Constraint Satisfaction over Connected Row Convex Constraints

Yves Deville **Olivier Barette**
Université catholique de Louvain,
Pl. Ste Barbe 2,
B-1348 Louvain-la-Neuve, Belgium
{yde,barette}@info.ucl.ac.be

Pascal Van Hentenryck
Brown University
Box 1910
Providence, RI 02912, USA
pvh@cs.brown.edu

Abstract

In this paper, we study constraint satisfaction over connected row convex (CRC) constraints, a large class of constraints subsuming, in particular, monotone constraints. We first show that CRC constraints are closed under composition, intersection, and transposition, the basic operations of path-consistency algorithms. This establishes that path consistency over CRC constraints produces a minimal and decomposable network, strenghtening the results of van Beek and Dechter [1995]. We then present a path-consistency algorithm for CRC constraints running in time $O(n^3d^2)$ and space $O(n^2d)$, where n is the number of variables and d is the size of the largest domain. This improves the traditional time complexity $O(n^3d^3)$ and space complexity $O(n^3d^2)$. Finally, we show that a solution can be found in time $O(n^2)$, once the graph is path-consistent.

1 Introduction

Constraint satisfaction techniques have been found useful in many areas such as Operations Research, hardware design, robotics, knowledge bases, and temporal resaoning to name a few. Some applications require to find one or all solutions, in which case consistency techniques such arc and path consistency are instrumental in reducing the size of the search space. Other applications require to put the graph of constraints in minimal form, e.g., to remove redundant information.

Increasing attention has been devoted recently to the study of special classes of constraints or contraint graphs. These studies are motivated both by practical considerations (e.g., constraint languages are based on a set of primitive constraints) and by theoretical considerations, since stronger results and more efficient algorithms can be obtained by exploiting special properties.

This paper considers the class of connected row convex (CRC) constraints. CRC constraints were motivated by van Beek's row convex constraints, their properties, and their applications to various tasks in artificial intelligence [van Beek and Dechter, 1995]. The class of CRC constraints include many constraints such as $ax + by + c \leq 0$, $ax + by + c \geq 0$, $axy + b \leq 0$, $axy + b \geq 0$, $af(x) + by + c \leq 0$, and $af(x) + by + c \geq 0$, where a, b, c are rationals and $f(x)$ is a function whose derivative does not change sign in the considered domain as well as conjunctions of these constraints, some of which being non-monotone. We show that, contrary to row convex constraints, CRC constraints are closed under intersection, composition, and transposition, the basic operations of path-consistency algorithms. We then propose a generic path-consistency algorithm PC-GEN which mimics the generic arc-consistency algorithm AC-5 and we show that it can be instantiated to produce a path-consistency algorithm running in time $O(n^3d^2)$ and in space $O(n^2d)$ for CRC constraints. Finally, we show that finding a solution to a path-consistent graph of CRC constraints can be done in $O(n^2)$ time.

The rest of the paper is organized as follows. Section 2 introduces the necessary background on Constraint Satisfaction Problems (CSPs) and Section 3 discusses related work. Section 4 introduces CRC constraints and studies their properties Section 5 presents PC-GEN and Section 6 instantiates it for CRC constraints. Section 7 concludes the paper.

2 Background

In this paper, variables are represented by the natural numbers $1, \ldots, n$. Each variable i has an associated finite domain D_i. All constraints are binary and relate two distinct variables. If i and j are variables ($i < j$), we assume, for simplicity, that there is at most one constraint relating them, denoted by C_{ij}. A constraint C_{ij} denotes a set of couples ($C_{ij} \subseteq D_i \times D_j$). The fact that $(v, w) \in C_{ij}$ is also be denoted by $C_{ij}(v, w)$. We denote by D the union of all domains and by d the size of the largest domain. We assume the existence of a total ordering over D. Following Montanari [1974], a constraint C_{ij} will also be seen as a (0,1)-matrix with $\mid D_i \mid$ rows and $\mid D_j \mid$ columns. Rows and columns are ordered according to the underlying order over D. A 1 (resp. 0) at position (v, w) in the matrix means $(v, w) \in C_{ij}$ (resp. $(v, w) \notin C_{ij}$). To simplify the presentation, each domain D_i is also represented by a (pseudo-binary) constraint C_{ii} such that $C_{ii}(v, v)$ holds iff $v \in D_i$. Domain D_i and constraint C_{ii} can be used in an interchangeable way.

Consistency algorithms generally work on the graph representation of the CSP. We associate a graph G to a CSP in the following way. G has a node i and an arc (i,i) for each variable i. The constraint associated to arc (i,i) is C_{ii}. For each constraint C_{ij} relating variables i and j $(i < j)$, G has two directed arcs, (i,j) and (j,i). The constraint associated to arc (i,j) is C_{ij} and the constraint associated to (j,i) is C_{ji}, which is the transposition of C_{ij}. The graph is thus defined by the nodes, the arcs, and their associated constraints. We use $arc(G)$ and $node(G)$ to denote the set of arcs and the set of nodes of graph G. A CSP uniquely defines a graph and vice versa. The tuple $(v_1, \ldots, v_n) \in D^n$ is a solution of G if $C_{ij}(v_i, v_j)$ holds for all $(i,j) \in arc(G)$. Two graphs G and G' are *equivalent* if G and G' have the same solutions.

Let us review some of the basics in the area of path consistency. A tuple $\langle v_{i_0}, v_{i_m}\rangle$ is *path-consistent* for path $(i_0, \ldots, i_m)$ wrt G if $\exists v_{i_1}, \ldots, v_{i_{m-1}}$ such that $v_{i_k} \in D_{i_k}$ $(1 \leq k < m)$ and $C_{i_k i_{k+1}}(v_{i_k}, v_{i_{k+1}})$ $(0 \leq k < m)$. The path $p = (i_0, \ldots, i_m)$ in G is *path-consistent wrt* G if for all $v_{i_0} \in D_{i_0}$ and $v_{i_m} \in D_{i_m}$ with $C_{i_0 i_m}(v_{i_0}, v_{i_m})$, $\langle v_{i_0}, v_{i_m}\rangle$ is path-consistent for p wrt G. A graph G is *path consistent* if for all paths p in G, p is path consistent wrt G. Montanari [1974] showed that a complete graph is path consistent iff all paths of length two are path-consistent. Typically, path-consistency algorithms work on complete graphs, and an incomplete graph can be easily transformed into a complete graph by adding *TRUE* constraints. The objective of a path-consistency algorithm is thus, given a complete graph G, to compute new constraints which are path consistent and have the same solutions as G. Path-consistency algorithms are generally defined in terms of interesection and composition of the matrix representation defined as

$$(C_{ik} \,.\, C_{kj})(v,w) = \vee_{u \in D} C_{ik}(v,u) \wedge C_{kj}(u,w)$$
$$(C_1 \cap C_2)(v,w) = C_1(v,w) \wedge C_2(v,w)$$

A graph G is *minimal* if, for all $i, j \in node(G)$ and for all $v, w \in D$, $C_{ij}(v,w)$ implies that (v,w) is part of some solution of G. A graph G is *decomposable* if, for all $v_{i_1} \ldots v_{i_k}$ satisfying all the constraints relating nodes $i_1 \ldots i_k (1 \leq k < n)$ and for any new node i_{k+1}, there exists $v_{i_{k+1}}$ such that $v_{i_1} \ldots v_{i_k}, v_{i_{k+1}}$ satisfy all the constraints relating nodes $i_1 \ldots i_k, i_{k+1}$. A decomposable graph is also called strongly n-consistent [Freuder, 1982]. Decomposable graphs have thus the property that any consistent instantiation of some variables can be extended to a solution without backtracking. A decomposable graph is of course minimal. In a minimal graph, it is not possible to prune further the constraints without removing solutions.

3 Related Work

This research was motivated by van Beek's result on row convex constraint. A constraint C_{ij} is *row convex* if, in each row of its matrix representation, all the ones are consecutive. Van Beek and Dechter [1995] show that, when the constraints of a path-consistent graph are row convex (or can be made row convex by permutation of values in the domain), then the graph is minimal and decomposable. One can thus compute a solution without backtracking in $O(n^2 d)$. Solving the CSP can then be done in $O(n^3 d^3)$, the time complexity of the PC algorithm. Unfortunately, row convex constraints are not closed under composition and intersection. As a consequence, no conclusion can be drawn a priori for a graph of row convex constraints, since its path consistent subgraph may or may not be row convex. CRC constraints remove this problem, since a graph of CRC constraints, after application of a path consistency algorithm, is still CRC and is thus minimal and decomposable.

In [1995], Jeavons and Cooper identify the class of max-closed constraints that can be solved in polynomial time ($O(n^4 d^4)$ for binary constraints). Our class of CRC constraints, which can be solved in $O(n^3 d^2)$, intersects with max-closed constraints, but is not a subset.

The idea of row convexity has also been exploited in the context of continuous constraints [Haroud and Faltings, 1996]. They start from the result that, when constraints are convex and binary, path-consistency is sufficient to ensure decomposability. They show that for continous domain, this result can be generalized to ternary and n-ary constraints using some other notion of consistency ((3,2)-relational consistency).

The class of CRC constraint is also related to discrete temporal reasoning [van Beek, 1992b]. Valdès-Peres [Valdès-Pérez, 1987] shows that path-consistency algorithms find the minimal network for a subclass of Allen's interval algebra [Allen, 1983]. Such a result has also been proposed in the context of point algebra [Vilain and Kautz, 1986; van Beek, 1989].

Montanari [Montanari, 1974] already shows that a path-consistent tree or distributive networks are minimal. He also shows that path consistency of (total) monotone constraints produces a decomposable network. Note that CRC constraints are not distributive and generalize the total monotone functions of Montanari.

Finally, recall that best path-consistency algorithms [Chmeiss, 1996; Han and Lee, 1988; Mohr and Henderson, 1986; Singh, 1995] run in time $O(n^3 d^3)$ and in space $O(n^3 d^2)$ (PC-6). These algorithms do not take advantage of properties of the constraints. A generic arc-consistency algorithm AC-5 has been proposed in [Van Hentenryck *et al.*, 1992] and it can be instantiated to produce linear algorithms for different classes of constraints such as functional and monotone constraints.

4 Connected Row Convex Constraints

Row convex constraints exhibits two problems during path-consistency algorithms. First, when a row convex constraint is composed of disjoint blocks of 1s, its composition with another row convex constraint may not be row convex. Second, even if disjoint blocks are forbidden, intersection may create empty rows and columns and

thus disjoint blocks. Here is an illustration of these two problems :
$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$
$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \cap \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

CRC constraints avoid both problems. Informally, a constraint is CRC if, after removing the empty rows, it is row convex and connected (two successive rows either intersect or are consecutive).

Definition 1 The *reduced form* of a constraint C_{ij}, denoted by C^*_{ij}, is obtained by removing all the empty rows and columns in its matrix representation. The *domain* of i through the constraint C_{ij}, denoted by $D_i(C_{ij})$, is the set $\{v \in D \mid \exists w : \langle v, w\rangle \in C_{ij}\}$.

Definition 2 Let C_{ij} be a row convex constraint and $v \in D_i(C_{ij})$. The *image* of v in C_{ij} is the set $\{w \mid \langle v, w\rangle \in C^*_{ij}\}$. Because of the row convexity of C_{ij}, this set is represented as an interval $[w_1, w_m]$ (over the domain $D_j(C_{ji})$) and we denote w_1 and w_m by $min(C_{ij}, v)$ and $max(C_{ij}, v)$ respectively. We also denote by $succ(w)$ and $pred(w)$ the successor and the predecessor of w in $D_j(C_{ji})$.

Definition 3 A row convex constraint C_{ij} is *connected* if the images $[a, b]$ and $[a', b']$ of two consecutive rows in C^*_{ij} is such that $b' \geq pred(a) \wedge a' \leq succ(b)$.

Definition 4 A constraint C_{ij} is *connected row convex* (CRC) if C^*_{ij} and C^*_{ji} are both row convex and connected.

We assume that C_{ij} is always the transposition of C_{ji}. Notice that CRC constraints are not necessarily row convex (because of empty rows) and that row convex constraints are not necessarily CRC (not connected rows).

The main result of this section is the fact that CRC constraints are closed under composition, intersection and transposition.

Lemma 1 The deletion of rows and columns in a CRC constraint produces a CRC constraint.

Connectivity in Lemma 1 can be proven by showing that all the columns between two non-connnected rows are necessarily empty and must thus be deleted when considering the connectivity property.

Lemma 2 Let C_{ij} be a CRC constraint. Let v_1, v, v_2 be in $D_i(C_{ij})$ such that $v_1 < v < v_2$ and their respective images are $[a_1, b_1]$, $[a, b]$ and $[a_2, b_2]$ in C_{ij}.

$$b_2 < a_1 \Rightarrow [a, b] \cap [b_2, a_1] \neq \emptyset$$
$$a_2 > b_1 \Rightarrow [a, b] \cap [b_1, a_2] \neq \emptyset$$
$$b_2 \geq a_1 \wedge a_2 \leq b_1 \Rightarrow [a_1, b_1] \cap [a_2, b_1] \subseteq [a, b]$$

Theorem 5 The intersection and composition of two CRC constraints and the transposition of a CRC constraint are CRC constraints.

Proof (Sketch) Transposition is obvious. For intersection and composition ($C = A \cap B$ or $C = A.B$), delete the empty rows and columns in the operands to give A^+ and B^+. By Lemma 1, it is sufficient to show that $C^+ = A^+ \cap B^+$ or $C^+ = A^+.B^+$ are CRC. Row convexity is obvious for intersection and results from Lemma 2 for composition. Connectivity can be proven by showing that all the columns between two non-connnected rows are necessarily empty and must thus be deleted when considering the connectivity property. □

Theorem 6 Let G be composed of CRC constraints. The application of a path-consistency algorithm to G produces a minimal and decomposable graph.

5 PC-GEN: Generic Path Consistency

We now present a path-consistency algorithm for CRC constraints. We proceed in two steps. We first present a generic path-consistency algorithm PC-GEN which is parametrized in a way similar to AC-5. We then instantiate PC-GEN to CRC constraints.

procedure PRUNE(**in** Δ, i, j)
Pre: $i, j \in node(G)$.
Post: $C_{ij} = C_{ij_0} \setminus \{\langle v, w\rangle \mid \langle v, w\rangle \in \Delta\}$,
$C_{ji} = C_{ji_0} \setminus \{\langle w, v\rangle \mid \langle v, w\rangle \in \Delta\}$.

procedure INITQUEUE(**out** Q)
Post: $Q = \{\}$.

function EMPTYQUEUE(**in** Q): Boolean
Post: EMPTYQUEUE $\Leftrightarrow$ ($Q = \{\}$).

procedure DEQUEUE(**inout** Q, **out** $i, k, j, \langle v, u\rangle$)
Post: $\langle i, k, j, \langle v, u\rangle\rangle \in Q_0$ and $Q = Q_0 \setminus \{\langle i, k, j, \langle v, u\rangle\rangle\}$.

procedure ENQUEUE(i, j, Δ, **inout** Q)
Pre: $\Delta \subseteq C_{ij}$.
Post: $Q = Q_0 \cup \{\langle i, j, k, \langle v, w\rangle\rangle \mid k \in node(G)$ and $j \neq k$ and $\langle v, w\rangle \in \Delta\}$.
$\cup\{\langle j, i, k, \langle w, v\rangle\rangle \mid k \in node(G)$ and $j \neq i \neq k$ and $\langle v, w\rangle \in \Delta\}$.

Let $PC_{ikj}(v, w) = \exists u : \langle v, u\rangle \in C_{ik}$ and $\langle u, w\rangle \in C_{kj}$.
$PC'_{ikj}(v, w) = \exists u : (\langle v, u\rangle \in C_{ik} \vee \langle i, k, j, \langle v, u\rangle\rangle \in Q)$ and $(\langle u, w\rangle \in C_{kj} \vee \langle j, k, i, \langle w, u\rangle\rangle \in Q)$

procedure PATHCONS(**in** i, k, j, **out** Δ)
Pre: $i, k, j \in node(G)$.
Post: $\Delta = \{\langle v, w\rangle \in C_{ij} \mid \neg PC_{ikj}(v, w)\}$.

procedure LOCALPATHCONS(**in** $i, k, j, \langle v, u\rangle$, **out** Δ)
Pre: $i, k, j \in node(G)$, and $\langle v, u\rangle \notin C_{ik}$.
Post: $\Delta_1 \subseteq \Delta \subseteq \Delta_2$, with
$\Delta_1 = \{\langle v, w'\rangle \in C_{ij} \mid \langle u, w'\rangle \in C^{init}_{kj}$ and $\neg PC_{ikj}(v, w')\}$.
$\Delta_2 = \{\langle v', w'\rangle \in C_{ij} \mid \neg PC_{ikj}(v', w')\}$.

Figure 1: Subproblems for for PC-GEN

The specification of the main operations in PC-GEN are given in Figure 1. In all specifications, a parameter p subscripted with 0 (p_0) represents the value of p at call time and C^{init}_{ij} is the original set of constraint tuples between i and j. This justifies the restriction $i \leq j$ in line 2 of PC-GEN. As is traditional, PC-GEN uses a queue to drive the algorithm. Procedure ENQUEUE is required to take $O(s)$ time, where s is the number of new elements to insert in the queue, and procedure DEQUEUE must take constant time. The deletion of tuples is performed by procedure PRUNE, which removes tuple $\langle v, w\rangle$ from C_{ij}, and $\langle w, v\rangle$ from C_{ji}. Hence, $\langle v, w\rangle \in C_{ij} \Leftrightarrow \langle w, v\rangle \in C_{ji}$ is an invariant of the algorithm.

PC-GEN is parametrized by two procedures, PATHCONS and LOCALPATHCONS whose implementations are left open. Procedure PATHCONS computes the set Δ of tuples in C_{ij} which are not path consistent for the path (i,k,j). Because of the relationship between C_{ij} and C_{ji}, Δ is also the set of tuples (in reverse order) of C_{ji} that are not path consistent for path (j,k,i).

Procedure LOCALPATHCONS returns in Δ a set of tuples of C_{ij} that are not path consistent for (i,k,j) after the tuple $\langle v,u\rangle$ has been removed from the constraint C_{ik}. The set Δ is also the set of tuples (in reverse order) of C_{ji} that are not path consistent in path (j,k,i) after tuple $\langle u,v\rangle$ has been removed from C_{ki}.

The size of Δ computed by LOCALPATHCONS can vary. The set Δ_1 contains the tuples in C_{ij} that become path inconsistent for (i,k,j) due to the removal of the tuple $\langle v,u\rangle$ from C_{ik}. Specifically, a tuple $\langle v,w'\rangle$ is included in Δ_1 if u was a support (i.e., $(u,w') \in C_{kj}^{init}$) and (v,w') is not supported anymore. In some cases, it is possible, but not always desirable, to prune a larger set of tuples. As an extreme case, Δ_2 prunes all tuples in C_{ij} which are not path inconsistent wrt (i,k,j) at call time, regardless of whether they can be supported by $\langle v,u\rangle$. In fact, the specifications can be made even less restrictive by replacing PC_{ikj} by PC'_{ikj} in the definition of Δ in PATHCONS, and in the definition of Δ_1 in LOCALPATHCONS. Our instantiation of PC-GEN to CRC constraints uses this flexibility to improve the time complexity of traditional path-consistency algorithms.

```
Algorithm PC-GEN
Post: G is the largest path-consistent graph for G0.
   begin
1     InitQueue(Q);
2     for each i,k,j ∈ node(G) with i ≤ j do
3     begin
4        PathCons(i,k,j,Δ);
5        Enqueue(i,j,Δ,Q);
6        Prune(Δ,i,j)
7     end;
8     while not EmptyQueue(Q) do
9     begin
10       Dequeue(Q,i,k,j,⟨v,u⟩);
11       LocalPathCons(i,k,j,⟨v,u⟩,Δ);
12       Enqueue(i,j,Δ,Q);
13       Prune(Δ,i,j)
14    end
   end
```

Figure 2: The Path Consistency Algorithm PC-GEN

PC-GEN is depicted in Figure 2 and mimics AC-5. In the loop on lines 2–7, procedure PATHCONS identifies the path-inconsistent tuples with respect to each path of length two. The inconsistent tuples are enqueued and processed in the second loop, on lines 8–14, where procedure LOCALPATHCONS is used to prune tuples of C_{ij} which become inconsistent after the removal of a tuple from C_{ik}. The removal of the tuple $\langle v,w\rangle$ in C_{ij} and $\langle w,v\rangle$ in C_{ji} induces to reconsider all length-two paths involving either (i,j) or (j,i) as the first or as the second arc. It is however unnecessary to explicitly consider the involvement as a second arc (in the ENQUEUE procedure) since LOCALPATHCONS$(i,j,k,\dots)$ will cover both paths (i,j,k) and (k,j,i), and LOCALPATHCONS$(j,i,k,\dots)$ will cover paths (j,i,k) and (k,i,j).

It can be shown that PC-GEN is correct and that it enqueues and dequeues at most $O(n^3d^2)$ elements.

Theorem 7 Given a time complexity of $O(d^2)$ for procedure PATHCONS and a time complexity of $O(\Delta)$ for procedure LOCALPATHCONS, the time complexity of algorithm PC-GEN is bounded by $O(n^3d^2)$.

PC-GEN can be instantiated for general constraints to produce a path-consistency algoritm with a time complexity of $O(n^3d^3)$ and a space complexity of $O(n^3d^2)$. We now turn to its instantiation for CRC constraints.

6 PC-GEN for CRC Constraints

Let $D = \{b,\dots,B\}$.
Let $C_{ij} = \{\langle v_1,v_1\rangle,\dots,\langle v_m,v_m\rangle\}$ if $i=j$
$\quad = \{\langle v_1,w_1\rangle,\dots,\langle v_m,w_m\rangle\}$ if $i \neq j$ (where $v_k, w_k \in D$)

Data Structure

Syntax

$C_{ij}.supmin$: array $[b..B]$ of element $\in D$.
$C_{ij}.supmax$: array $[b..B]$ of element $\in D$.
$C_{ij}.first$: element $\in D$.
$C_{ij}.succ$: array $[b..B]$ of element $\in D$.
$C_{ij}.pred$: array $[b..B]$ of element $\in D$.

Semantics

$C_{ij}.supmin[v] = min(C_{ij},v)$
$C_{ij}.supmax[v] = max(C_{ij},v)$
$C_{ij}.first = min\{v \in D_i(C_{ij})\}$
$C_{ij}.succ[v] = succ(v)$ in $D_i(C_{ij})$
$C_{ij}.pred[v] = pred(v)$ in $D_i(C_{ij})$

Invariant

$C_{ij} = C_{ji}^T$
$C_{ij}.supmin[v] \in D_j(C_{ji})$
$C_{ij}.supmax[v] \in D_j(C_{ji})$

Interface

function EMPTYSUPPORT(**in** v,w, i,k,j): Boolean
Post: EMPTYSUPPORT(v,w, i,k,j) $= \neg PC'_{ikj}(v,w)$

function FIRST(**in** i,j): Integer
Post: FIRST(i,j) $= min\{v \in D_i(C_{ij})\}$

function MIN(**in** v, i,j): Integer
Post: MIN(v, i,j) $= min(C_{ij},v)$

function MAX(**in** v, i,j): Integer
Post: MAX(v, i,j) $= max(C_{ij},v)$

function SUCC(**in** v, i,j): Integer
Post: SUCC(v, i,j) $= succ(v)$ in $D_i(C_{ij})$

function PRED(**in** v, i,j): Integer
Post: PRED(v, i,j) $= pred(v)$ in $D_i(C_{ij})$

Figure 3: The CRC CONSTRAINT Module

CRC constraints can be represented in space $O(d)$ as shown in Figure 3. It is necessary to keep a description of $D_i(C_{ij})$, since row convexity is only enforced on the reduced form. Figure 3 also specifies the operations on CRC constraints which are all implemented in constant time. For instance, EMPTYSUPPORT(v,w,i,k,j) can be implemented by $b' \geq a \wedge a' \leq b$ with $a =$MIN(v,i,k), $b =$MAX(v,i,k), $a' =$MIN(w,j,k), and $b' =$MAX(w,j,k).

procedure PathCons(**in** i, k, j, **out** Δ)
begin
$\Delta := \emptyset$;
for each $v \in D_i(C_{ij})$ **do**
begin
LocalPathCons(i, k, j, v, Δ_v);
$\Delta := \Delta \cup \Delta_v$;
end
end

procedure LocalPathCons(**in** i, k, j, v, **out** Δ)
Pre: $i, k, j \in node(G)$.
Post: $\Delta_1 \subseteq \Delta \subseteq \Delta_2$, with
$\Delta_1 = \{\langle v, w' \rangle \in C_{ij} \mid \neg PC'_{ikj}(v, w')\}$.
$\Delta_2 = \{\langle v', w' \rangle \in C_{ij} \mid \neg PC_{ikj}(v', w')\}$.
begin
BoundedMin(i, k, j, $\langle v$, Max(v, i,j) $\rangle$, Δ', w_{min});
if w_{min} =Max(v, i,j) **then** $\Delta := \Delta'$
else
begin
BoundedMax(i, k, j, $\langle v$, Min(v, i,j) $\rangle$, Δ'', w_{max});
Propagate(i,j,k,$\langle v, w_{min} \rangle$, BoundedMin, pred, Δ_1);
Propagate(i,j,k,$\langle v, w_{min} \rangle$, BoundedMin, succ, Δ_2);
Propagate(i,j,k,$\langle v, w_{max} \rangle$, BoundedMax, pred, Δ_3);
Propagate(i,j,k,$\langle v, w_{max} \rangle$, BoundedMax, succ, Δ_4);
$\Delta := \Delta' \cup \Delta'' \cup \Delta_1 \cup \Delta_2 \cup \Delta_3 \cup \Delta_4$;
end
end

Figure 4: PathCons and LocalPathCons for CRC constraints.

An implementation of Procedures PathCons and LocalPathCons is given in Figure 4. Note that the specification of LocalPathCons has been relaxed further by removing parameter u and that PathCons can now be expressed in terms of LocalPathCons. In LocalPathCons, BoundedMin computes the interval Δ' to be removed on the left of the interval in row v while BoundedMax computes the interval Δ'' to be removed on the right of the interval in row v. Although this pruning is sufficient, it may destroy the CRC property. To preserve the property, it is necessary to perform additional pruning on the rows above or below v. This is the role of the Propagate instructions. The specifications and implementation of the procedures are given in Figure 5 and the intuition behind LocalPathCons is captured in Figure 6. Because $C_{ij} := C_{ij} \cap C_{ik}.C_{kj}$ produces a CRC constraint, the implementation is guaranteed to keep C_{ij} connected row convex. Note that Propagate works from v to the exterior, while BoundedMin and BoundedMax work from the exterior to the interior.

Complexity Prune can be performed in $O(\Delta)$ assuming the elements of Δ are ordered to preserve the CRC property. The ordering can be performed during the construction of Δ during LocalPathCons without incurring any cost. An implementation of Δ as a doubly-linked list is sufficient for this purpose given the way Δ is constructed as mentioned in the previous section. The complexity of Procedures Propagate, BoundedMin and BoundedMax is obviously $O(\Delta)$. Hence LocalPathCons is $O(\Delta)$. By Theorem 7, the time complexity of PC-GEN is $O(n^3 d^2)$. The space complexity per con-

procedure Propagate(**in** i, k, j, $\langle v, w \rangle$, Bounded, Next, **out** Δ)
Let v_k = Next$^k(v)$,
w_k and Δ_k such that Bounded($i, k, j, \langle v_k, w \rangle, \Delta_k, w_k$),
$m = max\{k \mid \Delta_k \neq \emptyset \wedge w_k = w\}$.
Post: $\Delta = \bigcup_{1 \leq k \leq m+1} \Delta_k$
begin
$\Delta := \emptyset$;
$v_{calc} := v$;
repeat
v_{calc} :=Next(v_{calc});
Bounded(i, k, j, $\langle v_{calc}, w \rangle$, Δ_{calc}, w_{calc}) ;
$\Delta := \Delta \cup \Delta_{calc}$;
until ($w_{calc} \neq w$);
end

procedure BoundedMin(**in** i, k, j, $\langle v, w \rangle$, **out** Δ, w_{min})
Post: $w_{min} = max\{w \in D_j(C_{ji}) \mid \forall w' \in$ [Min$(v, i, j), w]$: EmptySupport$(v, w', i, k, j)\}$
$\Delta = \{ \langle v, w' \rangle \mid w' \in$ [Min$(v, i, j), w_{min}] \}$
begin
$\Delta := \emptyset$;
w_2 :=Min(v, i, j);
while $(w_2 \leq w) \wedge \neg$EmptySupport(v,w_2,i,k,j) **do**
begin
$\Delta := \Delta \cup \{\langle v, w_2 \rangle\}$;
w_2 := succ(w_2);
end ;
w_{min} := pred(w_2);
end

procedure BoundedMax(**in** i, k, j, $\langle v, w \rangle$, **out** Δ, w_{max})
Post: $w_{max} = min\{w \in D_j(C_{ji}) \mid \forall w' \in [w$,Max$(v, i, j)]$: EmptySupport$(v, w', i, k, j)\}$
$\Delta = \{ \langle v, w' \rangle \mid w' \in [w_{max}$,Max$(v, i, j)] \}$
begin
$\Delta := \emptyset$;
w_2 :=Max(v, i, j);
while $(w_2 \geq w) \wedge \neg$EmptySupport(v,w_2,i,k,j) **do**
begin
$\Delta := \Delta \cup \{\langle v, w_2 \rangle\}$;
w_2 := pred(w_2);
end ;
w_{max} := succ(w_2);
end

procedure Prune(**in** Δ, i, j)
Pre: $i, j \in node(G)$,
C_{ij} is a CRC constraint,
$C_{ij} \setminus \Delta$ is a CRC constraint.
Post: $C_{ij} = C_{ij_0} \setminus \{\langle v, w \rangle \mid \langle v, w \rangle \in \Delta\}$,
$C_{ji} = C_{ji_0} \setminus \{\langle w, v \rangle \mid \langle v, w \rangle \in \Delta\}$.

Figure 5: Subproblems for PC-CRC.

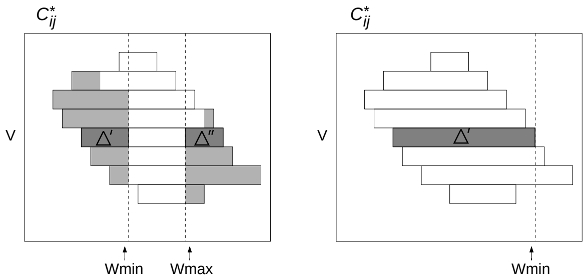

Figure 6: Illustrating LocalPathCons for CRC constraints : two possible cases.

straint is $O(d)$ and $O(nd)$ for all the constraints. The space complexity of the queue is bounded by $O(n^2 d)$ because procedure LocalPath does not use parameter u and elements in the queue can be grouped as tuples of the form $\langle i, j, E, v\rangle$, where the set E is initially $node(G) \backslash \{j\}$. The set E can be shared by all elements of the queue except the first one.

Theorem 8 For CRC constraints, PC-GEN has a time complexity of $O(n^3 d^2)$ and a space complexity of $O(n^2 d)$.

The above theorem is valid for incomplete graphs of CRC constraints as well, since the completion of the graph introduces *TRUE* constraints which are CRC.

Finding a Solution A path-consistent graph with CRC constraints is decomposable due to Helly's theorem (e.g., [Haroud and Faltings, 1996]). The proof in [van Beek and Dechter, 1995] is constructive and the author proposes a $O(n^2 d)$ algorithm to find a solution. We propose in Figure 7 an Instantiate procedure with a time complexity of $O(n^2)$ for CRC constraints. It is based on van Beek's algorithm, but takes advantage of the data structure.

```
procedure Instantiate(in G, out ⟨x_1,...,x_n⟩)
Pre: G has only CRC constraints, and is path-consistent,
     D_i ≠ ∅ (1 ≤ i ≤ n)
Post: ⟨x_1,...,x_n⟩ is a solution of G.

    begin
        for i := 1 to n do
        begin
            L := First(i,i);
            for j := 1 to i-1 do L := max( L, Min(x_j,j,i) ) ;
            x_i := L
        end
    end
```

Figure 7: Instantiate for CRC Constraints.

The total complexity to detect inconsistency or to find a solution of a graph composed with CRC constraints is thus $O(n^3 d^2)$, the time complexity of the path-consistency algorithm.

7 Conclusion

In this paper, we studied constraint satisfaction over connected row convex (CRC) constraints, a large class of constraints subsuming, in particular, monotone constraints. We showed that, contrary to row convex constraints, CRC constraints are closed under composition, intersection, and transposition, establishing that path consistency over CRC constraints produces a minimal and decomposable network and strenghtening the results of van Beek and Dechter [1995]. We then presented a path-consistency algorithm for CRC constraints running in time $O(n^3 d^2)$ and space $O(n^2 d)$, where n is the number of variables and d is the size of the largest domain, as an instantiation of a generic path-consistency algorithm. This improves the traditional time complexity $O(n^3 d^3)$ and space complexity $O(n^3 d^2)$. Finally, we show that a solution can be found in time $O(n^2)$, once the graph is path-consistent.

Further research will be devoted to studying how to improve path consistency and its approximation to continuous domains, since path consistency has been shown instrumental in speeding search considerably for some transistor modelling problems, and to other classes of discrete domains.

Acknowledgment We thank anonymous reviewers for helpful comments. This research is partially supported by the *Actions de recherche concertées (ARC/95/00-187)* of the Direction générale de la Recherche Scientifique – Communauté Française de Belgique, by the Office of Naval Research (ONR Grant N00014-94-1-1153 and a NSF National Young Investigator Award with matching funds of Hewlett-Packard.

References

[Allen, 1983] J.F. Allen. Maintaining Knowledge About Temporal Reasoning. *J.ACM*, 26:832–843, 1983.

[Chmeiss, 1996] A. Chmeiss. Sur la consistance de chemin et ses formes partielles. In *Actes du Congrès AFCET-RFIA'96, Rennes*, 1996.

[Freuder, 1982] E.C. Freuder. A Sufficient Condition for Backtrack-Free Search. *J.ACM*, 29:24–32, 1982.

[Han and Lee, 1988] C.C. Han and C.H. Lee. Comments on Mohr and Henderson's Path Consistency Algorithm. *Artif. Intel.*, 36:125–130, 1988.

[Haroud and Faltings, 1996] D. Haroud and B. Faltings. Consistency techniques for continuous constraints. *Constraints, An International Journal*, 1:85–118, 1996.

[Mohr and Henderson, 1986] R. Mohr and T.C. Henderson. Arc and Path Consistency Revisited. *Artif. Intel.*, 28:225–233, 1986.

[Jeavons and Cooper, 1995] P.G. Jeavons and M.C. Cooper. Tractable constraints on ordered domains. *Artif. Intel..*, 79:327-339, 1995.

[Montanari, 1974] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7(2):95–132, 1974.

[Singh, 1995] M. Singh. Path consistency revisited. In *IEEE-ICTAI'95, Washington DC*, 1995.

[Valdès-Pérez, 1987] R.E. Valdès-Pérez. The satisfiability of temporal constraint network. In AAAI-87, Seattle, pages 745–750, 1987.

[van Beek, 1989] P. van Beek. Approximation algorithms for temporal reasoning. In IJCAI-89, pages 745–750, 1989.

[van Beek, 1992b] P. van Beek. Reasoning About Qualitative Temporal Reasoning. *Artif. Intel.*, 58:297–326, 1992.

[van Beek and Dechter, 1995] P. van Beek and R. Dechter. On the Minimality and Global Consistency of Row Convex Networks. *J.ACM*, 42:543-561, 1995.

[Van Hentenryck *et al.*, 1992] P. Van Hentenryck, Y. Deville, and C. Teng. A generic arc-consistency algorithm and its specializations. *Artif. Intel.*, 57(2–3):291–321, 1992.

[Vilain and Kautz, 1986] M. Vilain and Kautz. Constraint propagation algorithms for temporal reasoning. In AAAI-86, Philadelphia, pages 132–144, 1986.