Applied Intelligence (2021) 51:1733–1746
https://doi.org/10.1007/s10489-020-01860-8

Speeding up distributed pseudo-tree optimization procedures with cross edge consistency to solve DCOPs

Mashrur Rashik[1] · Md. Musfiqur Rahman[1] · Md. Mosaddek Khan[1] · Md. Mamun-or-Rashid[1] · Long Tran-Thanh[2] · Nicholas R. Jennings[3]

Published online: 10 October 2020

Abstract

The Distributed Pseudo-tree Optimization Procedure (DPOP) is a well-known message passing algorithm that provides optimal solutions to Distributed Constraint Optimization Problems (DCOPs) in cooperative multi-agent systems. However, the traditional DCOP formulation does not consider constraints that must be satisfied (hard constraints), rather it concentrates only on constraints that place no restriction on satisfaction (soft constraints). This is a serious shortcoming as many real-world applications involve both types of constraints. Traditional DPOP algorithms are not able to benefit from the existence of hard constraints, where an additional calculation is required to handle such constraints. This results in longer runtimes. Thus scalability remains an issue. Additionally, in the standard DPOP, the agents are arranged as a Depth First Search (DFS) pseudo-tree, but recent work has shown that the construction of pseudo-trees in this way often leads to chain-like communication structures that greatly impair the algorithm's performance. To address these issues, we develop an algorithm that speeds up the DPOP algorithm by reducing the size of the messages exchanged and increases parallelism in the pseudo tree. For this purpose, initially, we improve the path for exchanging messages. Next, we introduce a new form of constraint propagation, which we call cross-edge consistency. Our theoretical evaluation shows that our proposed algorithm is complete and correct. In empirical evaluations, our algorithm achieves a significant reduction in the runtime, ranging from 4% to 96%, compared to the state-of-the-art.

Keywords Multiagent system · Distributed problem solving · Distributed constraint optimization · DPOP

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) is a commonly used framework involving multiple agents that interact with one another to achieve a common goal [23]. A number of real-world problems, such as distributed event scheduling [13], scheduling smart home devices [6] and allocating tasks in mobile sensor networks [11], can be modeled with this framework. Specifically, a DCOP consists of several distributed cost functions that collectively form a global objective function (i.e. the common goal). Each of these cost functions represents a constraint relationship among a set of variables that are controlled by the agents contributing to that constraint. In more detail, each agent is responsible for setting the value(s) of its variable(s) from a finite domain(s). However, they can communicate with their neighboring agents, and thus can influence the value assignment of each other. The goal of a DCOP solution is to set every variable to a value from its domain to minimize the number of constraint violations or maximize the global objective function.

Over the last couple of decades, a number of algorithms have been proposed to solve DCOPs. They are often classified into two types: incomplete and complete algorithms. The former experiences better computation and communication costs at the expense of solution quality. Among the incomplete DCOP algorithms DBA [9], DSA [24] and Max-Sum [4] are the most notable. Although this class of algorithms perform well in terms of computation and communication cost, a good number of applications, such as Wi-Fi Channel Assignment [16] and Reactive Network Resilience [10], cannot afford to sacrifice the quality of the solution. To solve this, researchers have developed complete algorithms. This class can be further classified as

✉ Mashrur Rashik
mashrur639@gmail.com

Extended author information available on the last page of the article.

search-based and inference-based algorithms. The former use a search technique to find the optimal solution from a set of possible assignments (e.g. SyncBB [8], ConcFB [15], ADOPT [14]). The latter, such as DPOP [17], Action-GDL [22], BrC-DPOP [5], are based on dynamic programming techniques. Among them, the Distributed Pseudo-tree Optimization Procedure (DPOP) has gained particular attention since it can often provide exact solutions for many real-life problems, often at the expense of low communication cost. This is achieved by following a synchronous message passing algorithm, where the agents exchange their utility and value assignments by following a synchronous message passing protocol.

To date, several DPOP variants have been proposed. O-DPOP [18] and MB-DPOP [19] have made improvements in terms of the memory requirements of the original algorithm and SS-DPOP [7] improves the participating agents' privacy. However, a notable issue with all of these variants is that they are not able to handle constraints that must be satisfied (i.e. hard constraints). Instead, they only deal with soft constraints. Unlike hard constraints, soft constraints poses a profit/loss for each possible value assignment to its corresponding variables. Nonetheless, hard constraints, along with soft constraints, are seen in many well-known DCOPs, such as distributed radio link frequency assignment [1] and distributed event scheduling [13]. To confront this shortcoming, two notable extensions of DPOP, H-DPOP [12] and BrC-DPOP [5], have been proposed.

In more detail, H-DPOP reduces the computation cost of DPOP by ruling out infeasible combinations of the variables, and thus generates smaller messages. Infeasibe combination of values for variables sharing a hard constraint, are those, that are restricted by that specific hard constraint. This is done by a Constraint Decision Diagram (CDD), which graphically represents a solution set for n-ary constraints [3]. To do so, H-DPOP performs join and projection operations on CDDs that are computationally expensive. At the same time, it is not possible to fully exploit hard constraints to prune the domain of a variable using this approach. Addressing these issues, BrC-DPOP introduced the notion of a Value Reachability Matrix (VRM). A VRM is a binary matrix representation of a constraint that requires a large number of matrix multiplications that largely affect the time complexity. Unfortunately, the issues of CDD, still exist for VRM to some extent. Therefore, it is worth noting that similar to the aforementioned DPOP extensions, BrC-DPOP uses a depth-first search pseudo tree to graphically represent a DCOP. Recently, it has been shown that this approach often results in a chain-like structure that impairs the performance of the algorithm due to the lack of parallelism [2]. Nevertheless, the algorithm proposed in the paper, the so-called BFS-DPOP, shows the significance of an alternative graphical representation – a breadth-first search pseudo tree. To be exact, BFS-DPOP enhances parallelism, and thus reduces the runtime of the algorithm. However, BFS-DPOP cannot handle hard constraints, and thus it is not directly applicable to BrC-DPOP.

Against this background, we propose a new variant of the DPOP algorithm, that we call Cross-Edge Consistent DPOP (CeC-DPOP). The contributions of our algorithm are as follows. Firstly, it takes advantage of increased parallelism through the use of a BFS pseudo tree as the communication structure. Our algorithm can also make use of hard constraints to reduce the domain size of a variable. In this context, unlike BrC-DPOP that enforces branch consistency, we develop a new form of consistency, called Cross-edge Consistency.[1] This particular form of consistency helps to remove nonassignable values from a variable's domain. In addition to this, we have also used arc consistency, for further reduction in domain size. This enables us to produce smaller message sizes that improve DPOP's runtime. Finally, we introduce a data structure, called a Consistency Matrix, which is used to store constraint information. Unlike VRM, the Consistency Matrix requires a smaller number of computations to remove nonassignable values from the domain of a variable. The use of a BFS pseudo-tree along with the cross-edge consistency enforced by Consistency Matrices enables CeC-DPOP to outperform the previous state-of-the-art algorithms in terms of runtime. We theoretically prove that our algorithm CeC-DPOP is complete and correct. We then evaluate the complexity of the algorithm and observe that it has polynomial-time computational complexity. Furthermore, we empirically evaluate the performance of our approach, and observe a significant reduction of runtime, up to 18-96% compared to DPOP, 10-89% compared to BFS-DPOP, 5-67% compared to BrC-DPOP, 55-75% compared to H-DPOP.

The remainder of this paper is structured as follows. We describe the problem in the section that follows. Then, in Section 3, we discuss the complete process of CeC-DPOP with a worked example. Afterward, we discuss the complexity of the algorithm in Section 4. Section 5 presents the theoretical analysis. In Section 6, we present the empirical results of our method compared to the current state-of-the-art, and Section 7 concludes.

2 Background and Problem Formulation

A DCOP model can be formally expressed as a 5-tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$ where:

[1] Cross-edge Consistency is a new form of consistency introduced in this paper for pruning out those possible values for the variables in a DCOP which cannot possibly be part of a consistent solution.

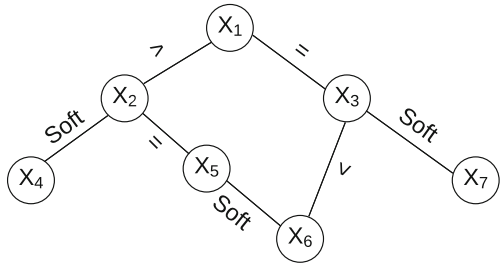


Fig. 1 A constraint graph representation of a DCOP. Here, the edges having relational operators are the hard constraints

- $\mathbf{A} = \{a_1, a_2, ..., a_k\}$ is a set of agents.
- $\mathbf{X} = \{x_1, x_2, ..., x_n\}$ is a set of variables[2], where n≥k.
- $\mathbf{D} = \{d_1, d_2, ..., d_n\}$ is a set of domains for the variables in $\mathbf{X}$, where each $d_i \in D$ is the available domain for the corresponding variable $x_i \in X$.
- $\mathbf{F} = \{f_1, f_2, ..., f_m\}$ is a set of constraint functions (also known as utility or cost functions). Constraints are used to represent the relationship among the variables and denotes the utility value for each possible assignment of those variables. In constrained optimization problems a value (penalty) is assigned for each possible value-combination of variables sharing a constraint. In a maximization problem the sum of the penalty of every constraint is maximized and it is the opposite for minimization problem. In more detail, each function $f_i(\mathbf{x_i})$ depends on a subset of variables $\mathbf{x_i} \subseteq \mathbf{X}$ that can be considered as the scope of that function. To represent the relationship among the variables in $\mathbf{x_i}$, the function $f_i(\mathbf{x_i})$ denotes the utility value for each possible assignment of those variables. Each constraint $f_i \in \mathbf{F}$ can be hard in which case the value combinations that must be avoided are denoted as 0, whereas the combinations that are allowed are represented by 1. In our paper, we used 1 to indicate a legal value assignment since we are considering a maximization problem. Relational operators like; greater than, less than, equal to, etc, are a common example of hard constraint. The remaining type is the soft constraint indicating that each value combination results in a finite utility/cost value and there is no restriction in value assignment. The dependencies among the variables can be used to construct a constraint graph that has been used to represent DCOPs graphically. In this representation, each variable is associated with a node and connected to each other through an edge.

[2]Throughout this paper, we consider agents and variables as interchangeable.

x_5	x_6	Cost
0	0	12
0	1	3
1	0	7
1	1	3

Fig. 2 A sample cost table for the soft constraint involving variable x_5 and x_6

- $\alpha : \mathbf{X} \rightarrow \mathbf{A}$ is an onto mapping function that assigns the variables $\mathbf{X}$ to the set of agents $\mathbf{A}$.

$$\mathbf{X}^* = \underset{\mathbf{X}}{\text{argmax}} \sum_{f_i \in F} f_i(\mathbf{x_i}) \quad (1)$$

Within this model, the main objective of a DCOP algorithm can be expressed as each agent assigning the values to its associated variable(s) from the corresponding domain(s) that can be expressed as $\mathbf{X}^*$, in the pursuit of the maximization or minimization of the sum of the utility functions (i.e. the global objective function). In this paper, we consider the maximization problem only (Equation 1). However, the algorithm can also be applied to a minimization problem. For example, in Fig. 1, a DCOP instance is graphically represented as a constraint graph. Here, we consider the set of variables $\mathbf{X} = \{x_1, x_2, ..., x_7\}$, each having domain $d_i = \{0, 1\}$. The cost matrix of the soft constraint involving variables x_5 and x_6 is shown in Fig. 2. The remaining constraints in the graph that are defined by relational operators are the hard constraints.

As mentioned in Section 1, DPOP is a complete, synchronous message passing algorithm for solving DCOPs. Specifically, it uses a dynamic programming technique on a DFS pseudo-tree in a distributed manner. DPOP is executed through three phases. In the first phase, a distributed DFS traversal is started from the root (held by an agent) of the constraint graph using the distributed DFS algorithm [20]. As a result, a DFS pseudo-tree structure is built where each agent labels its neighbors as parents, pseudo-parents, children or pseudo-children and edges are identified as tree or back edges. For example, after this phase, the constraint graph of Fig. 1 results in the DFS pseudo-tree of Fig. 3. The resulting pseudo-tree serves as a communication structure for the subsequent phases of DPOP. The second phase is the Util propagation phase in which each agent, starting from the leaves of the constraint graph, sends a UTIL message to its parent. The UTIL message is generated by aggregating the constraint utilities between the current node and the variables in its separator. Here, a separator is defined as the ancestors of the current node that are connected directly to this node or its descendants. Again, the utilities in the UTIL message received from its children are also aggregated with the constraint utilities of the current node. Lastly, the current

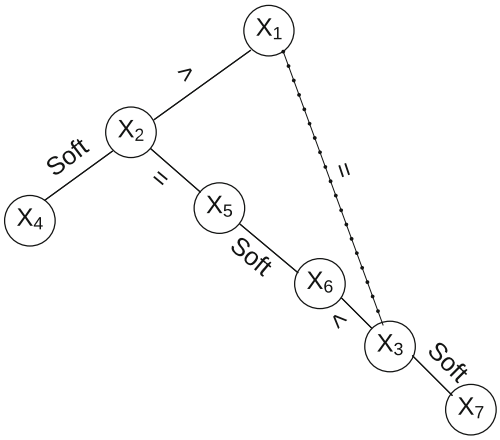


Fig. 3 DFS Pseudo-tree

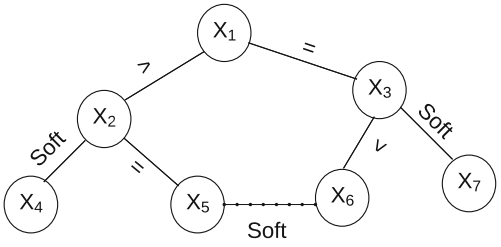


Fig. 4 BFS Pseudo-tree

node projects itself out by optimization over the received utilities. Finally, the value propagation phase is initiated by the root agent. Each agent selects its optimal assignment using the cost function computed in the UTIL propagation phase and the VALUE message received from its parent. Afterward, each agent broadcasts its assignment to its children. When every agent has chosen its optimal assignment, the algorithm terminates.

DPOP can be executed on different branches independently using a DFS pseudo-tree as its communication structure.[3] Although DPOP produces a linear number of messages, message size is exponential. This is because, a message consist of all possible value assignment to a variable pair, sharing a constraint. Another notable limitation of the DPOP algorithm is that it does not exploit hard constraints along with soft ones. These two limitations have been resolved by the BrC-DPOP algorithm [5].

In particular, to deal with hard constraints, BrC-DPOP enforces arc consistency and introduces a weaker form of the path consistency which can be applied along the path of a pseudo-tree to reduce message size. Specifically, the algorithm starts by generating a pseudo-tree structure followed by a path construction phase which is subsequently used to get the knowledge of the direct paths from each agent to its parent and pseudo-parents. In the next phase, arc consistency is enforced in a distributed manner. Then the most important phase is executed where branch consistency is exploited in a distributed way. This phase aims to ensure consistent pairs of assignable values between an agent and its pseudo-parents considering every pseudo-tree path between them. Finally, the UTIL and VALUE propagation phases are executed considering the updates of the pseudo-tree. BrC-DPOP reduces the message size due to branch-consistency enforcement, as well as faster runtime since it prunes the values of the variables. Though BrC-DPOP improves the DPOP algorithm to a greater scale, the communication structure is a DFS pseudo-tree, and as previously mentioned, this often becomes chain-like[4] in many experiments for example in Fig. 3. This condition greatly reduces the algorithm's performance. To deal with this drawback, [2] proposes the BFS-DPOP variant which uses the Breadth-First Search (BFS) pseudo-tree as the communication structure.

In more detail, BFS-DPOP operates on a Breadth-First Search (BFS) pseudo-tree that is used as the communication structure. This increases the parallelism because it produces more branches than that of its DFS counterpart. Here, Fig. 4 depicts the transformed BFS pseudo-tree of the corresponding constraint graph of Fig. 1. In BFS-DPOP, following the construction of BFS Pseudo-tree, an additional phase, namely, a cluster removal phase, is added. In this phase, for two end-points of a cross-edge, one of them is selected as the cross-edge belonger. The value of a cross-edge belonger is assigned by the agent itself. In contrast, the value of the non-belonger of a cross-edge is assigned by the root agent. Finally, the UTIL and VALUE propagation phase is executed on the BFS Pseudo-tree considering the changes that occurred in the previous phases. Even though BFS-DPOP experiences shorter communication paths, and hence less communication time, through the use of a BFS pseudo-tree, the algorithm produces messages with exponential size as the system grows. Moreover, this algorithm cannot deal with hard constraints.

To summarize, the primary issue of the current DPOP variants, based on this context, is the lack of an effective domain pruning process, in the presence of hard constraints. There is also a lack of an appropriate pseudo-tree structure, with more branches and reduced height. Our proposed approach, the Cross-Edge Consistent DPOP (CeC-DPOP), addresses both of these issues.

[3] A communication structure is a path along which DCOP message passing takes place.

[4] The work presented in [2] has shown that DFS traversal often leads to low-quality chain-like pseudo-trees with poor parallelism.

3 The Cross-Edge Consistent DPOP Algorithm (CeC-DPOP)

CeC-DPOP improves the DPOP algorithm by enforcing cross-edge consistency to reduce the domain size of the variables of a given DCOP. To be precise, cross-edge consistency is a new version of consistency, enforced by CeC-DPOP. Unlike branch-consistency, cross-edge consistency can be enforced in a shorter time. CeC-DPOP particularly achieves this through the use of a Consistency Matrix and a BFS pseudo-tree as the constraint representation data structure and the graphical representation of a DCOP, respectively. Cross-edge consistency is enforced along the path from two endpoints of a cross-edge to the lowest node, containing both these endpoints as descendants (i.e. lowest common ancestor). The use of cross-edge consistency eliminates the nonassignable values from a variable's domain. This decreases the time required to perform the join and projection operation on UTIL messages. This way, we obtain reduced message size and runtime compared to the current state-of-the-art. Moreover, by using a BFS pseudo-tree instead of a DFS pseudo-tree, CeC-DPOP can increase parallelism and shorten the tree depth. Based on the problem formulation in Section 2, we now formally define cross-edge consistency:

Definition 1 Given a BFS pseudo-tree associated with a DCOP problem instance, we define a CE relationship **(Cross-Edge relationship)** on its variables: x_i and x_j if and only the Lowest common Ancestor, $LCA(x_i, x_j) = x_{LCA}$, where $x_{LCA} \in Ancestor_i$ and $x_{LCA} \in Ancestor_j$.

Definition 2 A pair of values $(r, c) \in D_i \times D_j$ of two variables x_i, x_j that share a constraint f_{ij} is **cross edge consistent** (CeC) if and only for any sequence of variables $(x_i = x_{p_1}, x_{p_2}, \ldots, x_{LCA}, x_{c_1}, x_{c_2}, ..., x_{c_m} = x_j)$ such that $f_{p_r p_s} \in F$, $f_{c_r c_s} \in F$ where $r \leq s \leq r+1$, $par(c_s) = c_r$, $par(p_s) = p_r$ and x_i, x_j have a CE relationship (Definition 1) and there exists a tuple of values $(r = v_{k_1}, ..., v_{LCA}, ..., v_{k_m} = c)$ such that $v_{k_q} \in D_{k_q}$ and $(v_{k_p}, v_{k_q}) \in f_{k_p k_q}$, for each $1 \leq q \leq m$ and $p \leq q \leq p+1$.

Definition 3 A DCOP is cross edge consistent (CeC) if and only for any pair of variables (x_i, x_j) that share a CE relationship and any $(u, v) \in f_{ij}$, (u, v) is cross edge consistent.

Definition 4 Given a DCOP, the **Consistency Matrix** M_{ij} of two variables x_i and x_j is a binary matrix of size $D_i \times D_j$, where $M_{i,j}[r, c] = 1$ iff for a sequence of variables $(x_i = x_{p_1}, x_{p_2}, \ldots, x_{LCA}, x_{c_1}, x_{c_2}, ..., x_{c_m} = x_j)$, there exists tuple of values $(r = v_{k_1}, ..., v_{LCA}, ..., v_{k_m} = c)$ such that $v_{k_q} \in D_{k_q}$ and $(v_{k_p}, v_{k_q}) \in f_{k_p k_q}$, for each $1 \leq q \leq m$ and $p \leq q \leq p+1$.

Given this definition of cross-edge consistency, in Section 3.1, we give detailed description of the algorithm. Next, in Section 3.2, we provide a worked example of the algorithm.

Algorithm 1 Path-Construction(G_{bfs}, **P**, **C**).

Input: Pseudo tree G_{bfs}, set of parents **P**, set of child **C**.
Output: A list containing path information from current variable to enforce cross-edge consistency.

1: **for each** cross edge (x_i, x_j) of G_{bfs} **do**
2: $x_l \leftarrow LCA(x_i, x_j)$
3: send $NEXT\_UPDATE(x_l, x_i)$ to P_i
4: **if** x_i is not an end point of a cross edge **then**
5: send $NEXT\_UPDATE(NULL, x_i)$ to P_i
6: **while** $cnt\_next_i < |C_i|$ **do**
7: **if** receive $NEXT\_UPDATE(x_l, x_c)$ from $x_c \in C_i$ **then**
8: $NEXT_i \leftarrow NEXT_i \cup (x_l, x_c)$
9: **if** receive $complete(x_c)$ from $x_c \in C_i$ **then**
10: $cnt\_next_i \leftarrow cnt\_next_i + 1$
11: **if** $NEXT_i$ not equals $NULL$ **then**
12: **for each** x_l such that $(x_l, x_c) \in NEXT_i$ **do**
13: send $NEXT\_UPDATE(x_l, x_i)$ to P_i
14: send $complete(x_i)$ to P_i

3.1 Algorithm Description

CeC-DPOP consists of four phases: BFS pseudo tree construction, consistency enforcement, UTIL propagation, and VALUE propagation phase. Initially, a BFS pseudo tree is constructed from the constraint graph. In order to generate the corresponding BFS pseudo tree, we use the same method as prescribed in the BFS-DPOP algorithm. For example, Fig. 4 illustrates a sample BFS pseudo tree of the

constraint graph depicted in Fig. 1. Having a BFS pseudo tree G_{bfs} constructed, CeC-DPOP enforces arc-consistency. This phase uses the distributed Arc-Consistency (AC) algorithm that is introduced in BrC-DPOP. This algorithm results in a reduced domain for all the variables having hard constraints.

After arc-consistency is achieved, CeC-DPOP enforces a new form of consistency (i.e. cross-edge consistency) on the BFS pseudo tree. To do so, we need the lowest common ancestor $LCA(x_i, x_j)$ for every pair of variables x_i and x_j in G_{bfs}. To find the LCA of every pair of variables, we use [21], a distributed algorithm. To represent hard constraints we use Consistency Matrices (Definition 4), where a matrix M_{ij} represents a hard constraint between variables, x_i and x_j.

Algorithm 2 Cross-Edge-Consistency-Propagation(G_{bfs}, **M**, **C**, $NEXT$).

Input: Pseudo tree G_{bfs}, set of Consistency Matrix **M**, set of child **C**, next list $NEXT$.
Output: A cross edge consistent pseudo tree.

1: **if** x_i is root **then**
2: **for each** x_c in C_i **do**
3: send $CeC(x_{\mathrm{i}}, M_{\mathrm{ii}})$ to x_{c}
4: **if** received $CeC(x_{\mathrm{p}}, M_{\mathrm{pl}})$ from x_{p} such that x_p is parent of x_i **then**
5: **for each** $(x_l, x_{\mathrm{c}}) \in NEXT_{\mathrm{i}}$ **do**
6: **if** x_l equals x_{i} **then**
7: $M_{\mathrm{il}} \leftarrow M_{\mathrm{ii}}$
8: **else**
9: $M_{\mathrm{il}} \leftarrow M_{\mathrm{ip}} \times M_{\mathrm{pl}}$
10: **if** x_{c} not equals $NULL$ **then**
11: send $CeC(x_{\mathrm{i}}, M_{\mathrm{il}})$ to x_{c}
12: **for each** $x_{\mathrm{c}} \in C_{\mathrm{i}}$ such that x_{c} received no CeC_{i} message **do**
13: send $CeC(x_{\mathrm{i}}, NULL)$ to x_{c}
14: **for each** $ce_{ij} \in CE_i$ such that x_l equals LCA_{ij} **do**
15: $M_{\mathrm{lj}} \leftarrow {M_{\mathrm{jl}}}^{\mathrm{T}}$
16: $M_{\mathrm{ij}} \leftarrow M_{\mathrm{il}} \times M_{\mathrm{lj}}$

Now the algorithm enforces cross edge consistency on the pseudo tree G_{bfs}. For this, we need to construct a path for each cross-edge in G_{bfs} (Algorithm 1). In more detail, the BFS pseudo tree G_{bfs}, parent set **P** and set of child **C** are the inputs of the algorithm. We construct a list, $NEXT_i$, which contains the pair, (x_l, x_c), throughout this algorithm phase. It informs the current agent x_i about the next agent x_c to enforce cross-edge consistency for that edge whose endpoints have a LCA at x_l. The for loop in line 1 selects a cross edge having one end point x_i from G_{bfs}, and sends a message $NEXT\_UPDATE(x_l, x_i)$ to its parent P_i. This message contains information about the LCA x_l of two variables x_i and x_j and the current variable x_i. To do this, line 2 computes a LCA, x_l of x_i with another variable x_j with whom it holds cross edge. Then in line 3, x_i sends a $NEXT\_UPDATE(x_l, x_i)$ to its parent P_i. In line 4, CeC-DPOP checks whether x_i is a member of a cross edge, and if this is not the case, it sends a $NEXT\_UPDATE(NULL, x_i)$ to parent P_i. Here, NULL indicates that x_i is not an end point of any cross edge.

Afterwards, the while loop in line 6 compares a counter variable, cnt_next_i, with the child count of current variable (i.e. $|C_i|$) to check whether the current variable received a $NEXT\_UPDATE$ message from each child in C_i. Within this loop, if a $NEXT\_UPDATE(x_l, x_c)$ is received from a child then (x_l, x_c) is appended to the list, $NEXT_i$ (line 7-8). Then line 9 checks for any $complete(x_c)$ message received from a child. This message informs the current variable x_i that the path construction for the sub tree rooted at x_c is complete. For each received $complete(x_c)$ message, line 10 increments cnt_next_i by 1. The while loop terminates when each child x_c in C_i sends a $complete(x_c)$ message. Now, in line 11, the algorithm checks whether $NEXT_i$ is not empty. If this is true, the for loop in line 12 selects each (x_l, x_c) pair from the $NEXT_i$ list and line 13 sends a $NEXT\_UPDATE(x_l, x_i)$ message to P_i. Next, the algorithm terminates after sending a $complete(x_i)$ message to P_i after line 14. The $NEXT_i$ list thus computed holds information about the next variable to enforce cross-edge consistency, from current variable x_i.

Finally, we enforce cross-edge consistency on the path that is established on the pseudo tree (Algorithm 2). The BFS pseudo tree G_{bfs}, set of Consistency Matrices **M**, set of child **C** and the $NEXT$ list are the inputs of the algorithm. The algorithm works as follows. Line 1 checks whether the current variable x_i is root. If it is, it initiates CeC message propagation by iterating every child using the for loop in line 2. Line 3 then sends a $CeC(x_i, M_{ii})$ to every x_c in C_i, where x_i is the variable which sent the message along with its Consistency Matrix M_{ii}. Line 4 of the algorithm checks whether any CeC message has been received from its parent. If this is the case, line 5 iterates over each pair (x_l, x_c) of the $NEXT_i$ list to propagate a CeC message. For this purpose, lines 6-7 checks whether the current variable

equals the LCA x_l of a cross edge in the subtree. If this is the case, it initializes M_{il} with its unary constraint M_{ii} which represents the domain of the current variable x_i. Otherwise, lines 8-9 computes M_{il}, which is a multiplication of M_{ip} and M_{pl}. Next, line 10 checks whether x_c is not null. If this is true, line 11 sends a $CeC(x_i, M_{il})$ message to x_c. Now, lines 12-13 of the algorithm checks whether any child exists that did not receive any CeC message. If this is true, x_i sends a $CeC(x_i, NULL)$ to that child. Lines 14-16 finally compute the Consistency Matrices along each cross edge by iterating over every cross edge and multiplying the matrices obtained for each endpoint of the cross edge. After cross edge consistency is enforced, we obtain a set of variables with reduced domain size. Now, we execute the UTIL and VALUE propagation phase. These two steps correspond to the UTIL and VALUE propagation phases of the BFS-DPOP algorithm.

3.2 Worked Example

In this sub-section, we present a working example of the algorithm that we introduced in Section 3.1. Initially our algorithm constructs a BFS pseudo-tree (Fig. 5b) from the constraint graph (Fig. 5a). From the pseudo-tree, we observe a cross-edge $x_5 - x_6$. Then, for convenience we show CeC enforcement in a section of the original pseudo-tree (Fig. 5c). For our example, we assume each variable x_i has a domain D_i={0, 1, 2, 3, 4}. After pseudo-tree construction, the algorithm enforces arc-consistency. AC propagation reduces the domain size of a variable by removing values, which are not satisfied by the hard constraints related to that variable. The effect of AC propagation on Fig. 5c is shown in Fig. 6. Now the algorithm enforces cross-edge consistency following the steps described in the upcoming sections:

3.2.1 Path Construction

From the pseudo-tree the algorithm detects a cross-edge connecting x_5 and x_6 and finds $LCA(x_5, x_6) = x_1$. Now, agent a_5 sends $NEXT\_UPDATE(x_1, x_5)$ to its parent a_2. Afterwards a_5 sends $complete(x_5)$ to its parent a_2. This message indicates that agent a_5 has no other $NEXT\_UPDATE$ message to send to a_2. After receiving $NEXT\_UPDATE(x_1, x_5)$, agent a_2 appends (x_1, x_5) to $NEXT_2$. Now, the $NEXT_2$ list contains $\{(x_1, x_5)\}$. After updating the $NEXT$ list, agent a_2 sends $NEXT\_UPDATE(x_1, x_2)$ to its parent a_1. Agent a_2 completes its path construction by sending $complete(x_2)$ message to a_1.

In a similar process, agent a_6 sends *NEXT_UPDATE* (x_1, x_6) and later sends $complete(x_6)$ to its parent a_3. On receiving a *NEXT_UPDATE* message, agent a_3 appends (x_1, x_6) to $NEXT_3$ and sends $NEXT\_UPDATE(x_1, x_3)$ to its parent a_1. Finally, a_3 sends $complete(x_3)$ to a_1. On receiving $NEXT\_UPDATE(x_1, x_2)$ and $NEXT\_UPDATE(x_1, x_3)$, agent a_1, appends $\{(x_1, x_2), (x_1, x_3)\}$ to $NEXT_1$. The $NEXT_1$ list contains information about two agents a_2 and a_3. Likewise, the list, $NEXT_2$ and $NEXT_3$ contain information about agent a_5 and a_6 respectively. This way, we have two paths from agent a_1 to the endpoints of the cross-edge $x_5 - x_6$. The course to cross-edge consistency enforcement is now set. The path construction phase can be visualized with the help of Fig. 7. In the next phase CeC messages are propagated along these paths.

3.2.2 CeC Propagation Phase

The CeC propagation begins from the root agent, a_1. Agent a_1 sends $CeC(x_1, M_{11})$ to its child, a_2 and a_3. Agent a_2 on receiving $CeC(x_1, M_{11})$ calculates $M_{21} = M_{21} \times M_{11}$. Afterwards, agent a_2, sends $CeC(x_2, M_{21})$ to a_5. Agent a_5 on receiving this message, calculates $M_{51} = M_{52} \times M_{21}$.

Similarly, agent a_3 on receiving $CeC(x_1, M_{11})$ from a_1, calculates $M_{31} = M_{31} \times M_{11}$. Agent a_3 iterates its $NEXT_3$ list and sends $CeC(x_3, M_{31})$ to a_6. Agent a_6 on receiving this message calculates $M_{61} = M_{63} \times M_{31}$. Finally, the algorithm calculates $M_{56} = M_{51} \times M_{16}$. Consistency Matrix M_{56} gives the reduced number of assignable pairs along x_5 and x_6. In our example the algorithm results in a 76%

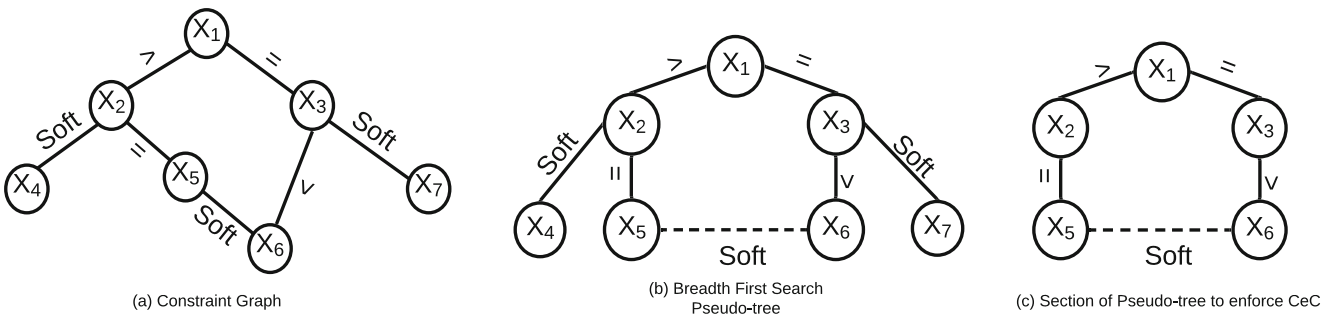
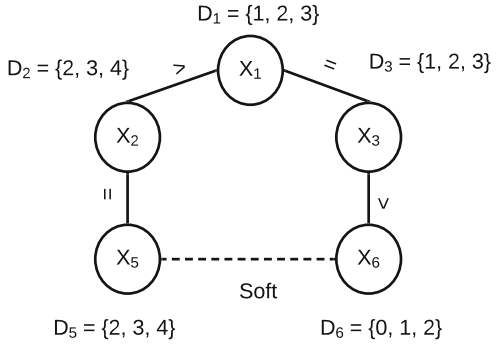


Fig. 5 Pseudo-tree construction phase



(a) Reduced Domain after AC Propagation

$X_1 < X_2$		$X_2 = X_5$	
0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
$X_3 > X_6$		$X_1 = X_3$	
0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4

(b) Reduced Assignable pairs after AC Propagation

Fig. 6 Arc consistency enforcement phase

reduction (Fig. 9). The simulation for CeC propagation in our example is shown in Fig. 8. In the next section, we will discuss the complexity analysis (Section 4) of the algorithm. Following, the complexity analysis, we provide a theoretical analysis of the algorithm, to show that CeC-DPOP is complete and correct (Section 5).

4 Complexity Analysis

Initially, we discuss the complexity of the DPOP algorithm. Given a graph and an ordering of its nodes, when processing the nodes in that order, the width of the current node is the number of neighbours that precede it in the ordering. The induced width of an ordering of nodes is the largest width of any node in that ordering. For DPOP, the complexity lies in the maximal utility message size which is exponential in the induced width of the DFS pseudo-tree, $\mathcal{O}(d^{w^*})$. Here, d is the domain size of the variables.

Improving DPOP for making use of hard constraints, BrC-DPOP employs branch consistency to reduce the domain d to d_{brc}, where $d_{brc} \leq d$. Hence, the complexity stands as $\mathcal{O}(d_{brc}^{w^*})$. On the other hand, BFS-DPOP reduces the induced width to the order of the maximal size of cross-edge cluster. The cross-edge cluster of a node is the number of cross-edges connecting the node. Thus, if the maximum size of a cross-edge cluster is $|CEC|$, the complexity of BFS-DPOP is $\mathcal{O}(|X|.d^{|CEC|+2})$. Here, $|X|$ is the number of variables.

As stated above, CeC-DPOP enforces cross-edge consistency and use a breadth-first search pseudo-tree. To enforce

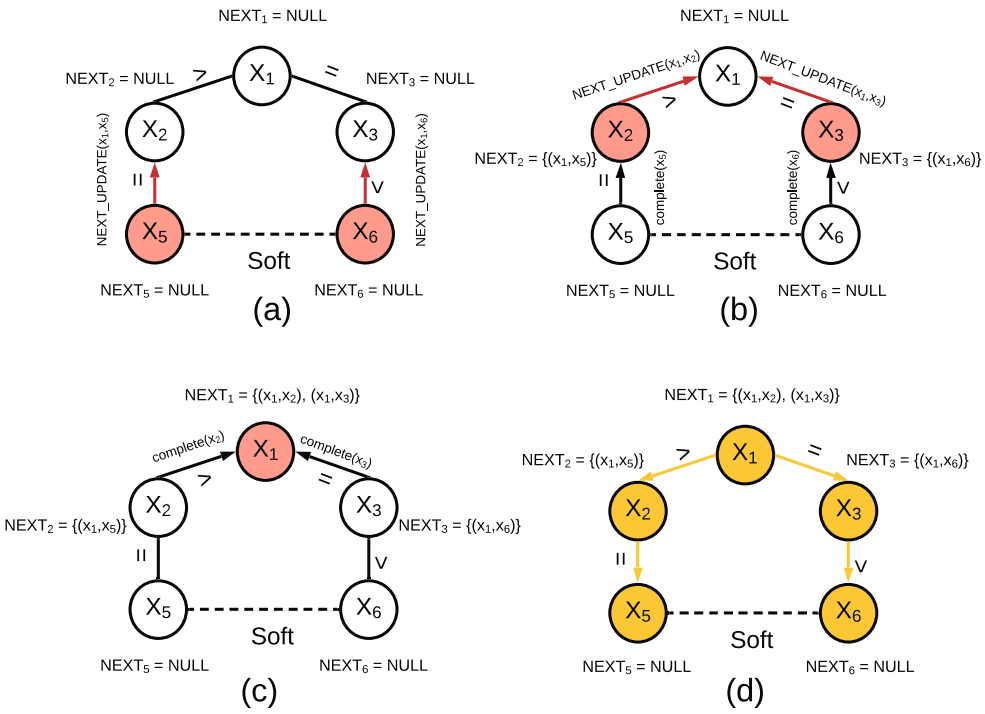


Fig. 7 Path construction

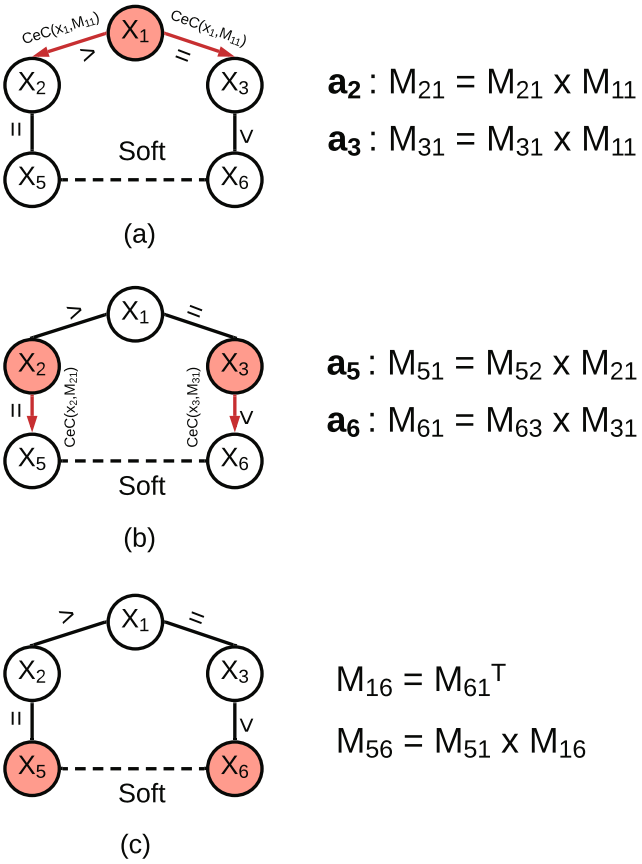
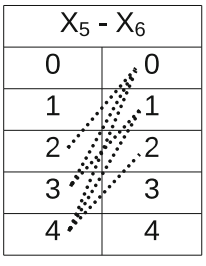


Fig. 8 Cross-Edge consistency (CeC) propagation phase

cross-edge consistency, CeC-DPOP constructs a path and then enforces arc-consistency along the path. In the path construction phase, each node sends a message to its parent containing the path information starting from one end of the cross edge to itself. This continues until the information reaches the least common ancestor of the two cross edge endpoints. Therefore, the complexity of this phase is $\mathcal{O}(|CE|\log(|X|))$, where $|CE|$ is the number of cross-edges and $|X|$ is the number of variables. This particular phase requires the lowest common ancestor for each pair of nodes associated with that path, which is found in a preprocessing phase having a complexity of $\mathcal{O}(\log(|X|))$. The next phase is the arc-consistency enforcement phase. In this phase, each hard constraint is evaluated to check whether the domain of both variables connecting the endpoints is consistent with each other. Given the number of hard constraints is C_H and the average domain of each variable is d, the complexity of this phase is $\mathcal{O}(C_H d^3)$. Here, in order to check whether each value in the domain of an endpoint is consistent with every value of the other endpoint, it requires three nested loops, resulting in d^3 computations. The final phase then enforces cross-edge consistency. In this phase, each agent waits for its parent agent to send a CeC message which it uses to find the final cross-edge consistent matrix. This requires a complexity of $\mathcal{O}(d^3)$ indicating a multiplication of two matrices. The process continues for each variable, and as such, the total complexity of the cross-edge consistency enforcement phase is $\mathcal{O}(|X|(d^3))$.



Possible assignments	5^2=25
After enforcing cross-edge consistency	6
% reduction	76%

Fig. 9 Consistent pairs along cross-edge after applying CeC-DPOP which shows an overall 76% domain size reduction (straight lines indicate a possible value assignment for x_5 and x_6)

The arc-consistency enforcement phase requires $\mathcal{O}(d|X|)$ messages, where the size of each message is $\mathcal{O}(d)$. In each step of arc-consistency enforcement, only the domain information of a variable needs to be propagated. Therefore, the cross-edge consistency enforcement phase requires $\mathcal{O}(C_H)$ messages and the size of each message is $\mathcal{O}(d^2)$. In this phase, we only propagate CeC messages each of which contains the Consistency Matrices and the size of a message depends on the size of these matrices. Therefore, the overall time complexity for enforcing cross-edge consistency is $\mathcal{O}(|CE|\log(|X|) + C_H d^3 + |X|(d^3))$ and the total number of messages exchanged is $\mathcal{O}(d|X| + C_H)$. The entire process of cross-edge consistency enforcement is insignificant to the complexity of UTIL and VALUE propagation and thus it does not affect the overall complexity. After, enforcing cross-edge consistency, the algorithm executes DPOP UTIL and VALUE propagation. For this the complexity is $\mathcal{O}(|X|.d_{cec}^{|CEC|+2})$.

5 Theoretical Analysis

In this section, we prove CeC-DPOP is both complete and correct. Similar to BFS-DPOP, CeC-DPOP makes use of the DPOP's UTIL and VALUE propagation phase, which produces an exact solution of a given DCOP. Therefore, CeC-DPOP is complete and correct if we can prove that a DCOP is arc and cross-edge consistent (Definition 2) after the AC propagation and CeC propagation phases, respectively. As we utilize the same AC propagation phase as the BrC-DPOP paper, the former is true (see [5] for the proof). Theorem 1 proves the latter.

Theorem 1 *The DCOP is cross edge consistent after the CeC propagation phase.*

Proof Let, x_i and x_j be the end points of a cross-edge. According to our path construction phase, we have two paths $x_{k_1} = x_l, ..., x_{k_n} = x_i$ and $x_{k'_1} = x_l, ..., x_{k'_m} = x_j$, where x_l is the lowest common ancestor of x_i and x_j. All the arcs in the path (x_l, x_i) and (x_l, x_j) are arc consistent after the AC propagation phase. Now, CeC-DPOP enforces cross-edge consistency along these paths and we obtain Consistency Matrices, M_{li}, M_{lj} from the paths (x_l, x_i) and (x_l, x_j), respectively (Algorithm 2: line 9).

$$M_{li} = M_{lk_2} \times M_{k_2k_3} \times ..... \times M_{k_{n-1}i} \tag{2}$$

$$M_{lj} = M_{lk'_2} \times M_{k'_2k'_3} \times ..... \times M_{k'_{m-1}j} \tag{3}$$

Finally, from lines 15-16 (Algorithm 2), we obtain the Consistency Matrix M_{ij} using the following equation: $M_{lj} = M^T_{jl}$.

$$M_{ij} = M_{il} \times M_{lj} \tag{4}$$

Matrix M_{ij} is cross edge consistent because M_{il} defines that x_i and x_l are path consistent and M_{lj} defines that x_l and x_j are path consistent by definition. So multiplying them will make x_i and x_j path consistent which in effect is cross edge consistent. Since this is true for a single cross-edge, it is also true for every other cross-edge. Therefore, the given DCOP is cross-edge consistent after cross-edge consistency enforcement. □

6 Experimental Results

We now empirically evaluate how much performance improvement can be attained using CeC-DPOP in comparison to the original DPOP algorithm and three important variants of DPOP (BFS-DPOP, BrC-DPOP, and H-DPOP). Unlike CeC-DPOP, the original DPOP uses a DFS pseudo-tree as the communication structure and does not actively exploit hard constraints. Therefore, it is reasonable to observe the attributes of CeC-DPOP (i.e. inclusion of soft constraints along with hard constraints and the use of BFS pseudo-tree as the communication structure) with respect to the original DPOP. Additionally, we consider the BFS-DPOP algorithm as a benchmark because it also uses a BFS pseudo-tree as the communication structure. Finally, we compare CeC-DPOP with BrC-DPOP and H-DPOP as these algorithms can deal with DCOPs having both types of constraints. To benchmark the runtime of our algorithm CeC-DPOP, we run our experiments on two standard types of DCOP settings: random constraint graphs and the distributed RLFA problems. We are particularly influenced by the BrC-DPOP paper in choosing the above experimental settings.

In the case of random DCOPs, the runtime of the algorithms are reported varying four parameters: domain size, graph density, the density of hard constraints (i.e. the ratio of the number of hard constraints and $n * (n-1)/2$ where n is the number of variables) and the number of variables. For the first parameter, we vary the domain size from 26 to 40 in Fig. 10. Here we set the number of variables X at 20 and edges are created by taking pairs of variables randomly and connecting them considering a fixed graph density, $\rho = 0.5$, and fixed hard constraint density 0.4. For the second parameter, we execute the algorithms by changing the graph density from 0.2 to 0.9 in Fig. 11. Here, we set the number of variables X at 20, domain size, D at 10 and hard constraint density at 0.4. Then for the third parameter, we increase the density of hard constraints from 0.2 to 0.9 in Fig. 12 setting the number of variables at 20, domain size at 10 and graph density at 0.4. Finally, we vary the number of variables, setting graph density $\rho = 0.5$, domain size, D at 10 and hard constraint density at 0.4. In our experiments, we generate 30 instances and calculate the average runtime of running each of the algorithms. We also recorded standard errors to ensure statistical significance. For the third experiment (Fig. 12), we only compared CeC-DPOP with BrC-DPOP and H-DPOP. Amongst DPOP, BFS-DPOP, BrC-DPOP, and H-DPOP, only BrC-DPOP and H-DPOP can make use of hard constraints to prune domain and thus, varying density of hard constraints (keeping the remaining parameters constant) would not affect the runtime of DPOP and BFS-DPOP. Therefore, for our third experiment, we used BrC-DPOP as the benchmark. All of the experiments were performed on a simulator implemented in an Intel $i7$ Octacore $3.4GHz$ machine with $16GB$ of RAM.

Our experimental results for solving random DCOPs are depicted in Figs. 10 – 13. Specifically, we use hard constraints that are either "less than", "greater than" or "equal" alongside soft constraints for which we randomly generated utility values from the range [0, 100]. In Fig. 10, we observe the runtime of CeC-DPOP by varying the

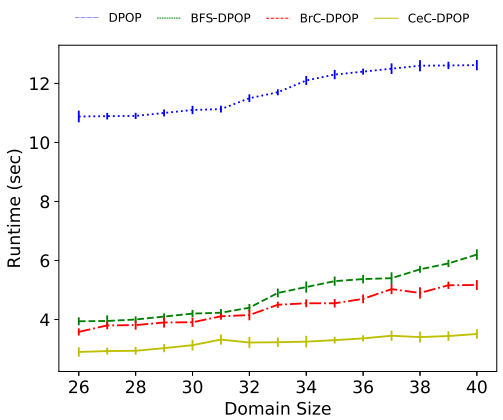


Fig. 10 Domain size vs runtime (Random DCOP)

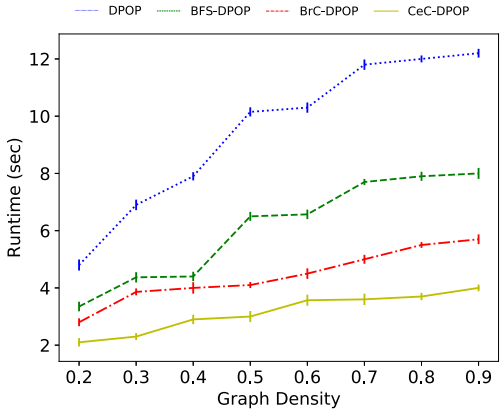


Fig. 11 Graph density vs runtime (Random DCOP)

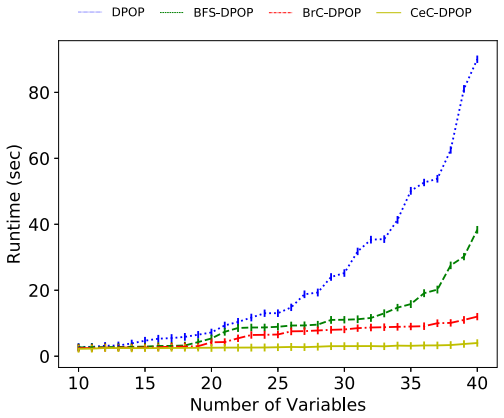


Fig. 13 Number of variables vs runtime (Random DCOP)

domain size. The reason behind this performance is that when the domain size increases, more values in each domain are pruned by CeC-DPOP through consistency enforcement which produces UTIL message of smaller dimensions. As a result, the required time to compute messages decreases at a significant rate. Though BrC-DPOP has a relatively smaller runtime than DPOP and BFS-DPOP for enforcing branch consistency, CeC-DPOP always outperforms through enforcing cross edge consistency. Here we observe that the runtime of CeC-DPOP is $70-73\%$ smaller than DPOP, $22-43\%$ than BFS-DPOP and $19-33\%$ than BrC-DPOP.

Figure 11 illustrates the results based on the next setting; that is, varying the graph density while setting the number of nodes, domain size and density of hard constraint as constants. Here we observe that the runtime of CeC-DPOP is $56-70\%$ smaller than DPOP, $34-54\%$ than BFS-DPOP and $21-40\%$ than BrC-DPOP. This behavior is explained by the fact that CeC-DPOP uses a BFS pseudo-tree as the communication structure which is generated from dense constraint graphs which give more branches. As a result, more parallelism is experienced. Another reason is that the number of edges is relatively higher in the dense constraint graphs, creating the opportunity of cross-edge consistency enforcement at a significant level. Thus, more domain values are pruned and shorter messages are produced resulting in smaller computation time. Overall, a significant reduction in runtime is observed.

In the third experimental setting, we vary the density of hard constraints and set the other three parameters as constants (Fig. 12). We observe a notable performance gain of CeC-DPOP in terms of runtime compared to the other algorithms. In particular, we detect a $16-20\%$ reduction of runtime in comparison to BrC-DPOP and a $55-75\%$ reduction in comparison to H-DPOP. With the increase in the density of hard constraints, CeC-DPOP is able to prune more values, compared to BrC-DPOP and H-DPOP. This results in shorter message sizes and an overall reduction in runtime. Finally, we vary the number of variables and set the other three parameters as a constant (Fig. 13). The results obtained are similar to the first three experiments on random DCOPs. More specifically, CeC-DPOP outperforms other algorithms contributing a $18-96\%$, $10-89\%$ and $5-67\%$ reduction in runtime relative to DPOP, BFS-DPOP, and BrC-DPOP.

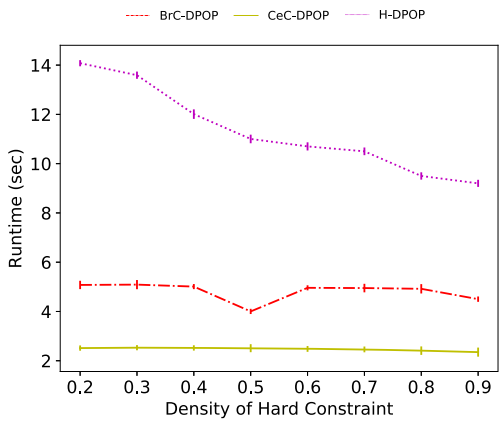


Fig. 12 Density of hard constraint vs runtime (Random DCOP)

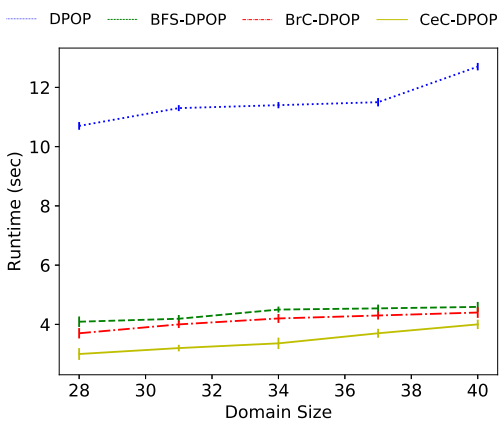


Fig. 14 domain size vs runtime (RLFA Problem)

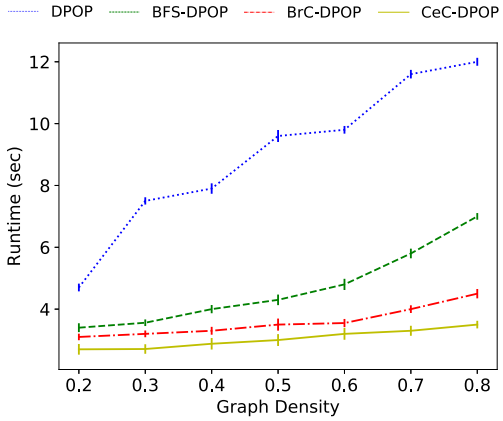


Fig. 15 graph density vs runtime (RLFA Problem)

As already mentioned, the Distributed RLFA Problem is the second type of problem to evaluate CeC-DPOP against the benchmarking algorithms. The distributed RLFA problem [1] consists of a set of channels, each having a transmitter and receiver at both ends. The aim is to assign a frequency from a given set F by minimizing the total interference at the receivers below an acceptable level (hard constraints) and at the same time using as few and also as low frequencies as possible (soft constraints). For our experiment, we mapped a transmitter as a variable and for simplicity, we assigned a single agent to a variable. The domain of a variable consists of frequencies (chosen from available spectral resources) that can be assigned to a variable. The interference between transmitter is modeled as a constraint of the form $x_i - x_j > s$ where x_i, x_j are variables and s is a random frequency separation. For this problem, we varied three parameters: domain size, graph density and the number of variables.

In Fig. 14 we varied the domain size, setting the number of variables at 20, $s \in \{3, 4\}$ and graph density, $\rho = 0.5$. The

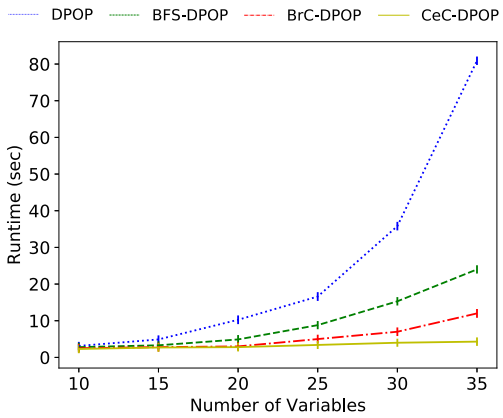


Fig. 16 number of variable vs runtime (RLFA Problem)

Table 1 Total number of assignable pairs after AC propagation and CeC propagation varying the domain size on random DCOPs

Domain size	After AC propagation	After CeC propagation
10	16384	14887
20	79524	77535
30	274576	262037
40	492804	489839

results are similar to those observed for random DCOPs. Here we observe that the runtime of CeC-DPOP is 68 − 72% smaller than DPOP, 13 − 27% than BFS-DPOP and 9 − 20% than BrC-DPOP. Next, in Fig. 15, we varied the graph density, setting number of variables at 20, $s \in \{3, 4\}$ and domain size, $D = 10$. Here we observe that the runtime of CeC-DPOP is 43 − 70% smaller than DPOP, 21 − 50% than BFS-DPOP and 10−22% than BrC-DPOP. Lastly, we varied the number of variables in Fig. 16, setting graph density, $\rho = 0.5$, $s \in \{3, 4\}$ and domain size, $D = 10$. Here we observe that the runtime of CeC-DPOP is 27 − 95% smaller than DPOP, 18 − 82% than BFS-DPOP and 4 − 64% than BrC-DPOP.

In order to compare CeC propagation to AC propagation, we conducted another experiment. Here, we varied the domain size of a random DCOP setting the number of variables at 20, hard constraint density, 0.4 and graph density, $\rho = 0.5$. We recorded the total number of assignable pairs after AC and CeC propagation. The results are shown in Table 1. This shows, even after AC propagation, we get an additional 30 − 40% reduction after CeC propagation (on average). This further reduction is observed since AC propagation is not entirely capable of removing all non-assignable value pairs. A summary of the experimental analysis is shown in Table 2. Here, we also

Table 2 Summary of experimental results showing the reduction of the runtime of CeC-DPOP in comparison to the other algorithms

	DPOP	BFS-DPOP	BrC-DPOP	H-DPOP
Random DCOPs				
Number of variables	18-96%	10-89%	5-67%	–
Domain Size	70-73%	22-43%	19-33%	–
Graph density	56-70%	34-54%	21-40%	–
Density of hard-constraints	–	–	16-20%	55-75%
Distributed RLFA problem				
Number of variables	27-95%	18-82%	4-64%	–
Domain size	68-72%	13-27%	9-20%	–
Graph density	43-70%	21-50%	10-22%	–

show the reduced runtime of CeC-DPOP in comparison to the benchmarks.

7 Conclusions

DCOPs have been used to solve different multi-agent coordination problems since the last one and a half decade. The existing algorithms that deal with DCOPs do not actively utilize hard constraints in solving such problems. To address the shortcoming, We present a new algorithm, CeC-DPOP, that significantly reduces the runtime of the DPOP algorithm that can be used to solve DCOPs having both soft and hard constraints. This is possible due to the introduction of a new type of consistency, that we call cross edge consistency. Additionally, CeC-DPOP uses BFS pseudo-tree as the communication structure that further accelerate the message passing process through enhance parallelism. Finally, we empirically observe that our algorithm performs around 5-96% faster than the current state-of-the-art algorithms.

Now, the contribution of our paper have thrown up few new questions that need further investigation. In the future, we intend to investigate how much speedup can be achieved applying our approach to other DPOP extensions (e.g. O-DPOP, MB-DPOP, SS-DPOP). Moreover, we intend to reduce the size of the consistency matrix while maintaining the quality of the solution. In so doing, we can make our approach more compatible with higher dimension constraints; and in effect, further reduce memory requirements of a given DPOP algorithm. As a result, CeC-DPOP would extend the use of DPOP in solving real-life problems including Distributed RLFA problems that include both hard and soft constraints. Furthermore, since the cross edge consistency has only been applied to exact inference based DCOP algorithm, we would like to further investigate whether it can be tailored to different non-exact DCOP algorithms.

References

1. Cabon B, De Givry S, Lobjois L, Schiex T, Warners JP (1999) Radio link frequency assignment. Constraints 4(1):79–89
2. Chen Z, He Z, He C (2017) An improved dpop algorithm based on breadth first search pseudo-tree for distributed constraint optimization. Appl Intell 47(3):607–623
3. Cheng KC, Yap RH (2005) Constrained decision diagrams. In: Proceedings of the national conference on artificial intelligence, vol 20. AAAI Press, MIT Press, 1999, Menlo Park, p 366
4. Farinelli A, Rogers A, Petcu A, Jennings NR (2008) Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, pp 639–646
5. Fioretto F, Le T, Yeoh W, Pontelli E, Son TC (2014) Improving dpop with branch consistency for solving distributed constraint optimization problems. In: International conference on principles and practice of constraint programming, Springer, pp 307–323
6. Fioretto F, Yeoh W, Pontelli E (2017) A multiagent system approach to scheduling devices in smart homes. In: Proceedings of the 16th conference on autonomous agents and multiagent systems, International Foundation for Autonomous Agents and Multiagent Systems, pp 981–989
7. Greenstadt R, Grosz B, Smith MD (2007) Ssdpop: improving the privacy of dcop with secret sharing. In: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, ACM, pp 171
8. Hirayama K, Yokoo M (1997) Distributed partial constraint satisfaction problem. In: International conference on principles and practice of constraint programming, Springer, pp 222–236
9. Hirayama K, Yokoo M (2005) The distributed breakout algorithms. Artif Intell 161(1-2):89–115
10. de la Hoz E, Gimenez-Guzman JM, Marsa-Maestre I, Cruz-Piris L, Orden D (2017) A distributed, multi-agent approach to reactive network resilience. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, International Foundation for Autonomous Agents and Multiagent Systems, pp 1044–1053
11. Jain M, Taylor M, Tambe M, Yokoo M (2009) Dcops meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. In: Twenty-first international joint conference on artificial intelligence
12. Kumar A, Petcu A, Faltings B (2008) H-dpop: Using hard constraints for search space pruning in dcop. In: AAAI, pp 325–330
13. Maheswaran RT, Tambe M, Bowring E, Pearce JP, Varakantham P (2004) Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In: Proceedings of the Third international joint conference on autonomous agents and multiagent systems-Volume 1, IEEE Computer Society, pp 310–317
14. Modi PJ, Shen WM, Tambe M, Yokoo M (2005) Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artif Intell 161(1-2):149–180
15. Netzer A, Grubshtein A, Meisels A (2012) Concurrent forward bounding for distributed constraint optimization problems. Artif Intell 193:186–216
16. Orden D, Gimenez-Guzman J, Marsa-Maestre I, de la Hoz E (2018) Spectrum graph coloring and applications to wi-fi channel assignment. Symmetry 10(3):65
17. Petcu A, Faltings B (2005) A scalable method for multiagent constraint optimization. Tech rep
18. Petcu A, Faltings B (2006) Odpop: an algorithm for open/distributed constraint optimization. In: AAAI, vol 6, pp 703–708
19. Petcu A, Faltings B (2007) Mb-dpop: a new memory-bounded algorithm for distributed optimization. In: IJCAI, pp 1452–1457
20. Petcu A, Faltings B, Parkes DC (2008) M-dpop: Faithful distributed implementation of efficient social choice problems. J Artif Intell Res 32:705–755
21. Schieber B, Vishkin U (1988) On finding lowest common ancestors: Simplification and parallelization. SIAM J Comput 17(6):1253–1262
22. Vinyals M, Rodriguez-Aguilar JA, Cerquides J (2009) Generalizing dpop: Action-gdl, a new complete algorithm for dcops. In: Proceedings of The 8th international conference on autonomous

agents and multiagent systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, pp 1239–1240
23. Yokoo M, Durfee EH, Ishida T, Kuwabara K (1998) The distributed constraint satisfaction problem: Formalization and algorithms. IEEE Transactions on knowledge and data engineering 10(5):673–685
24. Zhang W, Wang G, Xing Z, Wittenburg L (2005) Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artif Intell 161(1-2):55–87

Affiliations

Mashrur Rashik[1] · Md. Musfiqur Rahman[1] · Md. Mosaddek Khan[1] · Md. Mamun-or-Rashid[1] · Long Tran-Thanh[2] · Nicholas R. Jennings[3]

Md. Musfiqur Rahman
musfiq14shohan@gmail.com

Md. Mosaddek Khan
mosaddek@du.ac.bd

Md. Mamun-or-Rashid
mamun@cse.du.ac.bd

Long Tran-Thanh
ltt08r@ecs.soton.ac.uk

Nicholas R. Jennings
n.jennings@imperial.ac.uk

[1] Department of Computer Science and Engineering, University of Dhaka, Dhaka, Bangladesh

[2] School of Electronics and Computer Science, University of Southampton, Southampton, UK

[3] Departments of Computing and Electrical and Electronic Engineering, Imperial College London, London, UK