Ranking Constraints

Christian Bessiere
LIRMM-CNRS
Montpellier, France

Emmanuel Hebrard
LAAS-CNRS
Toulouse, France

George Katsirelos
INRA
Toulouse, France

Zeynep Kiziltan
Universit di Bologna
Bologna, Italy

Toby Walsh
University of New South Wales
Sydney, Australia

Abstract

We need to reason about rankings of objects in a wide variety of domains including information retrieval, sports tournaments, bibliometrics, and statistics. We propose a global constraint therefore for modeling rankings. One important application for rankings is in reasoning about the correlation or uncorrelation between sequences. For example, we might wish to have consecutive delivery schedules correlated to make it easier for clients and employees, or uncorrelated to avoid predictability and complacence. We therefore also consider global correlation constraints between rankings. For both ranking and correlation constraints, we propose efficient filtering algorithms and decompositions, and report experimental results demonstrating the promise of our proposed approach.

1 Introduction

In many problems we want to reason about the ranking of items. For example, in information retrieval, we often want to aggregate several search results. The results being aggregated together may contain ties, and so are rank orders (e.g. [Fagin *et al.*, 2004; Brancotte *et al.*, 2015]). As a second example, we may wish to construct an overall ranking of tennis player based on pairwise comparisons between players. One principled method for constructing a ranking is the Kemeny distance [Kemeny, 1959; Levenglick, 1975] as this is the unique scheme that is neutral, consistent, and Condorcet. Unfortunately, determining this ranking is NP-hard, and remains so when we permit ties in the input or output [Hemaspaandra *et al.*, 2005]. As a third example, as we discuss shortly, tasks in a scheduling problem may run in parallel, resulting in a ranking rather than a permutation of tasks.

In a ranking, unlike a permutation, we can have ties. Thus, 12225 is a ranking whilst 12345 is a permutation. To reason about permutations, we have some beautiful, efficient and effective global constraints. Regin [1994] proposed an $O(n^2d^2)$ domain consistency propagator for permutations where n is the number of variables and d is their domain size. This work won the 2013 AAAI Classic Paper award. For bound consistency, there is an even faster $O(n \log n)$ propagator [Ortiz *et al.*, 2003]. Every constraint toolkit now provides propagators for permutation constraints. Surprisingly, ranking constraints are not yet supported in any toolkit. We tackle this weakness by proposing a global ranking constraint.

One very important application for rankings is in reasoning about the correlation between two sequences. For instance, in a vehicle routing problem, we might wish to have a large correlation between routes on consecutive days. In this way, drivers can "learn the routes", and customers can see a predictable face at a predictable time. A standard method in statistics to achieve such correlation is to minimize the distance between the ranking of stops in a route. On the other hand, there are also applications where we want routes to be uncorrelated. For example, in delivering cash to ATMs, we may wish routes to be highly uncorrelated. We therefore also propose some global correlation constraints.

2 Background

Constraint Satisfaction

A Constraint Satisfaction Problem (CSP) is a triple $P = \langle X, D, C\rangle$ where X is a set of variables, D is a function from variables to sets of values and $D(X_i)$ is the domain of X_i. An assignment $\sigma(S)$ to $S \subseteq X$ is a function $S \to \cup_{X_i \in S} D(X_i)$ such that $\sigma(X_i) \in D(X_i)$. C is a set of constraints. A constraint c is a pair $\langle S_c, R_c\rangle$ where $S_c \subseteq X$ and R_c is a predicate over $\prod_{X_i \in S_c} D(X_i)$. If R_c is made true by $\sigma(S_c)$, c is satisfied by σ. We seek a $\sigma(X)$ that satisfies all constraints.

Let $\min(X)$ ($\max(X)$) be the minimum (max.) value in $D(X)$. A value $v \in D(X_i)$ is domain consistent (DC) in a constraint c if there is a $\sigma(S_c)$ that satisfies c such that $\sigma(X_i) = v$ and $\sigma(X_j) \in D(X_j) \forall X_j \in S_c \setminus \{X_i\}$ and range consistent (RC) if $\sigma(X_j) \in [\min(X_j), \max(X_j)] \forall X_j \in S_c \setminus \{X_i\}$. A constraint is DC (RC) if $\forall X_i \in S_c, v \in D(X_i)$, v is DC (RC). It is bound consistent (BC) if $\min(X_i)$ and $\max(X_i)$ are RC for all $X_i \in S_c$. It is domain (range) disentailed if all values are domain (range) inconsistent.

A RANKING on a set of variables ensures an assignment is a ranking under some ordering of the variables.

Definition 1 *A sequence R is a* ranking *(often called a* standard *ranking) iff either $R = (1)$, or $R = (x_1, \ldots, x_{m+1})$*

with $x_{m+1} = x_m$ or $x_{m+1} = m + 1$ and $(x_1, \ldots, x_m)$ is a ranking. $\text{RANKING}(X_1, \ldots, X_n)$ *iff the sorted sequence $(X_{\pi(1)}, \ldots, X_{\pi(n)})$ is a ranking.*

For example, $\text{RANKING}([4, 1, 2, 2])$ is satisfied but $\text{RANKING}([3, 1, 4, 3])$ is not.

Scheduling

In scheduling, we need to determine starting times for a set of tasks of a given duration, respecting their release and due dates, bounds on the utilization of resources, so that an objective like the makespan is minimized. In the "batching machine" model [Potts and Kovalyov, 2000], the resources (such as, for instance, an oven) have a capacity, but the processing time of the tasks scheduled in parallel equal the processing time of the longest task. In this model, the partial order on the equivalence classes "processed in parallel" is a ranking. In some scheduling problems the length of a task is time dependent [Gupta and Gupta, 1988], for instance because of a "learning curve" [Dutton and Thomas, 1984], or because of wear and tear, see [Biskup, 2008] for a survey. Some models relate the duration of a task to the number of times a similar task has been processed in the past [Biskup, 1999; Gordon *et al.*, 2008]. In our experiments, we used the model of Mosheiov [2005], where the processing time of task i is $r_i p_i$ where p_i is a constant and r_i the rank of task i.

Correlation Constraints

Let $\mathbf{X} = \{X_1, \ldots, X_n\}$ and $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$ be two rankings. They are positively correlated if the distance between $\mathbf{X}$ and $\mathbf{Y}$ is low, negatively correlated if this distance is high and uncorrelated if this distance is average. Several distance measures can be used to define a correlation coefficient. In this paper, we consider only Manhattan distance, a well known correlation metric. We measure correlation using the *Spearman's Footrule*, a simple measure of correlation between sequences based on Manhattan distance. We denote $m = \left\lfloor \frac{n^2}{4} \right\rfloor$ the median Manhattan distance between two rankings. If we consider the gap to the median $|\sum_{i=1}^{n} |X_i - Y_i| - m|$, we can enforce uncorrelation by stating an upper bound, or correlation (either positive or negative) by stating a lower bound. This gives the following two global correlation constraints where $\oplus \in \{\leq, \geq\}$:

Definition 2

$$\text{RANKINGCORRELATION}_{\oplus}(\mathbf{X}, \mathbf{Y}, C) \iff \left| \sum_{i=1}^{n} |X_i - Y_i| - m \right| \oplus C \& \text{RANKING}(\mathbf{X}) \& \text{RANKING}(\mathbf{Y})$$

3 The Ranking Constraint

We introduce two decompositions for the RANKING constraint. The first uses the $\text{SORTEDNESS}(\mathbf{X}, \mathbf{Y})$ constraint [Older *et al.*, 1995], which ensures that $\mathbf{Y}$ is a sorted version of the sequence $\mathbf{X}$ and which is itself efficiently decomposable [Schaus, 2010]:

$$\begin{aligned} &\text{SORTEDNESS}(\mathbf{X}, \mathbf{Y}), \quad Y_1 = 1 \\ &\qquad Y_i = Y_{i-1} \vee Y_i = i \quad \forall i \in [2, n] \end{aligned}$$

The second decomposition uses the $\text{GCC}(\mathbf{X}, V, \mathbf{Y})$ constraint [Oplobedu *et al.*, 1989], which ensures that each value $v \in V$ appears Y_v times[1] in the sequence $\mathbf{X}$:

$$\begin{aligned} &\text{GCC}(\mathbf{X}, \{1, \ldots, n\}, \mathbf{Y}), \quad Y_1 = Z_1 \\ &\qquad Z_i = Z_{i-1} + Y_i \quad \forall i \in [2, n] \\ &\qquad Z_i \geq i \quad \forall i \in [1, n] \\ &\qquad Y_i = 0 \iff Z_{i-1} \geq i \quad \forall i \in [2, n] \end{aligned}$$

Proposition 1 *Neither encoding achieves BC*

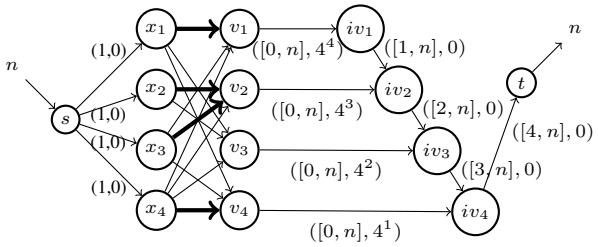
Proof: Consider $X_1, X_2 \in [1, 5], X_3 = 4$ and $X_4, X_5, X_6 \in [2, 3]$. This instance does not admit a ranking. However neither decomposition identifies this. □

We next show that domain consistency can be enforced in polynomial time. However, our propagator requires many $(O(n^2))$ calls to a min cost flow algorithm, so we also propose an efficient algorithm for reasoning with interval domains.

3.1 Domain Consistency

High level description. We reformulate the problem as a lexicographic maximum flow [Kozen, 2009], modeled using exponentially large costs. If the lex-max flow is not a ranking, we force a lexicographically smaller choice at the point where the ranking property is violated. Since there are only two choices at each point of the sorted sequence, the smaller choice is true in all solutions and will not be revisited, although previous choices may have to be revised. Hence, we make $O(n^2)$ revisions and get a polynomial runtime. We present this in Algorithm 1.

Figure 1: Network flow for a ranking with 4 variables. Arcs are labelled with $([demand, capacity], cost)$, or $(demand, cost)$ if $demand = capacity$.



Construction. The construction is similar to that of nested GCC [Zanarini and Pesant, 2007] over the intervals $[1, 1], [1, 2], \ldots, [1, n]$. There exists a unique source s and sink t, a vertex x_i for each variable X_i and arcs (s, x_i) with demand 1 and cost 0. There exists a vertex v_j for each value j and an arc (x_i, j) iff $j \in D(X_i)$. For each $j \in [1, n]$, there exists a vertex iv_j for the interval $[1, j]$ and an arc (v_j, iv_j) with cost $n^{n-i+1} = 2^{(n-i+1)\log n}$. For each $j \in [2, n]$ there is an arc (iv_{j-1}, iv_j) with demand $j - 1$. Finally, there is an arc (iv_n, t) with demand n. Unless otherwise mentioned, all arcs have maximal capacity, no demand and no cost. We denote this network by $N(X_1, \ldots, X_n)$.

[1] The version where $\mathbf{Y}$ is a set of integer variables is called "extended" GCC in [Quimper, 2006].

Algorithm 1: DCSupport($X_1, \ldots, X_n$)

$G = N(X_1, \ldots, X_n)$;
while *true* **do**
  $F = \text{MinCostFlow}(G, n)$; // n units of flow
  if *There exists no flow* **then** FAIL ;
  $\sigma = \{X_i = j \mid F(x_i, v_j) > 0\}$;
  if σ *is a ranking* **then return** σ ;
  $o \leftarrow sort(\sigma)$; // by assigned value
  $r \leftarrow$ first position in o that violates the ranking property ;
  $p \leftarrow \max\{j \mid \exists i, k \text{ s.t. } o(k) = \{X_i = j\}, k < r\}$;
  set demand of arc (iv_p, iv_{p+1}) in G to r;

Theorem 1 *Algorithm 1 returns a support of a* RANKING *constraint iff the constraint is satisfiable, in polynomial time.*

Proof: $\Rightarrow$ Immediate.

$\Leftarrow$ Consider a minimum cost feasible flow of G. In the usual way, we construct an assignment σ to the variables of the constraint. Let $Z_1, \ldots, Z_n$ be the sorted assignment, i.e., there exists a permutation π such that $Z_i = \sigma(X_{\pi(i)})$ and $Z_i \leq Z_{i+1}$ for all $i \in [1, n-1]$. This sequence has the property $Z_1 = 1$, $Z_r \in [Z_{r-1}, r]$ for $2 \leq r \leq n$, otherwise the demand on (iv_{r-1}, iv_r) would be violated. This is weaker than the ranking property $Z_1 = 1$, $Z_r \in \{Z_{r-1}, r\}$. Suppose $Z_1, \ldots, Z_n$ is not a ranking. Then there exists a minimum r such that $Z_1, \ldots, Z_{r-1} = p$ is a ranking and $Z_r \in (p, r)$. By construction, we place priority on higher values, hence no flow extends $Z_1, \ldots, Z_{r-1}$ with $Z_r = r$, nor is there a flow in which any of $Z_i, i \leq r$ gets a larger value. Hence, any ranking is lexicographically smaller than $Z_1, \ldots, Z_r$ and if it matches $Z_1, \ldots, Z_{r-1}$ then $Z_r = p$. This is enforced by requiring at least r values to be smaller than p, by setting the demand of the edge (iv_p, iv_{p+1}) to r.

Since the lexicographic upper bound of the flow is reduced at each iteration, the algorithm will eventually find a flow that is a ranking or report that none exists. In that case, the constraint is unsatisfiable, by the correctness of the bound. The demand of one of the n edges (iv_i, iv_{i+1}) is increased at each iteration and the total flow is n, so there are $O(n^2)$ iterations. The weights are represented with $n \log n$ bits, so computing the flow is polynomial, as is the entire algorithm. □

Note that given the above algorithm we can achieve domain consistency by probing (see also section 3.2).

3.2 Bounds Consistency

Support. Algorithm 2 computes the lexicographically maximum BC support σ, if one exists, and fails otherwise. It maintains a partition of the variables into assigned (A) or unassigned (U). Suppose it extracts the variables in U in the order $X_{i_1}, \ldots, X_{i_n}$. It always tries to assign to variable X_{i_k} the value k (line 2), while line 1 ensures that X_{i_k} has minimum upper bound among those that contain k or fails if no such variable exists. Construction of a ranking can continue either with the value k or $|A| + 1$ ($= k + 1$ in the first iteration of the loop at line 3). If some variables do not contain $|A|+1$, it assigns them k (line 5). These are *forced* variables. This may create further forced variables. If the domain of a forced variable does not contain k either, it fails (line 5).

Algorithm 2: RCSupport($X_1, \ldots, X_n$)

$U \leftarrow \{1, \ldots, n\}$; $A \leftarrow \emptyset$; $\sigma \leftarrow []$;
while $U \neq \emptyset$ **do**
  $k \leftarrow |A| + 1$;
1 pop $\arg\min_{i \mid k \in D(X_i)}(\max(X_j))$ from U or FAIL;
2 $\sigma[i] \leftarrow k$; $A \leftarrow A \cup \{i\}$; $F \leftarrow \emptyset$;
3 **while** $\exists X_j \in U$ *s.t.* $\max(X_j) < |A| + 1$ **do**
    $F \leftarrow F \cup \{j \mid j \in U \wedge \max(X_j) < |A| + 1\}$;
    $U \leftarrow U \setminus F$; $A \leftarrow A \cup F$;
  if $|F| > 0$ **then**
4 **if** $\exists j \in F$ *s.t.* $k \notin D(X_j)$ **then** FAIL;
5 **else** $\sigma[F] \leftarrow k$;

Example 1 *Consider the domains*

$D(X_1)$	=	***1***	2					
$D(X_2)$	=	1	***2***					
$D(X_3)$	=	1	2	***3***				
$D(X_4)$	=		2	***3***				
$D(X_5)$	=	1	2	***3***	4			
$D(X_6)$	=			3	4	5	***6***	
$D(X_7)$	=		2	3	4	5	6	***7***
$D(X_8)$	=				4	5	6	***7***
$D(X_9)$	=				4	5	6	***7***

Algorithm 2 orders the variables as given and finds the support marked in bold. X_4, X_5, X_8 and X_9 are forced.

Theorem 2 *Algorithm 2 returns a valid BC support if one exists and fails otherwise, in time $O(n \log n)$*

Proof: Algorithm 2 constructs an assignment as well as a total variable ordering which agrees with the partial ordering where variables are ordered by their rank. The total order is given by the order in which variables are popped from A (for variables in M, we choose arbitrarily). We show that the algorithm constructs the lexicographically maximal among the non-decreasing solutions in this ordering if and only if the constraint has a bounds support.

($\Rightarrow$) This is straightforward.

($\Leftarrow$) Suppose the RANKING constraint has a bounds support but algorithm 2 fails. At the point where it fails, it has constructed a total ordering o_a of a subset $\mathbf{X}' \subset \mathbf{X} = [X_1, \ldots, X_n]$. Now let σ_1 be a bounds support of the constraint. We extend the partial order induced by the ranking of σ_1 to a total order o_s arbitrarily and consider σ_{max}, the lexicographically maximal among the solutions that are non-decreasing in the order o_s. For these total orders, we write $o(i)$ to denote the variable in the ith position.

Since the algorithm failed without producing a solution, either o_a disagrees with o_s or σ disagrees with σ_{max} in $\mathbf{X}'$. We show that there exists an ordering and corresponding lex-max non-decreasing solution that do not disagree with o_a and σ, a contradiction. We use induction on the position q of the first disagreement between either o_a and o_s or σ and σ_{max}.

Base case. For $q = 1$, the only possible value is 1, so the only possible disagreement is in the ordering. Let $o_a(1) = i$, $o_s(1) = j$. Since X_i is chosen such that $\max(X_i)$ is minimum, it follows $\sigma(i) \geq \max(X_j)$. Thus we can swap X_i and X_j in the ordering and in σ and get a new ordering o'_s and

assignment σ'_{max} that agree with o_a and σ at position 1.

Inductive step. Assume now that o_a and σ agree with o_s and σ_{max} until the $q-1$th position.

(1) Suppose first that $o_a(q) = o_s(q) = i$ but $\sigma(i) \neq \sigma_{max}(i)$. Observe that the ranking $o_s(1)\dots o_s(q-1)$ can be only be extended by $p = \sigma_{max}(o_s(q-1))$ (i.e., continue a sequence of p ranks, $\sigma_{max}(i) = p$) or by $q > p$. Algorithm 2 chooses $\sigma(i) = q$, so it follows that $\sigma_{max}(i) = p$. Let $r \geq q$ be the greatest index such that $\sigma_{max}(o_s(r)) = p$ and hence by the above reasoning $\sigma_{max}(o_s(r+1)) = r+1$. For all variables $X_j, j \in [o_{max}(q+1), o_{max}(r)]$, it must be $\min(X_j) \leq p < q$ and $\max(X_j) \geq q$, otherwise they would have been chosen before X_i by algorithm 2, which is impossible since o_a and o_s agree until position q. Hence we can construct $\sigma'_{max}(j) = \sigma_{max}(j)$ if $j \notin [q, r]$ and q otherwise. In other words we transform the ranking $(\underbrace{1, \dots, p, \dots, p}_{1\dots r}, r+1, \dots)$ to $(\underbrace{1, \dots, p, \dots, p}_{1\dots q-1}, \underbrace{q, \dots, q}_{q\dots r}, r+1, \dots)$. The new assignment σ'_{max} is a ranking, non-decreasing over o_s and lexicographically greater than σ_{max}, a contradiction.

(2) Suppose o_a disagrees with o_s at position q so that $o_a(q) = i$ and $o_s(q) = j$ but $\sigma(i) = \sigma_{max}(j)$. As in the base case ($q = 1$), we can swap i and j in σ_{max} and o_s to get σ'_{max} and o'_s that agree with σ and o_a.

(3) Finally, assume $o_a(q) = i, o_s(q) = j \neq i$ and $\sigma(i) \neq \sigma_{max}(j)$. By the argument in case (1), σ_{max} cannot be a lexicographically maximal solution over o_s.

Complexity. In each iteration we assign at least one variable, hence the loop runs $O(n)$ times. All operations are in O(1), except line 1, which is in $O(\log n)$ using a binary heap, for a total $O(n \log n)$. □

Probing propagator

Based on algorithm 2, we can enforce range consistency on a RANKING constraint by "probing", i.e., asserting for each $v \in D(X_i)$ $X_i = v$ and looking for a bound support. If none exists, that value is pruned. The cost of this is $O(n^3 \log n)$, as it runs algorithm 2 once for each value in each domain. Hence, we explore computationally cheaper alternatives.

Pruning Conditions.

We first identify the conditions under which algorithm 2 can fail and for when a value is range inconsistent.

Definition 3 ((super-)Hall intervals) *Let $V_{\sqsubseteq}(a,b) = \{X \mid D(X) \subseteq [a,b]\}$ and $S(a,b) = |V_{\sqsubseteq}(a,b)|$. We say that $[a,b]$ is a Hall interval if $S(a,b) = b - a + 1$ and a super-Hall interval if $S(a,b) > b - a + 1$. We write $\Delta(a,b) = [b+1, a+S(a,b)-1]$, $\underline{\Delta}(a,b) = [b+1, a+S(a,b)]$.*

Lemma 1 *If a ranking constraint contains a super-Hall interval $[a,b]$, then $\forall X_i \in \mathbf{X}$, $X_i \notin \Delta(a,b)$. Moreover, if $[a,b]$ is a Hall interval, then $\forall X_i \in \mathbf{X} \setminus V_{\sqsubseteq}(a,b)$, $X_i \in [a,b] \rightarrow \forall X_j \in \mathbf{X}. X_j \notin \underline{\Delta}(a,b)$.*

Proof: Suppose there exists a ranking that violates this condition, i.e., $\exists X_i \in \Delta(a,b)$. Since $\min(\underline{\Delta}(a,b)) = b+1$, every variable in $V_{\sqsubseteq}(a,b)$ is ranked strictly higher than X_i, and the lowest possible rank among them is a. So the value assigned to X_i must be at least $a + S(a,b)$.

For the second claim, if the left hand side is satisfied, $S(a,b)$ increases, making it equivalent to the first claim. □

The first condition is also correct for Hall intervals, but $\Delta(a,b) = [b+1, a+S(a,b)-1] = [b+1, a+b-a] = \emptyset$, so we achieve no pruning.

Example 2 *Consider again the instance* RANKING $([X_1, \dots, X_9])$ *from Example 1. The interval $[1,2]$ is a Hall interval with $V_{\sqsubseteq}(1,2) = \{X_1, X_2\}$. The interval $[1,3]$ is a super-Hall interval with $V_{\sqsubseteq}(1,3) = \{X_1, X_2, X_3, X_4\}$ and entails that $[4,4]$ is inconsistent for all variables.*

Definition 4 (Saturated/Failed value) *A value v such that $v = |V_{\sqcap}(1,v)|$, where $V_{\sqcap}(1,v) = \{X_i \mid \min(X_i) \leq v\}$, is called a saturated value. It is a failed value if $v > |V_{\sqcap}(1,v)|$.*

Lemma 2 *If a* RANKING *constraint contains a failed value it is unsatisfiable. If it contains a saturated value v, then in every solution σ, $\sigma(X_i) \leq v, \forall X_i \in V_{\sqcap}(1,v)$.*

Proof: The first claim follows from the fact that in a ranking the kth value is at most k, but when there exists a failed value v, if X_j is at position $|V_{\sqcap}(1,v)| + 1 \leq v$, $[1,v] \notin D(X_j)$. The second claim follows because if we made any of these assignments, v would become a failed value. □

Theorem 3 *A* RANKING *constraint is bounds disentailed if and only if either (a) the constraint contains a failed value; or (b) The constraint contains a variable X and a set $\mathbf{S}$ of super-Hall intervals such that $dom(X) \subseteq \cup_{[a,b]\in\mathbf{S}} \Delta(a,b)$.*

Proof: $\Leftarrow$. This follows from lemmas 1 and 2.

$\Rightarrow$ By theorem 2 it suffices to show this for algorithm 2.

Algorithm 2 fails only in lines 1 and 4. In the first case (line 1), k is failed value. Indeed, $k - 1 = n - |A|$ variables have been assigned and none of the remaining variables contain k, hence their lower bound is greater than k. So $|vars(k)| = k - 1$ and k is a failed value.

For the second case (line 4), the algorithm fails because a variable X_j is in F but does not contain the value k, so $\max X_j < k$. Assume w.l.o.g. that the variables are chosen in the order $X_1, \dots, X_n$. We claim that each time $F \neq \emptyset$, the algorithm has identified a super-Hall set. We say that a variable assigned in line 5 is *forced*. Let $F = \{X_j, \dots, X_k\}$ and X_i be the latest unforced variable with $\sigma(X_i) = \min(X_i)$. (X_1 in the extreme). Then we show that the interval $[a,b]$ where $a = \min X_i$ and $b = \max_{p\in M} D(X_p)$ is a super-Hall interval and $V_{\sqsubseteq}(a,b) = \{X_i, \dots, X_k\}$.

We have that $b < k$ because $X_k \in F$ and, because X_i was not forced, $i < j$ and $a = i$. For all $X_p, i \leq p < j$, we have that $\max X_p \leq b$ as well. If not, suppose p is the greatest index such that $\max X_p > b$. Since $\max X_{p+1} \leq b$ and X_p is chosen before X_{p+1}, it must be because X_p gets a value v which is not in the domain of X_{p+1}. It also means that X_{p+1} gets $v + 1 = \min X_{p+1}$ and is unforced, contrary to the assumption that X_i is the latest such variable. Hence, $\{X_i, \dots, X_k\} \subseteq V_{\sqsubseteq}(a,b)$. Moreover, we have $|[a,b]| = b - a + 1 < k - i + 1 = |\{X_i, \dots, X_k\}| \leq |V_{\sqsubseteq}(a,b)|$, proving that $[a,b]$ is a super-Hall interval. Hence the values that the algorithm skips when constructing a support are exactly those that are in $\Delta(a,b)$ of some super-Hall interval $[a,b]$.

Consider now the failed variable X_j. Suppose algorithm 2 has used a value in $D(X_j)$ and let v be the largest one. When

it used v, X_j would be in F and it would be assigned v, which did not happen. Hence, algorithm 2 has skipped over all of $D(X_j)$, so there exists a series of super-Hall intervals whose $\Delta(a,b)$ cover $D(X_j)$, as required. □

Lemma 3 *An assignment $X_i = v$ is range inconsistent in* RANKING *constraint C if and only if (a) There exists a super-Hall interval $[a,b]$ s.t. $v \in \underline{\Delta}(a,b)$; or (b) The constraint contains a saturated value $v' < v$ and $X_i \in V_\sqcap(1,v')$; or (c) The constraint contains a variable X_j and two sets of Hall intervals S_1, S_2 such that $\forall [a,b] \in S_1, v \in [a,b] \wedge X_i \notin V_\sqsubseteq(a,b)$, and $dom(X_j) \subseteq \big(\cup_{[a,b]\in S_1} \underline{\Delta}(a,b)\big) \bigcup \big(\cup_{[a,b]\in S_2} \Delta(a,b)\big)$.*

Proof:[Sketch] $\Leftarrow$. This is immediate by lemmas 1 and 2.

$\Rightarrow$. We examine the constraint $C \mid_{X_i=v}$ which is range disentailed and derive the above conditions. □

The third condition means that there exists X_j whose values are either range inconsistent by super-Hall intervals in S_2 or incompatible with $X_i = v$ by Hall intervals in S_1.

Filtering Algorithm

From Lemma 3 we can design an algorithm to enforce range consistency (RC), but its cost is similar to that of the probing propagator, so we propose an incomplete method instead.

Saturated values. We iterate over the variables sorted by non-decreasing upper bounds. Since the upper bound is n, sorting is in $O(n)$ [Cormen *et al.*, 2009, Chapter 8.2]. At step j we explore X_{i_j}, and if $\min(X_{i_j}) \geq j$ then $|V_\sqcap(1, j-1)| \leq j-1$, so $j-1$ is failed (for strict inequality) or saturated, in which case we prune the upper bounds of $V_\sqcap(1, j-1)$.

Super-Hall intervals. The second type of pruning comes from super-Hall intervals, i.e., if $[a,b]$ is such that $S(a,b) > b - a + 1$, then no variable can take a value in the interval $[b+1, a+S(a,b)-1]$. This can be achieved by computing all left-maximal Hall intervals as described in [Ortiz *et al.*, 2003] with the difference that we continue when a Hall interval becomes a "super-Hall interval". This algorithm runs in $O(n \log n)$ and returns $O(n)$ left-maximal Hall intervals. It explores the variables ordered by non-decreasing upper bound. For convenience, let X_i be the i-th such variable. We maintain at each step i, and for the lower bound a of each left-maximal interval, the value of $S(a,b)$ where b is $\max(X_i)$.

Definition 5 *An interval [a,b] is left-maximal if there does not exist a value a' such that $S(a',b) \geq S(a,b) + a - a'$.*

Notice that if $[a,b]$ is not left-maximal because of $[a',b], a' < a$ any subsequent Hall interval $[a,c], c > b$ is also not left-maximal because of $[a',c]$. Lemma 4 shows that the pruning due to non-left-maximal super-Hall intervals is subsumed by left-maximal Hall intervals.

Lemma 4 *If the super-Hall interval $[a,b]$ is not left-maximal but $[a',b]$ is then $\underline{\Delta}(a,b) \subseteq \underline{\Delta}(a',b)$.*

Proof: By definition, we have $S(a',b) \geq S(a,b) + a - a'$ hence $a' + S(a',b) \geq a + S(a,b)$. □

Therefore, we do not need to keep a as a possible lower bound for a future Hall interval and we can use the algorithm described by Ortiz *et al.* (2003). However, we continue to maintain a value $S(a,b)$ even when it is strictly larger than $a - b + 1$, instead of pruning. When we move to the next variable X_{i+1}, if $\max(X_{i+1}) = \max(X_i)$ we increment $S(a, \max(X_i))$ and add $[a, \max(X_i)]$ to a list otherwise.

Backward pruning from (super-)Hall intervals. To enforce the pruning corresponding to case (c) in Lemma 3, we want to find a variable X whose domain is included into the union of $\underline{\Delta}(a,b)$ for some intervals $[a,b]$. As shown in Lemma 1, if a variable Y whose domain is not contained in the union of those intervals takes a value in their intersection, then as $\Delta(a,b)$ increases, it will wipe out the domain of X. The values in the intersection are thus inconsistent for Y.

We give a $O(n^2)$ algorithm to achieve this with respect to every subset of left-maximal Hall intervals. However, one can prune even with respect to non left-maximal Hall intervals, this algorithm is therefore not complete.

We can make the two following simple observations:

Lemma 5 *There are $O(n)$ left-maximal (super-)Hall intervals, each with a distinct upper bound.*

Proof: Consider two left-maximal intervals $[a,b]$ and $[a',b]$ with $a' > a$. Then by definition there exists no value a'' such that $S(a'',b) \geq S(a',b) + a' - a''$. But $a'' = a$ is exactly such a value, a contradiction. □

Lemma 6 *If $j > i$ then either $a_i \geq a_j$ or $a_j > b_i$.*

Proof: Suppose that $a_i < a_j$ and $a_j \leq b_i$. By Lemma 5, $[a_i, b_i]$ is the unique left-maximal (super-)Hall interval with upper bound b_i, so $[a_j, b_i]$ is not left-maximal. Therefore, at least $a_i - a_j$ variables have their domain in $[a_i, b_i]$ and overlapping $[a_i, a_j - 1]$. Since these variables are also in $[a_i, b_j]$, it is left maximal and $[a_j, b_j]$ is not, a contradiction. □

These observations allow to speed up the pruning. By Lemma 5, we know that the (super-)Hall intervals $[a_j, b_j]$ for $j \in [1,m]$ are ordered by upper bounds. Therefore, the intervals $\underline{\Delta}(a_j, b_j)$ are ordered both by lower and upper bounds (since they are left-maximal). We explore variables by non-decreasing lower bound and find, at step i, the greatest l and smallest u such that $D(X_i) \in \bigcup_{j=l}^{u} \underline{\Delta}(a_j, b_j)$, and continue otherwise. When we find such a set, we can prune the intersection of the Hall intervals from the domain of any variable whose domain is not included in their union.

Finding l such that $\min(X_i) \in \underline{\Delta}(a_j, b_j)$ can be done in logarithmic time by binary search (similarly for u). Then, by Lemma 6 there are two cases: Either $a_u > b_l$ and then the intervals $[a_l, b_l]$ and $[a_u, b_u]$ are disjoint, which means that there is no possible pruning, or $a_u \leq a_l$. However, we know by Lemma 5 that $b_u > b_l$. Therefore $[a_l, b_l] \subseteq, \ldots, \subseteq [a_u, b_u]$. Computing the intersection or the union can thus be done in constant time: the union is $[a_u, b_u]$ and the intersection $[a_l, b_l]$. Thus, this takes $O(n \log n)$ time. However, we need $O(n^2)$ time to actually achieve the pruning. At step i we review every variable to check if its domain is contained in $[a_u, b_u]$ and not disjoint to $[l_b, u_b]$, and if so, keep a memory of this pruning. After processing every variable, we actually perform the stored pruning in $O(n^2)$.

4 Experimental Evaluation

Here we compare our propagator for the RANKING constraint against the probing RC algorithm and the two decompositions. We used Choco 3 to implement the two propagators as well as the decompositions and we ran all the experiments on a cluster of AMD opteron 6176 2.3 GHz processors.

Choco 3 uses the algorithm of Mehlhorn and Thiel (2000) to propagate the SORTEDNESS constraint and an unspecified algorithm to propagate the GCC constraint.

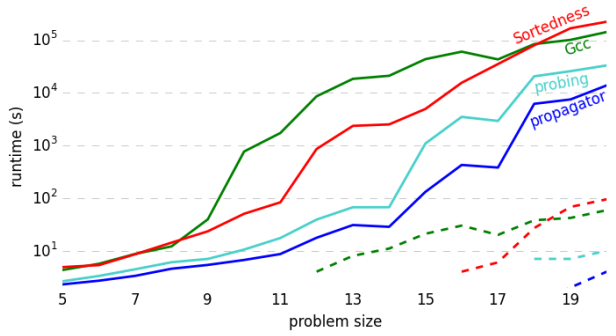


Figure 2: Uncorrelation: Random uniform intervals

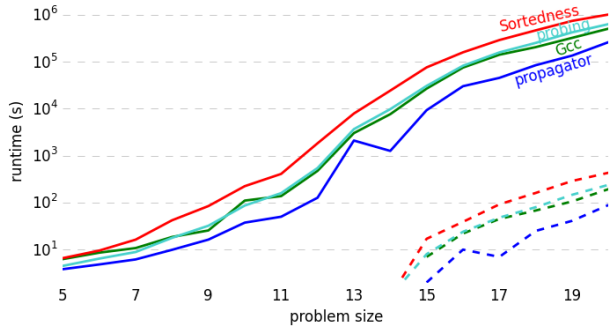


Figure 3: Uncorrelation: Solution embedded

Correlation and Uncorrelation. First, we considered the problem of minimizing the correlation between two vectors of variables $\mathbf{X}$ and $\mathbf{Y}$ using the RANKING CORRELATION constraint described in Section 2. We generated random ranges for all variables using two methods. First, we randomly picked two integers in $[1, n]$ for every variable and used these as bounds. As this method may produce unsatisfiable instances, we also used a second method, where we first generate a ranking r as follows: $r[1] = 1$, and for $i > 1$, we set $r[i] = r[i-1]$ with probability $\frac{1}{2}$ and $r[i] = i$ otherwise. Then for every variable X_i we set its bounds to a random interval that includes $r[i]$. We used lexicographic variable ordering and branched on the minimum value in all cases, so the same search tree is explored, modulo pruning.

For every value of n in $[5, 20]$, we generated 1000 random instances, and solved them with each method with a 30 minute time limit. Figure 2 shows the total runtime spent by the different methods for random intervals on each set of instances. The gain from the extra pruning is very clear (notice the logarithmic scale). At the end of the scale, the methods with weaker pruning cannot solve a significant proportion of the instance (dashed lines show the number of instances for which the time limit was reached). As a consequence, the gap in runtime is underestimated for the larger instances. The decomposition with SORTEDNESS seems to dominate (by orders of magnitude) the decomposition with GCC in this case.

Figure 3 shows results on instances with a solution embedded. The gain from stronger filtering is not as significant, but our propagator is still at least one order of magnitude faster than the decompositions. Interestingly, the relative performance of the decompositions is inversed here.

Scheduling. We also considered an application of the RANKING constraints to scheduling problems in the batching machine model with wear and tear. We generate such scheduling problems following the model of Mosheiov [Mosheiov, 2005]. For a number of tasks ranging from $n = 5$ to 10, we generate 50 such scheduling instances, choosing duration constants p_i and demand at random. Then we post a CUMULATIVE constraint, as well as the channeling constraints between overlap and ranking and between ranking and actual durations $r_i p_i$. The runtimes, in seconds, to solve each set of 50 instances are reported in Table 1, in the last column we also report the number of instances of size 10 that timed out after 3h. This problem is very hard for Choco, as only small problems can be tackled, but using our propagator improves scalability. In contrast to the uncorrelation problem, here the cost of propagating RANKING is small compared to the rest of the constraints, so the impact of the cost of the probing propagator is small. However, it remains measurably slower, suggesting that the extra pruning it may generate does not offset its higher computational cost. Once again, the two decompositions are worse than both propagators.

Table 1: Scheduling: Total runtime in seconds

size:	5	6	7	8	9	10	#U
propagator	97	**359**	**1203**	**3795**	**29302**	**275237**	**14**
probing	**73**	371	1428	4721	33975	276240	15
GCC	106	432	1389	4936	54495	323433	19
SORTEDNESS	104	458	1509	6118	65445	328331	17

5 Conclusions

We have proposed a global ranking constraint. We argued that simple decompositions of this global constraint hurt pruning. We proposed instead some efficient filtering algorithms. One application for such propagators is ensuring two sequences are correlated or uncorrelated. To demonstrate the promise of these methods, we ran experiments on two problem domains and observed significant speedups compared to the decompositions.

References

[Biskup, 1999] Dirk Biskup. Single-machine scheduling with learning considerations. *European Journal of Operational Research*, 115(1):173–178, 1999.

[Biskup, 2008] Dirk Biskup. A state-of-the-art review on scheduling with learning effects. *European Journal of Operational Research*, 188(2):315–329, 2008.

[Brancotte *et al.*, 2015] Bryan Brancotte, Bo Yang, Guillaume Blin, Alain Denise Sarah Cohen-Boulakia, and Sylvie Hamel. Rank aggregation with ties: Experiments and analysis. *Proceedings of the VLDB Endowment (PVLDB)*, 2015.

[Cormen *et al.*, 2009] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[Dutton and Thomas, 1984] John M. Dutton and Annie Thomas. Treating Progress Functions as a Managerial Opportunity. *The Academy of Management Review*, 9(2):235–247, April 1984.

[Fagin *et al.*, 2004] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *Proceedings of the Twenty-third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '04, pages 47–58, New York, NY, USA, 2004. ACM.

[Gordon *et al.*, 2008] V. S. Gordon, C. N. Potts, V. A. Strusevich, and J. D. Whitehead. Single machine scheduling models with deterioration andlearning: handling precedence constraints viaprioritygeneration. *Journal of Scheduling*, 11(5):357–370, 2008.

[Gupta and Gupta, 1988] Jatinder N.D. Gupta and Sushil K. Gupta. Single facility scheduling with nonlinear processing times. *Computers & Industrial Engineering*, 14(4):387 – 393, 1988.

[Hemaspaandra *et al.*, 2005] Edith Hemaspaandra, Holger Spakowski, and Jrg Vogel. The complexity of kemeny elections. *Theoretical Computer Science*, 349(3):382 – 391, 2005.

[Kemeny, 1959] John G Kemeny. Mathematics without numbers. *Daedalus*, 88(4):577–591, 1959.

[Kozen, 2009] Dexter Kozen. Lexicographic flow. Technical Report http://hdl.handle.net/1813/13018, Computing and Information Science, Cornell University, June 2009.

[Levenglick, 1975] Arthur Levenglick. Fair and reasonable election systems. *Behavioral Science*, 20(1):34–46, 1975.

[Mehlhorn and Thiel, 2000] Kurt Mehlhorn and Sven Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. In *Principles and Practice of Constraint Programming–CP 2000*, pages 306–319. Springer, 2000.

[Mosheiov, 2005] Gur Mosheiov. A note on scheduling deteriorating jobs. *Mathematical and Computer Modelling*, 41(89):883 – 886, 2005.

[Older *et al.*, 1995] W.J. Older, G.M. Swinkels, and M.H. van Emden. Getting to the real problem: experience with BNR Prolog in OR. In *Proceedings of the Third Conference on Practical Applications of Prolog*, pages 465–478, 1995.

[Oplobedu *et al.*, 1989] A. Oplobedu, J. Marcovitch, and Y. Tourbier. CHARME: Un langage industriel de programmation par contraintes, illustré par une application chez Renault. In *Ninth International Workshop on Expert Systems and their Applications: General Conference*, volume 1, pages 55–70, 1989.

[Ortiz *et al.*, 2003] Alejandro Ortiz, Claude-Guy Quimper, John Tromp, and Peter van Beek. A fast and simple algorithm for bounds consistency of the alldifferent constraint. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 245–250, Acapulco, Mexico, 2003.

[Potts and Kovalyov, 2000] Chris N. Potts and Mikhail Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228 – 249, 2000.

[Quimper, 2006] Claude-Guy Quimper. *Efficient propagators for global constraints*. PhD thesis, University of Waterloo, 2006.

[Régin, 1994] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on AI*, pages 362–367. AAAI, 1994.

[Schaus, 2010] Pierre Schaus. http://cp-is-fun.blogspot.com/2010/09/recent-trend-of-cp-that-i-noticed-is.html, 2010.

[Zanarini and Pesant, 2007] Alessandro Zanarini and Gilles Pesant. Generalizations of the global cardinality constraint for hierarchical resources. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, pages 361–375. Springer, 2007.