforce the problem constraints.

Symmetry on the slots is broken by specifying that the home team must be less than the away team (this is in fact essential for the *game* matrix to work correctly). In addition, the periods and the weeks are symmetrical and cannot be equal. Hence, we post a strict lexicographic ordering constraint between periods and weeks in *teams*. A static variable ordering on *teams*, tuned by initial experiments, is used as follows. We fill the first row left to right. We fill the 2nd row right to left. We fill the 1st column top to bottom. From then on we fill the remainder of the rows left to right, right to left from top to bottom. Again, alternating directions appears to help both the global constraint and the decomposition.

The results are presented in Table 3. They show that **GACLexLess** maintains a significant advantage over the decomposition on multi-valued domains. The variable ordering chosen is compatible with lexicographic ordering, so there is no difference in the size of the search tree. However, the benefits of having an efficient global constraint are again apparent.

7 Related work

Beldiceanu has classified many global constraints in terms of simple graph properties [1]. This classification provides hints at ways to implement pruning algorithms for the global constraints. However, our algorithms are developed by taking a different approach.

For ordering constraints, Gent et al. show that, GAC on a global monotonicity constraint on n variables (e.g. $x_0 < x_1 < \ldots < x_{n-1}$) is equivalent to AC on its decomposition into binary ordering constraints [9]. This is based on a result of Freuder as the decomposed constraint graph is a tree [7]. As arc-consistency can be efficiently enforced on the decomposed binary ordering constraints, there is no need for global consistency algorithms for simple orderings like monotonicity constraints. However, our results show both theoretically and empirically the value of global consistency algorithms for more complex orderings.

The ECLiPSe constraint solver provides a global constraint for lexicographically ordering two vectors. However, it is not documented what level of consistency is enforced with this constraint, nor the complexity of enforcement. It does no pruning on $\langle\{0,1\},\{0,1\},\{1\}\rangle \leq_{\text{lex}} \langle\{0,1\},\{0\},\{0\}\rangle$, even though the problem is not GAC.

8 Conclusions

We have proposed some global constraints for lexicographic orderings. These constraints are very useful for breaking symmetry. We show that decomposing such global constraints carries a penalty either in the amount or the cost of constraint propagation. We have therefore developed an $O(nd)$ global consistency algorithm which enforces a lexicographic ordering between two vectors of n variables and domains of size d. The algorithm can be modified very slightly to enforce a strict lexicographical ordering. Our experimental results confirm the efficiency and value of these new global constraints.

In our future work, we plan to use our global constraints in multi-criteria optimization problems. We also hope to develop algorithms for GAC on $\bar{x}_0 \leq_{\text{lex}} \bar{x}_1 \ldots \leq_{\text{lex}} \bar{x}_{m-1}$ and GAC on $\bar{x}_0 <_{\text{lex}} \bar{x}_1 \ldots <_{\text{lex}} \bar{x}_{m-1}$. The example in the proof of Theorem 7 suggests that this may be quite challenging. Such algorithms need to be able to perform sophisticated counting arguments and solve pigeonhole problems quickly. Global constraints for lexicographic orderings simultaneously along both rows and columns of a matrix would also present a significant challenge. Finally, we intend to look at global constraints for other orderings like the multiset ordering. Such orderings are also very useful for breaking symmetry.

References

1. N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. In *Proc. of CP'2000*, pages 52–66. Springer, 2000.
2. C. Bessière, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. In *Proc. of CP'99*, pages 88–102. Springer, 1999.
3. C. Bessière and J.C. Régin. Arc consistency for general constraint networks: Preliminary results. In *Proc. of IJCAI'97*, pages 398–404. Morgan Kaufmann, 1997.
4. M. Ehrgott and X. Gandibleux. A survey and annotated bibliography of multiobjective combinatorial optimization. *OR Spektrum*, 22:425–460, 2000.
5. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Symmetry in matrix models. Technical Report APES-30-2001, APES group, 2001. Available from http://www.dcs.st-and.ac.uk/~apes/reports/apes-30-2001.ps.gz. Presented at SymCon'01, CP'2001 post-conference workshop.
6. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Matrix modelling. Technical Report APES-36-2001, APES group, 2001. Available from http://www.dcs.st-and.ac.uk/~apes/reports/apes-36-2001.ps.gz. Presented at Formul'01, CP'2001 post-conference workshop.
7. E. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the Association for Computing Machinery*, 32(4):755–761, 1985.
8. A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Global constraints for lexicographical ordering. Technical Report. This paper is confidential, but will be made available to the CP program chair upon request.
9. I.P. Gent, K. Stergiou, and T. Walsh. Decomposable constraints. *Artificial Intelligence*, 123(1-2):133–156, 2000.
10. P. Van Hentenryck, L. Michel, L. Perron, and J.C. Regin. Constraint programming in OPL. In *Proc. of PPDP'99*, pages 98–116. Springer, 1999.