Sweep as a Generic Pruning Technique Applied to Constraint Relaxation

Nicolas Beldiceanu and Mats Carlsson

SICS, Lägerhyddsvägen 18, SE-75237 Uppsala, Sweden
{nicolas,matsc}@sics.se

Abstract. We introduce a new generic filtering algorithm for handling constraint relaxation within constraint programming. More precisely, we first present a generic pruning technique which is useful for a special case of the *cardinality* operator where all the constraints have at least two variables in common. This method is based on a generalization of a sweep algorithm which handles a conjunction of constraints to the case where one just knows the minimum and maximum number of constraints that have to hold. The main benefit of this new technique comes from the fact that, even if we don't know which, and exactly how many constraints, will hold in the final solution, we can still prune the variables of those constraints right from the beginning according to the minimum and maximum number of constraints that have to hold. We then show how to extend the previous sweep algorithm in order to handle preferences among constraints.

Finally, we specialize this technique to an extension of the non-overlapping rectangles constraint, where we permit controlling how many non-overlapping constraints should hold. This allows handling over-constrained placement problems and provides constraint propagation even if some non-overlapping constraints have to be relaxed.

1 Introduction

Since its introduction within constraint programming, the *cardinality* operator [9] has been recognized as a generic combinator [10], [11] which was integrated in several constraint systems. Its most general form is $\text{cardinality}(C, \{CTR_1(V_{11},..,V_{1n_1}),..,CTR_m(V_{m1},..,V_{mn_m})\})$ where C is a domain variable[1] and $\{CTR_1(V_{11},..,V_{1n_1}),..,CTR_m(V_{m1},..,V_{mn_m})\}$ is a set of constraints over domain variables. The *cardinality* operator holds iff $C = \sum_{i=1}^{m} \#CTR_i(V_{i1},..,V_{in_i})$, where $\#CTR_i(V_{i1},..,V_{in_i})$ is equal

[1] A *domain variable* is a variable that ranges over a finite set of integers; $\min(V)$ and $\max(V)$ respectively denote the minimum and maximum values of variable *V*, while $\text{dom}(V)$ designates the set of possible values of *V*.

to 1 if constraint $CTR_i(V_{i1},..,V_{in_i})$ holds and 0 otherwise[2]. Throughout this paper we consider a restricted case of the *cardinality* operator where all the constraints $CTR_1,..,CTR_m$ have at least two distinct variables X and Y in common. Let C denote the first argument of the *cardinality* operator throughout the rest of this paper.

From an operational point of view the *cardinality* operator used entailment [10] in order to implement constraint propagation. However a fundamental weakness is that it does not take advantage of the fact that some constraints may share some variables. The main contribution of this paper is to provide a stronger filtering algorithm for the case where all the constraints of the *cardinality* operator share at least two variables. This allows performing more constraint propagation even though some constraints may be relaxed.

The filtering algorithm is based on an idea which is widely used in computational geometry and which is called sweep [7, pp. 10-11]. Consider the illustrative example given in Fig. 1 where we have five constraints and their projections on two given variables X and Y ; assume that we want to find out the smallest value of X so that the conjunction of four or five of those constraints may hold for some Y . By trying out $X=0$, $X=1$ and $X=2$, we conclude that $X=2$ is the first value[3] that may be feasible. The new sweep algorithm performs this search efficiently; See Sect. 3.2 for details on this particular example.

In two dimensions, a plane sweep algorithm solves a problem by moving a vertical line from left to right. The algorithm uses the two following data structures:

– a data structure called the *sweep-line status*, which contains some information related to the current position Δ of the sweep-line,
– a data structure named the *event point series*, which holds the events to process, ordered in increasing order wrt. the abscissa.

The algorithm initializes the sweep-line status for the initial position of the sweep-line. Then the sweep-line jumps from event to event; each event is handled, updating the sweep-line status. In our context, the sweep-line scans the possible values of a domain variable X that we want to prune, and the sweep-line status contains for each value y of $\mathrm{dom}(Y)$ the minimum and maximum number of constraints that can be satisfied under the assumptions that $X=\Delta$ and $Y=y$. If, for some position Δ, all values of Y have an interval which does not intersect the possible number of constraints that should hold (i.e. C), then we will remove Δ from $\mathrm{dom}(X)$.

The sweep filtering algorithm will try to adjust the minimum[4] value of X wrt. the *cardinality* operator as well as the minimum and maximum value of C by moving a sweep-line from the minimum value of X to its maximum value. In our case, the events to process correspond to the starts and the ends of forbidden and safe

---

[2] As usual within constraint programming, this definition applies for the ground case when all the variables of the constraints $CTR_1,..,CTR_m$ are fixed.

[3] On this example, the propagation described for the classical *cardinality* operator [8] would not deduce anything, since none of the previous five constraints is neither always true nor always false.

[4] It can also be used in order to adjust the maximum value, or to prune completely the domain of a variable.

2-dimensional regions wrt. the constraints $CTR_1,..,CTR_m$ of the *cardinality* operator and variables X and Y.

Throughout this paper, we use the notation $\left(R_x^-..R_x^+, R_y^-..R_y^+, R_{type}\right)$ to denote for an ordered pair R of intervals, its lower and upper bounds and its type (i.e. forbidden or safe).

The next section presents the notion of forbidden and safe regions, which is a way to represent constraints that is suited for the sweep algorithm of this paper. Sect. 3 describes the sweep algorithm itself and analyzes its worst-case complexity, while Sect. 4 shows how to slightly modify the previous algorithm for handling the *weighted cardinality* operator. It is a more general case of the *cardinality* operator which allows specifying preferences, where we associate to each constraint CTR_i $(1 \le i \le m)$ a weight $W_i \in \mathbb{N}$; the *weighted cardinality* operator holds iff $C = \sum_{i=1}^{m}\left(W_i \cdot \# CTR_i\left(V_{i1},..,V_{in_i}\right)\right)$. Finally Sect. 5 presents its specialization to the relaxed non-overlapping rectangles constraint.

2 Forbidden and Safe Regions

We call R a *forbidden region of the constraint* CTR_i $(1 \le i \le m)$ *wrt. the variables* X *and* Y if: $\forall x \in R_x^-..R_x^+$, $\forall y \in R_y^-..R_y^+$: $CTR_i\left(V_{i1},..,V_{in_i}\right)$ with the assignment $X = x$ and $Y = y$ has no solution, no matter which values are taken by the other variables of constraint $CTR_i\left(V_{i1},..,V_{in_i}\right)$.

In a similar way, we name R a *safe region of the constraint* CTR_i $(1 \le i \le m)$ *wrt. the variables* X *and* Y if: $\forall x \in R_x^-..R_x^+$, $\forall y \in R_y^-..R_y^+$: $CTR_i\left(V_{i1},..,V_{in_i}\right)$ with the assignment $X = x$ and $Y = y$ always holds, no matter which values are taken by the other variables of constraint $CTR_i\left(V_{i1},..,V_{in_i}\right)$.

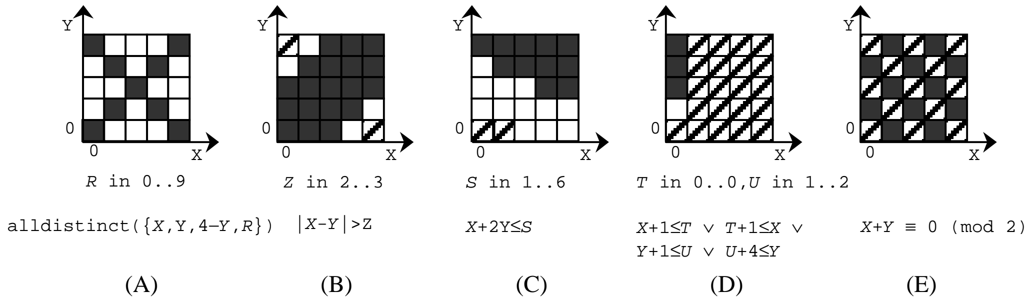

Fig. 1. Examples of forbidden ■ and safe ▨ regions. *X* in 0..4, *Y* in 0..4.

Fig. 1 shows 5 constraints and their respective forbidden (shaded) and safe (striped) regions wrt. two given variables X and Y and their domains. The statement *Var* in *min..max*, where *min* and *max* are two integers such that *min* is less than or equal to *max*, creates a domain variable *Var* for which the initial domain is made up from all

values between *min* and *max* inclusive. The first constraint requires X, Y, $4-Y$ and R be pairwise distinct, while the last four constraints correspond to arithmetic and disjunctive constraints.

The sweep algorithm computes the forbidden and safe regions on request, in a lazy evaluation fashion. The forbidden and safe regions of each constraint CTR_i $(1 \le i \le m)$ are gradually generated as a set of rectangles $R_{i1},\ldots,R_{ik}$ such that:

- $R_{i1} \cup \cdots \cup R_{ik}$ represents all forbidden and safe regions of constraint CTR_i wrt. variables X and Y,
- the rectangles $R_{i1},...,R_{ik}$ do not pairwise intersect,
- $R_{i1},...,R_{ik}$ are sorted by ascending start position on the X axis.

In practice, we use the following functions[5] for gradually getting the forbidden and safe regions for each triple (X, Y, CTR_i) $(1 \le i \le m)$ that we want to be used by the sweep algorithm:

- GETFIRSTREGIONS(X, Y, CTR_i): generates all the forbidden and safe regions R_{CTR_i} of constraint CTR_i such that:

$$\begin{cases} R_{CTR_i}{}_x^- \le first_{CTR_i} \le R_{CTR_i}{}_x^+ \\ R_{CTR_i}{}_y^+ \ge \min(Y) \wedge R_{CTR_i}{}_y^- \le \max(Y) \end{cases},$$

  where $first_{CTR_i}$ is the smallest value in $\min(X)..\max(X)$ such that there exists such a forbidden or safe region R_{CTR_i} of CTR_i.
- GETNEXTREGIONS$(X, Y, CTR_i, previous_i)$: generates all the forbidden and safe regions R_{CTR_i} of constraint CTR_i such that:

$$\begin{cases} R_{CTR_i}{}_x^- = next_{CTR_i} \\ R_{CTR_i}{}_y^+ \ge \min(Y) \wedge R_{CTR_i}{}_y^- \le \max(Y) \end{cases},$$

  where $previous_i$ is the position of the previous start event of constraint CTR_i and $next_{CTR_i}$ is the smallest value greater than $previous_i$ such that there exists such a forbidden or safe region R_{CTR_i} of CTR_i.

If we consider constraint (C) of Fig. 1 (i.e. $X + 2 \cdot Y \le S$), and we assume that $X \in 0..3$, $Y \in 0..3$ and $S \in 1..6$, then a complete scan of X would produce the following sequences of calls:

- GETFIRSTREGIONS$(X, Y, X + 2 \cdot Y \le S)$ returns region $(0..1, 0..0, \mathit{safe})$,
- GETNEXTREGIONS$(X, Y, X + 2 \cdot Y \le S, 0)$ returns region $(1..3, 3..3, \mathit{forbidden})$,
- GETNEXTREGIONS$(X, Y, X + 2 \cdot Y \le S, 1)$ returns region $(3..3, 2..2, \mathit{forbidden})$.

The complexity results of this paper assume that all the previous functions used for getting the forbidden and safe regions are performed in $O(nr)$, where nr is the number of regions returned by the function.

---

[5] Two analogous functions GETLASTREGIONS and GETPREVREGIONS are also provided for the case where the sweep-line moves from the maximum value of X to its minimum value.

3 A Sweep Algorithm for the *Cardinality* Operator

The purpose of this section is to describe the new sweep algorithm which can cope with the fact that we don't know exactly[6] how many constraints of the *cardinality* operator will hold. We first describe the data structures used by the algorithm and illustrate its main ideas on a concrete example. Finally we give the algorithm and analyze its worst-case complexity.

3.1 Data Structures

As is the case for most sweep algorithms, the new sweep algorithm uses one data structure for recording the sweep-line status and another data structure for storing the event points. For the current position Δ of the sweep-line, the *sweep-line status* contains for each possible value y of Y the following information:

- the number $nsafe[y]$ of safe regions that currently intersect the sweep-line at the point of coordinates Δ, y; the quantity $nsafe[y]$ gives, under the assumptions that both $X = \Delta$ and $Y = y$, a lower bound of the total number of constraints $CTR_1,..,CTR_m$ that hold,
- the number $nforbid[y]$ of forbidden regions that currently intersect the sweep-line at the point of coordinates Δ, y; the quantity $m - nforbid[y]$ gives, under the assumptions that both $X = \Delta$ and $Y = y$, an upper bound of the total number of constraints $CTR_1,..,CTR_m$ that hold,
- $nsafe_C[y]$ is the smallest value greater than or equal to $nsafe[y]$ such that both $nsafe_C[y] \in \mathrm{dom}(C)$ and $nsafe_C[y] \leq m - nforbid[y]$; it is equal to $m+1$ is no such value exists,
- $nforbid_C[y]$ is the smallest value greater than or equal to $nforbid[y]$ such that both $m - nforbid_C[y] \in \mathrm{dom}(C)$ and $m - nforbid_C[y] \geq nsafe[y]$; it is equal to $m+1$ is no such value exists.

When $nsafe_C[y]$ is equal to $m+1$ [7], it means that the interval $nsafe[y]..m - nforbid[y]$ has an empty intersection with the set of possible values of C.

Each array $nsafe[y]$, $nforbid[y]$, $nsafe_C[y]$ and $nforbid_C[y]$ is implemented with an (*a*,*b*)-tree [5] which stores for the values of Y the corresponding quantity (i.e. the endpoints of the intervals of consecutive values of Y for which the array contains the same value). Let k denote the number of changes of an array (i.e. the number of times the value stored at an entry i is different from the value kept at entry $i+1$). Incrementing a set of consecutives entries by a given constant, getting the entry with minimal value, and setting a set of consecutives entries, which are currently set to the same

---

[6] We only know that the number of constraints, that should hold, is one of the values of $\mathrm{dom}(C)$.

[7] $nsafe_C[y] = m+1 \Leftrightarrow nforbid_C[y] = m+1$.

value, to a given constant are all $O(\log k)$ operations. A whole iteration through all the intervals (i.e. consecutives entries with the same value) of the array takes $O(k \log k)$.

The *event point series*, denoted Q_{event}, contains the start and end+1 on the X axis, of those safe and forbidden regions of the constraints $CTR_1,..,CTR_m$ wrt. variables X and Y that intersect the sweep-line. These start and end events are sorted in increasing order and recorded in a heap. In addition an array $count_regions[1..m]$ records, for each constraint $CTR_1,..,CTR_m$, how many starts of safe or forbidden regions are recorded within Q_{event}. This allows to check if Q_{event} does not contain any start event associated to a given constraint in $O(1)$ (see line 6 of Algorithm 2).

3.2 Principle of the Algorithm

In order to check if $X = \Delta$ may be feasible wrt. the *cardinality* operator, the sweep-line status records the number of safe regions as well as the number of forbidden regions that intersect the current position of the sweep-line. If, for $X = \Delta$, $\forall y \in \mathrm{dom}(Y)$: $nsafe_C[y] = m+1$ (i.e. $nsafe[y].m - nforbid[y] \cap \mathrm{dom}(C) = \varnothing$), the sweep-line will move to the right to the next event to handle.

Before going more into the detail of the algorithm, let us first illustrate how it works on a concrete example. Assume that we want to find out the minimum value of variable X such that the conjunction of four or five of those constraints that were given in Fig. 1 hold. In addition we want to update the minimum and maximum value of the number C of constraints that hold. Table 1 shows the content of the sweep-line status for all positions Δ of the sweep-line. The smallest value of X which may be feasible is 2, since this is the first position where there exists a value $y = 0$ of Y such that $nsafe_C[0] = 4 \neq m+1 = 5+1$. Since for each position of the sweep-line at least one constraint does not hold we also update the maximum value of C to value 4.

Table 1. Status of the sweep-line at each stage of the algorithm. s, f, s_C, f_C respectively denote $nsafe[\ ]$, $nforbid[\ ]$, $nsafe_C[\ ]$, $nforbid_C[\ ]$ per Y position.

Y	Δ=0	Δ=1	Δ=2	Δ=3	Δ=4
4	2,3,6,6	1,2,6,6	2,2,6,6	1,3,6,6	2,3,6,6
3	0,2,6,6	2,3,6,6	1,3,6,6	2,3,6,6	1,3,6,6
2	1,2,6,6	1,2,6,6	2,2,6,6	1,3,6,6	2,2,6,6
1	0,2,6,6	2,2,6,6	1,2,6,6	2,2,6,6	1,1,4,1
0	3,2,6,6	2,2,6,6	2,1,4,1	1,1,4,1	3,1,4,1

3.3 The Main Procedure

The procedure FINDMINIMUM (see Algorithm 1) implements the sweep algorithm for adjusting the minimum value of a variable X wrt. a given *cardinality* operator as well as for adjusting the minimum and maximum number of constraints that hold. It can be easily adapted to a procedure that adjusts the maximum value of a variable. The main parts of FINDMINIMUM are:

- Lines 1-7 initialize the event queue to the start and end events associated to the left-most safe and forbidden regions of each constraint. Note that we only insert events that are effectively within $\min(X)..\max(X)$ and $\min(Y)..\max(Y)$. If no such events are found or if no safe or forbidden region intersects $\min(X)$, we exit the procedure.
- Line 8 initializes to 0 all the four arrays of the sweep-line status, while line 9 sets $nsafe[v]$, $nforbid[v]$, $nsafe_C[v]$ and $nforbid_C[v]$ to $m+1$, for those values v that do not belong[8] to $\mathrm{dom}(Y)$. These values will not be considered any more, since no safe or forbidden region which contains these values will be added.
- Lines 11-19 extract from the event queue all events associated to the current position Δ of the sweep-line and update the sweep-line status. Afterwards, check whether there may exist some feasible solution for $X=\Delta$ and, if so, record it and eventually update the minimum and maximum number of constraints which hold.
- Line 20 reports a failure since a complete sweep over the full domain of variable X was done without finding any solution.
- Lines 21-23 adjust the minimum and maximum of variable C and return a possibly feasible solution for X and Y.

Input: A *cardinality* operator $\mathrm{cardinality}(C, \{CTR_1(V_{11},..,V_{1n_1}),..,CTR_m(V_{m1},..,V_{mn_m})\})$ and two domain variables X and Y present in each constraint $CTR_1,..,CTR_m$.

Output: An indication that no solution exists or an indication that a solution may exist and values $\hat{x}$, $\hat{y}$.

Ensure: Either $\hat{x}$ is the smallest value of X such that $\hat{y} \in \mathrm{dom}(Y)$ and $(\hat{x},\hat{y})$ belongs to exactly s safe regions and to precisely f forbidden regions of $CTR_1,..,CTR_m$ wrt. variables X and Y, such that interval $s..m-f$ has a non-empty intersection with the domain variable C, or no solution exists. Also adjust the minimum and maximum values of C.

1: Q_{event} ←an empty event queue, $feasible$ ←0, $Cmin \leftarrow m+1$, $Cmax \leftarrow -1$.
2: **for all** constraint CTR_i $(1 \le i \le m)$ **do**
3:     **for all** region $R_{CTR_i} \in$ GETFIRSTREGIONS(X, Y, CTR_i) **do**
4:         Insert $\max(R_{CTR_i}{}^{-}_{x}, \min(X))$ into Q_{event} as a start event.
5:         **if** $R_{CTR_i}{}^{+}_{x} + 1 \le \max(X)$ **then** Insert $R_{CTR_i}{}^{+}_{x} + 1$ into Q_{event} as an end event.
6: **if** Q_{event} is empty **or** the leftmost position of any event of Q_{event} is greater than $\min(X)$ **then**
7:     $\hat{x} \leftarrow \min(X)$, $\hat{y} \leftarrow \min(Y)$, **return** (**true**, $\hat{x}$, $\hat{y}$).
8: $nsafe$, $nforbid$, $nsafe_C$, $nforbid_C$ ←arrays ranging over $\min(Y)..\max(Y)$ initialized to 0.
9: $nsafe[i]$, $nforbid[i]$, $nsafe_C[i]$, $nforbid_C[i] \leftarrow m+1$, **for** $i \in \min(Y)..\max(Y) \setminus \mathrm{dom}(Y)$.
10: **while** Q_{event} is not empty **and** ($feasible = 0$ **or** $Cmin > \min(C)$ **or** $Cmax < \max(C)$) **do**
11:     $\Delta \leftarrow$ the leftmost position of any event of Q_{event}.
12:     **for all** event E at position Δ of Q_{event} **do** HANDLEEVENT(E).
13:     $imin \leftarrow$ index such that $nsafe_C[imin]$ is minimal.
14:     **if** $nsafe_C[imin] \ne m+1$ **then**
15:         **if** $feasible = 0$ **then** $\hat{x} \leftarrow \Delta$, $\hat{y} \leftarrow imin$, $feasible \leftarrow 1$.

[8] $A \setminus B$ denotes the set difference between A and B.

16: $smin \leftarrow$ smallest value of $nsafe_C[\,]$.
17: **if** $smin < Cmin$ **then** $Cmin \leftarrow smin$.
18: $fmin \leftarrow$ smallest value of $nforbid_C[\,]$.
19: **if** $m - fmin > Cmax$ **then** $Cmax \leftarrow m - fmin$.
20: **if** $feasible = 0$ **then** **return** (**false**, 0, 0).
21: **if** $Cmin > \min(C)$ **then** adjust the minimum of C to $Cmin$.
22: **if** $Cmax < \max(C)$ **then** adjust the maximum of C to $Cmax$.
23: **return** (**true**, $\hat{x}$, $\hat{y}$).

Algorithm 1: FINDMINIMUM$(CTR_1,..,CTR_m, X, Y, C)$

Holes in the domain of variable X are handled by generating so called "contradiction" regions, which add $m+1$ to $nsafe[v]$ for all values $v \in \min(Y)..\max(Y)$ when we enter such regions. The next section describes the procedure HANDLEEVENT , which specifies how to modify the sweep-line status according to a given start or end event.

3.4 Handling Start and End Events

1: Extract E from Q_{event} and get the corresponding region R_E and constraint CTR_E .
2: $l \leftarrow \max\left(R_{E}{}_{y}^{-}, \min(Y)\right)$, $u \leftarrow \min\left(R_{E}{}_{y}^{+}, \max(Y)\right)$, $t \leftarrow R_{E_{type}}$.
3: **if** E is an end event **then** $inc \leftarrow -1$.
4: **else**
5: $inc \leftarrow 1$.
6: **if** Q_{event} does not contain any start event associated to constraint CTR_E **then**
7: $previous_x_E \leftarrow R_{E}{}_{x}^{-}$.
8: **for all** region $R_{CTR_i} \in$ GETNEXTREGIONS$(X, Y, CTR_E, previous_x_E)$ **do**
9: Insert $R_{CTR_E}{}_{x}^{-}$ into Q_{event} as a start event.
10: **if** $R_{CTR_E}{}_{x}^{+} + 1 \le \max(X)$ **then** Insert $R_{CTR_E}{}_{x}^{+} + 1$ into Q_{event} as an end event.
11: **if** $t = \text{safe}$ **then** Add inc to $nsafe[i]$ for all $i \in l..u$ **else** Add inc to $nforbid[i]$ for all $i \in l..u$.
12: **for** all intervals $a..b$ such that $\begin{cases} l \le a \ \wedge \ b \le u \\ nsafe[a] = nsafe[a+1] = \ldots = nsafe[b] \\ nforbid[a] = nforbid[a+1] = \ldots = nforbid[b] \\ a = l \vee nsafe[a-1] \ne nsafe[a] \vee nforbid[a-1] \ne nforbid[a] \\ b = u \vee nsafe[b+1] \ne nsafe[b] \vee nforbid[b+1] \ne nforbid[b] \end{cases}$ **do**
13: Set $nsafe_C[i](a \le i \le b)$ to the smallest value v such that:
14: $v \ge nsafe[a] \wedge v \in \text{dom}(C) \wedge v \le m - nforbid[a]$ [9].
15: Set $nforbid_C[i](a \le i \le b)$ to the smallest value v such that:
16: $v \ge nforbid[a] \wedge m - v \in \text{dom}(C) \wedge m - v \ge nsafe[a]$ [10].

Algorithm 2: HANDLEEVENT(E)

---

[9] $nsafe_C[i](a \le i \le b)$ is initialized to m+1 if no such value v exists.

[10] $nforbid_C[i](a \le i \le b)$ is initialized to m+1 if no such value v exists.

When E is the last start event of a given constraint CTR_E and since not all events were initially inserted in Q_{event}, we search for the next events of CTR_E and insert them in the event queue Q_{event} (lines 6-10). Depending on whether we have a start or an end event E that comes from a safe or a forbidden region we add 1 or -1 to $nsafe[i]$ or to $nforbid[i]$ ($l \le i \le u$), where l and u are respectively the start and the end on the Y axis of the region that is associated to the event E (lines 1-5,11). Finally, for each maximum interval $a..b$ such that $l \le a \le b \le u$ and such that the pair of values $(nsafe[i], nforbid[i])$ is constant for all $i \in a..b$ (line 12), we update $nsafe_C$ (line 13) as well as $nforbid_C$ (line 15).

3.5 Worst-Case Analysis

This section analyses the worst-case complexity of a complete sweep over the domain of X. Let r denote the total number of forbidden and safe regions intersecting the domain of the variables X, Y under consideration, and m the number of constraints. Furthermore assume the domain of C to be represented as two tables $low[1..d]$ and $up[1..d]$ such that the domain of C consists only of those values belonging to $low[j]..up[j]$ ($1 \le j \le d$). For a complete sweep, Table 2 indicates the number of times each operation is performed, and its total worst-case complexity. Hence, the overall worst-case complexity of a complete sweep is $O(m + r^2 \log r + r^2 \log d)$.

Table 2. Maximum number of calls and worst-case complexity per basic operation in a sweep

Operation	**Max. times**	**Total**
Initialize to empty the queue Q_{event}	$O(1)$	$O(1)$
Compute the first forbidden and safe regions of all constraints[11]	$O(1)$	$O(m + r)$
Add an event to the queue Q_{event}	$O(r)$	$O(r \log r)$
Extract the next event from the queue Q_{event}	$O(r)$	$O(r)$
Check if there exists a start event associated to a constraint	$O(r)$	$O(r)$
Initialize to a value nsafe[], nforbid[], nsafe_C[], nforbid_C[]	$O(1)$	$O(1)$
Update a range of nsafe[], nforbid[]	$O(r)$	$O(r \log r)$
Update a range of nsafe_C[], nforbid_C[]	$O(r^2)$	$O(r^2 \log d + r^2 \log r)$
Check if there exists an element of nsafe_C with a value $\neq m+1$	$O(r)$	$O(r \log r)$

Since the main difficulty is the update of $nsafe_C / nforbid_C$ we give the detail of this part (line 8 of Table 2). First note that finding the smallest value of $\mathrm{dom}(C)$ greater than or equal to $nsafe[i]$ (respectively $m - nforbid[i]$) can be done in $\log d$. In addition since there cannot be more than $O(r)$ changes in $nsafe_C / nforbid_C$, and since to

[11] This corresponds to lines 2-3 of Algorithm 1.

each change in $nsafe_C / nforbid_C$ corresponds a set of consecutives entries which are currently set to the same value, the worst-case complexity per change of $nsafe_C / nforbid_C$ is $O(\log d + \log r)$. Finally as we have at most $2r$ calls to HANDLEEVENT and because for each call there cannot be more than $O(r)$ intervals $a..b$ (line 12 of Algorithm 2) such that both $nsafe[a] = nsafe[a+1] = \ldots = nsafe[b-1] = nsafe[b]$, $a = l \vee nsafe[a-1] \neq nsafe[a] \vee nforbid[a-1] \neq nforbid[a]$, $nforbid[a] = nforbid[a+1] = \ldots = nforbid[b-1] = nforbid[b]$, $b = u \vee nsafe[b+1] \neq nsafe[b] \vee nforbid[b+1] \neq nforbid[b]$ the total complexity for updating $nsafe_C / nforbid_C$ is $O(r^2 \log d + r^2 \log r)$.

4 A Sweep Algorithm for the *Weighted Cardinality* Operator

This section explains how to slightly modify the sweep-line status of the algorithm presented in Sect. 3 in order to handle the *weighted cardinality* operator. We now record in $nsafe[y]$ (respectively $nforbid[y]$) the sum of the weights of the safe (respectively forbidden) regions which contain the point of coordinates Δ, j. Finally, $nsafe_C[y]$ (respectively $nforbid_C[y]$) is set to the smallest value greater than or equal to $nsafe[y]$ (respectively $nforbid_C[y]$) such that $nsafe_C[y]$ (respectively $m - nforbid_C[y]$) belongs to the domain of C. The domain variable C is the sum of the weights of the constraints which hold in the constraints $CTR_1,..,CTR_m$ of the *weighted cardinality* operator. In lines 1,10,15,20 of Algorithm 1 the quantity m is replaced by $\sum_{i=1}^{m} W_i$, while lines 11,13-16 of Algorithm 2 are modified as indicated below.

```
11: if t = safe then Add inc·W_i to nsafe[i] for all i ∈ l..u else Add inc·W_i to nforbid[i] for all i ∈ l..u .
13:     Set nsafe_C[i](a ≤ i ≤ b) to the smallest value v such that:
14:         v ≥ nsafe[a] ∧ v ∈ dom(C) ∧ v ≤ Σ_{j=1}^{m} W_j − nforbid[a] [12].
15:     Set nforbid_C[i](a ≤ i ≤ b) to the smallest value v such that:
16:         v ≥ nforbid[a] ∧ Σ_{j=1}^{m} W_j − v ∈ dom(C) ∧ Σ_{j=1}^{m} W_j − v ≥ nsafe[a] [13].
```

Algorithm 3: modifications of procedure HANDLEEVENT(E)

---

[12] $nsafe_C[i](a \le i \le b)$ is initialized to $\sum_{j=1}^{m} W_j + 1$ if no such value v exists.

[13] $nforbid_C[i](a \le i \le b)$ is initialized to $\sum_{j=1}^{m} W_j + 1$ if no such value v exists.

5 A Relaxation of the Non-Overlapping Rectangles Constraint

Assume that we want to implement a constraint $\text{RELAXEDNONOVERLAPPING}(C, P_1, \ldots, P_m)$ over a set of rectangles, which should hold if exactly C [14] pairs of rectangles P_i, P_j, $i < j$ do not overlap. A rectangle P_i with origin coordinates (X_i, Y_i), width w_i and height h_i is given as $\langle X_i, w_i, Y_i, h_i \rangle$, where X_i and Y_i are domain variables and w_i, h_i are non-negative integers. We have a total number of $(m^2 - m)/2$ non-overlapping constraints of the form:

$$non_overlap_{ij}(\langle X_i, w_i, Y_i, h_i \rangle, \langle X_j, w_j, Y_j, h_j \rangle) \Leftrightarrow$$
$$X_i + w_i \le X_j \vee X_j + w_j \le X_i \vee Y_i + h_i \le Y_j \vee Y_j + h_j \le Y_i$$

As it was mentioned in [2], there can be a most one non-empty forbidden region $R_{ij} = (r_x^- .. r_x^+, r_y^- .. r_y^+, \text{forbidden})$ of $non_overlap_{ij}$ wrt. (X_i, Y_i), where:

$r_x^- = \max(X_j) - w_i + 1$, $r_x^+ = \min(X_j) + w_j - 1$, $r_y^- = \max(Y_j) - h_i + 1$, $r_y^+ = \min(Y_j) + h_j - 1$.

On the other hand, there can be at most 4 non-empty safe regions $R_{ij} = (r_{k,x}^- .. r_{k,x}^+, r_{k,y}^- .. r_{k,y}^+, \text{safe})$ $(1 \le k \le 4)$ of $non_overlap_{ij}$ wrt. (X_i, Y_i), where:

$r_{1,x}^- = \min(X_i)$, $r_{1,x}^+ = \max(X_i)$, $r_{1,y}^- = \min(Y_i)$, $r_{1,y}^+ = \min(Y_j) - h_i$,
$r_{2,x}^- = \min(X_i)$, $r_{2,x}^+ = \max(X_i)$, $r_{2,y}^- = \max(Y_j) + h_j$, $r_{2,y}^+ = \max(Y_i)$,
$r_{3,x}^- = \min(X_i)$, $r_{3,x}^+ = \min(X_j) - w_i$, $r_{3,y}^- = \min(Y_j) - h_i + 1$, $r_{3,y}^+ = \max(Y_j) + h_j - 1$,
$r_{4,x}^- = \max(X_j) + w_j$, $r_{4,x}^+ = \max(X_i)$, $r_{4,y}^- = \min(Y_j) - h_i + 1$, $r_{4,y}^+ = \max(Y_j) + h_j - 1$.

To each rectangle P_i $(1 \le i \le m)$, we associate a variable C_i which gives the minimum and maximum number of non-overlapping constraints which hold and we link all these variables by the constraint $C = \sum_{i=1}^{m} C_i$. In order to adjust the minimum of variable X_i and to update the minimum and maximum value of variable C_i, we use Algorithm 1.

6 Discussion and Conclusion

The relevance of our approach compared to what is currently done is as follows. Constraint network based frameworks used to model constraint relaxation [3] require constraints to be defined as a set of tuples. However, in practice, a lot of constraints can't be defined extensionally since a huge number of allowed tuples is needed to model a given constraint. On the other hand, even if the constraints are defined intentionally, the

[14] C is a domain variable.

worst-case complexity of these consistency algorithms generally depends on the number of values present in the domains of the variables.

More recently, in order to take advantage of the structure of some specific constraint, Petit et al. [6] have proposed two filtering algorithms based on flow for two relaxed version of the *alldifferent*[15] constraint. Our approach can be situated between these two extremes: on one side we define constraints in a compact way by providing functions which return forbidden and safe regions. On the other side, the only point about the structure of the constraints that we exploit is the fact that two given variables occur in different constraints. It should be noted that using multi-dimensional data structures should allow to take advantage, without changing the filtering algorithm, of the fact that some constraints share more than two variables.

In the past within practical constraints systems, conventional wisdom had it that, in order to get the full benefit from the power of constraint propagation, one should enforce all constraints. As a corollary it has been generally assumed that constraint relaxation some how "kills" constraint propagation. In this paper we have shown that, for a specific type of constraint relaxation, this is not true. We have come up with a generic filtering algorithm which can cope with the fact that we just know the minimum and maximum number of constraints that have to hold. This algorithm was derived from our value sweep algorithm [1], where in addition to the concept of forbidden region we came up with the notion of safe region. Finally we introduced a small modification of the sweep-line status in order to handle relaxation. This is yet another useful application of the concept of sweep to constraint propagation. Implementing the algorithm of this paper would be needed in order to compare it with existing techniques [3], [4], [8] and further assess its practical value.

Acknowledgements

Thanks to Per Mildner as well as to anonymous referees for useful comments on an early version of this paper.

References

1. Beldiceanu, N.: Sweep as a generic pruning technique. In *TRICS: Techniques foR Implementing Constraint programming*, CP2000, Singapore (2000).
2. Beldiceanu, N., Carlsson, M.: Sweep as a Generic Pruning Technique Applied to the Non-Overlapping Rectangles Constraint.. In *Principles and Practice of Constraint Programming – CP'2001, 7th International Conference*, Paphos, Cyprus, (2001).
3. Bistarelli, S., Fargier, H., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G.: Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. *Constraints*, Vol.4, N.3, 199-240, Kluwer, (1999).
4. Freuder, E., Wallace, R.: Partial Constraint Satisfaction. *Artificial Intelligence*, Vol.58, 21-70, (1992).

[15] The constraint $\text{alldifferent}([X_1,\ldots,X_n])$ holds if all variables $X_1,\ldots,X_n$ are pairwise different.

5. Mehlhorn, K.: *Data Structures and Algorithms 1: Sorting and Searching*. EATCS Monographs. Springer Verlag, Berlin (1984).
6. Petit, T., Régin, J.C, Bessière, C.: Algorithmes de Filtrage Spécifiques pour les Problèmes Sur-Contraints. In *JNPC'2001*, 953-966, Toulouse, France (June 2001). in French.
7. Preparata F.P., Shamos M.I.: *Computational Geometry. An Introduction.* Springer-Verlag, 1985.
8. Schiex, T.: Arc Consistency for Soft Constraints. In *Principles and Practice of Constraint Programming – CP'2000, 6th International Conference*, Singapore. Lecture Notes in Computer Science, Vol. 1894, Springer, 411-424, (2000).
9. Van Hentenryck, P., Deville, Y.: The Cardinality Operator: A New Logical Connective for Constraint Logic Programming. In *International Conference on Logic Programming*. The MIT Press, 745-759, (1991).
10. Van Hentenryck, P., Saraswat, V., Deville, Y.: Design, Implementation and Evaluation of the Constraint Language cc(*FD*). In *A. Podelski, ed., Constraints: Basics and Trends*, vol. 910 of Lecture Notes in Computer Science, Springer-Verlag, (1995).
11. Würtz, J., Müller, T.: Constructive Disjunction Revisited. In *20th German Annual Conference on Artificial Intelligence*. LNAI vol. 1137, 377-386, Springer-Verlag, (1996).