A Deterministic Distributed Algorithm for Reasoning with Connected Row-Convex Constraints

Shufeng Kong
Centre for Quantum Software and Information
Faculty of Engineering & IT
University of Technology Sydney
Australia
shufeng.kong@student.uts.edu.au

Jae Hee Lee
Centre for Quantum Software and Information
Faculty of Engineering & IT
University of Technology Sydney
Australia
jaehee.lee@uts.edu.au

Sanjiang Li
Centre for Quantum Software and Information
Faculty of Engineering & IT
University of Technology Sydney
Australia
sanjiang.li@uts.edu.au

ABSTRACT

The class of CRC constraints generalizes several tractable classes of constraints and is expressive enough to model problems in domains such as temporal reasoning, geometric reasoning, and scene labelling. This paper presents the first distributed deterministic algorithm for connected row-convex (CRC) constraints. Our distributed (partial) path consistency algorithm efficiently transforms a CRC constraint network into an equivalent constraint network, where all constraints are minimal (*i.e.*, they are the tightest constraints) and generating all solutions can be done in a backtrack-free manner. When compared with the state-of-the-art distributed algorithm for CRC constraints, which is a randomized one, our algorithm guarantees to generate a solution for satisfiable CRC constraint networks and it is applicable to solve large networks in real distributed systems. The experimental evaluations show that our algorithm outperforms the state-of-the-art algorithm in both practice and theory.

Keywords

Connected row-convex constraints; deterministic algorithm; distributed algorithm; decomposable network

1. INTRODUCTION

Although solving general constraint satisfaction problems (CSPs) is known to be NP-hard, many subclasses have been identified as tractable. The class of *connected row-convex* (CRC) constraints [5] is an important tractable subclass of CSPs, which generalizes several subclasses of constraints such as 2SAT, binary integer linear constraints, and monotone constraints [5]. The CRC constraint class is very expressive and can model problems in domains such as temporal reasoning [18, 10], VLSI design [3], geometric reasoning [9], scene labelling [22] as well as logical filtering [12].

In this paper we are interested in handling CRC constraints from a distributed CSP perspective. Modelling problems from a distributed perspective is attractive when *privacy* and *autonomy* are of concern, as the conventional centralized approaches are often not applicable in this case. The following example illustrates a distributed CSP involving CRC constraints.

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

Example 1. Two agents A, B have their *private variables* $V_P^A = \{x^A, y^A\}$ and $V_P^B = \{y^B, z^B\}$, respectively, defined on finite domains. The private variables are subject to binary constraints such as $f^A(y^A) + x^A \cdot y^A \leq 0.5$, where f^A is a real-valued function that is either strictly increasing or decreasing. Agents A and B also have *shared variables* $V_S^A = \{z^A, t^A\}$ and $V_S^B = \{x^B, t^B\}$, respectively, which are connected with other agent's shared variables through constraints, such as $f^A(z^A) + x^B \leq 0.3$. They can be also connected with the agent's private variables through constraints. Each agent has control over only its own variables, where the values of its private variables are not known to the other agent. Figure 1 illustrates an example constraint graph, where all

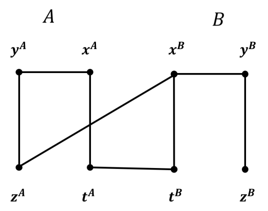


Figure 1: A constraint graph of Example 1.

edges correspond to the constraints mentioned in the example. All constraints here are CRC [5]. The problem cannot be solved in a centralized way; a distributed algorithm is therefore required so as to take consideration of the privacy of agents.

In the literature, Yokoo *et al.* [24] proposed a distributed backtracking algorithm for *general* distributed CSPs and, since then, efficient algorithms have been developed for *specific* distributed CSPs that are tailored to the problem domains of interest, *e.g.*, the distributed scheduling problem [14], the distributed plan coordination problem [4], and the distributed simple temporal problem (STP) [2].

Recently, Kumar *et al.* [11] proposed a distributed algorithm, called D-CRC, for solving CRC constraints, which was shown to be more efficient than the state-of-the-art centralized algorithm for CRC constraints. There are, however, several drawbacks of D-CRC: (i) it is based on randomization and as such it does not guarantee to return a solution even when the input CSP is consistent and (ii) cannot determine the inconsistency of the input; and (iii) it cannot assign more than one variable to each agent, which makes the algorithm unrealistic to solve large networks in real distributed systems.

The previous observations suggest us that designing an efficient *deterministic* distributed algorithm for reasoning with CRC constraints is critically important. To tackle this problem, we adopt in this paper a powerful method, called P^{3}C [20], which was originally designed to solve simple temporal networks (STNs) and makes use of the convex property of simple temporal constraints. Recently, Boerkoel and Durfee [2] successfully extended P^{3}C to STNs in distributed settings, which suggests that their approach can also be adapted to distributed solving of other convex constraints, such as CRC constraints. However, while the classes of CRC constraints and simple temporal constraints both share the convex properties, it was observed that, as opposed to simple temporal constraints, CRC constraints are not distributive in the sense that the relational composition operation is not distributive over nonempty intersections [15]. This presents a significant obstacle for a straightforward adaptation of the machinery devised for STNs to CRC constraints.

In this paper, we present a deterministic distributed algorithm, called D△CRC, for solving CRC constraint networks. To this end, we prove that P^{3}C can indeed be adapted to transform an input CRC constraint network into an equivalent constraint network, where all constraints are minimal (*i.e.*, they are the tightest constraints) and generating all solutions can be done in a backtrack-free manner. Then we extend the result to distributed settings by employing the communication mechanism by Boerkoel and Durfee [2] that was originally devised for distributed STNs. Our algorithm D△CRC does not suffer from the aforementioned deficiencies that the state-of-the-art algorithm D-CRC has, and it is more efficient than D-CRC in theory as well as in practice.

The remainder of the paper is organized as follows. After a short introduction of basic definitions and notations in section 2, we propose in section 3 a centralized strong partial path consistency algorithm (△CRC) for CRC constraint networks. Then in section 4 we extend this to a distributed setting and present the distributed △CRC algorithm (D△CRC) for CRC constraint networks, where we adopt the communication mechanism from [2]. In section 5 we evaluate the D△CRC algorithm against the state-of-the-art algorithm for solving CRC constraints and conclude the paper in section 6 with further discussion.

2. PRELIMINARIES

In this section, we recall some basic notions and results about binary constraint networks, triangulated constraint graphs, and connected row-convex constraints.

2.1 Constraints and Constraint Networks

Definition 1. A *binary* constraint network $\mathcal{N}$ is a triple $\langle V, D, C\rangle$, where:

- $V = \{v_1, \ldots, v_n\}$ is a non-empty finite set of variables;
- $D = \{D_1, \ldots, D_n\}$ is a collection of finite sets of values, where D_i serves as the domain of variable $v_i \in V$;
- $C = \{(s_k, R_k) \mid 1 \leq k \leq m\}$ is a set of *binary constraints*, where s_k is a pair of variables in V, say (v_i, v_j), and $R_k \subseteq D_i \times D_j$.

Let $\mathcal{N} = \langle V, D, C\rangle$ be a binary constraint network. Throughout this paper we assume that for any pair of variables (v_i, v_j), there exists at most one constraint $((v_i, v_j), R_{ij}) \in C$ between them. We also assume that $R_{ji} = R_{ij}^{-1}$, where

$$R_{ij}^{-1} := \{\langle y, x\rangle \mid \langle x, y\rangle \in R_{ij}\}$$

is the inverse of R_{ij}. For brevity, we often write R_{ij} for the constraint $((v_i, v_j), R_{ij})$.

The *image* of $a \in D_i$ under R_{ij}, denoted by $R_{ij}(a)$, is the set $\{b \in D_j \mid \langle a, b\rangle \in R_{ij}\}$. A *solution* of $\mathcal{N}$ is an assignment of values from $D_1, \ldots, D_n$ to the variables $v_1, \ldots, v_n$ s.t. all constraints in C are satisfied. We use $sol(\mathcal{N})$ to denote the set of solutions of $\mathcal{N}$ and say that $\mathcal{N}$ is *consistent* if $sol(\mathcal{N}) \neq \varnothing$, and *inconsistent* otherwise.

The underlying graph structure of a constraint network $\mathcal{N} = \langle V, D, C\rangle$ can be always represented as an undirected graph $G_{\mathcal{N}} = (V, E(\mathcal{N}))$, called the *constraint graph* of $\mathcal{N}$, where an edge $e_{ij} \in E(\mathcal{N})$ iff $R_{ij} \in C$ or $R_{ji} \in C$. In this paper, we assume that each edge $e_{ij} \in E(\mathcal{N})$ is associated with a relation $R_{ij} \in C$, and we use the constraint network $\mathcal{N}$ and its corresponding constraint graph $G_{\mathcal{N}}$ interchangeably to represent the same problem at hand.

Definition 2. [19] Let $\mathcal{N} = \langle V, D, C\rangle$ be a constraint network and $\prec = (v_1, \ldots, v_n)$ an ordering of variables in V. Write $V_{\leq i} = \{v_j \mid j \leq i\}$. We say that $\mathcal{N}$ is *decomposable w.r.t.* $\prec$ if for any $i < n$, any consistent assignment to $V_{\leq i}$ can be consistently extended to an assignment to $V_{\leq i+1}$. We say $\mathcal{N}$ is *decomposable* if it is decomposable w.r.t. every ordering of variables in V.

Definition 3. [17] Let $\mathcal{N} = \langle V, D, C\rangle$ be a constraint network and $G_{\mathcal{N}}$ its constraint graph. Let e_{ij} be an edge in $G_{\mathcal{N}}$ and $\pi_{ij} = (v_i = u_0, u_1, \ldots, u_k = v_j)$ a path in $G_{\mathcal{N}}$ with $e_{ij} \in E(\mathcal{N})$. We say that e_{ij} (or R_{ij}) is *arc-consistent* (AC) if for each element $a \in D_i$ there is an element $b \in D_j$ s.t. $\langle a, b\rangle \in R_{ij}$; we say π_{ij} is *path-consistent* (PC) if, for every $\langle c_0, c_k\rangle \in R_{ij}$, we can find values for all intermediate variables u_x $(0 < x < k)$ s.t. all the constraints $R_{u_x, u_{x+1}}$ $(0 \leq x < k)$ are satisfied.

Specially, a path of length 1 is always PC.

Definition 4. [17, 1] Let $\mathcal{N} = \langle V, D, C\rangle$ be a constraint network and $G_{\mathcal{N}}$ its constraint graph. We say $\mathcal{N}$ is AC iff all edges in $G_{\mathcal{N}}$ are AC, and say $\mathcal{N}$ is partial path-consistent (PPC) if every path π_{ij} in $G_{\mathcal{N}}$ with $e_{ij} \in E(\mathcal{N})$ is PC. Moreover, a PPC network is called PC if its constraint graph is complete.

Therefore, PC is a special case of PPC. We say a constraint network is *strong* PC (PPC, resp.) if it is both AC and PC (PPC, resp.).

Definition 5. [19] Let $\mathcal{N} = \langle V, D, C\rangle$ be a constraint network. We say a non-empty constraint $R_{ij} \in C$ is *minimal* if any assignment $\langle a_i, a_j\rangle \in R_{ij}$ to variables v_i, v_j can be extended to a solution of $\mathcal{N}$. We say $\mathcal{N}$ is minimal if it has a complete constraint graph and every constraint in $\mathcal{N}$ is minimal.

Decomposable network is always minimal and a constraint network that admits a minimal constraint is always consistent.

2.2 Triangulated Constraint Networks

Triangulated graphs play a key role in efficiently solving large sparse constraint networks [1, 23, 20, 16]. An undirected graph $G = (V, E)$ is said to be *triangulated* or *chordal* if every cycle of length greater than 3 has a chord, *i.e.*, an edge connecting two non-consecutive vertices of the cycle. A network $\mathcal{N}$ is said to be triangulated (resp. complete) if $G_{\mathcal{N}}$ is triangulated (resp. complete).

Triangulated constraint graphs have the following nice property.

THEOREM 1. [1] *A triangulated constraint graph is PPC if every path of length 2 is PC.*

If a constraint graph is not triangulated, we may add new edges (labeled with universal constraints) to make it triangulated. In the following, we give a characterization of triangulated graphs.

Definition 6. [7] An ordering $\prec$ in a graph $G = (V, E)$ is called a *perfect vertex elimination ordering* in G, if the set of successors of v

$$F_v := \{w \mid (v, w) \in E, v \prec w\}$$

induces a complete subgraph of G for all $v \in V$.

For example, consider the graph $G = (V, E)$ in Figure 2, the ordering (v_1, v_2, v_3, v_4) is a perfect vertex elimination ordering in G, whereas the ordering (v_2, v_1, v_3, v_4) is not.

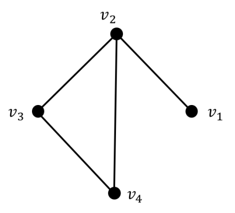


Figure 2: A graph $G = (V, E)$.

PROPOSITION 1. [7] *A graph G is triangulated iff there exists a perfect vertex elimination ordering in G.*

We next introduce the notion of row-convex constraints.

2.3 Connected Row-Convex Constraints

Suppose $R \subseteq D_1 \times D_2$ is a binary relation. The *(0,1)-matrix representation* of a relation R consists of $|D_1|$ rows and $|D_2|$ columns subject to orderings imposed on D_1, D_2. The entry in the i-th row and j-th column of the matrix is 1, if the corresponding pair of values from $D_1 \times D_2$ is in R, and is 0 otherwise.

Definition 7. [5] A binary relation R, represented as a (0,1)-matrix, is *row-convex* if in each row all of the '1's are consecutive. We say R is a *connected row-convex* (CRC) relation if, after removing empty rows and columns in the matrix of R, (i) both R and its transpose are row-convex, and (ii) the positions of the '1's in any two consecutive rows or columns intersect, or are consecutive

Fig. 3 shows two row-convex constraints, where the left one is CRC but the right one is not.

			1	1	1
			1	1	
		1			
	1				
1	1				
	1				

(a) A CRC constraint.

	1				
1	1	1			
	1	1	1		
		1			
		1	1	1	1
				1	1

(b) A non-CRC constraint.

Figure 3: A CRC constraint and a non-CRC constraint.

Row convex constraints are not closed under intersection and composition [5]—the two operations to enforce PC, but CRC constraints are closed under these operations.

LEMMA 1. [5] *CRC constraints are closed under intersection and composition.*

Row-convex constraint networks have the following property.

LEMMA 2. [22] *A row-convex constraint network is decomposable and minimal if it is strong PC.*

The following lemma shows that enforcing PPC on a triangulation of a CRC network has the same effect as enforcing PC as far as only edges in the triangulation are concerned.

LEMMA 3. [1] *Let $\mathcal{N}$ be a CRC constraint network. Suppose $G \supseteq G_{\mathcal{N}}$ is a triangulation of $G_{\mathcal{N}}$ and G^* is the completion of $G_{\mathcal{N}}$. Further, let $\mathcal{N}^{\triangle} = \langle V, D, C^{\triangle}\rangle$ and $\mathcal{N}^* = \langle V, D, C^*\rangle$ be the constraint networks obtained by enforcing PPC on $\mathcal{N}$ w.r.t G and G^*, respectively. Then $C^{\triangle} \subseteq C^*$.*

As a corollary of Lemmas 2 and 3, we have

COROLLARY 1. *Let $\mathcal{N}$ be a CRC constraint network and G a triangulation of $G_{\mathcal{N}}$. Suppose $\mathcal{N}' = \langle V, D, C'\rangle$ is the network obtained by enforcing PPC on $\mathcal{N}$ w.r.t. G. If no inconsistency is detected, then all constraints in C' are minimal.*

PROOF. Suppose $\mathcal{N}$ is consistent. Let $\mathcal{N}^* = \langle V, D, C^*\rangle$ be the path-consistent constraint network that is equivalent to $\mathcal{N}$, *i.e.*, $\mathcal{N}^*$ is obtained by enforcing PPC on $\mathcal{N}$ w.r.t. the completion of $G_{\mathcal{N}}$. By Lemma 2, $\mathcal{N}^*$ is decomposable and minimal. Further, by Lemma 3, we have $C' \subseteq C^*$, which finishes our proof. $\Box$

By Lemma 3 and Theorem 1, enforcing PPC on a CRC constraint network $\mathcal{N}$ can be done by the following two steps: (i) obtain a triangulation G of $G_{\mathcal{N}}$ by adding edges labeled with universal constraints, and (ii) make every path of length two in G path-consistent by using relational intersection and composition. The following section then presents such an algorithm for enforcing strong PPC.

3. AN EFFICIENT CENTRALIZED ALGORITHM FOR CRC CONSTRAINTS

In this section we present an efficient algorithm to enforce strong partial path-consistency (PPC) on CRC constraint networks. The algorithm, called $\triangle$CRC, is adapted from the well-known P^{3}C algorithm [20], which enforces PPC on simple temporal networks and was further generalized to enforce PPC on qualitative constraint networks defined over distributive subalgebras [16].

As opposed to simple temporal constraints and distributive subalgebras, CRC constraints are defined over finite domains and are not distributive [15] (*i.e.*, the relational composition operation is not distributive over nonempty intersections of CRC constraints). Consequently, naively adapting the algorithms from [20, 16] to CRC constraints does not lead to an algorithm that enforces strong PPC on CRC constraint networks. We solve this in the following way:

First, as the domains of CRC networks are finite, arc-consistency (AC) is not automatically achieved with a naive adaptation of P^{3}C. Therefore, including an AC enforcing mechanism in the algorithm is necessary for enforcing strong PPC on the CRC networks. Instead of simply including an AC enforcing mechanism as a preprocessing procedure (cf. [25]) that can cause extra computing effort, we integrate the AC enforcing mechanism tightly into the algorithm.

Second, contrary to the proofs in [20, 15], which implicitly make use of distributivity of constraints, we prove that $\triangle$CRC enforces strong PPC without making use of distributivity of CRC constraints (cf. Theroem 2).

Algorithm.

Our algorithm $\triangle$CRC is presented as Algorithm 1. It takes as input a CRC network $\mathcal{N} = \langle V, D, C\rangle$ and an ordering $\prec = (v_n, \ldots, v_1)$ on V. It first eliminates variables v_k along the ordering $\prec$ and propagates the information of constraints involving v_k to its successors $F_k := \{v_i \mid R_{ki} \in C, i < k\}$ by using the following elimination rule.

Algorithm 1: $\triangle$CRC

```
Input : A constraint network $\mathcal{N} = \langle V, D, C\rangle$, where $C$ is a
        set of CRC constraints;
        An ordering $\prec = (v_n, \dots, v_1)$ on $V$.
Output: A triangulated PPC network that is equivalent to $\mathcal{N}$ if
        $\mathcal{N}$ is consistent, or "inconsistent".
// Phase 1:  Eliminate variables
for $k \leftarrow n$ to 2 do
    (result, $\mathcal{N}$) $\leftarrow$ ELIMINATE($v_k, \mathcal{N}, \prec$)
    if result = False then
        return "inconsistent"
// Phase 2:  Reinstate variables
for $k \leftarrow 2$ to $n$ do
    $\mathcal{N} \leftarrow$ REINSTATE($v_k, \mathcal{N}, \prec$)
return $\mathcal{N}$
Function ELIMINATE($v_k, \mathcal{N} = \langle V, D, C\rangle, \prec$)
    for $i \leftarrow k-1$ to 1 s.t. $R_{ik} \in C$ do
        for $j \leftarrow i-1$ to 1 s.t. $R_{jk} \in C$ do
            if $R_{ij} \notin C$ then
                add constraint $R_{ij} := D_i \times D_j$ to $C$
            $R_{ij} \leftarrow R_{ij} \cap (R_{ik} \circ R_{kj})$
            if $R_{ij} = \varnothing$ then
                return (False, $\mathcal{N}$)
        $D_i \leftarrow D_i \cap R_{ki}(D_k)$
        if $D_i = \varnothing$ then
            return (False, $\mathcal{N}$)
    return (True, $\mathcal{N}$)
Function REINSTATE($v_k, \mathcal{N} = \langle V, D, C\rangle, \prec$)
    $C^* \leftarrow C$
    for $i \leftarrow k-1$ to 1 s.t. $R_{ik} \in C$ do
        for $j \leftarrow i-1$ to 1 s.t. $R_{jk} \in C$ do
            // $R^*_{kj}$ and $R^*_{ki}$ are in $C^*$
            $R_{ki} \leftarrow R_{ki} \cap (R^*_{kj} \circ R_{ji})$
            $R_{kj} \leftarrow R_{kj} \cap (R^*_{ki} \circ R_{ij})$
        $D_k \leftarrow D_k \cap R_{ik}(D_i)$
    return $\langle V, D, C\rangle$
```

Eliminate $\mathbf{v_k}$: For all $v_i, v_j \in F_k$ update R_{ij} as $R_{ij} \cap (R_{ik} \circ R_{kj})$ and update D_i as $D_i \cap R_{ki}(D_k)$.

After the elimination phase, Algorithm 1 reinstates the variables according the inverse ordering $\prec^{-1}$ and propagates the information of the constraints back to the neighboring predecessors.

Reinstate $\mathbf{v_k}$: For all $v_i, v_j \in F_k$, update R_{ik} as $R_{ik} \cap (R_{ij} \circ R_{jk})$ and R_{jk} as $R_{jk} \cap (R_{ji} \circ R_{ik})$ and update D_k as $D_k \cap R_{ik}(D_i)$.

We first show that Algorithm 1 outputs a triangulated network.

LEMMA 4. *Let $(\mathcal{N}, \prec)$ be an input to Algorithm 1 where $\mathcal{N}$ is consistent. Then Algorithm 1 outputs a triangulated CRC constraint network $\mathcal{N}'$ which is equivalent to $\mathcal{N}$ and has $\prec$ as its perfect elimination ordering.*

PROOF. Suppose $\mathcal{N}$ is consistent and $\mathcal{N}'$ is the output of Algorithm 1. Because CRC constraints are closed under intersection and composition (cf. Lemma 1), $\mathcal{N}'$ is also a CRC constraint network. Moreover, as the operations (*i.e.*, intersection, composition and adding universal relations) in Algorithm 1 do not modify the solution space of $\mathcal{N}$, $\mathcal{N}'$ is equivalent to $\mathcal{N}$.

We next prove that $\mathcal{N}'$ is triangulated. Write $\mathcal{N}^d = \langle V, D, C^d\rangle$ for the network obtained after the elimination phase (lines 1–4) of Algorithm 1. For any $1 < k \leq n$, note that the set of variables F_k induces a subnetwork of $\mathcal{N}^d$ whose constraint graph is complete. This shows that $\prec = (v_n, \dots, v_1)$ is a perfect vertex elimination ordering in $G_{\mathcal{N}^d}$. Since the reinstatement phase does not alter the graph structure, $\prec$ is also a perfect vertex elimination ordering in the constraint graph of $\mathcal{N}'$. By Proposition 1, we know $\mathcal{N}^d$ and $\mathcal{N}'$ are both triangulated graphs. $\square$

Algorithm 1 also decides the consistency of CRC constraint networks. The proof uses the *Helly property* as stated in the following lemma.

LEMMA 5. [22] *Let F be a finite collection of (0,1)-row vectors that are row-convex and of equal length such that every pair of row vectors in F have a non-zero entry in common. Then all row vectors in F have a non-zero entry in common.*

The following result follows from [25, Theorem 1]. For completeness we include a proof here.

PROPOSITION 2. *Let $(\mathcal{N}, \prec)$ be an input to Algorithm 1. Then $\mathcal{N}$ is consistent if and only if the algorithm does not return "inconsistent" in the elimination phase.*

PROOF. Suppose that the input constraint network $\mathcal{N}$ is consistent. Then, because the operations in Algorithm 1 do not modify the solution set of $\mathcal{N}$, the algorithm does not find any empty relation in lines 14–15 or empty domain in lines 17–18. Thus, it does not return "inconsistent" (line 4).

Let $\mathcal{N}^d = \langle V, D, C^d\rangle$ be the network obtained after the elimination phase (lines 1–4) of Algorithm 1. Now suppose that the elimination step (lines 1–4) of the algorithm did not return "inconsistent" as its output. We show that $\mathcal{N}^d$ is consistent. Let $\mathcal{N}^d_k$ be the subnetwork of $\mathcal{N}^d$ induced by variables $v_1, \dots, v_k$. We claim that $\mathcal{N}^d_k$ is consistent for $k = 1, \dots, n$ and prove the claim by induction on k. First, $\mathcal{N}^d_1$ is consistent, because D_1 is not empty; otherwise the elimination step would have detected the inconsistency in lines 17–18. Now, suppose that $\mathcal{N}^d_k$ is consistent and let $A_k := \langle a_1, \dots, a_k\rangle \in D_1 \times \cdots \times D_k$ be a solution for $\mathcal{N}^d_k$. We show that A_k can be extended to a solution $A_{k+1} = \langle a_1, \dots, a_{k+1}\rangle$ for $\mathcal{N}^d_{k+1}$. To this end, we need to show that $\langle a_i, a_{k+1}\rangle \in R_{i,k+1}$ for all $i \leq k$ with $R_{i,k+1} \in C^d$. Observe that after the elimination phase, (i) for any variable v_i with $i \leq k$, $R_{i,k+1}$ is arc-consistent, and (ii) for any pair of variables (v_i, v_j) with $i, j \leq k$ with $R_{i,k+1}, R_{j,k+1} \in C^d$, we have that $R_{ij} \subseteq R_{i,k+1} \circ R_{k+1,j}$. Since $\langle a_i, a_j\rangle \in R_{ij}$, we also have $R_{i,k+1}(a_i) \cap R_{j,k+1}(a_j) \neq \varnothing$ for all $i, j \leq k$ with $R_{i,k+1}, R_{j,k+1} \in C^d$.

Then, by Lemma 5, the intersection of all $R_{i,k+1}(a_i)$ with $i \leq k, R_{i,k+1} \in C^d$ is not empty. Thus there exists a value a_{k+1} with $\langle a_i, a_{k+1}\rangle \in R_{i,k+1}$ for all $i \leq k, R_{i,k+1} \in C^d$ and we showed that $A_{k+1} = \langle a_1, \dots, a_{k+1}\rangle$ is a solution of $\mathcal{N}^d_{k+1}$. Thus $\mathcal{N}^d_k$ is consistent for $k = 1, \dots, n$ and in particular for $\mathcal{N}^d = \mathcal{N}^d_n$. Since operations in Algorithm 1 do not modify the solution set of $\mathcal{N}$, networks $\mathcal{N}^d$ and $\mathcal{N}$ are equivalent. Thus, $\mathcal{N}$ is consistent. $\square$

The preceding proof also shows that one can generate *all* solutions of a consistent input network *backtrack-free* by applying $\triangle$CRC and instantiating the variables along the inverse of the input ordering $\prec$. This is because for all $1 \leq k < n$ a solution $\langle a_1, \dots, a_k\rangle$ of $\mathcal{N}^d_k$ can be extended to a solution $\langle a_1, \dots, a_{k+1}\rangle$ of $\mathcal{N}^d_{k+1}$ by choosing an element a_{k+1} from the intersection of all $R_{i,k+1}(a_i)$

with $i \leq k, R_{i,k+1} \in C^d$, which is not empty as shown in the preceding proof. This proves the following result.

Proposition 3. *Algorithm 1 returns, on input $(\mathcal{N}, \prec)$, an equivalent CRC network $\mathcal{N}'$ that is decomposable with respect to the inverse of $\prec$, if $\mathcal{N}$ is consistent.*

We next show that Algorithm 1 also enforces strong PPC.

Theorem 2. *Algorithm 1 returns, on input $(\mathcal{N}, \prec)$, a network $\mathcal{N}'$ that is AC and PPC, if $\mathcal{N}$ is consistent.*

Proof. Suppose $\mathcal{N}$ is consistent. Let $\mathcal{N}' = \langle V, D, C\rangle$ be the output of Algorithm 1 on $(\mathcal{N}, \prec)$, and $\mathcal{N}'_k$ be the subnetwork of $\mathcal{N}'$ induced by $v_1, v_2, ..., v_k$. We note that $\mathcal{N}'_k$ is triangulated for all k (cf. Lemma 4). Now we claim that $\mathcal{N}'_k$ is AC and PPC for $1 \leq k \leq n$. We prove the claim by induction on k.

First, $\mathcal{N}'_1$ is trivially AC and PPC, as D_1 is not empty.

Now suppose $\mathcal{N}'_{k-1}$ is AC and PPC. We first prove that $\mathcal{N}'_k$ is AC. Let $i, j < k$. Then all $R_{ij} \in C$ are AC by the induction hypothesis. Now suppose $R_{ik} \in C$. Note that R_{ki} is AC due to line 26 in Algorithm 1. We show that R_{ik} is also AC. If there is no $j \neq i$ with $v_j \in F_k$, then R_{ik} is AC by line16. Thus suppose there exists $j \neq i$ with $v_j \in F_k$ and let $a_i \in D_i$. Then by the induction hypothesis, R_{ij} is AC and PPC with respect to $\mathcal{N}'_{k-1}$, thus by Corollary 1 it is minimal with respect to $\mathcal{N}'_{k-1}$. There exists a solution for $\mathcal{N}'_{k-1}$ which assigns a_i to v_i. Furthermore, this solution can be extended to a solution σ of $\mathcal{N}'$ as $\mathcal{N}'$ is decomposable with respect to the inverse of $\prec$ by Proposition 3. Suppose σ assigns a_k to v_k. Then $(a_i, a_k) \in R_{ik}$ holds. This proves that $\mathcal{N}'_k$ is AC.

We now show that $\mathcal{N}'_k$ is PPC. Since all paths π in $\mathcal{N}'_{k-1}$ are PC by the induction hypothesis, we can assume that π includes v_k, and because $\mathcal{N}'_k$ is triangulated, by Theorem 1 we can assume that π is of length 2.

Suppose π has the form (v_i, v_k, v_j) with $i, j < k$ (see Figure 4 for an illustration). By induction hypothesis $\mathcal{N}'_{k-1}$ is PPC and,

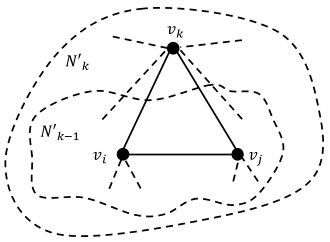


Figure 4: Illustration of proof of Theorem 2.

by Corollary 1, R_{ij} is minimal with respect to $\mathcal{N}'_{k-1}$. Then there exists a solution of $\mathcal{N}'_{k-1}$ that assigns values a_i and a_j to variables v_i and v_j, respectively. This solution can be extended to a solution σ of $\mathcal{N}'$ by Proposition 3. Suppose σ assigns a_k to v_k. Then, $(a_i, a_k) \in R_{ik}$ and $(a_k, a_j) \in R_{kj}$. Thus (v_i, v_k, v_j) is PC.

Now suppose π has the form (v_i, v_j, v_k) with $i, j < k$. We show that for any $(a_i, a_k) \in R_{ik}$ there exists a_j such that $(a_i, a_j) \in R_{ij}$ and $(a_j, a_k) \in R_{jk}$, *i.e.*, $R_{ij}(a_i) \cap R_{kj}(a_k) \neq \varnothing$. By lines 22–25 in the algorithm, we have for all $\mu < k$ with $R_{k\mu} \in C$ that

$$R_{k\mu} = R^*_{k\mu} \cap \bigcap_{v_\nu \in F_k} (R^*_{k\nu} \circ R_{\nu\mu}) \tag{1}$$

This in particular means that

$$R_{kj} = R^*_{kj} \cap \bigcap_{v_\nu \in F_k} (R^*_{k\nu} \circ R_{\nu j}) \tag{2}$$

Thus, to show that $R_{ij}(a_i) \cap R_{kj}(a_k)$ is not empty, it suffice to show by the Helly property (cf. Lemma 5) that the intersection of any two of the following sets is not empty:

$$R_{ij}(a_i), \quad R^*_{kj}(a_k), \quad \bigcap_{v_\nu \in F_k} (R^*_{k\nu} \circ R_{\nu j})(a_k)$$

First, from equation (1) it follows that $R_{ki} \subseteq R^*_{k\nu} \circ R_{\nu i}$ for all $v_\nu \in F_k$. Thus $(a_k, a_i) \in R_{ki} \subseteq R^*_{kj} \circ R_{ji}$ and the intersection of $R_{ij}(a_i)$ and $R^*_{kj}(a_k)$ is not empty.

Then, because $R_{kj} \neq \varnothing$ is AC by the induction hypothesis, we know $R_{kj}(a_k) \neq \varnothing$. Then equation (2) implies that the intersection of $R^*_{kj}(a_k)$ and $\bigcap_{v_\nu \in F_k} (R^*_{k\nu} \circ R_{\nu j})(a_k)$ is not empty.

Finally, for an arbitrary $\nu < k$ with $v_\nu \in F_k$ we have by equation (1) that $R_{ki} \subseteq R^*_{k\nu} \circ R_{\nu i}$. Since $R_{\nu i}$ is PC with respect to $\mathcal{N}'_{k-1}$ we also have that $R_{ki} \subseteq R^*_{k\nu} \circ (R_{\nu j} \circ R_{ji})$. Then, because $(a_k, a_i) \in R_{ki} \neq \varnothing$, we have that $(a_k, a_i) \in (R^*_{k\nu} \circ R_{\nu j}) \circ R_{ji}$. Thus, the intersection of $R_{ij}(a_i)$ and $(R^*_{k\nu} \circ R_{\nu j})(a_k)$ is not empty for all $\nu < k$ with $v_\nu \in F_k$. Then, by the Helly property (cf. Lemma 5), the intersection of $R_{ij}(a_i)$ and $\bigcap_{v_\nu \in F_k} (R^*_{k\nu} \circ R_{\nu j})(a_k)$ is not empty. Altogether, we have that $R_{ij}(a_i) \cap R_{kj}(a_k) \neq \varnothing$ and have showed π is PC with respect to $\mathcal{N}'_k$.

In summary, we have proved that $\mathcal{N}'_k$ is AC and PPC for all $1 \leq k \leq n$. Thus $\mathcal{N}' = \mathcal{N}'_n$ is AC and PPC. $\square$

By Corollary 1 and Theorem 2, we have the following result.

Corollary 2. *Algorithm 1 transforms an input CRC constraint network into an equivalent network where all its constraints are minimal, if no inconsistency is detected.*

By the above result, the minimal network of a consistent CRC constraint network $\mathcal{N}$ can be computed as follows: (i) complete $\mathcal{N}$ by adding edges labeled with universal constraints, and (ii) apply Algorithm 1 on the completion of $\mathcal{N}$.

We now turn our attention to the analysis of the time complexity of Algorithm 1. We first recall some concepts in graph theory required for the analysis of the time complexity of Algorithm 1.

An *ordered graph* is a pair $(G, \prec)$, where $G = (V, E)$ is an undirected graph and $\prec$ is an ordering on V. The nodes adjacent to v that succeeds it in the ordering are called its *children*. The *width of a node* in an ordered graph is its number of children. The *width of an ordering* $\prec$, denoted by $w(\prec)$, is the maximum width among all of its nodes. The *induced graph* of an ordered graph $(G, \prec)$ is an ordered graph $(G^*, \prec)$, where $G^* = (V, E^*)$ is obtained from G as follows: the nodes of G are processed from first to last with respect to $\prec$; when a node v is processed, we connect all of its children with edges. The *induced width* of an ordering $\prec$, denoted by $w^*(\prec)$, is the width of the ordering $\prec$ with respect to G^*.

Proposition 4. *Algorithm 1 runs in time $O(w^*(\prec)ed)$, where e is the number of edges of the induced graph of $G_\mathcal{N}$ and d is the largest domain size.*

Proof. Because the elimination phase and the reinstatement phase have the same time complexity, we will only consider the elimination phase. Let $(G^* = (V, E^*), \prec)$ be the induced graph of $(G_\mathcal{N}, \prec)$. Given $v_k \in V$, let $F_k = \{v_i \mid i < k, e_{ik} \in G_\mathcal{N}\}$. We first analyze the time complexity of function Eliminate. Lines 11–18 are executed at most $|F_k|^2$ times. Since both the composition operation and the intersection operation run in $O(d)$ [25, 5], the operations in line 13 and line 16 takes $O(d)$ time. We can conclude that it takes $O(|F_k|^2 d)$ time for function Eliminate to eliminate variable v_k. Therefore, the time complexity of the elimination phase is $O(\sum_{k=1}^{n} |F_k|^2 d)$. Because $|F_k| \leq w^*(\prec)$, we have

$$\sum_{k=1}^{n} |F_k|^2 \leq \sum_{k=1}^{n} (|F_k| w^*(\prec)) = w^*(\prec) \sum_{k=1}^{n} |F_k| \quad (3)$$
$$= w^*(\prec) e, \quad (4)$$

where $e = |E^*|$. Therefore, Algorithm 1 runs in $O(w^*(\prec)ed)$. □

Note that when the constraint network is complete, Algorithm 1 runs in time $O(n^3 d)$. This can be compared with PC-CRC [5], the state-of-the-art PC enforcing algorithm for CRC constraints, which runs in time $O(n^3 d^2)$. It is worth mentioning that Algorithm 1 performs even more efficiently when the input networks are sparse.

4. AN EFFICIENT DISTRIBUTED ALGORITHM FOR CRC CONSTRAINTS

In this section, we extend △CRC to a distributed algorithm, called D△CRC, for solving distributed CRC networks. Except for some details, we follow the work by Boerkoel and Durfee [2], which extends P^{3}C to solving simple temporal networks in distributed settings. For further details, we refer the readers to [2].

Definition 8. [24] A binary *distributed constraint network* is a pair $\mathcal{M} = \langle \mathcal{P}, C_X \rangle$, where

- $\mathcal{P} = \{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$ is a set of *local* binary constraint networks, where each $\mathcal{N}_i$ is defined as $\mathcal{N}_i = \langle V_i, D_i, C_i \rangle$.
- $C_X = \{(s_k, R_k) \mid 1 \leq k \leq m\}$ is a set of *external constraints*, where each s_k is a pair of variables from $V_{i_k} \times V_{j_k}$ with $i_k \neq j_k$ and $R_k \subseteq D_{i_k} \times D_{j_k}$.

We note that in Definition 8 each local constraint network $\mathcal{N}_i$ corresponds to an agent i and that C_X is a set of constraints that are shared by two different agents.

Definition 9. Given a binary distributed constraint network $\mathcal{M} = (\mathcal{P}, C_X)$, a variable $v \in V_j$ is called an *external variable* of agent i if there is an external constraint $((v, w), R) \in C_X$ with $v \in V_i$ and $w \in V_j$, $i \neq j$. We use the notation V_X^i for the set of external variables of agent i.

Definition 10. Let $\mathcal{V}(C_X)$ be the set of all variables that appear in some constraints in C_X. Then each V_i can be partitioned into two disjoint sets: the *private variable* set $V_P^i = \{v_i \mid v_i \in V_i, v_i \notin \mathcal{V}(C_X)\}$ and the *shared variable* set $V_S^i = \{v_i \mid v_i \in V_i, v_i \in \mathcal{V}(C_X)\}$. The *private subnetwork* of $\mathcal{N}_i$, denoted by $\mathcal{N}_p^i$, is the subnetwork induced by V_P^i, and the *shared subnetwork* of $\mathcal{M}$, denoted by $\mathcal{M}_S$, is the subnetwork induced by $\bigcup_{\mathcal{N}_i \in \mathcal{P}} V_S^i$. $\prec_S$ is an ordering of variables of the shared subnetwork and $\prec_P^i$ is an ordering of agent i's private variables.

D△CRC, which is presented as Algorithm 2, is a distributed version of Algorithm 1. Given a distributed constraint network $\mathcal{M}$, the algorithm takes as its input agent i's part of $\mathcal{M}$ (*i.e.*, $\mathcal{N}_i = \langle V_i, D_i, C_i \rangle$) as well as private and shared elimination orderings $\prec_P^i$ and $\prec_S$. Like Algorithm 1, Algorithm 2 eliminates all variables of $\mathcal{N}_i$ and then reinstates them. Concerning private variables in V_P^i, agent i can eliminate and reinstate them independently. However, agent i needs to collaborate with other agents that are connected through external constraints so as to correctly eliminate and reinstate shared variables in V_S^i. In order to avoid using outdated information, the algorithm adopts the communication mechanism of the distributed PPC algorithm introduced in [2].

We explain the algorithm based on Example 1.

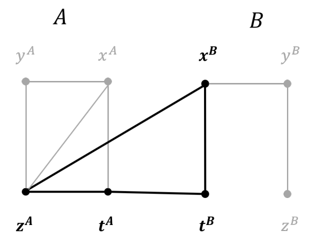
Algorithm 2: D△CRC

```
Input  : $\mathcal{N}_i = \langle V_i, D_i, C_i \rangle$: Agent $i$'s part of a distributed
         constraint network $\mathcal{M}$.
         $\prec_P^i = (v_p, \ldots, v_1)$: $\mathcal{N}_i$'s private variable elimination
         ordering.
         $\prec_S = (w_s, \ldots, w_1)$: $\mathcal{M}$'s shared variable elimination
         ordering.
Output : Agent $i$'s part of a PPC constraint network that is
         equivalent to $\mathcal{M}$.
   // Phase 1: Eliminate private variables
1  for $\ell \leftarrow p$ to 1 do
2      (Result, $\mathcal{N}_i$) ← ELIMINATE($v_\ell, \mathcal{N}_i, \prec_P^i$);
3      if $\mathcal{N}_i$ = False then
4          Broadcast "inconsistent"
5          return "inconsistent"
   // Phase 2: Eliminate shared variables
6  for $\ell \leftarrow s$ to 1 s.t. $w_\ell \in V_S^i$ do
7      foreach $w_j \in V_X^i$ s.t. $j > \ell$ and $R_{j\ell} \in C_X$ do
8          Wait for the elimination of $w_j$ by other agent and for
           the updated information about constraints involving
           $w_\ell$.
9      (Result, $\mathcal{N}_i$) ← ELIMINATE($w_\ell, \mathcal{N}_i, \prec_S$, )
10     if Result = False then
11         Broadcast "inconsistent"
12         return "inconsistent"
13     else
14         for $j, k < \ell$ s.t. $R_{j\ell}, R_{\ell k} \in C_X$ do
15             Send updated information about $R_{jk}$ to the agents
               to whom variables $w_j$ and $w_k$ belong.
   // Phase 3: Reinstate shared variables
16 for $\ell \leftarrow 1$ to $s$ s.t. $w_\ell \in V_S^i$ do
17     for $j \leftarrow \ell - 1$ to 1 s.t. $R_{j\ell} \in C_X$ do
18         for $k \leftarrow j - 1$ to 1 s.t. $R_{k\ell} \in C_X$ do
19             Wait for the reinstatement of $w_j$ and $w_k$ by
               another agents and for the updated information
               about $R_{jk}$.
20             $R_{j\ell} \leftarrow R_{j\ell} \cap (R_{jk} \circ R_{k\ell})$
21             $R_{k\ell} \leftarrow R_{k\ell} \cap (R_{kj} \circ R_{j\ell})$
22         $D_\ell \leftarrow D_\ell \cap R_{j\ell}(D_j)$
23         for $k \leftarrow \ell + 1$ to $s$ s.t. $w_k \in V_X^i$, $R_{k\ell}, R_{kj} \in C_X$ do
24             Send updated relation about $R_{j\ell}$ to the agent to
               whom variable $w_k$ belongs.
   // Phase 4: Reinstate private variables
25 for $\ell \leftarrow 1$ to $p$ do
26     $\mathcal{N}_i$ ← REINSTATE($v_\ell, \mathcal{N}_i, \prec_P^i$)
```

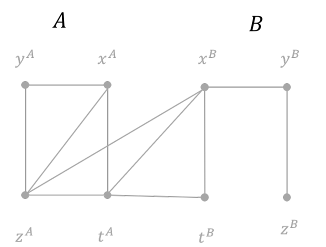
Phase 1: Agent i eliminates its private variables along the ordering $\prec_P^i$ independently (see Figure 5a).

Phase 2: Agent i eliminates its shared variables along the ordering $\prec_S$. Before eliminating a shared variable $v \in V_S^i$, agent i must wait for possible updates related to v (line 8). In Figure 5b, when agent B eliminates x^B, because z^A precedes x^B and connected to x^B, z^A should be eliminated before x^B. Therefore, agent B must wait for agent A to eliminate z^A and update $R_{t^A x^B}$.

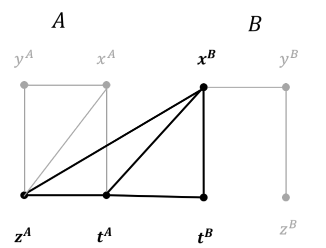
Phase 3: Agent i reinstates its shared variables along the ordering



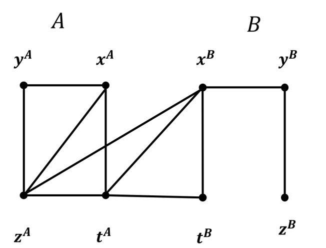
(a) A and B eliminate their own private variables following the orderings $\prec_P^A = (y^A, x^A)$ and $\prec_P^B = (z^B, y^B)$



(b) A and B collaborate on eliminating the shared variables following $\prec_S = (z^A, t^B, x^B, t^A)$;



(c) A and B collaborate on reinstating the shared variables following the ordering $(\prec_S)^{-1}$.



(d) A and B reinstate their own private variables following the orderings $(\prec_P^A)^{-1}$ and $(\prec_P^B)^{-1}$, respectively.

Figure 5: Execution of DΔCRC on Example 1.

$(\prec_S)^{-1}$. When agent i reinstates a variable $w_\ell \in V_S^i$, for any pair (w_j, w_k) of other agents' shared variables that precede w_ℓ and connected to w_ℓ through a relation, agent i must wait for all updates on R_{jk} before updating $R_{\ell j}$ and $R_{\ell k}$ using R_{jk} (line 19). In Figure 5c, when agent A reinstates variable z^A, it needs to update relations $R_{z^A x^B}$ and $R_{z^A t^A}$ using $R_{t^A x^B}$. In order to avoid using outdated information, all possible changes on $R_{t^A x^B}$ must be made beforehand.

Phase 4: Agent i reinstates its private variables following the ordering $(\prec_P^i)^{-1}$ (see Figure 5d).

We can prove the following using the proof idea in [2, Theorem 6]

THEOREM 3. *Algorithm 2 decides the consistency of its input distributed constraint network and enforces strong PPC on it if no inconsistency is detected.*

Because Algorithm 2 returns the same networks as the its centralized counterpart Algorithm 1, we can also show the following results similarly to Proposition 3 and Theorem 2,

PROPOSITION 5. *Given an input $(\mathcal{M}, \prec_P^1, \ldots, \prec_P^m, \prec_S)$, Algorithm 2 returns a network $\mathcal{M}'$ that is decomposable with respect to the inverse of $\prec = (\prec_P^1, \ldots, \prec_P^m, \prec_S)$, if $\mathcal{M}$ is consistent.*

THEOREM 4. *Algorithm 2 transforms a consistent input CRC constraint network into an equivalent one where all its constraints are minimal.*

Consequently, agents can jointly generate any solution of a consistent $\mathcal{M}$ backtrack-free by applying Algorithm 2 to it and instantiating the variables following the inverse of $\prec = (\prec_P^1, \ldots, \prec_P^m, \prec_S)$.

PROPOSITION 6. *Algorithm 2 has the same time complexity as its centralized counterpart Algorithm 1.*

PROOF. At each constraint update at most one message is sent or received. Thus the runtime added for communication is $O(e)$, where e is the number of edges of the triangulated constraint graph $G_{\mathcal{N}_i}$. Since the agents may need to wait for constraints update from other agents, in the worst case, the elimination and reinstatement of all shared variables must be done sequentially. Consequently, Algorithm 2 has the same time complexity class as Algorithm 1. □

5. EVALUATIONS

In this section we theoretically and experimentally compare our algorithm DΔCRC against the state-of-the-art algorithm D-CRC (sections 5.1 and 5.2). Furthermore, we analyze our algorithm in more detail (section 5.3).

For the experimental comparisons we implemented both DΔCRC and D-CRC on the FRODO 2.0 [13] platform that simulates parallel algorithms for CSPs on a single machine. We measured the computing time of both algorithms using the *simulated time metric* [21], which is a common metric for computing times of parallel algorithms that run in a simulated environment. In all our experiments we set for both distributed algorithms the default communication latency to zero. Our experiments were carried out on a computer with an Intel Core i5-4570 processor with a 3.2 GHz frequency per CPU core, 4 GB memory.

5.1 Theoretical Comparisons

First of all, D-CRC by Kumar *et al.* [11] is a randomized algorithm for solving CRC constraint networks. As such, D-CRC cannot detect inconsistency of an input CRC constraint network; at best, the user can stop it after a time-out to declare the input as inconsistent; nevertheless this does not guarantee the inconsistency of its input, *i.e*, it can generate false negatives. By contrast, our algorithm DΔCRC is deterministic, sound and complete.

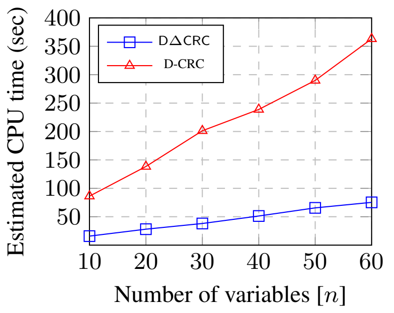
The expected time complexity of D-CRC is $O(\gamma n^2 d^2)$, where γ is the largest vertex degree of the constraint graph. By contrast, the time complexity of our algorithm DΔCRC is $O(w^*(\prec)ed)$ (see Proposition 6). Since $w^*(\prec) \leq \gamma$ and $e \leq n^2$, our algorithm DΔCRC outperforms D-CRC at least by a factor of d on average.

For our experimental comparisons, we also included a preprocessing procedure for D-CRC and procedures for generating the input orderings and solutions for DΔCRC. The theoretical time complexities of those procedures are, however, all dominated by the main algorithms.

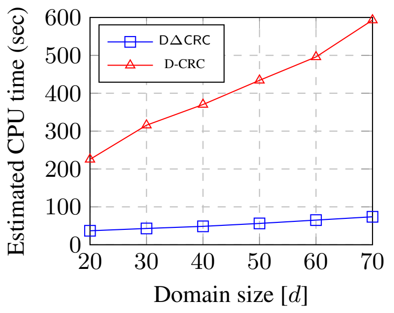
5.2 Experimental Comparisons

We considered random consistent CRC constraint networks that were used in the literature for evaluations (cf. [5], [25] and [11]). These CRC constraint networks were generated by varying three parameters that affect the time complexity of both algorithms: (i) the number n of variables; (ii) the size d of the largest domain; (iii) the density $\rho = 2|C|/n(n+1)$ of the input CRC constraint network. We fixed two from three parameters and varied the remaining parameter. When we fixed ρ we set it 0.5, as distributed problems usually involve sparse networks, $\rho = 0.5$ meaning a high density value for practical distributed problems.

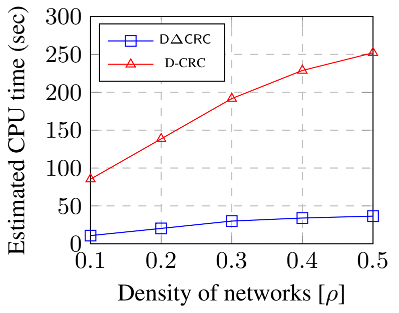
For the comparisons we assigned to each agent only one single variable (*i.e.*, the number of agents is equal to the number of variables), because D-CRC has the limitation that it does not allow each agent to possess more than one variable. By contrast, our algorithm



(a) Performance evaluation of the two algorithms in the number n of variables. We set $\rho = 0.5$ and $d = 20$.



(b) Performance evaluation of the two algorithms in the size d of domains. We set $\rho = 0.5$ and $n = 30$.



(c) Performance evaluation of the two algorithms in the density ρ of the input networks. We set $n = 30$ and $d = 20$.

Figure 6: Performance comparisons between DΔCRC and D-CRC.

DΔCRC is more flexible and can assign multiple variables to the agents. But for fair comparisons, we used the same agent setting for D-CRC and DΔCRC.

The graphs in Figures 6a–c illustrate the experimental comparisons between algorithms DΔCRC and D-CRC. The data points in each graph are computing times averaged over 20 instances.

We observe in the graphs that both algorithms show linear time behaviors with respect to n and d and a sublinear time behavior with respect to ρ. We also observe that the performance differences between the two algorithms are remarkable. DΔCRC not only runs faster than D-CRC, but it also scales up to 7 times better than D-CRC with increasing parameter values. This owes to the theoretical property of DΔCRC, *i.e.*, it leverages the input network structure and scales better also in theory with the increasing size of the domain. All in all, we can conclude that DΔCRC is more suitable than D-CRC for distributed problems.

5.3 In-depth Evaluation of DΔCRC

As mentioned previously, the setting for the experimental comparisons did not allow assign an agent to two or more variables, *i.e.*, the number of agents had to be equal to the number of variables. Therefore, we also evaluated our algorithm exclusively (see Figure 7), where we changed the number n_A of agents in the input networks. Each agent is assigned to around $\lfloor n_A/n \rfloor$ variables that are chosen randomly from the input network.

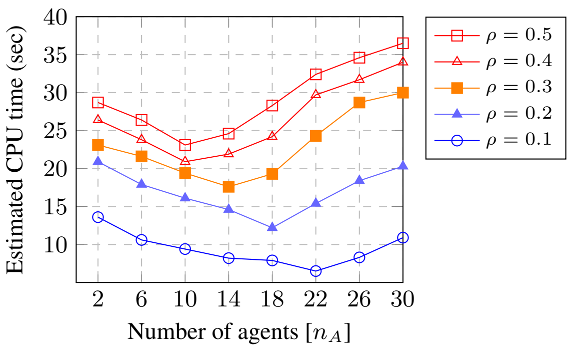


Figure 7: Evaluation of our algorithm DΔCRC for different number of agents and network density. We set $n = 30$ and $d = 20$.

We observe in Figure 7 that the performance graph of DΔCRC forms a U-shape for all the network densities. This is because more agents allows for more constraints to be handled concurrently, but from a certain number of agents on, this effect is dominated by the delays caused by the inter-agent communications.

In the figure we also notice that the number of agents needed for the optimal performance shifts to the right with decreasing network density. This phenomenon owes to the fact that networks with lower densities involve less constraints between agents, allowing the agents to handle more constraints concurrently.

The results also show that DΔCRC runs up to two times faster when the agents are assigned to two or more variables and not to only one variable. Consequently, DΔCRC can outperform D-CRC even more significantly.

6. DISCUSSION AND CONCLUSION

In this paper, we have proposed the first deterministic distributed algorithm, called DΔCRC, for solving CRC constraints. The algorithm can efficiently transform an input CRC constraint network into an equivalent constraint network, where all constraints are minimal, and can generate all solutions in a backtrack-free manner.

DΔCRC does not suffer from the problems that the state-of-the-art algorithm D-CRC has: (i) it is sound and complete and (ii) it can assign more than one variable to each agent, allowing the algorithm to solve large networks in real distributed systems. Furthermore, our theoretical and experimental comparisons showed that DΔCRC significantly outperforms D-CRC.

Since the class of CRC constraints generalizes several subclasses of constraints such as 2SAT, binary integer linear constraints, and monotone constraints [5], and can model problems in domains such as VLSI design [3], scene labelling [22] as well as logical filtering [12], our algorithm for solving CRC constraints can benefit many multi-agent applications that are based on CRC constraints.

As the class of CRC constraints is closed under a majority operation [6], one may wonder if the algorithms can be extended to other majority closed constraint languages, *e.g.*, the class of tree-preserving constraints [8]. However, since the proofs of Propositon 2 and Theorem 2 heavily rely on the Helly property, which is not enjoyed by all majority closed constraint languages, we may need to devise new methods to establish these results.

Acknowledgments

This work was partially supported by an ARC Discovery Project (No. DP120104159). The work of Jae Hee Lee was also supported by a Feodor Lynen Research Fellowship sponsored by the Alexander von Humboldt Foundation, and the work of Sanjiang Li was also supported by NSFC (No. 11671244).

REFERENCES

[1] C. Bliek and D. Sam-Haroud. Path consistency on triangulated constraint graphs. In *IJCAI*, pages 456–461, 1999.

[2] J. C. Boerkoel and E. H. Durfee. Distributed reasoning for multi-agent simple temporal problems. *J. Artif. Intell. Res.*, 47:95–156, 2013.

[3] E. Boros, P. L. Hammer, M. Minoux, and D. J. Rader. Optimal cell flipping to minimize channel density in vlsi design and pseudo-boolean optimization. *Discrete Applied Mathematics*, 90(1):69–88, 1999.

[4] J. Cox and E. Durfee. Efficient and distributable methods for solving the multiagent plan coordination problem. *Multiagent and Grid Systems*, 5(4):373–408, 2009.

[5] Y. Deville, O. Barette, and P. van Hentenryck. Constraint satisfaction over connected row-convex constraints. *Artificial Intelligence*, 109(1):243–271, 1999.

[6] P. Jeavons, D. Cohen, and M. C. Cooper. Constraints, consistency and closure. *Artificial Intelligence*, 101(1):251 – 265, 1998.

[7] U. Kjærulff. Triangulation of graphs – algorithms giving small total state space. Technical report, 1990.

[8] S. Kong, S. Li, Y. Li, and Z. Long. On tree-preserving constraints. In *CP*, pages 244–261, 2015.

[9] T. K. S. Kumar. On geometric CSPs with (near)-linear domains and max-distance constraints. In *Workshop on Modelling and Solving Problems with Constraints*, pages 125–138, 2004.

[10] T. K. S. Kumar. On the tractability of restricted disjunctive temporal problems. In *ICAPS*, pages 110–119, 2005.

[11] T. K. S. Kumar, D. T. Nguyen, W. Yeoh, and S. Koenig. A simple polynomial-time randomized distributed algorithm for connected row convex constraints. In *AAAI*, pages 2308–2314, 2014.

[12] T. K. S. Kumar and S. J. Russell. On some tractable cases of logical filtering. In *ICAPS*, pages 83–92, 2006.

[13] T. Léauté, B. Ottens, and R. Szymanek. Frodo 2.0: An open-source framework for distributed constraint optimization. In *Proceedings of the IJCAI Distributed Constraint Reasoning Workshop*, pages 160–164, 2009.

[14] J.-S. Liu and K. P. Sycara. Multiagent coordination in tightly coupled task scheduling. In *Proceedings of the Second International Conference on Multi-Agent Systems*, pages 181–188, 1996.

[15] Z. Long and S. Li. On distributive subalgebras of qualitative spatial and temporal calculi. In *COSIT 2015*, pages 354–374, 2015.

[16] Z. Long, M. Sioutis, and S. Li. Efficient path consistency algorithm for large qualitative constraint networks. In *IJCAI 2016,*, pages 1202–1208, 2016.

[17] A. K. Mackworth. Consistency in networks of relations. *Artificial intelligence*, 8(1):99–118, 1977.

[18] R. Miyashiro and T. Matsui. A polynomial-time algorithm to find an equitable home–away assignment. *Operations Research Letters*, 33(3):235–241, 2005.

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

[20] L. Planken, M. de Weerdt, and R. van der Krogt. P^3C: A new algorithm for the simple temporal problem. In *ICAPS*, pages 256–263, 2008.

[21] E. A. Sultanik, R. N. Lass, and W. C. Regli. Dcopolis: A framework for simulating and deploying distributed constraint optimization algorithms. In *Proceedings of the Workshop on Distributed Constraint Reasoning*, 2007.

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

[23] L. Xu and B. Y. Choueiry. A new efficient algorithm for solving the simple temporal problem. In *TIME*, pages 212–222, 2003.

[24] M. Yokoo, T. Ishida, E. H. Durfee, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 614–621, 1992.

[25] Y. Zhang and S. Marisetti. Solving connected row convex constraints by variable elimination. *Artificial Intelligence*, 173(12):1204–1219, 2009.