Solving Connected Row Convex Constraints by Variable Elimination

Yuanlin Zhang and Satyanarayana Marisetti

Department of Computer Science, Texas Tech University, Lubbock, USA

Abstract

We propose an algorithm for the class of connected row convex constraints. In this algorithm, we introduce a novel variable elimination method to solve the constraints. This method is simple and able to make use of the sparsity of the problem instances. One of its key operations is the composition of two constraints. We have identified several nice properties of connected row convex constraints. Those properties enable the development of a fast composition algorithm whose complexity is linear to the size of the variable domains. Compared with the existing work including randomized algorithms, the new algorithm has favorable worst case time and working space complexity. Experimental results also show a significant performance margin over the existing consistency based algorithms.

Key words: Constraint Satisfaction Problems, Connected Row Convex Constraints, Variable Elimination, Path Consistency, Constraint Composition

1 Introduction

Constraint satisfaction techniques have found widespread applications in combinatorial optimisation, scheduling, configuration, and many other areas [3]. However, Constraint Satisfaction Problems (CSP) are NP-hard in general. One active research area is to identify tractable CSP problems and find efficient algorithms for them.

Many interesting tractable problems have been identified (see [3]). The focus of this paper is on a class of connected row convex constraints (CRC). Some problems, e.g., the scene labeling problem and constraint based grammar examples given in [8], are CRC constraints.

Row convex constraints were first proposed by van Beek and Dechter [8]. If a problem composed of row convex constraints is path consistent, it is tractable to find one of its solutions. However, in general path consistency does not preserve the row convexity of constraints. Global consistency is therefore not guaranteed after path consistency is enforced on a row convex problem. Deville et al. [4] restrict row convexity to connected row convexity (CRC) which is preserved under the path consistency enforcing operations – intersection and composition of constraints. One can find a solution of CRC constraints by enforcing path consistency. Deville et al. also provide an algorithm more efficient than the general path consistency algorithm by making use of certain properties of row convexity. The algorithm has a worst case time complexity of $O(n^3d^2)$ with space complexity of $O(n^2d)$ where n is the number of variables, d the maximum domain size. Recently, Kumar [5] has proposed a randomized algorithm for CRC constraints with time complexity of $O(\gamma n^2d^2)$ and space complexity $O(ed)$ (personal communication) where e is the number of constraints and γ the maximum degree of the constraint graph.

In this paper, we propose a new algorithm to solve CRC constraints with time complexity of $O(n\sigma^2d+e'd^2)$ where σ is the *elimination degree* and e' the number of edges of the triangulated graph of the given problem. We observe that the satisfiability of CRC constraints is preserved when a variable is eliminated with proper modification of the constraints on the neighbors of the eliminated variable. The new algorithm simply eliminates the variables one by one until it reaches a special problem with only one variable.

In the matrix representation of a CRC constraint, the contour of the 1's shows some "monotonicity" which results in very nice properties. Those properties make it possible to develop a fast algorithm with time complexity of $O(d)$ for the composition of two constraints, a key operation of the elimination algorithm.

In this paper, we present the elimination algorithm (Section 3) after the preliminaries on CRC constraints (Section 2). The properties of CRC constraints and methods to compute their composition are then shown in Section 4. We discuss the elimination algorithm on problems with sparse constraint graphs in Section 5. Empirical study of the algorithm is presented in Section 6 before we conclude the paper.

2 Preliminaries

A *binary constraint satisfaction problem (CSP)* [7,6] is a triple (V, D, C) where V is a finite set of variables, $D = \{D_x \mid x \in V$ and D_x is the finite domain of $x\}$, and C is a finite set of binary constraints over the variables of V. As usual, we assume there is only one constraint on a pair of variables. We use n, e, and d to denote the number of variables, the number of constraints, and the maximum domain size of a CSP problem. We use $i, j, \ldots$ and $x, y, \ldots$ to denote variables in this paper. The *constraint graph* of a problem (V, D, C) is a graph with vertices V and edges $E = \{\{i, j\} \mid c_{ij} \in C\}$. A CSP is *satisfiable* if there is an assignment of values to variables such that all constraints are satisfied.

Assume there is a total ordering on each domain of D. Functions $\texttt{succ}(u, D_i)$ $(u \in D_i)$ and $\texttt{pred}(u, D_i)$ $(u \in D_i)$ denote respectively the successor and predecessor of u in the current domain $D_i \cup \{\textsf{head}, \textsf{tail}\}$ where $\textsf{head}$ and $\textsf{tail}$ do not belong to any domain and $\textsf{head}$ ($\textsf{tail}$ respectively) is smaller (larger respectively) than any other value of the domain. The domain D_i is omitted when it is clear from the context.

Given a constraint c_{ij} and a value $a \in D_i$, the *extension set* $c_{ij}[a]$ is $\{b \in D_j \mid (a, b) \in c_{ij}\}$. $c_{ij}[a]$ is also called the *image* of a with respect to c_{ij}. We use $c_{ij}[a].\textsf{min}$ to denote $\min\{v \in c_{ij}[a]\}$, and $c_{ij}[a].\textsf{max}$ to denote $\max\{v \in c_{ij}[a]\}$. Clearly, $c_{ij}[\textsf{head}] = c_{ij}[\textsf{tail}] = \emptyset$.

Standard operations of intersection and composition can be applied to constraints. The *composition* of c_{ix} and c_{xj}, denoted by $c_{ix} \circ c_{xj}$, is a constraint

$$\{(a, c) \mid a \in D_i, c \in D_j, \exists b \in D_x \text{ such that } (a, b) \in c_{ix} \text{ and } (b, c) \in c_{xj}\}$$

on i and j. It is convenient to use a Boolean matrix to represent a constraint c_{ij}. Its rows and columns are ordered by the ordering of the values of D_i and D_j.

A constraint c_{ij} is *arc consistent* (AC) if every value of D_i has a support in D_j and every value of D_j has a support in D_i. A CSP problem is *arc consistent* if all its constraints are arc consistent. A path $x, \ldots, y$ of a constraint graph, i.e., a sequence of variables, is *consistent* if for any assignments $x = a$ and $y = b$ such that $(a, b) \in c_{xy}$, there is an assignment for each of other variables in the path such that all constraints over the path are satisfied by the assignments. A constraint graph is *path consistent* if every path of the graph is consistent. A CSP is *path consistent* if the completion of its constraint graph is path consistent. A CSP is *partially path consistent* if its constraint graph is path consistent [1].

A constraint c_{ij} is *row convex* if there exists a total ordering on D_i and D_j respectively such that the 1's are consecutive in each row and column of the matrix of c_{ij}, i.e., for any $u \in D_i$ ($v \in D_j$ respectively) $c_{ij}[u] = [c_{ij}[u].\textsf{min}, c_{ij}[u].\textsf{max}]$ ($c_{ji}[v] = [c_{ji}[v].\textsf{min}, c_{ji}[v].\textsf{max}]$ respectively). The *reduced form* of a constraint c_{ij} is obtained by removing from D_i and D_j those values whose images with respect to c_{ij} are empty. For a row convex constraint c_{ij}, the image of $a \in D_i$ can be represented as an *interval* $[u, v]$ where u is the minimal and v is the maximal value of D_j such that $(a, u), (a, v) \in c_{ij}$. A row convex constraint c_{ij} is *connected* if the images $[a, b]$ and $[a', b']$ of any two consecutive rows or columns of c_{ij} are not empty and satisfy that their intersection is not empty, $b = \texttt{pred}(a')$, or $b' = \texttt{pred}(a)$. A constraint c_{ij} is *connected row convex* if its reduced form is row convex and connected. The constraints obtained from the intersection or composition of two CRC constraints are still connected row convex. The transposition of a CRC constraint is still connected row convex. Enforcing path consistency on a CSP of CRC constraints will make the problem globally consistent [4].

The consistency property on row convex constraints is due to some nice property on convex sets. Given a set U and a total ordering $\leq$ on it, a set $A \subseteq U$ is *convex* if its elements are consecutive under the ordering, that is

$$A = \{v \in U \mid \min A \leq v \leq \max A\}.$$

Consider a collection of sets $S = \{E_1, \ldots, E_k\}$ and an ordering $\leq$ on $\cup_{i=1..k} E_i$ such that every $E_i (1 \leq i \leq k)$ is convex. The intersection of the sets of S is not empty if and only if the intersection of every pair of sets of S is not empty [8,12].

3 Variable elimination in CRC

Consider a problem (V, D, C) and a variable $x \in V$. The *relevant* constraints of x, denoted by R_x, are the set of constraints $\{ c_{yx} \mid c_{yx} \in C\}$. To *eliminate* x is to transform (V, D, C) to $(V - \{x\}, D, C')$ where $C' = C \cup \{c_{ix} \circ c_{xj} \cap c_{ij} \mid c_{jx}, c_{ix} \in R_x \text{ and } i \neq j\} - R_x$. In the elimination, when composing c_{ix} and c_{xj}, if $c_{ij} \notin C$ we simply take c_{ij} as a universal constraint, i.e., $D_i \times D_j$.

Theorem 1 *Consider an arc consistent problem P=(V, D, C) of CRC constraints and a variable $x \in V$. Let P'=(V', D', C') be the problem after x is eliminated. P is satisfiable iff P' is satisfiable.*

Proof. We first prove if P is satisfiable, so is P'. Let s be a solution of P, s_x an assignment of x by s, and $s_{\bar{x}}$ be the restriction of s to V'. We only need to show that $s_{\bar{x}}$ satisfies $c'_{ij} \in C'$ for all $c_{ix}, c_{xj} \in C$. Since s is a solution of P, $s_{\bar{x}}$ satisfies c_{ix}, c_{jx} and c_{ij}. Hence, $s_{\bar{x}}$ satisfies c'_{ij}.

Next we prove if P' is satisfiable, so is P. Let t be a solution of P'. We will show that t is extensible consistently to x in P. Let V_x be $\{i \mid c_{ix} \in R_x\}$. For each $i \in V_x$, let the assignment of i in t be a_i. Let $S = \{c_{ix}[a_i] \mid i \in V_x\}$. Since all constraints of P are row convex and P is arc consistent, the sets of S are convex and none of them is empty.

Consider any two sets $c_{ix}[a_i], c_{jx}[a_j] \in S$. Since t is a solution of P', $(a_i, a_j) \in c'_{ij}$ where c'_{ij} is a constraint of P'. The fact that $c'_{ij} = c_{ix} \circ c_{xj} \cap c_{ij}$, where c_{ij} is either in C or universal, implies that there exists a value $b \in D_x$ such that a_i, a_j and b satisfy c_{ix}, c_{jx} and c_{ij}. Hence, $c_{ix}[a_i] \cap c_{jx}[a_j] \neq \emptyset$. By the property on the intersection of convex sets, the intersection of the sets of S is not empty. For any $v \in \cap_{E \in S} E$, it is easy to verify that (t, v) is a solution of P. Therefore, P is satisfiable. □

Based on Theorem 1, we can reduce a CSP with CRC constraints by eliminating the variables one by one until a trivial problem is reached.

Algorithm 1: Basic elimination algorithm for CRC constraints

```
eliminate (inout$(V, D, C)$, out consistent, s)
1  // $(V, D, C)$ is a CSP problem, s is a stack
2  enforce arc consistency on $(V, D, C)$
3  if some domain of $D$ becomes empty then
4      consistent ← false, return
5  consistent ← true
6  $C' \leftarrow C, C'' \leftarrow \emptyset, L \leftarrow V$
7  while $L \neq \emptyset$ do
8      select and remove a variable $x$ from $L$
9      $C'_x \leftarrow \{c_{yx} \mid c_{yx} \in C'\}$
10     foreach $c_{ix}, c_{jx} \in C'_x$ where $i < j$ do
11         $c'_{ij} \leftarrow c_{ix} \circ c_{xj}$
12         if $c_{ij} \in C'$ then $c'_{ij} \leftarrow c'_{ij} \cap c_{ij}$
13         $C' \leftarrow (C' - \{c_{ij}\}) \cup \{c'_{ij}\}$
14         collect to $Q$ the values not valid under $c'_{ij}$
15     remove from the domains the values in $Q$ and propagate the
       removals
16     if some domain becomes empty then
17         consistent ← false, return
18     $C' \leftarrow C' - C'_x$
19     $C'' \leftarrow C'' \cup C'_x$
20     s.push $(x)$
21 $C \leftarrow C''$, consistent ← true
```

The procedure `eliminate`((V, D, C), consistent, s) in Algorithm 1 eliminates the variables of (V, D, C). When it returns, consistent is false if some domain becomes empty and true otherwise; the eliminated variables are pushed to the stack s in order, and C will contain only the "removed" constraints associated with the eliminated variables. Most parts of the algorithm are clear by themselves. The body of the while loop (lines $7 - 20$) eliminates the variable x. Line 14 and 15 remove values no longer supported by the newly generated constraints and propagate these removals, whose nature is very close to arc consistency enforcing. The main purpose of this processing is for the correctness of the algorithm. Details will be discussed in Section 4.2. Line 18 discards from C' the constraints incident on x, i.e., C'_x. and Line 19–20 push x to the stack and put the constraints C'_x, which are associated to x, into C''. After `eliminate`, the stack `s`, D (revised in lines 2, 15), and C will be used to find a solution of the original problem.

With the elimination algorithm, it is rather straightforward to design an algorithm to find a solutions of a problem of CRC constraints. A procedure `solve` is shown in Algorithm 2. L (line 5) represents the assigned variables. C_x in line 8 consists of only those constraints that involve x and an instantiated variable. In line 10, when C_x is empty, the domain D_x is not modified.

Algorithm 2: Find a solution of CRC constraints

```
solve (in $(V, D, C)$, out consistent)
 1 // $(V, D, C)$ is a CSP problem
 2 create an empty stack s
 3 eliminate ( $(V, D, C)$, consistent, s)
 4 if not consistent then return
 5 $L \leftarrow \emptyset$
 6 while not s.empty () do
 7     $x \leftarrow$ s.pop ()
 8     $C_x \leftarrow \{c_{ix} \mid c_{ix} \in C, i \in L\}$
 9     for each $i \in L$, let $b_i$ the assignment of $i$
10     $D_x \leftarrow \cap_{c_{ix} \in C_x} c_{ix}[b_i]$
11     choose any value $a$ of $D_x$ as the assignment of $x$
12     $L \leftarrow L \cup \{x\}$
13 output the assignment of the variables of $L$
```

Theorem 2 *Assume the time and working space complexity of the composition (and intersection respectively) of two constraints are $O(\alpha)$ and $O(1)$. Further assume the time and space complexity of enforcing arc consistency are $O(ed^2)$ and $O(\beta)$. Given a CRC problem P=(V, D, C), a solution of the problem can be found in $O(n^3\alpha + n^2d^2)$ with working space $O(n + \beta)$.*

Proof. In the worst case, the constraint graph of P is complete. For every variable, there are at most $n - 1$ neighbors. So, to eliminate a variable (line 10–14 of Algorithm 1) takes $O(n^2\alpha)$. A total of n variables are removed. So, the time complexity of `eliminate` is $O(n^3\alpha)$. The procedure `eliminate` dominates the complexity of `solve` and thus to find a solution of P takes $O(n^3\alpha{+}x)$ where x is the cost of removing values and its propagation (Line 15 of Algorithm 1). In fact, x is $O(n^2d^2)$ because of the following reasons. Line 15, playing a role of arc consistency enforcing, is called every time a variable is eliminated. In the analysis of the worst case complexity of an arc consistency algorithm (e.g., [11]), one assumes all values of the domains have been removed. Therefore, the worst case complexity ($O(ed^2)$) of a single invocation of Line 15 is the same as that of n invocations. The number of constraints of the problem may be increased to as many as n^2.

Working space here *excludes the space for the representation of the constraints and the new constraints created by elimination.* Throughout this paper, space complexity refers to working space complexity by default. A stack `s` and a set L are used by `solve` and `eliminate` to hold variables. They need $O(n)$ space. The total space used by `solve` is $O(n + \beta)$ where β is the space cost (amortizable) of removing values and its propagation. □

4 Composing two CRC constraints

In this section, we will introduce a basic composition method, a procedure to remove values without support, the properties of CRC constraints and finally a linear composition algorithm.

Since in the main algorithm `eliminate`, arc consistency is enforced (line 2 and 15 of Algorithm 1) during the variable elimination, all constraints, including the original and the newly generated ones, are row convex and connected before composition is applied to them. Therefore, we assume all constraints are row convex and connected in this section.

The following property is useful across this section.

Property 1 *Let c_{ij} be the composition of c_{ix} and c_{xj}. If both c_{ix} and c_{xj} are row convex and connected, for any $u \in D_i$, $c_{ij}[u] \neq \emptyset$.*

Proof. Since c_{ix} is row convex and connected, $c_{ix}[u]$ is not empty. For any value $v \in c_{ix}[u]$, c_{xj} being row convex and connected implies that $c_{xj}[v]$ is not empty. Therefore, for all $d \in c_{xj}[v]$, $d \in c_{ij}[u]$. Hence, $c_{ij}[u] \neq \emptyset$. □

To compose two constraints c_{ix} and c_{xj}, one can simply multiply their matrices, which amounts to the complexity of $O(d^3)$. We will present fast algorithms to compute the composition in this section. Thanks to the row convexity, a constraint c_{ij} is represented here as *intervals*: $\{[c_{ij}[a].\mathsf{min}, c_{ij}[a].\mathsf{max}] \mid a \in D_i\}$.

4.1 Basic algorithm to compute composition

With the interval representation, we have procedure `compose` in Algorithm 3. For any value $u \in D_i$ and $v \in D_j$, lines 6–8 compute whether $(u, v) \in c_{ix} \circ c_{xj}$.

Proposition 1 *The procedure of `compose` has a time complexity of $O(d^2)$ with working space complexity of $O(1)$.*

Algorithm 3: Basic algorithm for computing the composition of two constraints

```
compose (in $c_{ix}, c_{xj}$ , out $c_{ij}$)
1  $u \leftarrow$ succ (head, $D_i$)
2  while $u \neq$ tail do
3      $v \leftarrow$ succ (head, $D_j$)
4      min $\leftarrow$ tail, max $\leftarrow$ head
5      while $v \neq$ tail do
6          if not disjoint ($c_{ix}[u]$, $c_{jx}[v]$) then
7              if $v >$ max then max $\leftarrow v$
8              if $v <$ min then min $\leftarrow v$
9          $v \leftarrow$ succ (v, $D_j$)
10     $c_{ij}[u]$.min $\leftarrow$ min, $c_{ij}[u]$.max $\leftarrow$ max
11     $u \leftarrow$ succ ($u$, $D_i$)
disjoint (in $c_{ix}[u], c_{jx}[v]$)
12 if ($c_{ix}[u]$.min $> c_{jx}[v]$.max) or ($c_{ix}[u]$.max $< c_{jx}[v]$.min) then
13     return true
14 else return false
```

Proof. The two while loops (lines 2, 5) give a time complexity of $O(d^2)$. The procedure needs only constant working space. □

Note that, due to the interval representation of constraints, for any constraint c_{ij}, we need separate representation of c_{ij} and c_{ji}. For any c_{ix} and c_{xj}, we call `compose` twice (with c_{ix}, c_{xj} and c_{jx}, c_{xi} respectively) to compute c_{ij} and c_{ji} respectively. This does not affect the complexity of those algorithms using `compose`.

4.2 Remove values without support

Although composition does not lead to the removal of values under our assumption, the intersection will inevitably cause the removal of values. In this case, to maintain the row convexity and connectedness, we need to remove values without support from their domains (line 15 of Algorithm 1). The algorithm `removeValues`, listed in Algorithm 4, makes use of the interval representation (line 6–11) to propagate the removal of values. If a domain becomes empty (line 13), we let the program involving this procedure exit with an output indicating inconsistency. One can verify that algorithm `removeValues`((V, D, C), Q)), where Q is the "initially" unsupported values of the problem, achieves arc consistency on the problem (V, D, C).

Proposition 2 *Given a CSP problem (V, D, C) of CRC constraints with an interval representation, the worst case time complexity of* ***`removeValues`*** *is*

Algorithm 4: Remove values

```
removeValues (in $(V, D, C)$, $Q$)
1  // $Q$ is a queue of values to be removed
2  while $Q \neq \emptyset$ do
3      take and delete a value $(u, x)$ from $Q$
4      foreach variable $y$ such that $c_{yx} \in C$ do
5          foreach value $v \in D_y$ do
6              if $c_{yx}[v]$.min $= u = c_{yx}[v]$.max then
7                  $Q \leftarrow Q \cup \{(v, y)\}$
8              else if $u = c_{yx}[v]$.min then
9                  $c_{yx}[v]$.min $\leftarrow$ succ $(u, D_x)$
10             else if $u = c_{yx}[v]$.max then
11                 $c_{yx}[v]$.max $\leftarrow$ pred $(u, D_x)$
12     delete $u$ from $D_x$
13     if $D_x = \emptyset$ then output inconsistency, exit
```

$O(ed^2)$ with space complexity of $O(nd)$.

Proof. Let δ_i be the degree of variable $i \in V$. To delete a value (line 3–12), the cost is $\delta_i d$. In the worst case, nd values are removed. Hence the time complexity is $\sum_{i \in 1..n} \delta_i d \times d = O(ed^2)$. The space cost for Q is $O(nd)$. □

Given a problem of CRC constraints that are represented by matrices, for each constraint c_{ij} and $u \in D_i$, we set up $c_{ij}[u]$.min and $c_{ij}[u]$.max and collect the values of D_i without support. Let Q contain all the removed values during the setup stage, we then call `removeValues` to make the problem arc consistent.

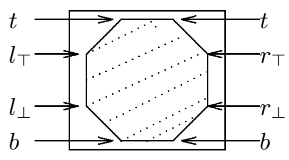
By the process above, Theorem 1, and Proposition 2, it is clear that the procedure `solve` equipped with `compose` and `removeValues` has the following property. Note that the time and space cost of `removeValues` are “amortized” in `eliminate`.

Corollary 1 *Given a problem of CRC constraints, `solve` can find a solution in time $O(n^3d^2)$ with space complexity $O(nd)$.*

4.3 Properties of row convex and connected constraints

In this section, we will present the properties of row convex and connected constraints. As one may see, `compose` makes use of the row convexity to the minimal degree. In fact, we can do better.

It is useful to point out that “a constraint is row convex and connected” is not identical to “a constraint is CRC” since the latter means that its reduced form is row convex and connected.



(a)

$$c_{ij} = \begin{array}{c|cccc|} & a_1 & a_2 & a_3 & a_4 \\ a_1 & 0 & 1 & 0 & 0 \\ a_2 & 1 & 1 & 1 & 0 \\ a_3 & 0 & 1 & 1 & 0 \\ a_4 & 0 & 0 & 1 & 1 \\ a_5 & 0 & 0 & 1 & 1 \end{array}$$

(b)

Fig. 1. (a) The abstract shape of 1’s in the matrix of a CRC constraint. (b) The matrix of a row convex and connected constraint c_{ij}. Its rows and columns are indexed by a_1 to a_5. Assume the total ordering on the values is $a_1, a_2, \cdots, a_5$. $c_{ij}[a_2] = \{a_1, a_2, a_3\}$, $c_{ij}[a_2].\mathsf{min} = a_1$ and $c_{ij}[a_2].\mathsf{max} = a_3$. For c_{ij}, $t = a_1, l_\top = l_\perp = a_2, b = a_5, r_\top = a_4$, and $r_\perp = a_5$.

The 1’s in the matrix of a CRC constraint form an *abstract* shape (the shaded area in Fig. 1(a)) where the slant edges mean monotonicity rather than real boundaries. It is characterised by the following fields associated with c_{ij}. Let $min = \min\{c_{ij}[u].\mathsf{min} \mid u \in D_i\}$ and $max = \max\{c_{ij}[u].\mathsf{max} \mid u \in D_i\}$. The field $c_{ij}.t$ denotes the value of D_i corresponding to the first row that contains at least a 1, $c_{ij}.b$ the value of D_i corresponding to the last row that contains at least a 1, $c_{ij}.l_\top$ the *first* value u of D_i such that $c_{ij}[u].\mathsf{min}{=}min$, $c_{ij}.l_\perp$ the *last* value v of D_i such that $c_{ij}[v].\mathsf{min}{=}min$, $c_{ij}.r_\top$ the *first* value u of D_i such that $c_{ij}[u].\mathsf{max}{=}max$, and $c_{ij}.r_\perp$ the *last* value v of D_i such that $c_{ij}[v].\mathsf{max}{=}max$. As an example, see Fig. 1(b). If c_{ij} is row convex and connected, $c_{ij}.t = \mathsf{succ}(\mathsf{head}, D_i)$ and $c_{ij}.b = \mathsf{pred}(\mathsf{tail}, D_i)$. The fields are related as follows.

Property 2 *Given a row convex and connected constraint c_{ij}, for all $u \in D_i$ such that $c_{ij}.l_\top \le u \le c_{ij}.l_\perp$, $c_{ij}[u].\mathsf{min}{=}\ min$; for all u such that $c_{ij}.r_\top \le u \le c_{ij}.r_\perp$, $c_{ij}[u].\mathsf{max}{=}\ max$; and the relation between $c_{ij}.l_\top$ ($c_{ij}.l_\perp$) and $c_{ij}.r_\top$ ($c_{ij}.r_\perp$) can be arbitrary.*

We present below the *connectedness* and *monotonicity* property of row convex and connected constraints that are the basis of other properties in this section. Intuitively, the monotonicity results from the observation that in general from the first row to the last row of the matrix of a row convex and connected constraint, the left ends of the rows decrease first and then increase while the right end of the rows increase first and then decreases. This phenomenon is clearly shown in the picture of Figure 1(a).

Property 3 (Connectedness) *Consider a row convex and connected constraint c_{ij} and values $e, f \in D_j$ and $u, v \in D_i$ such that $e \in c_{ij}[u]$ and $f \in c_{ij}[v]$*

and $e \leq f$. For any value a of D_j such that $a \in [e, f]$, there exists a value $w \in D_i$ such that $w \in [u, v]$ and $a \in c_{ij}[w]$.

Proof. Without loss of generality, we assume $u \leq v$. When $u = v$, for any $a \in [e, f]$, there exists $w(= u)$ such that $a \in c_{ij}[w]$. Now assume $u < v$. We also assume $a \neq e$ and $a \neq f$ because otherwise the property holds immediately.

We prove the claim by contradiction. Assume for all $w \in [u, v]$, $a \notin c_{ij}[w]$. Since c_{ij} is row convex, the values in $[u, v]$ fall into two classes: $left = \{w \mid c_{ij}[w].\mathsf{max} \leq \mathtt{pred}(a)\}$ and $right = \{w \mid c_{ij}[w].\mathsf{min} \geq \mathtt{succ}(a)\}$. Set *left* is not empty because $e \in c_{ij}[u]$ and $e < a$. Similarly, *right* is not empty because of f. Starting from u, we can find $u' \in left$ and $\mathtt{succ}(u') \in right$ because $u \in left$ and $v \in right$. Since $a \notin c_{ij}[u']$ and $a \notin c_{ij}[\mathtt{succ}(u')]$, row u' is not connected to $\mathtt{succ}(u')$, contradicting the connectedness of c_{ij}. □

Remember that the slant edges in Fig. 1 denote the monotonicity of the ends of consecutive 1's in the matrix, which is formalized in the following property.

Property 4 (Monotonicity) *Consider a row convex and connected constraint c_{ij}. The value $c_{ij}[u].\mathsf{min}$ is* decreasing *for $u \in [c_{ij}.t, c_{ij}.l_\top]$, i.e., $\forall u \in [c_{ij}.t, c_{ij}.l_\top]$, $c_{ij}[u].\mathsf{min} \geq c_{ij}[\mathsf{succ}(u)].\mathsf{min}$, and* increasing *for $u \in [c_{ij}.l_\top, c_{ij}.b]$, i.e., $\forall u \in [c_{ij}.l_\top,\ c_{ij}.b)$, $c_{ij}[u].\mathsf{min} \leq c_{ij}[\mathsf{succ}(u)].\mathsf{min}$. The value $c_{ij}[u].\mathsf{max}$ is increasing for $u \in [c_{ij}.t, c_{ij}.r_\top]$ and decreasing for $u \in [c_{ij}.r_\top, c_{ij}.b]$.*

Proof. We first show that $c_{ij}[u].\mathsf{min}$ is decreasing for $u \in [c_{ij}.t, c_{ij}.l_\top]$. We prove this claim by contradiction. Assume there exists $w \in [c_{ij}.t, c_{ij}.l_\top)$ such that $c_{ij}[w].\mathsf{min} < c_{ij}[\mathsf{succ}(w)].\mathsf{min}$. By definition of $l_\top$, $c_{ij}[w].\mathsf{min} > c_{ij}[c_{ij}.l_\top].\mathsf{min}$. So, $c_{ij}[w].\mathsf{min} \in [c_{ij}[\mathsf{succ}(w)].\mathsf{min}, c_{ij}[c_{ij}.l_\top].\mathsf{min}]$. By Property 3, there exists $u_3 \in [\mathsf{succ}(w), c_{ij}.l_\top]$ such that $c_{ij}[w].\mathsf{min} \in c_{ij}[u_3]$. Since $c_{ij}[w].\mathsf{min} \notin c_{ij}[\mathsf{succ}(w)]$, the 1's in the column $c_{ij}[w].\mathsf{min}$ of c_{ij} is not consecutive, contradicting the row convexity of c_{ij}.

In a similar fashion, we can prove the rest of this property. □

Consider a row convex and connected constraint c_{ij}. Let l_1, l_2, l_3, l_4 be the sorted (ascendingly) values of $c_{ij}.l_\top$, $c_{ij}.r_\top$, $c_{ij}.l_\bot$, and $c_{ij}.r_\bot$. For example, in Fig. 1(b), $l_1 = l_\top = a_2, l_2 = l_\bot = a_2, l_3 = r_\top = a_4, l_4 = r_\bot = a_5$. The matrix of c_{ij} can be divided into the following strips.

(1) *Top strip*: the rows from $c_{ij}.t$ to l_2,
(2) *middle strip*: the rows from l_2 to l_3, and
(3) *bottom strip*: the rows from l_3 to $c_{ij}.b$

By definition of the strips and the row convexity and connectedness of the constraints, the 1's in the top strip can be of only 'b' shape or 'd' shape, the 1's in the middle strip of only '\' shape, 'o' shape, or '/' shape, and the

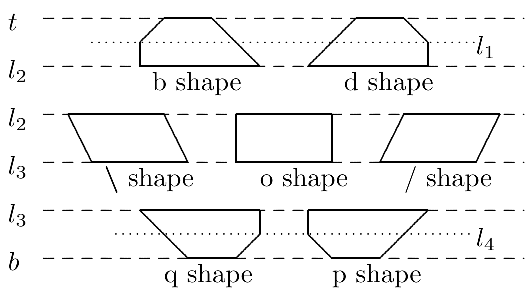


Fig. 2. The possible shapes of the strips of a constraint that is row convex and connected

1's in the bottom strip of only 'q' shape or 'p' shape (see Fig. 2). Note that these shapes are *abstract* shapes and do not have the ordinary *geometrical* properties. The strips and shapes are characterised by the following property.

Property 5 (Shapes) *Top strip: for any $u_1, u_2 \in [c_{ij}.t, l_2]$, $u_1 \leq u_2$ implies $c_{ij}[u_1] \subseteq c_{ij}[u_2]$.*

Middle strip: one of the following cases holds.

- *For any $u_1, u_2 \in [l_2, l_3]$, $u_1 \leq u_2$ implies $c_{ij}[u_1].\mathsf{min} \leq c_{ij}[u_2].\mathsf{min}$ and $c_{ij}[u_1].\mathsf{max} \leq c_{ij}[u_2].\mathsf{max}$, corresponding to '\' shape.*
- *For any $u_1, u_2 \in [l_2, l_3]$, $c_{ij}[u_1] = c_{ij}[u_2]$, corresponding to 'o' shape.*
- *For any $u_1, u_2 \in [l_2, l_3]$, $u_1 \leq u_2$ implies $c_{ij}[u_2].\mathsf{min} \leq c_{ij}[u_1].\mathsf{min}$ and $c_{ij}[u_2].\mathsf{max} \leq c_{ij}[u_1].\mathsf{max}$, corresponding to '/' shape.*

Bottom strip: for any $u_1, u_2 \in [l_3, c_{ij}.b]$, $u_1 \leq u_2$ implies $c_{ij}[u_2] \subseteq c_{ij}[u_1]$.

Proof. In this proof, $t, b, l_\top, l_\perp, r_\top$, and $r_\perp$ without prefix denote those of c_{ij}.

Top strip. By Property 2, Property 4 and definition of $l_\perp$ and $r_\perp$, for all $u \in [t, l_\perp)$, $c_{ij}[u].\mathsf{min} \geq c_{ij}[\mathsf{succ}(u)].\mathsf{min}$, and for all $u \in [t, r_\perp)$, $c_{ij}[u].\mathsf{max} \leq c_{ij}[\mathsf{succ}(u)].\mathsf{max}$. Note that $l_2 \leq l_\perp$ and $l_2 \leq r_\perp$. So, for all $u \in [t, l_2]$, $u \leq l_\perp$ and $u \leq r_\perp$. Therefore, $c_{ij}[u] \subseteq c_{ij}[\mathsf{succ}(u)]$. Hence, $u_1 \leq u_2$ implies $c_{ij}[u_1] \subseteq c_{ij}[u_2]$.

Similarly, we can prove the property of bottom strip.

Middle strip. Since $l_\top \leq l_\perp$ and $r_\top \leq r_\perp$, there are only four cases for the relations among them: 1) $l_\perp < r_\top$, 2) $l_\top > r_\perp$, 3) $r_\top \in [l_\top, l_\perp]$, and 4) $l_\top \in [r_\top, r_\perp]$. In cases 1, rows from l_2 to l_3 form '\' shape, in case 2 they form '/' shape, and in other cases they form 'o' shape.

We only prove case 1 for '\' shape. The rest is similar. In case 1, $l_2 = l_\perp$ and $l_3 = r_\top$. By Property 4, for every $u \in [l_2, l_3)$, $c_{ij}[u].\mathsf{min} \leq c_{ij}[\mathsf{succ}(u)].\mathsf{min}$ and $c_{ij}[u].\mathsf{max} \leq c_{ij}[\mathsf{succ}(u)].\mathsf{max}$. So, for any $u_1, u_2 \in [l_2, l_3]$, $u_1 \leq u_2$ implies $c_{ij}[u_1].\mathsf{min} \leq c_{ij}[u_2].\mathsf{min}$ and $c_{ij}[u_1].\mathsf{max} \leq c_{ij}[u_2].\mathsf{max}$. □

Let c_{ij} be the composition of c_{ix} and c_{xj}. In the rest of this section, we present the properties of c_{ij} in terms of the strips (rows) of c_{ix}.

Both c_{ix} and c_{xj} are assumed to be row convex and connected in Properties 6–11. Notations of t, b, l_1, l_2, l_3, and l_4 without prefix denote those of c_{ix}.

For any $u_1, u_2 \in D_i$ such that $u_1, u_2 \in [t, l_2]$ (top strip of c_{ix}), there is a nice relation between $c_{ij}[u_1].\mathsf{min}$ and $c_{ij}[u_2].\mathsf{min}$ (and between $c_{ij}[u_1].\mathsf{max}$ and $c_{ij}[u_2].\mathsf{max}$).

Property 6 *Consider the top strip of c_{ix}. For all $u_1, u_2 \in [t, l_2]$, $u_1 \leq u_2$ implies $c_{ij}[u_1].\mathsf{min} \geq c_{ij}[u_2].\mathsf{min}$ and $c_{ij}[u_1].\mathsf{max} \leq c_{ij}[u_2].\mathsf{max}$.*

Proof. For any $u_1, u_2 \in [t, l_2]$, by Property 5, $u_1 \leq u_2$ implies $c_{ix}[u_1] \subseteq c_{ix}[u_2]$. Let $c_{ij}[u_1].\mathsf{min} = d$, which implies that $c_{ix}[u_1] \cap c_{jx}[d] \neq \emptyset$. Therefore, $c_{ix}[u_2] \cap c_{jx}[d] \neq \emptyset$. Hence, $c_{ij}[u_2].\mathsf{min} \leq d = c_{ij}[u_1].\mathsf{min}$. Similarly, we have $c_{ij}[u_1].\mathsf{max} \leq c_{ij}[u_2].\mathsf{max}$. □

We have symmetric results for $u_1, u_2 \in [l_3, b]$ (bottom strip of c_{ix}).

Property 7 *Consider the bottom strip of c_{ix}. For all $u_1, u_2 \in [l_3, b]$, $u_1 \leq u_2$ implies $c_{ij}[u_1].\mathsf{min} \leq c_{ij}[u_2].\mathsf{min}$ and $c_{ij}[u_1].\mathsf{max} \geq c_{ij}[u_2].\mathsf{max}$.*

Proof. The property can be proved in a fashion similar to Property 6. □

For $u \in [l_2, l_3]$ (the middle strip of c_{ix} of shape '\' or '/'), the next four properties (Property 8 to 11) show that there exists a value $w \in [l_2, l_3]$ such that the monotonicity of the left ends (right ends respectively) of the rows of c_{ij} between l_2 and l_3 changes from non-increasing for rows above w to non-decreasing for those below w (from non-decreasing for rows above w to non-increasing respectively for those below w respectively). We give the proof of Property 8 in the Appendix. The proofs of the other properties are similar and thus are omitted.

Property 8 *Consider the '\' shape of the middle strip of c_{ix}. Let $w \in D_i$ be the first value in $[l_2, l_3]$ such that*

$$c_{ij}[w].\mathsf{min} = \min_{u \in [l_2, l_3]} \{c_{ij}[u].\mathsf{min}\}.$$

(1) For all $u_1, u_2 \in [l_2, w]$, $u_1 \leq u_2$ implies $c_{ij}[u_1].\mathsf{min} \geq c_{ij}[u_2].\mathsf{min}$, and
(2) for all $u_1, u_2 \in [w, l_3]$, $u_1 \leq u_2$ implies $c_{ij}[u_1].\mathsf{min} \leq c_{ij}[u_2].\mathsf{min}$,

Property 9 *Consider the '\' shape of the middle strip of c_{ix}. Let $w \in D_i$ be* *the first value in* $[l_2, l_3]$ *such that*

$$c_{ij}[w].\mathsf{max} = \max_{u \in [l_2, l_3]} \{c_{ij}[u].\mathsf{max}\}.$$

(1) For all $u_1, u_2 \in [l_2, w]$, $u_1 \le u_2$ *implies* $c_{ij}[u_1].\mathsf{max} \le c_{ij}[u_2].\mathsf{max}$, *and*
(2) for all $u_1, u_2 \in [w, l_3]$, $u_1 \le u_2$ *implies* $c_{ij}[u_1].\mathsf{max} \ge c_{ij}[u_2].\mathsf{max}$,

Property 10 *Consider the '/' shape of the middle strip of* c_{ix}. *Let* $w \in D_i$ *be the first value in* $[l_2, l_3]$ *such that*

$$c_{ij}[w].\mathsf{min} = \min_{u \in [l_2, l_3]} \{c_{ij}[u].\mathsf{min}\}.$$

(1) For all $u_1, u_2 \in [l_2, w]$, $u_1 \le u_2$ *implies* $c_{ij}[u_1].\mathsf{min} \ge c_{ij}[u_2].\mathsf{min}$, *and*
(2) for all $u_1, u_2 \in [w, l_3]$, $u_1 \le u_2$ *implies* $c_{ij}[u_1].\mathsf{min} \le c_{ij}[u_2].\mathsf{min}$,

Property 11 *Consider the '/' shape of the middle strip of* c_{ix}. *Let* $w \in D_i$ *be the first value in* $[l_2, l_3]$ *such that*

$$c_{ij}[w].\mathsf{max} = \max_{u \in [l_2, l_3]} \{c_{ij}[u].\mathsf{max}\}.$$

(1) For all $u_1, u_2 \in [l_2, w]$, $u_1 \le u_2$ *implies* $c_{ij}[u_1].\mathsf{max} \le c_{ij}[u_2].\mathsf{max}$, *and*
(2) for all $u_1, u_2 \in [w, l_3]$, $u_1 \le u_2$ *implies* $c_{ij}[u_1].\mathsf{max} \ge c_{ij}[u_2].\mathsf{max}$,

Finally, we have the property for the mid strip of 'o' shape which follows immediately from Property 5.

Property 12 *Consider the 'o' shape of the middle strip of* c_{ix}. *For all* $u_1, u_2 \in [l_2, l_3]$, $c_{ij}[u_1].\mathsf{min} = c_{ij}[u_2].\mathsf{min}$, $c_{ij}[u_1].\mathsf{max} = c_{ij}[u_2].\mathsf{max}$.

4.4 Fast composition of constraints

The new algorithm to compute $c_{ix} \circ c_{xj}$, listed in Algorithm 5, is based on the following two ideas. 1) We first compute $c_{ij}[u].\mathsf{min}$ for all $u \in D_i$ (line 2–21), which is called *min phase*, and then compute $c_{ij}[u].\mathsf{max}$ for all $u \in D_i$ (line 22–41), which is called *max phase*. 2) In the two phases, the properties of c_{ij} in terms of the shapes and strips of c_{ix} are employed to speed up the computation. We present below the two phases before discussing some other issues related to the composition algorithm.

In the min phase, the algorithm starts from the top strip of c_{ix}. Let $u = c_{ix}.t$. Find $c_{ij}[u].\mathsf{min}$ (line 5) and let it be v. Thanks to the property of the top strip (Property 6), we can find $c_{ij}[u].\mathsf{min}$ for all $u \in [c_{ij}.t, l_2]$ in order by scanning *once* from v down to head of D_j, i.e., searching to the left of v (line 7). The search procedure `searchToLeft` is listed in Algorithm 6 where one needs to

Algorithm 5: Fast algorithm for computing the composition of two row convex and connected constraints

```
fastCompose (in $c_{ix}, c_{xj}$, out $c_{ij}$)
1  let $l_1, \ldots, l_4$ be the ascendingly sorted values of $l_\top, l_\bot, r_\top, r_\bot$ of $c_{ix}$
2  // min phase
3  // process the top strip of $c_{ix}$
4  $u \leftarrow c_{ix}.b$
5  find from head to tail the first $v \in D_j$ such that $c_{ix}[u] \cap c_{jx}[v] \neq \emptyset$
6  $c_{ij}[u]$.min $\leftarrow v$
7  searchToLeft $(c_{ix}, c_{xj}, u, l_2, v, c_{ij})$
8  // process the middle strip
9  if the middle strip is of 'o' shape then
10     $u \leftarrow l_2$
11     while $u \leq l_3$ do $\{c_{ij}[u]$.min $\leftarrow v$, $u \leftarrow$ succ $(u, D_i)\}$
12 if the middle strip is of '\' shape then
13     if $v \neq c_{jx}.t$ and $c_{ix}[u]$.max $< c_{jx}[$pred$(v)]$.min then
14         searchToLeftWrap $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
15     else searchToRight $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
16 if the middle strip is of '/' shape then
17     if $v \neq c_{jx}.t$ and $c_{ix}[u]$.min $> c_{jx}[$pred$(v)]$.max then
18         searchToLeftWrap $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
19     else searchToRight $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
20 // bottom strip
21 searchToRight $(c_{ix}, c_{xj}, u, c_{ij}.b, v, c_{ij})$
22 // max phase
23 // process the top strip
24 $u \leftarrow c_{ix}.b$
25 find the last $v \in D_j$ such that $c_{ix}[u] \cap c_{jx}[v] \neq \emptyset$
26 $c_{ij}[u]$.max $\leftarrow v$
27 searchToRightMax $(c_{ix}, c_{xj}, u, l_2, v, c_{ij})$
28 // process the middle strip
29 if the middle strip is of 'o' shape then
30     $u \leftarrow l_2$
31     while $u \leq l_3$ do $\{c_{ij}[u]$.max $\leftarrow v$, $u \leftarrow$ succ $(u, D_i)\}$
32 if the middle strip is of '\' shape then
33     if $v \neq c_{jx}.b$ and $c_{ix}[u]$.max $< c_{jx}[$succ$(v)]$.min then
34         searchToRightWrap $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
35     else searchToLeftMax $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
36 if the middle strip is of '/' shape then
37     if $v \neq c_{jx}.b$ and $c_{ix}[u]$.min $> c_{jx}[$succ$(v)]$.max then
38         searchToRightWrap $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
39     else searchToLeftMax $(c_{ix}, c_{xj}, u, l_3, v, c_{ij})$
40 // bottom strip
41 searchToLeftMax $(c_{ix}, c_{xj}, u, c_{ij}.b, v, c_{ij})$
```

Algorithm 6: Search methods for computing the composition of two constraints

```
searchToLeft (inout $c_{ix}, c_{xj}, u, l, v, c_{ij}$)
1  // search to the left of $v$
2  while $u \leq l$ do
3      find first $v_1$ from $v$ down to head of $D_j$ such that
       $c_{ix}[u] \cap c_{jx}[\texttt{pred}(v_1)] = \emptyset$
4      $c_{ij}[u]$.min $= v_1$, $v \leftarrow v_1$, $u \leftarrow$ succ ($u$, $D_i$)
searchToLeftWrap (inout $c_{ix}, c_{xj}, u, l, v, c_{ij}$)
5  // search to the left of $v$
6  wrapToRight $\leftarrow$ false
7  while $u \leq l$ do
8      find first $v_1$ from $v$ down to head of $D_j$ such that
       $c_{ix}[u] \cap c_{jx}[\texttt{pred}(v_1)] = \emptyset$ and $c_{ix}[u] \cap c_{jx}[v_1] \neq \emptyset$
9      if $v_1$ does not exist then {wrapToRight $\leftarrow$ true, break }
10     else {$c_{ij}[u]$.min $\leftarrow v_1$, $v \leftarrow v_1$, $u \leftarrow$ succ ($u$, $D_i$)}
11 if wrapToRight is true and $u \leq l$ then
12     searchToRight ($c_{ix}, c_{xj}, u, l$, succ (head, $D_j$), $c_{ij}$)
searchToRight (inout $c_{ix}, c_{xj}, u, l, v, c_{ij}$)
13 // search to the right of $v$
14 while $u \leq l$ do
15     find first $v_1$ from $v$ to tail of $D_j$ such that $c_{ix}[u] \cap c_{jx}[\texttt{pred}(v_1)] = \emptyset$
       and $c_{ix}[u] \cap c_{jx}[v_1] \neq \emptyset$
16     $c_{ij}[u]$.min $\leftarrow v_1$, $v \leftarrow v_1$, $u \leftarrow$ succ ($u$, $D_i$)
searchToRightMax (inout $c_{ix}, c_{xj}, u, l, v, c_{ij}$)
17 // search to the right of $v$
18 while $u \leq l$ do
19     find last $v_1$ from $v$ to tail of $D_j$ such that $c_{ix}[u] \cap c_{jx}[v_1] \neq \emptyset$
20     $c_{ij}[u]$.max $\leftarrow v_1$, $v \leftarrow v_1$, $u \leftarrow$ succ ($u$, $D_i$)
searchToRightWrap (inout$c_{ix}, c_{xj}, u, l, v, c_{ij}$)
21 // search to the right of $v$
22 wrapToLeft $\leftarrow$ false
23 while $u \leq l$ do
24     find last $v_1$ from $v$ to tail of $D_j$ such that $c_{ix}[u] \cap c_{jx}[\texttt{succ}(v_1)] = \emptyset$
       and $c_{ix}[u] \cap c_{jx}[v_1] \neq \emptyset$
25     if $v_1$ does not exist then {wrapToLeft $\leftarrow$ true, break }
26     else {$c_{ij}[u]$.max $= v_1$, $v \leftarrow v_1$, $u \leftarrow$ succ ($u$, $D_i$) }
27 if wrapToLeft is true then
28     searchToLeftMax ($c_{ix}, c_{xj}, u, l$, pred (tail, $D_j$), $c_{ij}$)
searchToLeftMax (inout $c_{ix}, c_{xj}, u, l, v, c_{ij}$)
29 // search to the left of $v$
30 while $u \leq l$ do
31     find first $v_1$ from $v$ down to head of $D_j$ such that $c_{ix}[u] \cap c_{jx}[v_1] \neq \emptyset$
32     $c_{ij}[u]$.max $\leftarrow v_1$, $v \leftarrow v_1$,$u \leftarrow$ succ ($u$, $D_i$)
```

note that v is replaced by v_1 in line 4. Similarly, by Property 7, we can process the bottom strip by searching to the right of $v \in D_j$ (line 21). For the middle strip, we have three cases for the three shapes in accordance with Property 5. For the 'o' shape, lines 9–11 are quite straightforward, by Property 12. For the '\' shape (line 12–15), if v is not the first column of c_{xj} (i.e., not the first value of D_j) and $c_{ix}[u]$ is "above" the interval of the column before v of c_{xj} (line 13), i.e., $c_{ix}[u].\mathsf{max} < c_{jx}[\mathsf{pred}(v)].\mathsf{min}$, from case b) in the proof of Property 8, we need to search to the left of v until the w defined in Property 8 is found. After we hit the head of D_j and no support is found, we need to search to the right until tail (line 14) if necessary because $c_{ij}[u].\mathsf{min}$ is increasing for $u \in [w, l_3]$ according to the second claim of Property 8. This process is implemented as `searchToLeftWrap` (line 5–12 of Algorithm 6). However, if v is the first column of c_{xj} or $c_{ix}[u]$ is "below" the interval of the column before v of c_{xj} (line 19), i.e., $c_{ix}[u].\mathsf{min} > c_{jx}[\mathsf{pred}(v)].\mathsf{max}$, there is no need to search to left of v in terms of the proof of case a) for Property 8. In other words, $v = w$. Therefore, we only need to search to the right of v by Property 8, which is implemented by the procedure `searchToRight` (line 13 - 16). The process for the '/' shape is similar to that for the '\' shape with some "symmetrical" differences (line 17). The max phase is similar.

To simplify the analysis of the complexity of the algorithm, we assume the quantities of $l_1, \ldots, l_4$ (line 1), b and t of a constraint c_{ix} are computed whenever c_{ix} is composed with another constraint. Recall that a constraint is represented as intervals. By scanning once $c_{ix}[u]$ ($c_{ix}[u].\mathsf{min}$ and $c_{ix}[u].\mathsf{max}$) with u changing from the minimum value of D_u to the maximum, we can find t, b, $l_\top, l_\perp, r_\top$ and $r_\perp$, which takes $O(d)$ steps (due to their definitions and the monotonicity property of c_{ix}). By sorting $l_\top, l_\perp, r_\top$ and $r_\perp$, we obtain $l_1, \ldots, l_4$ in constant time. In the implementation of the algorithm, one may use an incremental way to maintain these quantities.

The shape of the middle strip of c_{ix} (line 9, 12, 16) can be decided as follows. If $c_{ix}[l_2].\mathsf{min} = c_{ix}[l_3].\mathsf{min}$ and $c_{ix}[l_2].\mathsf{max} = c_{ix}[l_3].\mathsf{max}$, the strip is of 'o' shape. Otherwise, it is of '\' shape if $c_{ix}[l_2].\mathsf{min} < c_{ix}[l_3].\mathsf{min}$, or else of '/' shape.

Proposition 3 *The algorithm `fastCompose` is correct and composes two constraints in time complexity of $O(d)$ with working space complexity of $O(1)$.*

Proof. Consider the composition, c_{ij}, of c_{ix} and c_{xj}. By Property 6 and 7, the algorithm processes the top and bottom strips correctly. For each phase, it scans at most $O(d)$ elements of D_j and thus has complexity of $O(d)$.

For the middle strips, we consider only '\' shape for the min phase. The other cases are similar. Property 8 shows that $c_{ij}[u]$ increases for $u \in [l_2, w]$ and decreases for $u \in [w, l_3]$ where w is defined in Property 8. Let $c_{ij}[l_2].\mathsf{min} = v$. If v is the first column of c_{xj}, $v = w$. Otherwise, if $c_{ix}[l_2].\mathsf{max} < c_{jx}[\mathsf{pred}(v)].\mathsf{min}$,

$v = w$ as shown in the proof of case a) for Property 8. In either case above, the algorithm does not need to search to the left of v, and thus `searchToRight` is sufficient in terms of the second claim of Property 8. If $c_{ix}[l_2]$.min $> c_{jx}[\mathtt{pred}(v)]$.max, `searchToLeftWrap` finds w first (and find $c_{ij}[u]$.min for all $u \in [l_2, w]$ in the process) by searching to the left of v, which is correct by the first claim of Property 8. After w is found, it searches to the right for the values between w and l_3, which is correct in terms of the second claim of Property 8. Property 12 and 10 assures the correctness of the min phase for 'o' and '/' shape respectively. In the worst case, `searchToLeftWrap` scans the values of D_j twice. So, its complexity is $O(d)$. Hence, the complexity of the min phase is $O(d)$. Similarly, Property 9, 12, and 11 guarantee the correctness of the max phase of the algorithm. The time complexity of the max phase is also $O(d)$.

The algorithm only needs constant space for data $t, b, l_\top, l_\perp, r_\top, r_\perp$ and l_1 to l_4. So, its working space complexity is $O(1)$. □

5 CSP's with sparse constraint graphs

The efficiency of `eliminate` is affected by the ordering of the variables to be eliminated. Consider a constraint graph with variables $\{1, 2, 3, 4, 5\}$ that is shown in the top left corner of Fig. 3. In the first row, we choose to eliminate 1 first and then 3. In this process, no constraints are composed. However, if we first eliminate 2 and then 4 as shown in the second row, `eliminate` needs to make 3 compositions in eliminating each of variable 2 and 4.

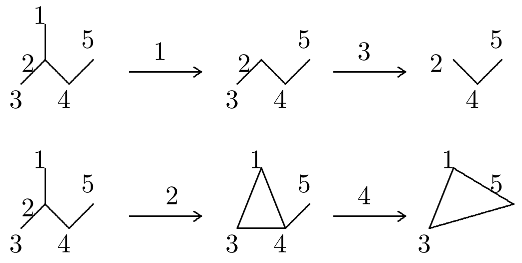


Fig. 3. Example on elimination variable ordering

The topology of a constraint graph can be employed to find a good variable elimination ordering. Here we consider triangulated graphs. An undirected graph G is *triangulated* if for every cycle of length 4 or more in G, there exists two non-consecutive vertices of the cycle such that there is an edge between them in G. Given a vertex $x \in G$, $\mathrm{N}(x)$ denotes neighbors of x: $\{y \mid \{x, y\}$ is an edge of $G\}$. A vertex x is *simplicial* if the subgraph of G induced by $\mathrm{N}(x)$ is complete. A nice property of triangulated graphs is that there is a simplicial vertex for each triangulated graph and a triangulated graph remains triangulated after a simplicial vertex and its incident edges are removed from the graph. A *perfect vertex elimination order* of a graph $G=(\{x_1, x_2, \ldots, x_n\}, E)$ is an ordering $\langle y_1, y_2, \ldots, y_n\rangle$ of the vertices of G such that for $1 \leq i \leq n-1$, y_i is a simplicial vertex of the subgraph of G induced by $\{y_i, y_{i+1}, \ldots, y_n\}$.

Given a perfect elimination order $\langle y_1, y_2, \ldots, y_n\rangle$ of a graph G, the *elimination degree* of $y_i (1 \leq i \leq n)$, denoted by σ_i, is the degree of y_i in the subgraph of G that is induced by $\{y_i, y_{i+1}, \ldots, y_n\}$. We use σ to denote the maximum elimination degree of the vertices of a perfect elimination order.

It is well known that, for a graph G that is not complete, it can be triangulated in time $O(n(e+f))$ where f is the number of edges added to the original graph and e the number of edges of G [1]. A perfect elimination order can be found in $O(n+e)$.

For CSP problems whose constraint graph is triangulated, the elimination algorithm has a better time complexity bound.

Theorem 3 *Consider a CSP problem P whose constraint graph G is triangulated. The procedure* `eliminate` *equipped with* `fastCompose` *has a time complexity of $O(n\sigma^2 d + ed^2)$ and space complexity of $O(nd)$.*

Proof. Let $\langle y_1, y_2, \ldots, y_n\rangle$ be a perfect elimination for G. Clearly, to eliminate y_i, `eliminate` has to compose σ_i^2 constraints. Since $n-1$ variables are eliminated by `eliminate`, its complexity is $O(n\sigma^2 d + ed^2)$ where $O(ed^2)$ is due to the `removeValues`. The space complexity is also due to `removeValues`. □

6 Experimental results

We have carried out experiments to evaluate empirically our new algorithm. As in [4], we use random connected row convex problems. Four parameters are used to generate a random connected row convex problem instance: n – the number of variables, d – the size of the domains (all variables have the same domain with values from 1 to d), e – the number of constraints, and l – the looseness of the constraints, i.e., the ratio of the number of allowed tuples over d^2.

For comparison purpose, we select the existing algorithm PC-CRC proposed by Deville et al. [4]. The reason we did not use the path consistency on triangulated graphs (PPC) is as follows. In terms of the (major) operations of composition and intersection, the initial phase of PPC uses exactly the same number of operations needed by the whole of our algorithm. It involves more operations in its propagation phase. Therefore, the practical difference between PPC and our algorithm lies in the implementation of the detailed operations on CRC constraints. Bliek and Sam-Haroud [1] recommend to use those proposed in PC-CRC algorithm [4] that is an efficient version of path consistency algorithm specialized for CRC constraints. In addition, we are able to obtain an efficient implementation from the authors of PC-CRC algorithm.

Both our program for the elimination algorithm and Deville et al's program for PC-CRC are written in C++. The programs are run on a DELL PowerEdge 1850 (two 3.6GHz Intel Xeon CPUs) with Linux.

Our algorithm can deal with problem instances whose constraint graphs are not complete. Due to the nature of path consistency, PC-CRC program requires the graphs to be complete. When the constraint graph of a problem instance is not complete, we create its *completed version* by adding a universal constraint between variables on which there is no constraint. In our experiments, we feed the original problem instances to our program while their completed versions to PC-CRC program.

The difference of worst time complexity of our algorithm and PC-CRC lies in the size of the domain and the sparsity of the problem instances. The performance of the algorithms is shown in Fig. 4 with varying sparsity, and Fig. 5 with varying domain size.

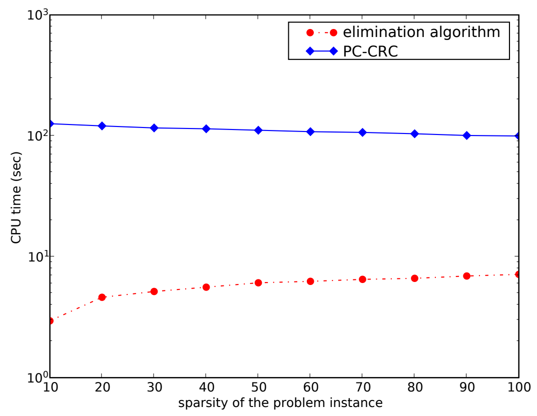


Fig. 4. The other parameters of the problem instances in this diagram are $n = 100, d = 100, l = 30\%$. The values on X axis are the percentage of the number of constraints of an instance over the number of all possible constraints over the variables of the instance.

As expected, the more sparse a problem is, the more speedup is gained by the elimination algorithm. One observation is that the PC-CRC program becomes faster as the instance becomes less sparse. By [4], the CPU time of PC-CRC program increases as the looseness of the constraints increases. The more sparse a problem is, the more universal constraints (whose looseness is 100%) we have in its completed version. This explains that the performance of PC-CRC improves as the number of constraints in a problem is increased (with other parameters fixed). Note that when the problem instances' constraint graphs become complete, completion of the graphs as required by the PC algorithms does not introduce any new constraints. Therefore, a PPC algorithm equipped with PC-CRC methods has the same performance as the PC-CRC algorithm in this case. However, our algorithm still has a significant performance margin (about 10 times) over PC-CRC.

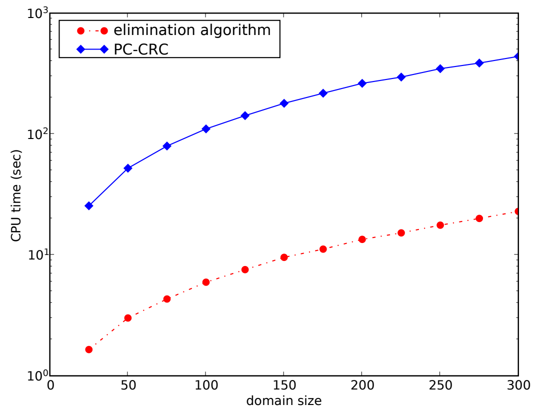


Fig. 5. The other parameters of the problem instances in this diagram are $n = 100, e = 2475$ (50% of all possible constraints), $l = 30\%$.

For problems with varying domain sizes, the new algorithm is more than ten times faster than PC-CRC. However, the speedup increases only slightly as d increases. This can be explained by the observation in [4] that CPU time for PC-CRC is linear to the domain size when n is fixed.

We also did an exhaustive experiment on varying all the parameters: n from 30 to 150 with step 30, d from 20 to 100 with step 20, e from 10 to 90% with step 20%, l from 10 to 50% with step 10%. The scatter graph of the performance data is shown in Fig. 6.

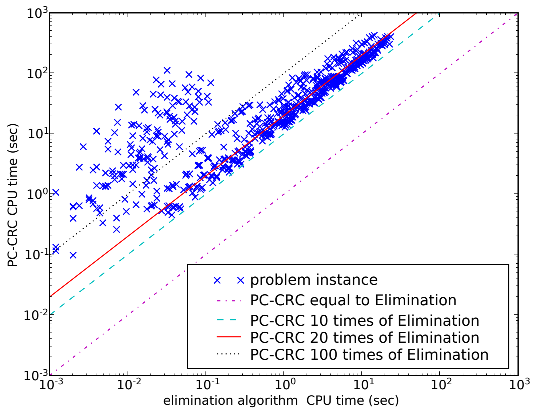


Fig. 6. Results of an exhaustive experiment.

For most of the problem instances, the performance improvement of our algorithm is around 20 times. For the improvement around hundreds of times, the problem instances are usually arc inconsistent, i.e., some domain becomes empty during arc consistency enforcing. In this case, the huge time saving results mainly from the fact that the elimination algorithm is not invoked.

In the experiments above, our implementation uses a perfect elimination ordering to eliminate the variables. We have also carried out an experiment on the effectiveness of the perfect elimination ordering against a lexicographical ordering. The result is shown in Fig 7.

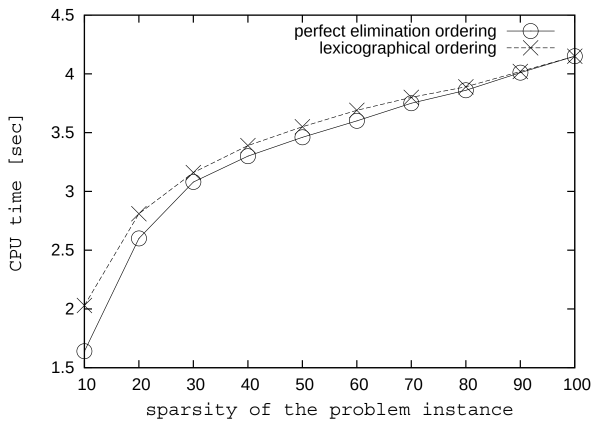


Fig. 7. Perfect elimination ordering vs lexicographical ordering on random problem instances with $n = 100, d = 100, l = 30\%$ and density of the constraint graphs changing from 10% to 100%.

From the experimental result, although perfect elimination ordering is better than lexicographical ordering on the sparse graphs, the difference is not very significant. In summary, the better performance of our algorithm results from a combination of factors: faster composition algorithm, less number of constraints are involved (compared with path consistency algorithms) and good variable orderings.

7 Related work and conclusion

We have proposed a simple elimination algorithm to solve CRC constraints. Thanks to this algorithm, we are able to focus on developing a fast algorithm to compose constraints that are row convex and connected. We show that the composition can be done in $O(d)$ time, which benefits from a new understanding of the properties of row convex and connected constraints. In addition to the simplicity, our deterministic algorithm has some other advantages over the existing ones. The working space complexity $O(nd)$ of our algorithm is the best among existing deterministic or randomized algorithms of which the best is $O(ed)$. However, when a graph is sparse, in contrast to the randomized algorithms, our algorithm needs space $O(fd)$ to store newly created constraints where f is the number of edges needed to triangulate the sparse graph.

For problems with dense constraint graphs ($e = \Theta(n^2)$), our algorithm ($O(n^3d+ed^2)$) is better than the best ($O(n^3d^2)$) of the existing algorithms.

For problems with sparse constraint graphs, the traditional path consistency method [4] does not make use of the sparsity. Bliek and Sam-Haroud [1] proposed to triangulate the constraint graph and introduced path consistency on triangulated graphs. For CRC constraints, their (deterministic) algorithm achieves path consistency on the triangulated graph with time complexity of $O(\delta e'd^2)$ and space complexity of $O(\delta e'd)$ where δ is the maximum degree of the triangulated graph and e' the number of constraints in the triangulated graph. The randomized algorithm by Kumar [5] has a time complexity of $O(\gamma n^2d^2)$ where γ is the maximum degree of the original constraint graph. Our algorithm can achieve $O(n\sigma^2d + e'd^2)$ where σ is the maximum elimination degree of the triangulated graph. Since $\sigma \leq \delta, \gamma \leq \delta, \sigma^2 \leq e' \leq n^2$ (σ and γ are not comparable), our algorithm is still favorable in comparison with the others.

Our extensive experiments on random problems of CRC constraints also show that the new algorithm has a clear performance margin over the existing deterministic algorithms.

It is worth mentioning that, in addition to "determinism", a deterministic algorithm has a great efficiency advantage over randomized algorithms when more than one solution is needed.

Dechter has proposed variable elimination (bucket elimination) to solve general CSP problems [2]. To eliminate a variable, one needs to *join* all the constraints on this variable, which may lead to exponential time and space complexity. We propose a variable elimination method (not through join) for connected row convex CSPs, which takes only polynomial time.

We also notice the work by Xu and Choueiry [9] who proposed an efficient algorithm, based on [1], to solve simple temporal problems, a special class of CRC constraints. The ideas and algorithms presented here may be used to produce more efficient algorithms for simple temporal algorithms and other problems with (connected) row convex constraints.

We point out that we introduce `removeValues` just for simplifying the design and analysis of the composition algorithms. It might be possible to design a refined propagation mechanism and/or composition algorithms to discard the ed^2 component from the time complexity and decrease the working space complexity of the elimination algorithm to $O(n)$.

Acknowledgment

The research leading to the results in this paper was funded in part by NASA-NNG05GP48G. We thank Yves Deville for providing us with the source code for the random problem generator and PC-CRC algorithm and many discussions on these programs and experiments. Part of the work reported here has been published in [10].

References

[1] Christian Bliek and Djamila Sam-Haroud. Path consistency on triangulated constraint graphs. In *IJCAI-99*, pages 456–461, Stockholm, Sweden, 1999. IJCAI Inc.

[2] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[3] R. Dechter. *Constraint Processing*. Morgan Kaufmann, San Francisco, CA, 2003.

[4] Yves Deville, Olivier Barette, and Pascal Van Hentenryck. Constraint satisfaction over connected row-convex constraints. *Artif. Intell.*, 109(1-2):243–271, 1999.

[5] T. K. Satish Kumar. Simple randomized algorithms for tractable row and tree convex constraints. In *Proceedings of National Conference on Artificial Intelligence 2006*, page to appear, 2006.

[6] Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):118–126, 1977.

[7] Ugo Montanari. Networks of constraints: fundamental properties and applications. *Information Science*, 7(2):95–132, 1974.

[8] Peter van Beek and Rina Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of The ACM*, 42(3):543–561, 1995.

[9] Lin Xu and Berthe Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In *Proceedings of 10th International Symposium on Temporal Representation and Reasoning / 4th International Conference on Temporal Logic (TIME-ICTL 2003), 8-10 July 2003, Cairns, Queensland, Australia*, pages 212–, 2003.

[10] Yuanlin Zhang. Fast algorithm for connected row convex constraints. In *IJCAI*, pages 192–197, 2007.

[11] Yuanlin Zhang and Roland H. C. Yap. Making AC-3 an optimal algorithm. In *Proceedings of International Joint Conference on Artificial Intelligence 2001*, pages 316–321, Seattle, 2001. IJCAI Inc.

[12] Yuanlin Zhang and Roland H. C. Yap. Consistency and set intersection. In *Proceedings of International Joint Conference on Artificial Intelligence 2003*, pages 263–268, Acapulco, Mexico, 2003. IJCAI Inc.

Appendix

Here is a proof of Property 8.

Proof. By Property 1, for any $u \in [l_2, l_3]$, $c_{ij}[u]$ is not empty. Let $c_{ij}[l_2].\mathsf{min} = a$ and $c_{ij}[w].\mathsf{min} = f$.

We now prove 1).

If a is the first value of D_j, the proof of 1) is trivial because $w = l_2$. So, we can assume $\mathtt{pred}(a)$ exists. Similarly, we can assume $l_2 < w$.

Since $c_{ij}[l_2].\mathsf{min} = a$, $c_{ix}[l_2] \cap c_{jx}[\mathtt{pred}(a)] = \emptyset$. There are two cases: a) $c_{ix}[l_2].\mathsf{min} > c_{jx}[\mathtt{pred}(a)].\mathsf{max}$ or b) $c_{ix}[l_2].\mathsf{max} < c_{jx}[\mathtt{pred}(a)].\mathsf{min}$.

For case a), we will show $w = l_2$ and thus the proof of 1) is trivial.

We show first that for all $a_1 \in [c_{jx}.t, a)$, $c_{jx}[a_1].\mathsf{max} < c_{ix}[l_2].\mathsf{min}$. Since $c_{ij}[l_2].\mathsf{min} = a$, $c_{jx}[a_1].\mathsf{max} < c_{ix}[l_2].\mathsf{min}$ or $c_{jx}[a_1].\mathsf{min} > c_{ix}[l_2].\mathsf{max}$. Assume there exists $a_2 \in [c_{jx}.t, a)$ such that $c_{jx}[a_2].\mathsf{min} > c_{ix}[l_2].\mathsf{max}$. Without loss of generality, assume a_2 is the maximal value of D_j satisfying the conditions. So, $c_{jx}[\mathtt{succ}(a_2)].\mathsf{max} < c_{ix}[l_2].\mathsf{min}$, implying $c_{jx}[a_2].\mathsf{min} > c_{jx}[\mathtt{succ}(a_2)].\mathsf{max}$. Therefore, $c_{jx}[a_2]$ and $c_{jx}[\mathtt{succ}(a_2)]$ are not connected, contradicting that c_{xj} is connected.

We next show that for all $u \in [l_2, l_3]$ and $a_1 \in [c_{jx}.t, a)$, $c_{ix}[u] \cap c_{jx}[a_1] = \emptyset$. We have $c_{ix}[u].\mathsf{min} \geq c_{ix}[l_2].\mathsf{min}$ because of the '\' shape. Since $c_{ix}[l_2].\mathsf{min} > c_{jx}[a_1].\mathsf{max}$, $c_{ix}[u] \cap c_{jx}[a_1] = \emptyset$.

Therefore, $w = l_2$.

Consider case b), i.e.,

(1) $c_{ix}[l_2].\mathsf{max} < c_{jx}[\mathtt{pred}(a)].\mathsf{min}$.

We first show that $a \leq c_{jx}.l_\top$. Assume $a > c_{jx}.l_\top$. Since c_{jx} is row convex and connected, by monotonicity Property 4, $c_{jx}[\mathtt{pred}(a)].\mathsf{min} \leq c_{jx}[a].\mathsf{min}$. By (1), we have $c_{ix}[l_2].\mathsf{max} < c_{jx}[a].\mathsf{min}$ and thus $c_{ix}[l_2] \cap c_{jx}[a] = \emptyset$, which contradicts that $c_{ij}[l_2].\mathsf{min} = a$.

Since $a \leq c_{jx}.l_\top$, $\forall u \in [c_{jx}.b, a)$, by Property 4,

(2) $c_{jx}[\mathsf{succ}(u)].\mathsf{min} \leq c_{jx}[u].\mathsf{min}$.

Consider $u_1, u_2 \in [l_2, w]$ such that $u_1 \leq u_2$. Let $c_{ij}[u_1].\mathsf{min} = d_1$ and $c_{ij}[u_2].\mathsf{min} = d_2$.

We next show that $d_1 \in [f, a]$. Construct $g' = \min(c_{ix}[l_2] \cap c_{jx}[a])$ and $g'' = \max(c_{ix}[w] \cap c_{jx}[f])$.

To help ease the memory of the many involved quantities and their relationships, we give a typical situation on these quantities in Fig. 8.

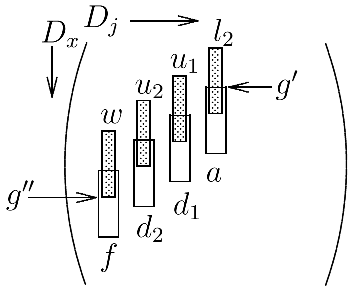


Fig. 8. The diagram represents c_{xj} with rows and columns indexed by D_x and D_y respectively. The shaded bars together with the letters on top of them denote the images of the corresponding letters with respect to c_{ix}. For example, the shaded bar with w denotes $c_{ix}[w]$. Note that $c_{ix}[w] \subseteq D_x$. The white bars with letters below them denote the images of of the corresponding letters with respect to c_{jx}. For example, the white bar with letter f denotes $c_{jx}[f]$. Note that $c_{jx}[f] \subseteq D_x$.

By definition of g' and g'', we have

(3) $g' \leq c_{ix}[l_2].\mathsf{min}$ *or* $g' \leq c_{jx}[a].\mathsf{min}$, and

(4) $g'' \geq c_{ix}[w].\mathsf{min}$ *and* $g'' \geq c_{jx}[f].\mathsf{min}$.

By the property of '\' shape,

(5) $c_{ix}[l_2].\mathsf{min} \leq c_{ix}[w].\mathsf{min}$.

By (2),

(6) $c_{jx}[a].\mathsf{min} \leq c_{jx}[f].\mathsf{min}$.

(3), (4), (5) and (6) imply $g'' \geq g'$.

Since $u_1 \leq w$, $c_{ix}[u_1].\mathsf{min} \leq c_{ix}[w].\mathsf{min}$. So, by (4),

(7) $g'' \geq c_{ix}[u_1].\mathsf{min}$.

By (3), $g' \leq c_{ix}[l_2].\mathsf{max}$. Hence, $c_{ix}[l_2].\mathsf{max} \leq c_{ix}[u_1].\mathsf{max}$ implies

(8) $g' \leq c_{ix}[u_1].\mathsf{max}$.

(7) and (8) imply that $c_{ix}[u_1] \cap [g', g''] \neq \emptyset$.

Take $u \in D_x$ such that $u \in c_{ix}[u_1] \cap [g', g'']$. Note that $g' \in c_{jx}[a]$ and $g'' \in c_{jx}[f]$. By Property 3, there exists $e \in D_j$ such that $e \in [f, a]$ and $u \in c_{jx}[e]$. Since $u \in c_{ix}[u_1]$, $c_{ix}[u_1] \cap c_{jx}[e] \neq \emptyset$. Therefore, $c_{ij}[u_1].\mathsf{min} \leq e \leq a$. By definition of w, $c_{ij}[u_1].\mathsf{min} \geq f$. Hence, $d_1 \in [f, a]$.

Since $d_1 \in [f, a]$, by replacing u_1 by u_2, l_2 by u_1, d_1 by d_2 and a by d_1 in the proof above for $d_1 \in [f, a]$, we can show that $d_2 \in [f, d_1]$. Therefore, $c_{ij}[u_1].\mathsf{min} = d_1 \geq d_2 = c_{ij}[u_2].\mathsf{min}$.

We next prove 2).

Let $w_1 \in [w, l_3]$ be the last row of c_{ix} such that $c_{ij}[w_1].\mathsf{min} = c_{ij}[w].\mathsf{min}$. We will show for all $w_2 \in [w, w_1]$, $c_{ij}[w_2].\mathsf{min} = c_{ij}[w].\mathsf{min}$. Let $g' = \min(c_{ix}[w] \cap c_{jx}[f])$ and $g'' = \max(c_{ix}[w_1] \cap c_{jx}[f])$. Similarly to the proof above, we can show $g' \leq g''$ and $c_{ix}[w_2] \cap [g', g''] \neq \emptyset$. Therefore, there exists $u \in c_{ix}[w_2]$ and $e \in [f, f]$ such that $c_{ix}[w_2] \cap c_{jx}[e] \neq \emptyset$, which, together with the definition of w, implies that $c_{ij}[w_2].\mathsf{min} = c_{ij}[w].\mathsf{min}$.

When $l_3 = w_1$, the proof of 2) is trivial. Let $w_1 < l_3$.

By definition of w_1, $c_{ij}[\mathsf{succ}(w_1)].\mathsf{min} > f$, i.e., $c_{ij}[\mathsf{succ}(w_1)].\mathsf{min} > c_{ij}[w_1].\mathsf{min}$.

Consider any $u_1, u_2 \in (w_1, l_3]$ and $u_1 \leq u_2$. Let $c_{ij}[u_2].\mathsf{min} = h$.

Since w_1 is the last row such that $c_{ij}[w_1].\mathsf{min} = c_{ij}[w].\mathsf{min}$, the definition of w implies that $h > f$.

By definition of w_1, $c_{ix}[\mathsf{succ}(w_1)] \cap c_{jx}[f] = \emptyset$. So, we have either $c_{ix}[\mathsf{succ}(w_1)].\mathsf{min} > c_{jx}[f].\mathsf{max}$ or $c_{ix}[\mathsf{succ}(w_1)].\mathsf{max} < c_{jx}[f].\mathsf{min}$. Consider the latter case. Since $c_{ix}[\mathsf{succ}(w_1)].\mathsf{max} \geq c_{ix}[w_1].\mathsf{max}$, $c_{ix}[w_1].\mathsf{max} < c_{jx}[f].\mathsf{min}$, which contradicts $c_{ix}[w_1] \cap c_{jx}[f] \neq \emptyset$. Therefore, the former case holds: $c_{ix}[\mathsf{succ}(w_1)].\mathsf{min} > c_{jx}[f].\mathsf{max}$. Since $c_{ix}[u_2].\mathsf{min} \geq c_{ix}[\mathsf{succ}(w_1)].\mathsf{min}$,

(9) $c_{ix}[u_2].\mathsf{min} > c_{jx}[f].\mathsf{max}$.

Let $g' = \min(c_{ix}[w_1] \cap c_{jx}[f])$ and $g'' = \max(c_{ix}[u_2] \cap c_{jx}[h])$. Since $g'' \geq c_{ix}[u_2].\mathsf{min}$, (9) implies that $g'' > c_{jx}[f].\mathsf{max}$. By definition of g', $g' \leq c_{jx}[f].\mathsf{max}$. Therefore, $g' < g''$. We have $c_{ix}[u_1].\mathsf{min} \leq c_{ix}[u_2].\mathsf{min} \leq g''$, and $c_{ix}[u_1].\mathsf{max} \geq c_{ix}[u_1].\mathsf{min} \geq c_{ix}[\mathsf{succ}(w_1)].\mathsf{min} > c_{ix}[f].\mathsf{max} > g'$. Therefore, $c_{ix}[u_1] \cap [g', g''] \neq \emptyset$.

Take $v \in D_x$ such that $v \in c_{ix}[u_1] \cap [g', g'']$. Note that $g' \in c_{jx}[f]$ and $g'' \in c_{jx}[h]$. By Property 3, there exists $e \in D_j$ such that $e \in [f, h]$ and $v \in c_{jx}[e]$. So,

$c_{ix}[u_1] \cap c_{jx}[e] \neq \emptyset$. Therefore, $c_{ij}[u_1].\mathsf{min} \leq e \leq h = c_{ij}[u_2].\mathsf{min}$. $\square$