A Local Search Based Approach to Solve Continuous DCOPs

Amit Sarker
Department of Computer Science and Engineering, University of Dhaka
Dhaka, Bangladesh
amitcsedu99@gmail.com

Moumita Choudhury
Department of Computer Science and Engineering, University of Dhaka
Dhaka, Bangladesh
moumitach22@gmail.com

Md. Mosaddek Khan
Department of Computer Science and Engineering, University of Dhaka
Dhaka, Bangladesh
mosaddek@du.ac.bd

ABSTRACT

Distributed Constraint Optimization Problems (DCOPs) are a suitable formulation for coordinating interactions (i.e. constraints) in cooperative multi-agent systems. The traditional DCOP model assumes that variables owned by the agents can take only discrete values and constraints' cost functions are defined for every possible value assignment of a set of variables. While this formulation is often reasonable, there are many applications where the decision variables are continuous-valued and constraints are in functional form. To overcome this limitation, Continuous DCOPs (C-DCOPs), an extension of the DCOPs model has been proposed that is able to formulate problems having continuous variables. The existing methods for solving C-DCOPs come with a huge computation and communication overhead. In this paper, we apply continuous non-linear optimization methods on Cooperative Constraint Approximation (CoCoA) algorithm, which is a non-iterative, fast incomplete local search approach for solving DCOPs. We empirically show that our algorithm is able to provide high-quality solutions at the expense of smaller communication cost and execution time compared to the state-of-the-art C-DCOP algorithms.

KEYWORDS

Distributed Problem Solving; DCOPs; C-DCOPs

ACM Reference Format:
Amit Sarker, Moumita Choudhury, and Md. Mosaddek Khan. 2021. A Local Search Based Approach to Solve Continuous DCOPs. In *Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

Distributed Constraint Optimization Problems (DCOPs) are a powerful framework to model cooperative multi-agent systems wherein multiple agents communicate directly or indirectly with each other. The agents act autonomously in a common environment in order to optimize a global objective which is an aggregation of their corresponding constraint cost functions. Each of the functions is associated with a set of variables controlled by the corresponding agents. In DCOPs, agents need to coordinate value assignments to their variables in such a way that maximize their aggregated utility or minimize the overall cost [6, 18, 19]. A number of multi-agent coordination problems, such as meeting scheduling [15], sensor networks [4, 27], multi-robot coordination [25] and smart homes [7, 21], have been dealt with this model.

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

Over more than a decade, a number of algorithms have been proposed to solve classical DCOPs. These approaches can be broadly classified as exact and non-exact algorithms. On the one hand, Synchronous Branch-and-Bound (SyncBB) [10], Asynchronous Distributed OPTimization (ADOPT) [18], Distributed Pseudo-tree Optimization Procedure (DPOP) [19, 20] are some of the former type of algorithms. They are able to find the global optimal solution of a given DCOP at the expense of either or both exponential memory and computational cost since DCOPs are NP-Hard. On the other hand, non-exact algorithms, such as DSA [26], MGM & MGM2 [14], Max-Sum [5, 13], CoCoA [23], AED [16], MIF-DCOP [17], and ACO_DCOP [1] trade solution quality for lower memory requirements as well as shorter execution time.

The traditional DCOP model is based on an assumption; that is, each of the variables that are involved in the constraints can take values from the discrete domain(s) and a constraint is typically represented in a cost (i.e. utility) table. Nevertheless, several applications, such as target tracking sensor orientation [8], cooperative air and ground surveillance [9], network coverage using low duty-cycled sensors [12] and many others besides, can be best modeled with continuous-valued variables. Therefore, the traditional DCOP formulation is not well-suited to such applications. To deal with this issue, the regular DCOP model is extended for continuous-valued variables [22]. Notably, there are two key differences between DCOPs and Continuous DCOPs (C-DCOPs). Firstly, variables controlled by the agents are continuous in C-DCOPs rather than discrete in traditional DCOPs, and as such, a variable can take any value between an upper bound and a lower bound of its domain. Secondly, the constraints utilities are represented in a functional form in C-DCOPs instead of a tabular form, which has been commonly seen in traditional DCOPs.

In more detail, [22] propose a new version of the Max-Sum algorithm [5] in order to solve continuous-valued DCOPs, and that is named as Continuous Max-Sum (CMS). CMS approximates the constraint utility functions as piece-wise linear functions. However, this approximation has not been become popular due to the lack of availability of real-world applications having piece-wise linear functions. Afterward, Hybrid CMS (HCMS) [24] is proposed, which uses discrete Max-Sum as the underlying algorithmic framework with the addition of a continuous non-linear optimization method. Later, [2] propose a Particle Swarm Based Algorithm (PFD) to solve Continuous DCOPs, while [11] have made the latest contribution to this field. In this paper, the authors propose an exact algorithm namely Exact Continuous DPOP (EC-DPOP), three non-exact methods – inference-based Approximate Continuous DPOP (AC-DPOP), Clustered AC-DPOP (CAC-DPOP), and a local search based algorithm, named Continuous DSA (C-DSA). The main limitation of these algorithms is that both AC-DPOP and CAC-DPOP incur exponential

memory and computation overhead even though CAC-DPOP cuts the communication cost by providing a bound on message size.

In the wake of the above background, we extend the Cooperative Constraint Approximation (CoCoA) [23] algorithm so that it can solve Continuous DCOPs. We particularly inspired tailoring CoCoA for C-DCOPs due to two main reasons. Firstly, it is a non-iterative, semi-greedy local search based approach that is able to find high-quality solutions with a smaller communication overhead. Secondly, it is already reported in [23] that the original DSA algorithm (i.e. the traditional DCOP version of C-DSA) is significantly outperformed by CoCoA. Moreover, our continuous version of CoCoA, which we call C-CoCoA, is an approximate local search algorithm that can solve C-DCOPs without any restriction on the graph structure and with a very lower communication cost. We empirically evaluate the performance of C-CoCoA in terms of solution quality, number of messages, and completion time. We show that C-CoCoA provides up to 25% better solution, requires up to 268 times and 3.23 times fewer messages and less execution time, respectively, compared to the state-of-the-art C-DCOP algorithms.

The remainder of this paper is structured as follows. In Section 2, we describe the background and research challenges. In Section 3, we discuss the details of the Continuous Cooperative Constraint Approximation (C-CoCoA) algorithm with a worked example. Afterward, Section 4 presents the theoretical analysis. Section 5 then reports the empirical evaluation of our approach as opposed to the current state-of-the-art, and Section 6 concludes.

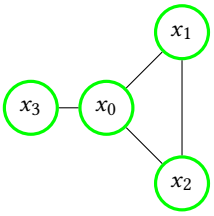
2 BACKGROUND AND PROBLEM FORMULATION

In this section, we formulate the problem and discuss the background necessary to understand our proposed method. We first describe the general DCOP framework and then move to the C-DCOP framework, which is our problem of interest in this paper. We then discuss the CoCoA algorithm and the challenges in incorporating CoCoA with the C-DCOP framework.

2.1 Distributed Constraint Optimization Problem

A Distributed Constraint Optimization Problem (DCOP) can be defined as a tuple $\langle A, X, D, F, \alpha \rangle$ [18] where,

- A is a set of agents $\{a_1, a_2, \ldots, a_n\}$.
- X is a set of discrete variables $\{x_1, x_2, \ldots, x_m\}$, where each variable x_j is controlled by one of the agents $a_i \in A$.
- D is a set of discrete domains $\{D_1, D_2, \ldots, D_m\}$, where each D_i corresponds to the domain of variable x_i.
- F is a set of cost functions $\{f_1, f_2, \ldots, f_l\}$, where each $f_i \in F$ is defined over a subset $x^i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ of variables X, called the *scope* of the function, and the cost for the function f_i is defined for every possible value assignment of x^i, that is, f_i: $D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}$, where the arity of the function f_i is k. In this paper, we consider only binary cost functions (i.e., there are only two variables in the scope of all functions).
- $\alpha : X \rightarrow A$ is a variable-to-agent mapping function that assigns the control of each variable $x_j \in X$ to an agent $a_i \in A$. Each agent can hold several variables. However, for ease of understanding, we assume each agent controls only one variable in this paper.



(a) Constraint Graph

$$\forall x_i \in X : D_i = [-20, 20]$$
$$f(x_0, x_1) = x_0^2 - 2x_0x_1 + 2x_1^2$$
$$f(x_0, x_2) = x_0x_2 + 3x_2^2$$
$$f(x_0, x_3) = x_0x_3 + x_3^2$$
$$f(x_1, x_2) = x_1^2 - x_1x_2 + 2x_2^2$$

(b) Cost Functions

Figure 1: Example of a C-DCOP

An optimal solution of a DCOP is an assignment X^* that minimizes the sum of cost functions as shown in Equation 1:[1]

$$X^* = \underset{X}{\operatorname{argmin}} \sum_{f_i \in F} f_i(x^i) \tag{1}$$

2.2 Continuous Distributed Constraint Optimization Problem

Similar to the DCOP formulation, C-DCOPs can be defined as a tuple $\langle A, X, D, F, \alpha \rangle$ [11]. In C-DCOPs, A, F, and α are the same as defined in DCOPs. Nonetheless, the set of variables X and the set of domains D are defined as follows:

- X is the set of *continuous* variables $\{x_1, x_2, \ldots, x_m\}$, where each variable x_j is controlled by one of the agents $a_i \in A$.
- D is a set of *continuous* domains $\{D_1, D_2, \ldots, D_m\}$, where each $D_i = [LB_i, UB_i]$ corresponds to the domain of variable x_i. In other words, variable x_i can take on any value in the range of LB_i to UB_i.

As discussed in the previous section, a notable difference between DCOPs and C-DCOPs can be found in the representation of the cost functions. In DCOPs, the cost functions are conventionally represented in the form of a table, while in C-DCOPs, they are represented in the form of a function [11]. However, the goal of C-DCOP remains the same as depicted in Equation 1. Figure 1 presents an example C-DCOP, where Figure 1(a) shows a constraint graph with four variables with each variable x_i controlled by an agent a_i. Each edge in the constraint graph represents a cost function and the definition of each function is shown in Figure 1(b). In this particular example, the domains of all variables are the same – each variable x_i can take values from the range $[-20, 20]$.

2.3 Cooperative Constraint Approximation

The Cooperative Constraint Approximation (CoCoA) algorithm starts with randomly activating an agent. Upon activation, the agent sends an **InquiryMessage** to its neighboring agents. We define the set of direct neighbors of the agent a_i is $\mathcal{N}_i$. When an agent

[1] For a maximization problem, the argmin operator should be replaced by the argmax operator.

a_i sends an **InquiryMessage** to the neighboring agents $a_j \in \mathcal{N}_i$, each a_j calculates cost maps for every value in the domain of a_i using the following equation:

$$\zeta_{j,k} = \min_{x_{j,l} \in D_j} \sum_{C \in F_j} C(\widetilde{x_j} \cap x_{i,k} \cap x_{j,l}) \tag{2}$$

Where $\zeta_{j,k}$ is the cost for the k^{th} value of agent a_i's domain which is calculated by the neighbor a_j, $x_{j,l}$ indicates that x_j is assigned the l^{th} value of a_j's domain[2], D_j, C is the cost for the function which is an element of the constraint function set F_j between agent a_j and one of the agent $\in \mathcal{N}_j$, $\widetilde{x_j}$ is the Current Partial Assignment (CPA) that contains the known assigned values of the neighbors of a_j, $x_{i,k}$ indicates that x_i is assigned the k^{th} value of agent a_i's domain D_i. Agent a_j calculates $\zeta_{j,k}$ for all the values of $k \in D_i$ and the resulting cost map $\zeta_j = \{\zeta_{j,1}, \zeta_{j,2}, \ldots, \zeta_{j,|D_i|}\}$ is sent to the inquiring agent a_i via **CostMessage**. Then, a_i finds the value of its variable x_i from Equation 3.

$$\delta = min \sum_{j=1}^{|\mathcal{N}_i|} \zeta_{j,k}; \qquad \rho = \{k : \sum_{j=1}^{|\mathcal{N}_i|} \zeta_{j,k} = \delta\} \tag{3}$$

In Equation 3, δ is the minimum aggregated cost received from the neighbors for each $k \in D_i$, ρ is a set of values from agent a_i's domain for which the cost is minimum. However, more than one value in a_i's domain can achieve the minimum cost. In this case, a *unique-first* approach is followed to determine whether the current solution is accepted or not. In this approach, $|\rho|$ is compared with a bound β. The initial value of β is set to 1. This means the value is acceptable if it is a unique local optimum. If $|\rho| > \beta$, agent a_i goes into HOLD state and waits for more information. Otherwise, a value is randomly selected from ρ and is assigned to its controlled variable. After assigning a value to x_i, every agent $a_j \in \mathcal{N}_i$ updates its current partial assignment and repeats the algorithm. If the value assignment is not possible for all the agents, β is increased by 1, and the algorithm is repeated. This approach prevents the agents from assigning a value prematurely to their variables.

2.4 Challenges

We need to address the following challenges to develop a C-DCOP algorithm that adapts the Cooperative Constraint Approximation (CoCoA).

- **Non-iterative Characteristics:** As CoCoA is a non-iterative, semi-greedy DCOP solver, it assigns a value to a variable only once. For its semi-greedy disposition, early assignments may turn out to be sub-optimal.
- **Infinite Domain:** For Continuous DCOPs, the domain is an infinite number of values within a range denoted by [Lower Bound, Upper Bound]. An agent needs to assign a value to its variables from an infinite number of points. For this reason, a C-DCOP solver needs a substantial amount of time to converge as well as massive memory overhead for the computation.
- **Discretization:** C-DCOP solvers need to discretize the continuous state space to operate. The choice of discrete points can be random, however, settling up the number of discrete points is critical. The quality of solutions found by a C-DCOP algorithm increases with the increasing number of points, because, with more discrete points, the agents can more accurately represent the cost function. Nevertheless, the increasing number of discrete points increases the overhead.
- **Initializing Parameters:** If the cost functions are not convex, initializing the parameters in continuous non-linear optimization methods matters a lot. Because, even with infinite computing power and time, the gradient approach can still be stuck with local minimum or saddle point.

In the following section, we devise a novel method to apply CoCoA in C-DCOPs.

3 THE C-COCOA ALGORITHM

To address the challenges discussed in the previous section, we propose C-CoCoA, a non-exact local search algorithm that uses Cooperative Constraint Approximation (CoCoA) as the underlying algorithmic framework. To be precise, we combine the discrete CoCoA algorithm and the continuous non-linear optimization technique. C-CoCoA is also a non-iterative algorithm like CoCoA in the sense that each agent can only assign its value once and once assigned, it cannot change its value.

In C-CoCoA, we assume that an agent a_i communicates only with those agents whose variables affect a_i's cost function. In other words, a_i communicates only with $a_j \in \mathcal{N}_i$. This ensures a low communication overhead as well as a fully decentralized solution. For this reason, the total cost of an individual agent a_i only depends on $|\mathcal{N}_i|$ rather than the size of the constraint graph. We also assume that each agent knows its neighbors' discretized domain and the nodes of the constraint graph are reachable from any other node.

The C-CoCoA algorithm uses the same message passing technique as described in Section 2.3 for the discrete CoCoA, using the current discretizations of the domain of each variable x_i. However, as the cost functions are not in the tabular form, each agent calculates the cost by evaluating $C = f_i(x^i)$, where x^i is the set of variables related to f_i. The key difference between the C-CoCoA and discrete CoCoA is that, in C-CoCoA, each agent a_i calculates the cost by considering its domain discretizations, $x_i(1)$, $x_i(2)$,..., $x_i(d)$ (Algorithm 1: Line 1) instead of the actual continuous domain, where d is the total number of random discrete points taken from D_i. We select the discrete points randomly because, as aforementioned, we use the non-linear optimization technique to adjust these random discrete points later. For the example of Figure 1, for simplicity, let us assume that $d = 2$. So, we discretize the domains of x_0, x_1, x_2, and x_3 into two random discrete points (x_0: {1, 2}, x_1: {3, 4}, x_2: {7, 8}, and x_3: {5, 9}) from the domain range [-20, 20].

The states of the agents are defined as IDLE, ACTIVE, HOLD and DONE [23]. At the start of the algorithm, all the agents are in IDLE state and $\beta = 1$. Then C-CoCoA activates an agent a_i randomly and as soon as the agent finishes the algorithm, it will trigger the algorithm for its neighbors. Agent a_i sends an **UpdateStateMessage**($\{i$, ACTIVE$\}$) and an **InquiryMessage**($\{i, \widetilde{x_i}\}$) to each neighbor $a_j \in \mathcal{N}_i$ and waits for the **CostMessage**($\{\zeta_j\}$) from the neighbors

[2]In Equation 2, for continuous valued variables, D_j indicates the discretized domain of the agent a_j.

Algorithm 1: The C-CoCoA Algorithm

1 Discretize each domain into d points, $x_i(1), x_i(2), \ldots, x_i(d)$

When started or a_i receives **UpdateStateMessage**($\{j$, DONE$\}$)**:**
require: $STATE$:= IDLE or HOLD
2 $STATE \leftarrow$ ACTIVE
3 **for** *each agent* $a_j \in \mathcal{N}_i$ **do**
4 sends **UpdateStateMessage**($\{i$, ACTIVE$\}$) to a_j
5 sends **InquiryMessage**($\{i, \widetilde{x_i}\}$) to a_j
6 waits for all **CostMessage**($\{\zeta_j\}$) from $a_j \in \mathcal{N}_i$
7 calculate ρ using Equation 3
8 $\chi \leftarrow$ values of x_j that results ζ_j
9 **if** $|\rho| \leq \beta$ **or** IdleActiveNeighbors(i) = 0 **then**
10 $\Theta \leftarrow$ randomly select from ρ
11 $\chi \leftarrow \chi \cup \Theta$
12 calculate $F_{\mathcal{N}_i}^{a_i}$ using Equation 4
13 $x_{\mathcal{N}_i}^{a_i} \leftarrow$ set of related variables with $F_{\mathcal{N}_i}^{a_i}$
14 **for** *each variable* $x \in x_{\mathcal{N}_i}^{a_i}$ **do**
15 initialize x with the corresponding value from χ
16 **while** *the terminating condition is not met* **do**
17 $\forall x \in x_{\mathcal{N}_i}^{a_i}$ update v_x using Equation 5
18 $x_i \leftarrow v_{x_i}$
19 $STATE \leftarrow$ DONE
20 **for** *each agent* $a_j \in \mathcal{N}_i$ **do**
21 sends **UpdateStateMessage**($\{i$, DONE$\}$) to a_j
22 sends **SetValueMessage**($\{i, \widetilde{x_i}\}$) to a_j
23 **else**
24 $STATE \leftarrow$ HOLD
25 **for** *each agent* $a_j \in \mathcal{N}_i$ **do**
26 sends **UpdateStateMessage**($\{i$, HOLD$\}$) to a_j

When received **InquiryMessage**($\{i, \widetilde{x_i}\}$) at a_j**:**
27 $\zeta_j \leftarrow \emptyset$
28 **for** *all* $x_{i,k}$ *in* $D_{i_{dis}}$ **do**
29 $cost \leftarrow \emptyset$
30 **if** $a_j \in \widetilde{x_i}$ **then** // a_j already assigned a value to its variable x_j
31 $D_{j_{dis}} \leftarrow$ value of a_j
32 **for** *all* $x_{j,l}$ *in* $D_{j_{dis}}$ **do**
33 calculate $\zeta_{j,k}$ using Equation 2
34 $cost \leftarrow cost \cup \zeta_{j,k}$
35 $C \leftarrow \min(cost), v_j \leftarrow \text{argmin}_j(cost)$
36 $\zeta_j \leftarrow \zeta_j \cup \{v_j : C\}$
37 sends **CostMessage**($\{\zeta_j\}$) to a_i

When received **UpdateStateMessage**($\{i, S\}$) at a_j**:**
38 a_j stores $STATE$ of $a_i \leftarrow S$
39 **if** S = HOLD **and** $STATE$ *of* a_j = HOLD **and** IdleActiveNeighbors(j) = 0 **then**
40 β++
41 Repeat algorithm
42 **if** S = DONE **and** $STATE$ *of* a_j = HOLD **then**
43 Repeat algorithm

Procedure 1: IdleActiveNeighbors

1 **Function** IdleActiveNeighbors(i)**:**
2 **for** *each agent* $a_j \in \mathcal{N}_i$ **do**
3 **if** $STATE$ *of* a_j = IDLE **or** ACTIVE **then**
4 $count$++
5 **return** $count$

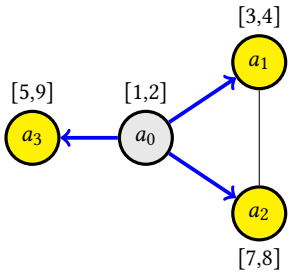
(Algorithm 1: Lines 3-6). After receiving the **InquiryMessage**, each neighbor a_j calculates the lowest cost for every possible value assignment for the discretized domain of x_i, $D_{i_{dis}}$ (Algorithm 1: Lines 27-34). We define $\zeta_j = \{\zeta_{j,x_i(1)}, \zeta_{j,x_i(2)}, ..., \zeta_{j,x_i(d)}\}$ as the overall cost map that contains the minimum cost for each of the discrete points of a_i's domain and is calculated by the agent a_j. We define each element of the cost map as $\zeta_{j,x_i(d)} = \{v_j : C\}$, where C is the minimum cost and v_j denotes the value of a_j's domain that results the minimum cost (Algorithm 1: Lines 35-36). a_j sends the cost map to a_i via **CostMessage**($\{\zeta_j\}$). After receiving all the cost maps, the agent a_i then calculates ρ using the Equation 3. ρ contains the values of x_i that is near-optimal within the discretized points $x_i(1)$, $x_i(2)$,..., $x_i(d)$ of the agent a_i's domain. a_i also stores the values $v_j \in \zeta_{j,x_i(d)}$ in a set χ (Algorithm 1: Lines 7-8). For the example of Figure 1, we assume that the agent a_0[3] is randomly selected. a_0 then sends **UpdateStateMessage** and **InquiryMessage** to its neighbors a_1, a_2, a_3 and waits for the **CostMessage** from all of them. Upon receiving the **InquiryMessage**, a_1, a_2 and a_3 calculate the cost map ζ_1 = {3: 13, 3: 10}, ζ_2 = {7: 154, 7: 161}, ζ_3 = {5: 30, 5: 35}, respectively, and send these cost maps to the inquiring agent a_0 via **CostMessage**. After receiving the cost maps, a_0 calculates ρ by using the Equation 3 and for this example a_0 assigns $\rho = \{1\}$ and $\chi = \{x_1 = 3, x_2 = 7, x_3 = 5\}$. We describe this example elaborately in Figure 2.

Similar to the discrete CoCoA algorithm, more than one value in a_i's domain can achieve the minimum cost, that is $|\rho| > 1$. In this case, we follow a *unique-first* approach which is described in the CoCoA algorithm (Section 2.3). Algorithm 1: Lines 24-26 describes the case when $|\rho| > \beta$. In this case, a_i goes into the HOLD state and waits until another agent has completed its assignment before the algorithm is run again. Otherwise (when $|\rho| \leq \beta$), a value is randomly selected from the set ρ. We assign this value to Θ and add this Θ to the set χ (Algorithm 1: Lines 10-11). This assignment is near-optimal within the discretized domain $D_{i_{dis}}$ of a_i. To find the best solution within the actual domain D_i, we use a non-linear optimization technique. We choose a gradient-based optimization approach because we can implement it in a decentralized way using only local information. Now, for employing the gradient-based non-linear optimization, agent a_i calculates the local objective function $F_{\mathcal{N}_i}^{a_i}$ (Algorithm 1: Line 12) by using the following equation.

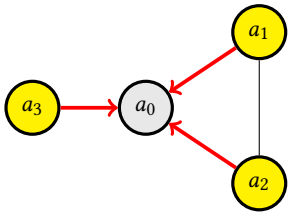
$$F_{\mathcal{N}_i}^{a_i} = \sum_{a_j \in \mathcal{N}_i} f(a_i, a_j) \tag{4}$$

Where $f(a_i, a_j)$ is the cost function that is related to agent a_i and its direct neighbor $a_j \in \mathcal{N}_i$. For the example of Figure 1, agent a_0 assigns $\Theta = 1$ from ρ and appends this value with the set χ. Hence, the set $\chi = \{x_0 = 1, x_1 = 3, x_2 = 7, x_3 = 5\}$. Thereafter, the

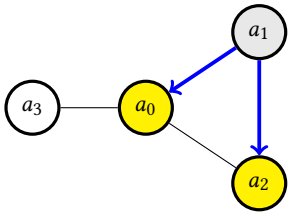
[3]We use a_i and x_i interchangeably throughout the paper since we assume that each agent controls exactly one variable.



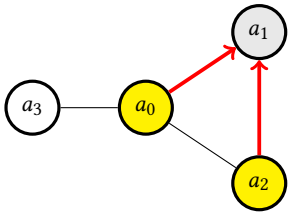
(a) **InquiryMessage** from a_0



(b) **CostMessage** to a_0



(c) Inquiry messages from a_1



(d) Cost messages to a_1

Algorithm starts by random activation of an agent a_0.

- a_0 sends **InquiryMessage** to a_1, a_2 and a_3 and waits for the **CostMessage** from them, *blue arrows* represent the **InquiryMessage**, *grey node* represents the inquiring agent.
- a_1, a_2 and a_3 calculates the cost map, ζ, *yellow nodes* represent the neighbors.

- a_1 calculates:

$\zeta_{3,1}$ = 13, $\zeta_{4,1}$ = 25, therefore, appends [3: 13] with ζ_1.
$\zeta_{3,2}$ = 10, $\zeta_{4,2}$ = 20, therefore, appends [3: 10] with ζ_1.
a_1 sends the final ζ_1 = **[3: 13, 3: 10]** to a_0 via **CostMessage**.

- a_2 calculates:

$\zeta_{7,1}$ = 154, $\zeta_{8,1}$ = 200, therefore, appends [7: 154] with ζ_2.
$\zeta_{7,2}$ = 161, $\zeta_{8,2}$ = 208, therefore, appends [7: 161] with ζ_2.
a_2 sends the final ζ_2 = **[7: 154, 7: 161]** to a_0 via **CostMessage**.

- a_3 calculates:

$\zeta_{5,1}$ = 30, $\zeta_{9,1}$ = 90, therefore, appends [5: 30] with ζ_3.
$\zeta_{5,2}$ = 35, $\zeta_{9,2}$ = 99, therefore, appends [5: 35] with ζ_3.
a_3 sends the final ζ_3 = **[5: 30, 5: 35]** to a_0 via **CostMessage**.

- a_0 receives cost maps from a_1, a_2 and a_3, *red arrows* represent the **CostMessage**.
- a_0 calculates ρ using Equation 3:

For the discretized domain value 1: cost = 13 + 154 + 30 = 197.
For the discretized domain value 2: cost = 10 + 161 + 35 = 206.
Therefore, ρ = 1 and $\chi = \{x_0 = 1, x_1 = 3, x_2 = 7, x_3 = 5\}$.
After 100 iterations of the gradient-based approach, we get, x_0 = -0.572.
a_0 marks its state to DONE and sends a **SetValueMessage** to a_1, a_2, and a_3 that contains the assigned value of a_0.

After the completion of a_0, the algorithm is triggered for the neighboring agents a_1, a_2, a_3.

- a_1 sends **InquiryMessage** to a_0 and a_2, *blue arrows* represent the **InquiryMessage**.
- a_0 and a_2 calculates the cost map.

- a_0 calculates:

$\zeta_{-0.572,3}$ = 21.756, therefore, appends [-0.572: 21.756] with ζ_0.
$\zeta_{-0.572,4}$ = 36.899, therefore, appends [-0.572: 36.899] with ζ_0.
a_0 sends the final ζ_0 = **[-0.572: 21.756, -0.572: 36.899]** to a_1 via **CostMessage**.

- a_2 calculates:

$\zeta_{7,3}$ = 86, $\zeta_{8,3}$ = 113, therefore, appends [7: 86] with ζ_2.
$\zeta_{7,4}$ = 86, $\zeta_{8,4}$ = 112, therefore, appends [7: 86] with ζ_2.
a_2 sends the final ζ_2 = **[7: 86, 7: 86]** to a_1 via **CostMessage**.

- a_1 receives cost maps from a_0 and a_2, *red arrows* represent the cost message.
- a_1 calculates ρ using Equation 3:

For the discretized domain value 3: cost = 21.756 + 86 = 107.756.
For the discretized domain value 4: cost = 36.899 + 86 = 122.899.
Therefore, ρ = 3 and $\chi = \{x_0 = -0.572, x_1 = 3, x_2 = 7\}$.
After 100 iterations of the gradient-based approach, we get, x_1 = -0.122.
a_1 marks its state to DONE and sends a **SetValueMessage** to a_0 and a_2 that contains the assigned value of a_1.

Agents a_2 and a_3 also calculate the values for their variables simultaneously.
We get x_2 = 0.124 and x_3 = 0.911 when the algorithm terminates.
Hence, the near-optimal assignment is, $\mathbf{X}^* = \{\mathbf{x_0} = \mathbf{-0.572}, \mathbf{x_1} = \mathbf{-0.122}, \mathbf{x_2} = \mathbf{0.124}$ and $\mathbf{x_3} = \mathbf{0.911}\}$

Figure 2: Example partial trace of the C-CoCoA algorithm to solve the C-DCOP shown in Figure 1. In this example, we assume that the variable x_i is controlled by the agent a_i and x_0: [1, 2], x_1: [3, 4], x_2: [7, 8], x_3: [5, 9] are the two random discrete points taken from the actual domain [-20, 20]. We also assume that, $\zeta_{j,k}$ is the cost for the k^{th} value of agent a_i's domain which is calculated by neighbor a_j and ζ_j is the overall cost map that is calculated by the neighbor a_j. We use the arrows between the nodes of the constraint graph to indicate the direction of the corresponding messages.

agent a_0 calculates the local objective function $F^{a_0}_{\mathcal{N}_0} = x_0^2 - 2x_0x_1 + 2x_1^2 + x_0x_2 + 3x_2^2 + x_0x_3 + x_3^2$.

After that, the agent a_i performs gradient-based approach for optimizing its local objective function $F^{a_i}_{\mathcal{N}_i}(x^{a_i}_{\mathcal{N}_i})$ where $x^{a_i}_{\mathcal{N}_i}$ is the set of all the related variables with $F^{a_i}_{\mathcal{N}_i}$ (Algorithm 1: Line 13). Agent a_i assigns every variable $x \in x^{a_i}_{\mathcal{N}_i}$ with the corresponding value from the set χ as the initial values in the gradient-based optimization (Algorithm 1: Lines 14-15). Specifically, the agent a_i minimizes the local objective function $F^{a_i}_{\mathcal{N}_i}$ and updates the value v_x of each variable $x \in x^{a_i}_{\mathcal{N}_i}$ according to the following equation:

$$v_x(t) = v_x(t-1) - \alpha \frac{\partial F^{a_i}_{\mathcal{N}_i}}{\partial x^{a_i}_{\mathcal{N}_i}}\Bigg|^{v_x}_{\text{argmin}_{x_i} F^{a_i}_{\mathcal{N}_i}(x^{a_i}_{\mathcal{N}_i})} \tag{5}$$

Here, α is the *learning rate* of the algorithm (Algorithm 1: Lines 16-17). For the example of Figure 1, $x^{a_0}_{\mathcal{N}_0} = \{x_0, x_1, x_2, x_3\}$. Agent a_0 initializes all the variables in $x^{a_0}_{\mathcal{N}_0}$ from the set χ in the gradient-based optimization. In this example, the initial values are set as $x_0 = 1, x_1 = 3, x_2 = 7, x_3 = 5$ (see Figure 2). Then the agent a_0 starts updating the values of the variables $x^{a_0}_{\mathcal{N}_0}$ by using the Equation 5.

The agent continues this update process until it converges or a maximum number of iterations is reached. After termination, the current value of v_x is actually the approximate optimal assignment for the variable x_i (Algorithm 1: Line 18). Then the agent a_i updates its state to DONE and sends its neighbors $a_j \in \mathcal{N}_i$ an **UpdateStateMessage**($\{i$, DONE$\}$) and a **SetValueMessage**($\{i, \widetilde{x_i}\}$) that contains the assigned value of x_i. Upon receiving the **SetValueMessage**, the neighbors update their CPA with the value of x_i and trigger the algorithm for them. For our example, after 100 iterations, the final assignment of the variable x_0 is -0.572. Agent a_0's state is marked as DONE and a_0 sends a **SetValueMessage** to all the neighbors a_1, a_2, and a_3. The neighbors update their CPA with the value of x_0 and trigger the algorithm for them. Note that, each agent can only assign its value once, and once assigned it cannot change its value. To be precise, each agent updates its value locally with gradient descent and sends the **SetValueMessage** only once to a neighbor, and thus C-CoCoA is a non-iterative approach.

4 THEORETICAL ANALYSIS

In this section, we describe some theoretical properties of C-CoCoA in terms of communication, computation, and memory.

Theorem 4.1. *In a binary constraint graph $G = (N, E)$, the total number of messages required by C-CoCoA is $5|N|^2 + d|N|^2$ in the worst case, where d is the total number of discrete points taken from the agents' domain and we assume that the total number of agents $|A| = |N|$.*

Proof. In C-CoCoA, we define $\mathcal{N}_i$ is the set of direct neighbors of the agent a_i. After the activation of an agent a_i, it sends $2|\mathcal{N}_i|$ messages (**UpdateStateMessage** and **InquiryMessage**) to its neighbors as well as a_i receives $|\mathcal{N}_i|$ messages from its neighbors against the reply of the inquiry messages in the form of **CostMessage** (Algorithm 1: Lines 4-6). Therefore, at this stage, the number of messages is $3|\mathcal{N}_i|$. Then, after a successful assignment to its variable, the agent a_i sends $2|\mathcal{N}_i|$ messages (**UpdateStateMessage** and **SetValueMessage**) to its neighbors (Algorithm 1: Lines 20-22). As a result, the number of messages transmitted so far is $5|\mathcal{N}_i|$. However, an agent sends an additional $|\mathcal{N}_i|$ messages each time it enters into the HOLD state (Algorithm 1: Lines 25-26). Although an agent may never enter into the HOLD state, in the worst case, it may enter into the HOLD state d times at most. For this reason, $5|\mathcal{N}_i| + H|\mathcal{N}_i|$ is the total number of messages an agent sends and receives, where $H = 0, 1, ..., d$ defines the number of times an agent enters into the HOLD state. In the worst case, the graph is complete where $|\mathcal{N}_i| = |N| - 1 \approx |N|$ and $H = d$. Therefore, the total number of messages sent or received by an agent a_i is $5|N| + d|N|$ and C-CoCoA requires total $5|N|^2 + d|N|^2$ messages in the worst case. □

Theorem 4.2. *The message size complexity of C-CoCoA is $O(2|N|^2 + d|N|)$ in the worst case.*

Proof. In C-CoCoA, the size of each **UpdateStateMessage** is constant and in each of the **InquiryMessage** and **SetValueMessage**, the agent a_i sends the CPA $\widetilde{x_i}$ that contains the set of known assigned values of all the neighbors of a_i. Hence, the size of each **InquiryMessage** and **SetValueMessage** is $|\mathcal{N}_i|$. a_i sends total $|\mathcal{N}_i|$ **InquiryMessage** and **SetValueMessage** to its neighbors. So, the summation of message size complexity is $|\mathcal{N}_i|^2 + |\mathcal{N}_i|^2 = 2|\mathcal{N}_i|^2$. When the neighboring agents send **InquiryMessage** to a_i, it sends a **CostMessage** of size d as well that contains the cost map ζ. Therefore, a_i sends $|\mathcal{N}_i|$ **CostMessage** of size d to the neighbors. Hence, the total message size for an agent a_i is $O(2|\mathcal{N}_i|^2 + d|\mathcal{N}_i|) \approx O(2|N|^2 + d|N|)$ in the worst case in C-CoCoA. □

Theorem 4.3. *The overall computational complexity of C-CoCoA is $O(|N|(d^2 + b))$ in the worst case, where b is the number of times an agent updates the values of the variables in Equation 5.*

Proof. After the transmission of **UpdateStateMessage** and **InquiryMessage** (Algorithm 1: Lines 3-5), the computational complexity of an agent is $|\mathcal{N}_i|d^2$ (d^2 is the complexity of calculating an **InquiryMessage**). In the gradient-based optimization, an agent needs $|x^{a_i}_{\mathcal{N}_i}| + b|x^{a_i}_{\mathcal{N}_i}|$ computational complexity. After a successful assignment or each of the unsuccessful attempt (HOLD state) to assign a value, an agent again iterates over the set of its neighbors (Algorithm 1: Lines 20-26). This step adds $|\mathcal{N}_i| + H|\mathcal{N}_i|$ complexity, where $H = 0, 1, ..., d$. After adding all these, the overall computational complexity is $O(|N|d^2 + |N| + |N|b + |N| + H|N|) \approx O(|N|(d^2 + b))$; where in the worst case $|\mathcal{N}_i| \approx |N|$, $|x^{a_i}_{\mathcal{N}_i}| \approx |N|$ and $H = d$. □

5 EXPERIMENTAL RESULTS

In this section, we empirically evaluate the performance of C-CoCoA against the state-of-the-art C-DCOP solvers: PFD, AC-DPOP[4], C-DSA, and HCMS. The performance metrics that we consider for the evaluation are solution quality, time, and the number of messages. For PFD, we use the parameter settings suggested in [2] with $K = 500$. We set DSA-B (i.e. $p = 0.6$) for C-DSA. For HCMS, AC-DPOP, and C-CoCoA, we choose the learning rate, $\alpha = 0.01$ (which

[4] Although [11] proposed three versions of Continuous DPOP, we only compare with AC-DPOP in this paper. The reason is that AC-DPOP is reported to provide the best solution among the approximate C-DPOP algorithms proposed in their work.

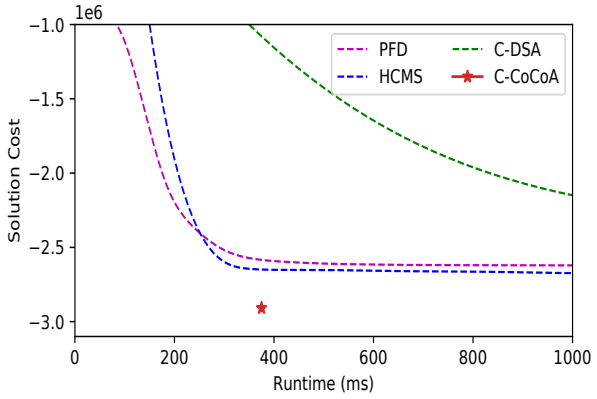


Figure 3: Comparison of C-CoCoA and the benchmarking algorithms on random C-DCOPs (Dense graphs).

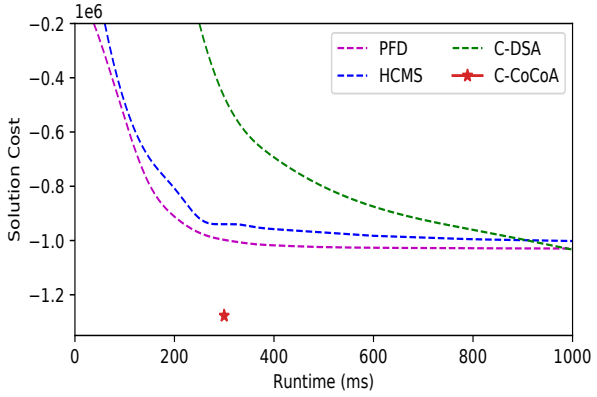


Figure 4: Comparison of C-CoCoA and the benchmarking algorithms on random C-DCOPs (Sparse graphs).

is the best result found on the empirical evaluation), and the number of discrete points to be 3. For the experimental settings, we use the settings described in [2]. To be precise, two types of graphs are used for comparison, namely, *Random Graphs* and *Random Trees*. Binary quadratic functions are used as the constraints which are of the form $ax^2 + bxy + cy^2$ (used in both [2] and [11]). The coefficients of the cost function (a, b, c) are chosen randomly between $[-5, 5]$, and the domains of each agent are set to $[-50, 50]$. However, it is worth mentioning that although we choose binary quadratic functions for evaluation, C-CoCoA is broadly applicable to other classes of problems. In all of the settings described above, we run the benchmarking algorithms on 25 independently generated problems and 20 times on each problem for a fixed amount of time which we describe in detail in the following part of our discussion. In order to conduct these experiments, we use a computer with an Intel Core i5-4200M CPU with 2.5 GHz processor and 8GB RAM. Note that unless stated otherwise, all differences shown in this section are statistically significant for p-value < 0.05.

Random Graphs: We use three different settings for random graphs - sparse, dense, and scale-free. For all the settings, we set the maximum number of iterations for Equation 5 of C-CoCoA to be 100.

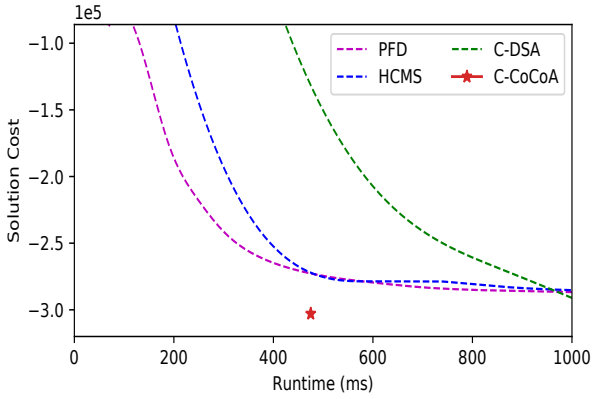


Figure 5: Comparison of C-CoCoA and the benchmarking algorithms on random C-DCOPs (Scale-free graphs).

Table 1: Solution quality of C-CoCoA, PFD, HCMS, and C-DSA on random graphs with varying number of agents.

		C-CoCoA	PFD	HCMS	C-DSA
$\|A\| = 30$	$p = 0.2$	-554,339	-469,093	-423,383	-493,730
	$p = 0.6$	-1,251,118	-1,042,611	-1,055,878	-1,096,852
$\|A\| = 50$	$p = 0.2$	-1,311,988	-1,030,328	-974,416	-1,118,344
	$p = 0.6$	-3,086,005	-2,623,534	-2,730,943	-2,334,385
$\|A\| = 70$	$p = 0.2$	-2,253,114	-1,858,646	-2,049,757	-1,636,508
	$p = 0.6$	-5,486,958	-4,494,102	-5,060,671	—

Note that, although AC-DPOP requires fewer messages than HCMS and PFD, we do not limit the number of messages for HCMS and PFD since AC-DPOP requires much more computation to calculate one message. Figure 3 shows the comparison between C-CoCoA and the benchmarking algorithms on Erdős-Rényi [3] topology with dense settings (edge probability 0.6). In this setting, we set the $no.\ of\ agents = 50$. Moreover, we stop PFD, HCMS, and C-DSA after 1000 milliseconds (ms) for each problem which is much larger than the time needed to complete C-CoCoA. C-CoCoA produces solutions of significantly improved quality and outperforms its competitors HCMS, PFD, and C-DSA by 7.97%, 10.85%, and 24.58%, respectively.

We choose sparse graphs as our second random graph settings. Figure 4 shows the comparison of costs on Erdős-Rényi topology with sparse settings (edge probability 0.2). The experimental setting is similar to setting in the dense graphs with $no.\ of\ agents = 50$ and the stopping criterion of iterative algorithms being 1000 ms. In this setting, C-CoCoA manages to outperform C-DSA by 18.75%. The other benchmarking algorithms (PFD and HCMS) are outperformed by $23.95\% - 25.43\%$.

Table 1 shows further comparisons on random graph settings varying the number of agents. For this experiment, we run each algorithm for 1500 ms using 30, 50, and 70 agents, each with both sparse ($p = 0.2$) and dense ($p = 0.6$) settings. For smaller graphs (i.e., $|A| = 30$), in both sparse and dense settings, the closest competitor of C-CoCoA is C-DSA. For larger graphs and sparse settings, the trends remain the same as the smaller instances. However, as the

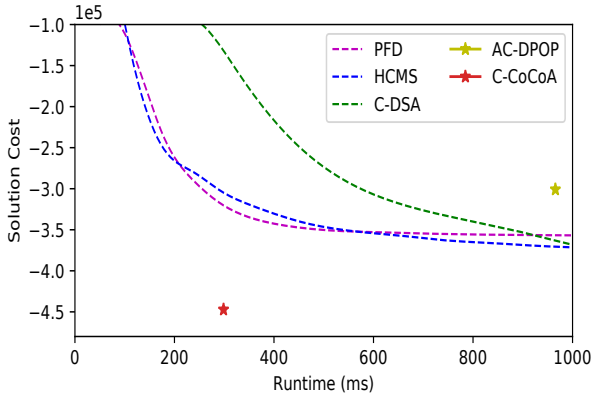


Figure 6: Comparison of C-CoCoA and the benchmarking algorithms on random C-DCOPs (Random tree).

density and graph size increases, C-DSA takes more time than the competing algorithms and HCMS becomes the closest competitor of C-CoCoA. In all the settings, C-CoCoA outperforms the existing algorithms given the same time. We omit the result for C-DSA in $|A| = 70$, $p = 0.6$, as it does not produce any output within the given time. A key insight we can draw from this experiment is that neither the graph size nor the density has any adverse effect on the performance of C-CoCoA in random graph settings.

Table 2: Comparison between C-CoCoA and the competing algorithms in terms of Solution Cost, Time and No. of messages.

Graph Type	Algorithm	C	T (ms)	M
Sparse	C-CoCoA	-1,277,088	297.4	2,502
	HCMS	-974,416	428.5	436,950
	PFD	-1,030,328	416.6	347,625
	C-DSA	-1,118,344	1000.0	218,475
Dense	C-CoCoA	-3,086,005	375.5	3,765
	HCMS	-2,730,943	525.1	1,012,266
	PFD	-2,623,534	515.4	775,733
	C-DSA	-2,334,385	1000.0	506,133
Scale-Free	C-CoCoA	-302,895	475.1	1,376
	HCMS	-261,848	856.0	284,532
	PFD	-272,023	781.7	228,600
	C-DSA	-281,995	1000.0	142,266
Tree	C-CoCoA	-447,163	298.6	490
	HCMS	-365,917	420.5	98,000
	PFD	-368,272	429.0	89,066
	C-DSA	-379,123	1000.0	49,000
	AC-DPOP	-300,825	965.54	100

Figure 5 shows the performance comparison for scale-free networks in a larger graph setting ($|A| = 100$). Although PFD, C-DSA, and HCMS show similar performances, C-CoCoA outperforms the existing algorithms by a margin of 5.94%, 5.62%, and 2.63% for HCMS, PFD, and C-DSA, respectively. Note that we do not include AC-DPOP in dense, sparse, and scale-free settings as it runs out of memory in these settings.

Random Trees: We use the random tree configuration as our last experimental setting since the memory requirement of AC-DPOP is less on trees. The experimental configurations are similar to random graph settings. Figure 6 shows the comparison graph between C-CoCoA and the benchmarking algorithms on random trees. HCMS, PFD, and C-DSA show similar performances in this setting, however, C-CoCoA outperforms these algorithms by 13.12%, 21.42%, and 11.31%, respectively. Moreover, C-CoCoA provides 1.49 times better solution than AC-DPOP.

Table 2 summarizes the performance evaluation of C-CoCoA against the benchmarking algorithms in terms of solution cost (C), convergence time in ms (T), and the number of messages (M). The settings for these experiments are the same as described above. We use the time reported in the table as the convergence time for each of the iterative algorithms. C-CoCoA outperforms all the benchmarking algorithms in each setting. Moreover, the closest competitor of C-CoCoA in sparse settings is C-DSA which requires 87 times more messages than C-CoCoA. The closest competitor of C-CoCoA in dense settings is HCMS which requires 268 times more messages. We see similar trends for Scale-free and random tree settings. The closest competitor C-DSA requires $100 - 103$ times more messages in these settings. Although AC-DPOP requires fewer messages than C-CoCoA in a random tree setting, it requires much more time (3.23 times) than C-CoCoA in the random tree setting.

6 CONCLUSIONS AND FUTURE WORK

The classical DCOP model is only able to effectively deal with discrete variables. However, this is recently reported that this assumption significantly limits DCOPs' ability to be applicable to many real-world problems. In the wake of this, the C-DCOP framework has been proposed which is an extension of the DCOP framework having continuous valued variables. In this paper, we propose an algorithm, namely C-CoCoA, that combines the discrete CoCoA algorithm with a gradient-based non-linear optimization method to solve C-DCOPs. We then empirically evaluate our algorithm in four different experimental settings and compare the results with those of the state-of-the-art C-DCOP algorithms. In all the experimental settings, C-CoCoA performs better than the competing algorithms in terms of solution quality, time, and the number of messages. In the future, we would like to explore whether C-CoCoA can be tailored to solve multi-objective and asymmetric C-DCOPs.

ACKNOWLEDGMENTS

This research is primarily supported by the University Grants Commission of Bangladesh. We also acknowledge the use of the BdREN High Performance Computing Facility.

REFERENCES

[1] Ziyu Chen, Tengfei Wu, Yanchen Deng, and Cheng Zhang. 2018. An Ant-Based Algorithm to Solve Distributed Constraint Optimization Problems. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI).*

[2] Moumita Choudhury, Saaduddin Mahmud, and Md. Mosaddek Khan. 2020. A Particle Swarm Based Algorithm for Functional Distributed Constraint Optimization Problems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI).* 7111–7118.

[3] Paul Erdős and Alfréd Rényi. 1960. On the Evolution of Random Graphs. *Publ. Math. Inst. Hung. Acad. Sci* 5 (1960), 17–60.

[4] Alessandro Farinelli, Alex Rogers, and Nicholas R. Jennings. 2014. Agent-Based Decentralised Coordination for Sensor Networks Using the Max-Sum Algorithm. *Autonomous Agents and Multi-Agent Systems* 28 (2014), 337–380.

[5] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R. Jennings. 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 Multi-Agent Systems (AAMAS)*. 639–646.

[6] Ferdinando Fioretto, Enrico Pontelli, and William Yeoh. 2018. Distributed Constraint Optimization Problems and Applications: A Survey. *Journal of Artificial Intelligence Research* 61 (2018), 623–698.

[7] Ferdinando Fioretto, William Yeoh, and Enrico Pontelli. 2017. A Multiagent System Approach to Scheduling Devices in Smart Homes. In *Proceedings of the 16th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 981–989.

[8] Stephen Fitzpatrick and L. Meetrens. 2003. Distributed Sensor Networks A Multiagent Perspective, Chapter Distributed Coordination Through Anarchic Optimization.

[9] Ben Grocholsky, James Keller, Vijay Kumar, and George Pappas. 2006. Factor Graphs and the Sum-Product Algorithm. *IEEE Robotics and Automation Magazine* 13 (2006), 16–25.

[10] Katsutoshi Hirayama and Makoto Yokoo. 1997. Distributed Partial Constraint Satisfaction Problem. In *Proceedings of The 3rd International Conference on Principles and Practice of Constraint Programming (CP)*. 222–236.

[11] Khoi D. Hoang, William Yeoh, Makoto Yokoo, and Zinovi Rabinovich. 2020. New Algorithms for Continuous Distributed Constraint Optimization Problems. In *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 502–510.

[12] Chih-fan Hsin and Mingyan Liu. 2004. Network Coverage Using Low Duty-Cycled Sensors: Random & Coordinated Sleep Algorithms. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks (IPSN)*. 433–442.

[13] Md. Mosaddek Khan, Long Tran-Thanh, and Nicholas R. Jennings. 2018. A Generic Domain Pruning Technique for GDL-Based DCOP Algorithms in Cooperative Multi-Agent Systems. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 1595–1603.

[14] Rajiv T. Maheswaran, Jonathan P. Pearce, and Milind Tambe. 2004. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS)*. 432–439.

[15] Rajiv T. Maheswaran, Milind Tambe, Emma Bowring, Jonathan P. Pearce, and Pradeep Varakantham. 2004. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 310–317.

[16] Saaduddin Mahmud, Moumita Choudhury, Md. Mosaddek Khan, Long Tran-Thanh, and Nicholas R. Jennings. 2020. AED: An Anytime Evolutionary DCOP Algorithm. In *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 825–833.

[17] Saaduddin Mahmud, Md. Mosaddek Khan, Moumita Choudhury, Long Tran-Thanh, and Nicholas R. Jennings. 2020. Learning Optimal Temperature Region for Solving Mixed Integer Functional DCOPs. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*. 268–275.

[18] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* 161 (2005), 149–180.

[19] Adrian Petcu and Boi Faltings. 2005. DPOP: A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*. 266–271.

[20] Mashrur Rashik, Md. Musfiqur Rahman, Md. Mosaddek Khan, Md. Mamunor Rashid, Long Tran-Thanh, and Nicholas R. Jennings. 2020. Speeding Up Distributed Pseudo-tree Optimization Procedure with Cross Edge Consistency to Solve DCOPs. *Applied Intelligence* (2020), 1–14.

[21] Pierre Rust, Gauthier Picard, and Fano Ramparany. 2016. Using Message-Passing DCOP Algorithms to Solve Energy-Efficient Smart Environment Configuration Problems. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. 468–474.

[22] Ruben Stranders, Alessandro Farinelli, Alex Rogers, and Nicholas R. Jennings. 2009. Decentralised Coordination of Continuously Valued Control Parameters Using the Max-Sum Algorithm. In *Proceedings of the 8th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 601–608.

[23] Cornelis Jan van Leeuwen and Przemyslaw Pawelczak. 2017. CoCoA: A Non-Iterative Approach to a Local Search (A)DCOP Solver. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*.

[24] Thomas Voice, Ruben Stranders, Alex Rogers, and Nicholas R. Jennings. 2010. A Hybrid Continuous Max-Sum Algorithm for Decentralised Coordination. In *Proceedings of The 19th European Conference on Artificial Intelligence (ECAI)*. 61–66.

[25] Harel Yedidsion and Roie Zivan. 2016. Applying DCOP_MST to a Team of Mobile Robots with Directional Sensing Abilities. In *Proceedings of the 15th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 1357–1358.

[26] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. 2005. Distributed Stochastic Search and Distributed Breakout: Properties, Comparison and Applications to Constraint Optimization Problems in Sensor Networks. *Artificial Intelligence* 161 (2005), 55–87.

[27] Roie Zivan, Harel Yedidsion, Steven Okamoto, Robin Glinton, and Katia Sycara. 2015. Distributed Constraint Optimization for Teams of Mobile Sensing Agents. *Autonomous Agents and Multi-Agent Systems* 29 (2015), 495–536.