Fast Algorithm for Connected Row Convex Constraints[*]

Yuanlin Zhang

Texas Tech University

Computer Science Department

yzhang@cs.ttu.edu

Abstract

Many interesting tractable problems are identified under the model of Constraint Satisfaction Problems. These problems are usually solved by forcing a certain level of local consistency. In this paper, for the class of connected row convex constraints, we propose a novel algorithm which is based on the ideas of variable elimination and efficient composition of row convex and connected constraints. Compared with the existing work including randomized algorithms, the new algorithm has better worst case time and space complexity.

1 Introduction

Constraint satisfaction techniques have found wide applications in combinatorial optimisation, scheduling, configuration, and many other areas. 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.

An interesting class of row convex constraints was identified by van Beek and Dechter (1995). It is known that if a problem of row convex constraints is path consistent, it is tractable to find a solution for this problem. However, when the problem is not path consistent, path consistency enforcing might not lead to global consistency due to the possibility that the row convexity of some constraints is destroyed. Deville et al. (1997) restrict row convexity to connected row convexity (CRC). In fact, the scene labeling problem and constraint based grammar examples given in [van Beek and Dechter, 1995] are CRC 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 $O(n^3 d^2)$ with space complexity of $O(n^2 d)$ where n is the number of variables, d the maximum domain size. Recently, Kumar (2006) has proposed a randomized algorithm for CRC constraints with time complexity of $O(\gamma n^2 d^2)$ 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, making use of the row convexity and connectedness of constraints, we propose a new algorithm to solve CRC constraints with time complexity of $O(n\sigma^2 d + ed^2)$ where σ is the *elimination degree* 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.

A key operation in the elimination algorithm is to compose two constraints. The properties of connectedness and row convexity of the constraints make it possible to get a fast composition algorithm with time complexity of $O(d)$.

In this paper, we present the elimination algorithm after the preliminaries on CRC constraints. The methods to compute composition of row convex and connected constraints are then proposed. We examine the elimination algorithm on problems with sparse constraint graphs before we conclude the paper.

2 Preliminaries

A *binary constraint satisfaction problem (CSP)* 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. When necessary, we introduce head and tail for each variable domain such that head (tail respectively) is smaller (larger respectively) than any other value of the domain. Functions $\mathrm{succ}(u, D_i)$ $(u \in D_i \cup \{\mathrm{head}\})$ and $\mathrm{pred}(u, D_i)$ $(u \in D_i \cup \{\mathrm{tail}\})$ denote respectively the successor and predecessor of u in the current domain $D_i \cup \{\mathrm{head, tail}\}$. 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}. Clearly $c_{ij}[\text{head}] = c_{ij}[\text{tail}] = \emptyset$. Standard operations of intersection and composition can be applied to constraints. The composition of c_{ix} and c_{xj} is denoted by $c_{xj} \circ c_{ix}$. It is convenient to use a Boolean matrix to represent a constraint c_{ij}. The 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 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 [Bliek and Sam-Haroud, 1999].

A constraint c_{ij} is *row convex* if there exists a total ordering on D_j such that the 1's are consecutive in each row of the matrix of c_{ij}. The *reduced form* of a constraint c_{ij}, denoted by c_{ij}^*, is obtained by removing from D_i (and D_j respectively) those values whose image with respect to c_{ij} (c_{ji} respectively) is 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 first and v is the last 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 (and columns respectively) of c_{ij} are not empty and satisfy $[a, b] \cap [\, \texttt{pred}(a'), \, b'\,] \neq \emptyset$ or $[a, b] \cap [a', \, \texttt{succ}(b')] \neq \emptyset$. Note that, for our purposes, the definition of connectedness here is *stronger* than that by Deville et al. (1997). If a constraint is row convex and connected, it is arc consistent. 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 [Deville *et al.*, 1997].

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 $\cup_{i=1..k} E_i$ where 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 [van Beek and Dechter, 1995; Zhang and Yap, 2003].

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 $C' = C \cup \{c_{xj} \circ c_{ix} \cap c_{ij} \mid c_{jx}, c_{ix} \in R_x \text{ and } i \neq j\} - R_x$. *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 $c'_{ij} = c_{xj} \circ c_{ix} \cap c_{ij}$, 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. $\square$

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' ← C, C'' ← ∅, L ← V
7  while L ≠ ∅ do
8      select and remove a variable x from L
9      C'_x ← {c_yx | c_yx ∈ C'}
10     foreach c_ix, c_jx ∈ C'_x where i < j do
11         c'_ij ← c_xj ∘ c_ix
12         if c_ij ∈ C' then c'_ij ← c'_ij ∩ c_ij
13         C' ← (C' - {c_ij}) ∪ {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' ← C' - C'_x
19     C'' ← C'' ∪ C'_x
20     s.push (x)
21 C ← 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 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.

On top of the elimination algorithm, it is rather straightforward to design an algorithm to find the solutions of a problem of CRC constraints (Algorithm 2). L (line 5) represents the assigned variables. C_x in line 8 contains 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 ← ∅
6   while not s.empty () do
7       x ← s.pop ()
8       C_x ← {c_ix | c_ix ∈ C, i ∈ L}
9       for each i ∈ L, let b_i the assignment of i
10      D_x ← ∩_{c_ix ∈ C_x} c_ix[b_i]
11      choose any value a of D_x as the assignment of x
12      L ← L ∪ {x}
13  output the assignment of the variables of L
```

Theorem 2 Assume the time and 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 $O(n^3\alpha)$ with working space $O(n + \beta)$.

Assume the constraint graph of P is complete. For every variable, there are at most n neighbors. So, to eliminate a variable (line 10–14) takes $O(n^2\alpha)$. Totally, n variables are removed. So, the 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 $O(n^3\alpha + ed^2)$ where ed^2 is the cost (amortizable) of removing values and its propagation. *Working space* here *excludes the space for the representation of the constraints and the new constraints created by elimination*. It is useful to distinguish the existing nonrandomized algorithms. 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 consider only constraints that are row convex and connected. These constraints are arc consistent in accordance with our definition. Remember that our definition

of connectedness is stronger than the original definition. The following property is clear and useful across this section.

Property 1 Given two row convex and connected constraints c_{ix} and c_{xj}, let c_{ij} be their composition. For any $u \in D_i$, $c_{ij}[u]$ is not empty.

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. Constraints here can use an *interval representation* defined below. For every $c_{ij} \in C$ and $u \in D_i$, $c_{ij}[u]$.min is used for $\min\{v \mid (u, v) \in c_{ij}\}$, and $c_{ij}[u]$.max for $\max\{v \mid (u, v) \in c_{ij}\}$.

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 $(u, v) \in c_{xj} \circ c_{ix}$. By Property 1, min $\leq$ max is always true for line 10.

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

```
compose (in c_ix, c_xj, out c_ij)
1    u ← succ (head, D_i)
2    while u ≠ tail do
3        v ← succ (head, D_j)
4        min ← tail, max ← head
5        while v ≠ tail do
6            if not disjoint (c_ix[u], c_jx[v]) then
7                if v > max then max ← v
8                if v < min then min ← v
9            v ← succ (v, D_j)
10       c_ij[u].min ← min, c_ij[u].max ← max
11       u ← 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
```

Proposition 1 The procedure of compose has a time complexity of $O(d^2)$ and space complexity is $O(1)$.

The two while loops (lines 2, 5) give a time complexity of $O(d^2)$. □

We emphasize that, due to the interval representation of constraints, for any c_{ix} and c_{xj} we need to call compose twice to compute c_{ij} and c_{ji} separately. This does not affect the complexity of those algorithms using compose. For example, for eliminate to use compose we need to change $i < j$ (line 10 of Algorithm 1) to $i \neq j$.

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. 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.

Algorithm 4: Remove values

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

Proposition 2 Given a CSP problem (V, D, C) of CRC constraints with an interval representation, the worst case time complexity of `removeValues` is $O(ed^2)$ with space complexity of $O(nd)$.

Let δ_i be the degree of variable $i \in V$. To delete a value (line 4–12), 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 matrix, for each constraint c_{ij} and $u \in D_i$, we setup $c_{ij}[u]$.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. This process has a time complexity of $O(ed^2)$ with working space complexity $O(nd)$ (due to Q).

By the above process, 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 $O(n^3 d^2)$ with space complexity $O(nd)$.

4.3 Fast composition of constraints

As one may see, `compose` makes use of the row convexity to the minimal degree. In fact, we can do better.
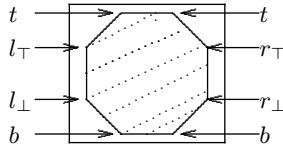

Figure 1: The area of 1's in the matrix of a CRC constraint

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

Proposition 3 Given a row convex and connected constraint c_{ij}, for all $u \in D_i$ such that $c_{ij}.l_\top \leq u \leq c_{ij}.l_\perp$, $c_{ij}[u].\text{min}= min$; for all u such that $c_{ij}.r_\top \leq u \leq c_{ij}.r_\perp$, $c_{ij}[u].\text{max}= max$; and the relation between $c_{ij}.l_\top$ ($c_{ij}.l_\perp$) and $c_{ij}.r_\top$ ($c_{ij}.r_\perp$) can be arbitrary.
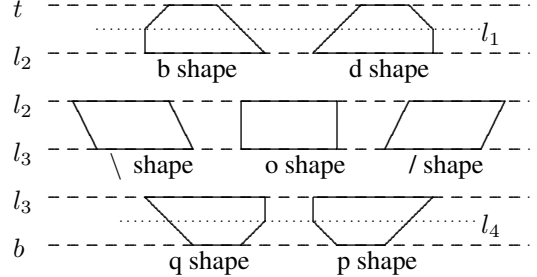

Figure 2: The possible shapes of the strips of a constraint that is row convex and connected

Consider a row convex and connected constraint c_{ij}. Let l_1, l_2, l_3, l_4 be the sorted values of $c_{ij}.l_\top$, $c_{ij}.r_\top$, $c_{ij}.l_\perp$, and $c_{ij}.r_\perp$. The matrix of c_{ij} consists of the following strips. 1) *Top strip* denotes 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 $c_{ij}.b$ (b in the diagram).

The row convexity and connectedness of c_{ij} implies that the 1's in its top strip can be of only 'b' shape or 'd' shape, the 1's in its middle strip of only '\' shape, 'o' shape, or '/' shape, and the 1's in its bottom strip of only 'q' shape or 'p' shape (see Figure 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 properties.

Property 2 Top strip: for every $u_1, u_2 \in [c_{ij}.t, l_2]$ where $u_1 \leq u_2$, $c_{ij}[u_1] \subseteq c_{ij}[u_2]$. Middle strip: for every $u_1, u_2 \in [l_2, l_3]$ where $u_1 = \text{pred}(u_2)$, shape '\' implies $c_{ij}[u_1].\text{min} \leq c_{ij}[u_2].\text{min}$ and $c_{ij}[u_1].\text{max} \leq c_{ij}[u_2].\text{max}$; shape 'o' implies $c_{ij}[u_1] = c_{ij}[u_2]$; and shape '/' implies $c_{ij}[u_2].\text{min} \leq c_{ij}[u_1].\text{min}$ and $c_{ij}[u_2].\text{max} \leq c_{ij}[u_1].\text{max}$. Bottom strip: for every $u_1, u_2 \in [l_3, c_{ij}.b]$ where $u_1 \leq u_2$, $c_{ij}[u_2] \subseteq c_{ij}[u_1]$.

Assume c_{ix} and c_{xj} are row convex and connected. The new algorithm to compute $c_{xj} \circ c_{ix}$, listed in Algorithm 5, is based on the following two ideas. 1) We first compute $c_{ij}[u].\text{min}$ for all $u \in D_i$ (line 2–21), which is called *min phase*, and then compute $c_{ij}[u].\text{max}$ for all $u \in D_i$ (line 22–41), which is called *max phase*. 2) In the two phases, the properties of the shapes and strips of c_{ix} are employed to speed up the computation.

In the min phase, the algorithm starts from the top strip of c_{ix}. Let $u = c_{ix}.t$. Find $c_{ij}[u].\text{min}$ (line 5) and let it be v.

Algorithm 5: Fast algorithm for computing the composition of two constraints

```
fastCompose (in c_ix, c_xj, out c_ij)
1   let l_1, ..., l_4 be the ascending sorted values of l_⊤, l_⊥, r_⊤, r_⊥ of c_ix
2   // min phase
3   // process the top strip of c_ix
4   u ← c_ix.b
5   find from head to tail the first v ∈ D_j such that c_ix[u] ∩ c_jx[v] ≠ ∅
6   c_ij[u].min ← 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 ← l_2
11      while u ≤ l_3 do {c_ij[u].min ← v, u ← succ (u, D_i)}
12  if the middle strip is of '\' shape then
13      if v ≠ 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 ≠ 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 ← c_ix.b
25  find the last v ∈ D_j such that c_ix[u] ∩ c_jx[v] ≠ ∅
26  c_ij[u].max ← 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 ← l_2
31      while u ≤ l_3 do {c_ij[u].max ← v, u ← succ (u, D_i)}
32  if the middle strip is of '\' shape then
33      if v ≠ 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 ≠ 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)
42  set the fields of c_ij: t, b, l_⊤, l_⊥, r_⊤, r_⊥
```

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 ≤ l do
3       find first v_1 from v down to head of D_j such that
        c_ix[u] ∩ c_jx[pred(v_1)] = ∅
4       c_ij[u].min = v_1, v ← v_1, u ← succ (u, D_i)
searchToLeftWrap (inout c_ix, c_xj, u, l, v, c_ij)
5   // search to the left of v
6   wrapToRight ← false
7   while u ≤ l do
8       find first v_1 from v down to head of D_j such that
        c_ix[u] ∩ c_jx[pred(v_1)] = ∅ and c_ix[u] ∩ c_jx[v_1] ≠ ∅
9       if v_1 does not exist then {wrapToRight ← true, break }
10      else {c_ij[u].min ← v_1, v ← v_1, u ← succ (u, D_i)}
11  if wrapToRight is true and u ≤ 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 ≤ l do
15      find first v_1 from v to tail of D_j such that
        c_ix[u] ∩ c_jx[pred(v_1)] = ∅ and c_ix[u] ∩ c_jx[v_1]
16      c_ij[u].min ← v_1, v ← v_1, u ← 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 ≤ l do
19      find last v_1 from v to tail of D_j such that c_ix[u] ∩ c_jx[v_1] ≠ ∅
20      c_ij[u].max ← v_1, v ← v_1, u ← succ (u, D_i)
searchToRightWrap (inout c_ix, c_xj, u, l, v, c_ij)
21  // search to the right of v
22  wrapToLeft ← false
23  while u ≤ l do
24      find last v_1 from v to tail of D_j such that
        c_ix[u] ∩ c_jx[succ(v_1)] = ∅ and c_ix[u] ∩ c_jx[v_1] ≠ ∅
25      if v_1 does not exist then {wrapToLeft ← true, break }
26      else {c_ij[u].max = v_1, v ← v_1, u ← 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 ≤ l do
31      find first v_1 from v down to head of D_j such that
        c_ix[u] ∩ c_jx[v_1] ≠ ∅
32      c_ij[u].max ← v_1, v ← v_1, u ← succ (u, D_i)
```

Due to the property of the top strip, we can find $c_{ij}[u]$.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 note that v is replaced by v_1 in line 4. Similarly, 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. By Property 2, lines 9–11 are quite straightforward for the 'o' shape. For the '\' shape (line 12–15), if v is not the first column of c_{xj} and $c_{ix}[u]$ is "above" the interval of the column before v of c_{xj} (line 13), we need to search to the left of v to be sure we do not miss any value of D_j that is smaller than v but is a support of $a \in [u, l_3]$. Due to the property of the '\' shape, after we hit the head of D_j and no support is found, we need to search to the right to tail if necessary (line 14). This process is implemented as `searchToLeftWrap` (line 5–12 of Algorithm 6). The correctness of this method is assured by the connectedness as well as row convexity of c_{ix} and c_{xj}. The details are not given here due to space limit. Otherwise (line 15), we only need to search to the right of v for values in $[u, l_3]$. The process for the '/' shape is similar to that for the '\' shape with some "symmetrical" differences (line 17).

The max phase is similar. Finally, according to the new c_{ij}, we set the attributes of c_{ij} in a proper way (line 42). Clearly, for each phase, we only need a time cost of $O(d)$.

Proposition 4 The algorithm `fastCompose` is correct and composes two constraints in time complexity of $O(d)$ with space complexity of $O(1)$.

5 CSP's with sparse constraint graphs

The practical 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 Figure 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.

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 $x \in G$, N(x) denotes
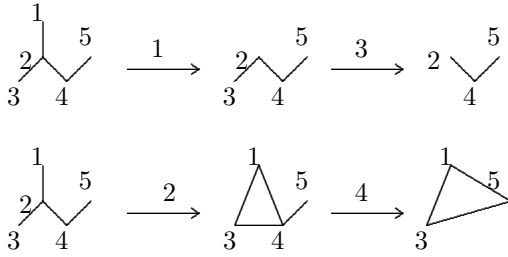
Figure 3: Example on elimination variable ordering

neighbors of x: $\{y \mid \{x, y\}$ is an edge of $G\}$. A vertex x is *simplicial* if the subgraph of G induced by $N(x)$ 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 [Bliek and Sam-Haroud, 1999]. 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 $O(nd)$.

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 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 fast algorithms 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, a deterministic 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 ($O(n^3 d + ed^2)$ where $e = n^2$) is better than the best ($O(n^3 d^2)$) of the existing algorithms.

For problems with sparse constraint graphs, the traditional path consistency method [Deville *et al.*, 1997] can not make use of the sparsity. Bliek and Sam-Haroud (1999) 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 (2006) has a time complexity of $O(\gamma n^2 d^2)$ where γ is the maximum degree of the original constraint graph. Our algorithm can achieve $O(n\sigma^2 d + e'd^2)$ 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.

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.

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 space complexity of the elimination algorithm to $O(n)$.

References

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

[Deville *et al.*, 1997] Y. Deville, O. Barette, and P. Van Hentenryck. Constraint satisfaction over connected row convex constraints. In *IJCAI-97*, volume 1, pages 405–411, Nagoya, Japan, 1997. IJCAI Inc.

[Kumar, 2006] 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.

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

[Zhang and Yap, 2003] 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.