A Particle Swarm Inspired Approach for Continuous Distributed Constraint Optimization Problems

Moumita Choudhury AMCHOUDHURY@CS.UMASS.EDU
College of Information and Computer Sciences,
University of Massachusetts Amherst,
Amherst, USA

Amit Sarker AMITCSEDU99@GMAIL.COM
Department of Computer Science and Engineering,
Washington University in St. Louis,
One Brookings Dr., St. Louis, MO 63130, USA

Samin Yaser SAMIN-2017014979@CS.DU.AC.BD
Department of Computer Science and Engineering,
University of Dhaka,
Science Complex, Dhaka 1000, Bangladesh

Md. Maruf Al Alif Khan ALIF-2017014997@CS.DU.AC.BD
Department of Computer Science and Engineering,
University of Dhaka,
Science Complex, Dhaka 1000, Bangladesh

William Yeoh WYEOH@WUSTL.EDU
Department of Computer Science and Engineering,
Washington University in St. Louis,
One Brookings Dr., St. Louis, MO 63130, USA

Md. Mosaddek Khan MOSADDEK@DU.AC.BD
Department of Computer Science and Engineering,
University of Dhaka,
Science Complex, Dhaka 1000, Bangladesh

Abstract

Distributed Constraint Optimization Problems (DCOPs) are a widely studied framework for coordinating interactions in cooperative multi-agent systems. In classical DCOPs, variables owned by agents are assumed to be discrete. However, in many applications, such as target tracking or sleep scheduling in sensor networks, continuous-valued variables are more suitable than discrete ones. To better model such applications, researchers have proposed *Continuous DCOPs* (C-DCOPs), an extension of DCOPs, that can explicitly model problems with continuous variables. The state-of-the-art approaches for solving C-DCOPs experience either onerous memory or computation overhead and are unsuitable for non-differentiable optimization problems. To address this issue, we propose a new C-DCOP algorithm, namely *Particle Swarm Optimization Based C-DCOP* (PCD), which is inspired by *Particle Swarm Optimization* (PSO), a well-known *centralized* population-based approach for solving continuous optimization problems. In recent years, population-based algorithms have gained significant attention in classical DCOPs due to their ability in producing high-quality solutions. Nonetheless, to the best of our knowledge, this class of

algorithms has not been utilized to solve C-DCOPs and there has been no work evaluating the potential of PSO in solving classical DCOPs or C-DCOPs. In light of this observation, we adapted PSO, a centralized algorithm, to solve C-DCOPs in a decentralized manner. The resulting PCD algorithm not only produces good-quality solutions but also finds solution without any requirement for derivative calculations. Moreover, we design a crossover operator that can be used by PCD to further improve the quality of solutions found. Finally, we theoretically prove that PCD is an anytime algorithm and empirically evaluate PCD against the state-of-the-art C-DCOP algorithms in a wide variety of benchmarks.

1. Introduction

Distributed Constraint Optimization Problems (DCOPs) are an important constraint-handling framework for multi-agent systems in which multiple agents communicate with each other in order to optimize a global objective. The global objective is defined as the aggregation of cost functions (i.e., constraints) among the agents. Each of the cost functions involves a set of variables controlled by the corresponding agents. The structure of DCOPs has made it suitable for deploying in various real-world problems. It has been widely applied to solve a number of multi-agent coordination problems including multi-agent task scheduling (Sultanik, Modi, & Regli, 2007), sensor networks (Farinelli, Rogers, & Jennings, 2014), multi-robot coordination (Yedidsion & Zivan, 2016), etc.

Over the years, several algorithms have been proposed to solve DCOPs, and they are broadly categorized into exact and non-exact algorithms. Exact algorithms, such as ADOPT (Modi, Shen, Tambe, & Yokoo, 2005; Yeoh, Felner, & Koenig, 2010), DPOP (Petcu & Faltings, 2005; Rashik, Rahman, Khan, Mamun-or Rashid, Tran-Thanh, & Jennings, 2020), and PT-FB (Litov & Meisels, 2017) are designed in such a way that they provide a global optimal solution of a given DCOP. However, since DCOPs are NP-Hard, exact algorithms experience exponential memory requirements and/or exponential computational costs as the system grows. On the contrary, non-exact algorithms such as DSA (Zhang, Wang, Xing, & Wittenburg, 2005), MGM & MGM2 (Maheswaran, Pearce, & Tambe, 2004), Max-Sum (Farinelli, Rogers, Petcu, & Jennings, 2008; Tassa, Grinshpoun, & Zivan, 2017; Khan, Tran-Thanh, & Jennings, 2018a; Khan, Tran-Thanh, Ramchurn, & Jennings, 2018c), CoCoA (van Leeuwen & Pawelczak, 2017), ACO_DCOP (Chen, Wu, Deng, & Zhang, 2018), D-Gibbs (Nguyen, Yeoh, Lau, & Zivan, 2019), and AED (Mahmud, Choudhury, Khan, Tran-Thanh, Jennings, et al., 2020b) compromise some solution quality for scalability.

In general, DCOPs assume that the variables of participating agents are discrete. Nevertheless, many real-world applications (e.g., target tracking sensor orientation (Fitzpatrick & Meetrens, 2003), sleep scheduling of wireless sensors (Hsin & Liu, 2004)) can be best modeled with continuous variables. Therefore, for discrete DCOPs to be applied in such problems, we need to discretize the continuous domains of the variables. However, the discretization process needs to be coarse for a problem to be tractable and must be sufficiently fine to find high-quality solutions of the problem (Stranders, Farinelli, Rogers, & Jennings, 2009). To overcome this issue, a continuous version of DCOPs have been proposed (Stranders et al., 2009), which is later referred to as both *Functional DCOPs* (Choudhury, Mahmud, Khan, et al., 2020; Mahmud, Khan, Choudhury, Tran-Thanh, & Jennings, 2020a) and *Continuous DCOPs* (C-DCOPs) (Hoang, Yeoh, Yokoo, & Rabinovich, 2020). In this paper, we will refer to it as C-DCOPs following the most popular convention. There are two main differences

between C-DCOPs and DCOPs. Firstly, instead of having discrete decision variables, C-DCOPs have continuous variables that can take any value between a range. Secondly, the constraint functions are represented in functional forms in C-DCOPs rather than in tabular forms in DCOPs.

In order to cope with the modification of the DCOP formulation, several C-DCOP algorithms have been proposed. Similar to DCOP algorithms, C-DCOP algorithms are also classified as exact and non-exact approaches (detailed discussions can be found in Section 2). In this paper, we focus on the latter class of C-DCOP algorithms as the ensuing exponential growth of search space can make exact algorithms computationally infeasible to deploy in practice. Now, the state-of-the-art algorithms for C-DCOPs are based on either inference (Stranders et al., 2009; Voice, Stranders, Rogers, & Jennings, 2010; Hoang et al., 2020) or local search (Hoang et al., 2020). In the inference-based C-DCOP algorithms, discrete inference-based algorithms, such as Max-Sum and DPOP, have been used in combination with continuous non-linear optimization methods. And, in the only local search-based C-DCOP algorithm, the discrete local search-based algorithm DSA has been extended with continuous optimization methods. However, continuous optimization methods, such as gradient-based optimization require derivative calculations and are thus not suitable for non-differentiable optimization problems.

Against this background, we propose a *Particle Swarm Optimization* (PSO) based C-DCOP algorithm called *PSO-Based C-DCOP* (PCD).[1] PSO is a stochastic optimization technique inspired by the social metaphor of bird flocking (Eberhart & Kennedy, 1995). It has been successfully applied to many optimization problems such as Function Minimization (Shi & Eberhart, 1999), Neural Network Training (Zhang, Zhang, Lok, & Lyu, 2007), and Power-System Stabilizers Design Problems (Abido, 2002). However, to the best of our knowledge, no previous work has been done to incorporate PSO in distributed scenarios similar to DCOPs or C-DCOPs. In PCD, agents cooperatively keep a set of particles where each particle represents a candidate solution and iteratively updates the solutions using a series of update equations over time. Since PSO requires only primitive mathematical operators such as addition and multiplication, it is computationally less expensive (both in memory and speed) than the gradient-based optimization methods. Furthermore, PSO is a widely studied technique with a variety of parameter choices and variants developed over the years. Hence, the wide opportunity for developing PCD as a robust population-based algorithm has inspired us to analyze the challenges and opportunities of PSO in C-DCOPs. Our main contributions are as follows.

- We develop a new algorithm PCD by tailoring PSO. In so doing, we redesign a series of update equations that utilize the communication topology in a distributed scenario.
- We introduce a new crossover operator that further improves the quality of solutions found and name the version PCD_CrossOver.
- We analyze the various parameter choices of PCD that balance exploration and exploitation.

1. A preliminary version of this research has appeared previously (Choudhury et al., 2020). This paper contains a more efficient approach and comprehensive description of the algorithm and comes with broader theoretical and experimental analysis to other state-of-the-art C-DCOP algorithms.

- We provide a theoretical proof of anytime convergence of our algorithm, and show empirical evaluations of PCD and PCD_CrossOver on various C-DCOP benchmarks. The results show that the proposed approach finds solutions with better quality by exploring a large search space compared to existing C-DCOP solvers.

In Section 2, we briefly review related work. In Section 3, we formulate the DCOP and C-DCOP frameworks as well as introduce PSO. Section 4 illustrates the details of our proposed PCD framework. Section 5 provides a theoretical proof of the anytime property and complexity analyses of PCD. In Section 6, we show empirical evaluations of PCD against existing C-DCOP algorithms. Finally, Section 7 concludes the findings of the paper and provides insights for future work.

2. Related Work

In this section, we discuss existing state-of-the-art exact and non-exact C-DCOP algorithms. The only exact algorithm for C-DCOP is the *Exact Continuous DPOP* (EC-DPOP), which only provides exact solutions to linear and quadratic cost functions and is defined over tree-structured graphs only (Hoang et al., 2020). While there are several non-exact algorithms exist, the first non-exact algorithm for C-DCOP is the *Continuous Max-Sum* (CMS) algorithm. CMS extends the discrete Max-Sum (Stranders et al., 2009) by approximating constraint cost functions as piece-wise linear functions. Subsequently, researchers introduced *Hybrid Continuous Max-Sum* (HCMS), which extends CMS by combining it with continuous non-linear optimization methods (Voice et al., 2010). However, continuous optimization methods, such as gradient-based optimization (Sarker, Choudhury, & Khan, 2021; Hendrikx, 2021) require derivative calculations and are thus not suitable for non-differentiable optimization problems. Finally, Hoang et al. (Hoang et al., 2020) made the most recent contributions to this field. In their paper, the authors proposed four algorithms – one exact and three non-exact C-DCOP solvers. The exact algorithm is EC-DPOP, which we discussed earlier. The non-exact algorithms are *Approximate Continuous DPOP* (AC-DPOP), *Clustered AC-DPOP* (CAC-DPOP), and *Continuous DSA* (C-DSA). Both AC-DPOP and CAC-DPOP are based on the discrete DPOP algorithm with non-linear optimization techniques. The discrete DPOP algorithm (Petcu & Faltings, 2005) is an inference-based DCOP algorithm that performs dynamic programming on a pseudo-tree representation of the given problem. This algorithm only requires a linear number of messages but has an exponential memory requirement and sends exponentially large message sizes. Since the underlying algorithm for AC-DPOP is DPOP, it also suffers from the same exponentially large message sizes, which is a limiting factor for communication-constrained applications. Although CAC-DPOP provides a bound on the message size by limiting the number of tuples to be sent in the messages, each agent still needs to maintain the original set of tuples in their memory for better accuracy in calculation. Hence, CAC-DPOP still incurs an exponential memory requirement. Nevertheless, the authors also provide C-DSA, a local search algorithm based on DSA. Unlike the DPOP variants, C-DSA's memory requirement is linear in the number of variables of the problem and it sends constant-size messages.

3. 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 centralized PSO algorithm and the challenges in incorporating PSO with the C-DCOP framework.

3.1 Distributed Constraint Optimization Problems

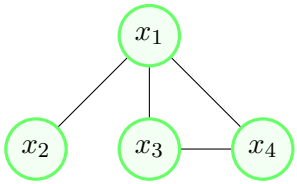
Distributed Constraint Optimization Problems (DCOPs) involve multiple agents collaborating to find a solution that satisfies their aggregated global constraint. Cost functions are utilized to express these constraints in a DCOP, with each agent having a local cost function that determines the cost of a specific value combination for its variables based on the degree of constraint violation. The global cost function is a sum of the local cost functions of all agents. A DCOP algorithm aims to minimize the global cost function by finding a joint solution that satisfies the constraints.

A DCOP can be defined as a tuple $\langle A, X, D, F, \alpha \rangle$ (Modi et al., 2005) 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 at least 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 (constraints) $\{f_1, f_2, \ldots, f_l\}$, where each $f_i \in F$ is defined over a subset $\mathbf{x}^i = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$ 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 $\mathbf{x}^i$, 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 (Khan, Tran-Thanh, Yeoh, & Jennings, 2018b) 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 the ease of understanding, we assume each agent controls only one variable in this paper.

An optimal solution of a DCOP is an assignment X^* that minimizes the sum of cost functions as shown in Equation 1[2]:

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



(a) Constraint Graph

$$\forall x_i \in X : D_i = [-10, 10]$$

$$f_{12}(x_1, x_2) = x_1^2 - x_2^2$$

$$f_{13}(x_1, x_3) = x_1^2 + 2x_1x_3$$

$$f_{14}(x_1, x_4) = 2x_1^2 - 2x_4^2$$

$$f_{34}(x_3, x_4) = x_3^2 + 3x_4^2$$

(b) Cost Functions

Figure 1: Example of a C-DCOP.

3.2 Continuous Distributed Constraint Optimization Problems

Similar to the DCOP formulation, C-DCOPs can be defined as a tuple $\langle A, X, D, F, \alpha \rangle$ (Hoang et al., 2020). 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 (Hoang et al., 2020). However, the goal of a C-DCOP remains the same as depicted in Equation 1. Figure 1 presents an example C-DCOP, where Figure 1a 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 1b. In this particular example, the domains of all variables are the same – each variable x_i can take values from the range $[-10, 10]$.

3.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a population-based optimization[3] technique inspired by the movement of a bird flock or a fish school (Eberhart & Kennedy, 1995). Recently, PSO has been reported to be effective at solving complex CSPs and COPs in a wide range of application domains. The applications that related to the former includes scheduling challenges (Yu, Gao, Wang, & Meng, 2020; Wei, Li, Jiang, Hu, & Hu, 2018), robotics (Dewang,

2. For a maximization problem, the argmin operator should be replaced by the argmax operator.
3. For simplicity, we are going to consider the terms 'optimization' and 'minimization' interchangeably throughout the paper.

Algorithm 1: Particle Swarm Optimization

```
1  Generate an n-dimensional population P
2  Randomly initialize positions and velocities of each particle
3  while termination condition is not met do
4      foreach P_i ∈ P do
5          calculate the current velocity
6          calculate the next position given current velocity
7          move to next position
8          if fitness of current position < fitness of local best then
9              update local best
10         if fitness of current position < fitness of global best then
11             update global best
```

Mohanty, & Kundu, 2018; Ever, 2017), and data mining (Ghosh, Karmakar, Sharma, & Phadikar, 2019). While, supply chain management (Jia, Chen, Tianlong, Zhang, Yuan, Lin, Yu, & Zhang, 2017), resource allocation (Hao, Wang, & Wang, 2022; Khireddine, Larbi, Sylia, Gueguen, & Lamine, 2020), and vehicle routing (Marinakis, Marinaki, & Migdalas, 2019; ?) are the notable recent works that address different problems formulated as COPs.

In PSO, each individual of the population is called a particle. PSO solves the problem by moving the particles in a multi-dimensional search space by adjusting the position and velocity of each particle. As shown in Algorithm 1, each particle is initially assigned a random position and velocity (Line 2). A fitness function is defined, which is used to evaluate the position of each particle. In each iteration, the movement of a particle is guided by both its *local best position* found so far in the search space and the *global best position* found by the entire swarm (Lines 5-7). The combination of the local and global best positions ensures that when a global better position is found through the search process, the particles will move closer to that position and explore the surrounding search space more thoroughly. Then, the local best position of each particle and the global best position of the entire population is updated when necessary (Lines 8-11). Over the last couple of decades, several versions of PSO have been developed. The standard PSO often converges to a sub-optimal solution since the velocity component of the global best particle tends to zero after some iterations. Consequently, the global best position stops moving, and the swarm behavior of all other particles leads them to follow the global best particle. To cope with the premature convergence property of standard PSO, Guaranteed Convergence PSO (GCPSO) has been proposed that provides convergence guarantees to a local optima (van den Bergh & Engelbrecht, 2002).

3.4 Challenges

Over the years, PSO and its improved variant *Guaranteed Convergence PSO* (GCPSO) have shown promising performance in centralized continuous optimization problems (Shi & Eberhart, 1999; van den Bergh & Engelbrecht, 2002). Motivated by its success, we seek to

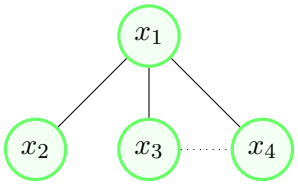


Figure 2: A sample BFS pseudo-tree representation of the C-DCOP depicted in Figure 1.

explore its potential in solving C-DCOPs. However, there are several challenges that must be addressed when developing an anytime C-DCOP algorithm using GCPSO:

- **Particles and Fitness Representation:** We need to define a representation for the particles where each particle represents a solution of the C-DCOPs. Moreover, a distributed method for calculating the fitness for each of the particles needs to be devised.
- **Creating the Population:** In centralized optimization problems, creating the initial population is a trivial task. However, in the case of C-DCOPs, different agents control different variables. Hence, a method needs to be devised to generate the initial population cooperatively.
- **Evaluation:** Centralized PSO deals with an n-dimensional optimization task. In C-DCOPs, each agent holds one variable and each agent is responsible for solving the optimization task related to that variable only where the global objective is still an n-dimensional optimization process. Thus, a decentralized evaluation needs to be devised.
- **Maintaining the Anytime Property:** To maintain the anytime property in a C-DCOP approach, we need to identify the global best particle and the local best position for each particle. A distribution method needs to be devised to notify all the agents when a new global best particle or local best position is found. Finally, a decentralized coordination method is needed among the agents to update the position and velocity considering the current best position.

In the following section, we devise a novel method that addresses the above challenges and applies PSO to solve C-DCOPs.

4. The PCD Algorithm

We now turn to describe our proposed *Particle Swarm Optimization Based C-DCOP* (PCD) algorithm. To facilitate an easier understanding of the algorithm, we first describe what each particle represents in the context of C-DCOPs. Like in PSO, each particle in PCD has two attributes – position and velocity. The position of a particle corresponds to a value assignment to all variables in the C-DCOP. In other words, it is a solution to a given C-DCOP. Moreover, each agent also maintains the local best position of the particle. The velocity of a particle defines the step size that a particle takes in each iteration to change its position and is influenced by the combination of the direction of its local best and global

Table 1: Population Representation in PCD

	Agent a_1	Agent a_2	Agent a_3	Agent a_m
Particle P_1	$P_1^1.v_1$ $P_1^1.x_1$	$P_1^2.v_2$ $P_1^2.x_2$	...	$P_1^m.v_m$ $P_1^m.x_m$
Particle P_2	$P_2^1.v_1$ $P_2^1.x_1$	$P_2^2.v_2$ $P_2^2.x_2$	...	$P_2^m.v_m$ $P_2^m.x_m$
...	...	...	...	...
Particle P_K	$P_K^1.v_1$ $P_K^1.x_1$	$P_K^2.v_2$ $P_K^2.x_2$	...	$P_K^m.v_m$ $P_K^m.x_m$

best position. However, unlike in PSO, where a centralized entity controls all particles, each particle in PCD is controlled in a decentralized manner by all deployed agents. Specifically, for each particle, each agent controls only the position and velocity corresponding to its variable.

In PCD, we define population P as a set of particles that are collectively maintained by all the agents and local population $P^i \subseteq P$ as the subset of the population maintained by an agent a_i. For further clarification, we present an example of a population in Table 1. Here, each row represents a particle $P_k = \{P_k^1, P_k^2, \ldots, P_k^m\}$, which is the k^{th} solution of the problem. Each column represents an agent a_i and the corresponding attributes that it holds for each particle. For example, in the table, each agent a_i holds two attributes, namely the position attribute $P_k^i.x_i$ and the velocity attribute $P_k^i.v_i$, for each particle $P_k^i \in P^i$. Additionally, we use the following notations:

- $P_k.X = \{P_k^1.x_1, P_k^2.x_2, \ldots, P_k^m.x_m\}$ and $P_k.V = \{P_k^1.v_1, P_k^2.v_2, \ldots, P_k^m.v_m\}$ to represent the complete position and velocity assignment for each particle P_k, respectively.
- $P^i.x_i = \{P_1^i.x_i, P_2^i.x_i, \ldots, P_K^i.x_i\}$ and $P^i.v_i = \{P_1^i.v_i, P_2^i.v_i, \ldots, P_K^i.v_i\}$ to represent the position and velocity assignments of each agent a_i for all the particles, respectively.
- $P_k^i.local\_fitness$ to represent the fitness of particle P_k^i, that is, the aggregated cost of constraints associated with the neighbors of agent a_i.
- $P_k.fitness$ and $P_k^i.fitness$ to represent the complete fitness and the fitness that agent a_i calculates for each particle $P_k \in P$ and $P_k^i \in P^i$, respectively.
- $P^i.fitness \leftarrow \{P_1^i.fitness, P_2^i.fitness, \ldots, P_K^i.fitness\}$ to represent the set of $P_k^i.fitness$ for all the particles.
- $P_k^i.p_{best}$ and $P_k^i.p_{best}.fitness$ to represent the best position of particle P_k^i thus far and the fitness value of that position, respectively.
- P^* to represent the global best particle among all particles.

Algorithm 2: PCD Algorithm

Input : K – Number of particles
w – Inertia weight
c_1 – Cognitive constant
c_2 – Social constant
max_{s_c} – Threshold for success count
max_{f_c} – Threshold for failure count

```
12 foreach a_i ∈ A do
13     INITIALIZATION()
14     while Termination condition not met do
15         P^i.fitness ← EVALUATION()
16         P* ← BEST_UPDATE(P^i.fitness)
17         t ← t + 1
18         VARIABLE_UPDATE(P*)
```

- $P^i.g_{best}$ and $P^i.g_{best}.fitness$ to represent the position attribute of the global best particle P^* and the fitness value of that position for each agent a_i, respectively.

PCD is a PSO-based iterative algorithm that first constructs a *Breadth First Search* (BFS) pseudo-tree (Chen, He, & He, 2017), which orders the agents, in a pre-processing step. Figure 2 illustrates a BFS pseudo-tree constructed from the constraint graph shown in Figure 1 having ${x_1}^4$ as the root. From this point, we use the notation N_i to refer to the neighboring agents of agent a_i in the constraint graph and the notations PR_i and $CH_i \subseteq N_i$ to refer to the parent agent and set of children agents of agent a_i in the pseudo-tree, respectively. For example, for agent x_3 of Figure 2, $N_3 = \{x_1, x_4\}$, $PR_3 = x_1$, and $CH_3 = \emptyset$.

The pseudocode of our PCD algorithm can be found in Algorithm 2. After constructing the pseudo-tree, it runs the following three phases:

- **Initialization Phase:** The agents create an initial population of K particles and initialize their parameters.
- **Evaluation Phase:** The agents calculate the fitness value for each particle in a distributed way.
- **Update Phase:** Each agent keeps track of the best solution found so far, propagates this information to the other agents, and updates its value assignment according to that information.

The agents repeat these last two phases in a loop until some termination condition is met.

We now describe these phases in more detail. In the **initialization phase**, each agent $a_i \in A$ executes the `INITIALIZATION` procedure (Procedure 3), which consists of the following: It first creates a set of K particles P^i and initializes the cycle counter t as well as three other variables s_c, f_c, and ρ that are used to update the velocity of the particles (Lines 19-21). It then initializes the velocity $P^i_k.v_i$ and position $P^i_k.x_i$ of each particle $P^i_k \in P^i$ to 0 and

4. We use a_i and x_i interchangeably throughout the paper since each agent controls exactly one variable.

Procedure 3: INITIALIZATION( )

19 $P^i \leftarrow$ set of K particles
20 $t \leftarrow s_c \leftarrow f_c \leftarrow 0$
21 $\rho \leftarrow 1$
22 **foreach** $P_k^i \in P^i$ **do**
23 $P_k^i.v_i \leftarrow 0$
24 $P_k^i.x_i \leftarrow$ a random value from D_i
25 $P_k^i.p_{best} \leftarrow$ null
26 $P_k^i.p_{best}.fitness \leftarrow \infty$
27 $P^i.g_{best} \leftarrow$ null
28 $P^i.g_{best}.fitness \leftarrow \infty$
29 Send VALUE($P^i.x_i$) to each agent $a_j \in N_i$

Procedure 4: EVALUATION( )

30 Wait until VALUE($P^j.x_j$) is received from each agent $a_j \in N_i$
31 **foreach** $P_k^i \in P^i$ **do**
32 $P_k^i.local\_fitness \leftarrow \sum_{a_j \in N_i} f_{ij}(P_k^i.x_i, P_k^j.x_j)$
33 Wait until COST($P^j.fitness$) is received from each agent $a_j \in CH_i$
34 **foreach** $P_k^i \in P^i$ **do**
35 $P_k^i.fitness \leftarrow P_k^i.local\_fitness + \sum_{a_j \in CH_i} P_k^j.fitness$
36 **if** $a_i = root$ **then**
37 **foreach** $P_k^i \in P^i$ **do**
38 $P_k^i.fitness \leftarrow P_k^i.fitness/2$
39 **else**
40 Send COST($P^i.fitness$) to PR_i
41 **return** $P^i.fitness$

a random value in D_i, respectively (Lines 23-24). This initialization is aimed at distributing the initial positions of the particles randomly throughout the search space. It then initializes the best position $P_k^i.p_{best}$ and the corresponding fitness value $P_k^i.p_{best}.fitness$ of each particle $P_k^i \in P^i$ to null and infinity, respectively, since the position has not been evaluated yet (Lines 25-26). Similarly, it initializes the best global position $P^i.g_{best}$ and the corresponding fitness value $P^i.g_{best}.fitness$ to null and infinity, respectively, as well (Lines 27-28). Finally, it sends its position assignments for all particles $P^i.x_i$ in a VALUE message to each of its neighboring agents (Line 29).

Next, in the **evaluation phase**, the agents collectively calculate the complete fitness $P_k.fitness$ of each particle P_k using the fitness function shown in Equation 2 in the `EVALUATION` procedure (Procedure 4).

$$P_k.fitness = \frac{1}{2} \sum_{a_i \in A} \sum_{f_j \in F^i} f_j(P_k.x^j) \tag{2}$$

$$F^i = \{f_j \in F \mid x^j = \{x_i, x_k\}\} \tag{3}$$

Here, F^i is the set of constraints whose scope x^i includes a_i (see Equation 3) and $P_k.x^j$ is the value assignment of the set of variables in the scope x^j of function f_j for each particle P_k. Note that a single agent cannot calculate the complete fitness value. Instead, it is calculated in a decentralized way by all the agents and then accumulated up the BFS tree towards the root. Specifically, each agent a_i is in charge of computing only $\sum_{f_j \in F^i} f_j(P_k.x^j)$ for each particle P_k. Further, note that the cost of each function f_j is summed up twice by the two agents in its scope.[5] Therefore, the complete fitness value is divided by two.

To calculate the complete fitness value in a decentralized way, each agent first waits for VALUE messages from its neighboring agents (Line 30). Upon receiving all the VALUE messages, it calculates the costs of all its functions $f^j \in F^i$ and aggregates them in local fitness values $P_k^i.local\_fitness$ (Line 32). If the agent does not have any children agent, then it assigns $P_k^i.local\_fitness$ to $P_k^i.fitness$ for all particles P_k^i (Line 35) and sends the set of fitness values of all particles in a COST message to its parent agent (Line 40). If an agent does have children agents, then it waits for COST messages from all its children agents (Line 33). After receiving the fitness values from all its children, it aggregates the fitness values received with its own local fitness values (Line 35) and sends the set of aggregated fitness values of all particles in a COST message to its parent agent (Line 40).

This process repeats until the root agent receives all COST messages from all its children agents and calculates the aggregated fitness values. At this point, note that the cost of each constraint is doubly counted in the aggregated fitness values because the local fitness values of both agents in the scope of the constraint are aggregated together. Thus, the root agent divides the aggregated fitness values by two (Line 38) before starting the next phase.

Finally, in the **update phase**, the agents synchronize on their best local and global particles in the `BEST_UPDATE` procedure (Procedure **??**) and update the positions and velocities of their particles in the `VARIABLE_UPDATE` procedure (Procedure 6).

To synchronize their best local and global particles, the root agent first checks if a better local position has been found for each particle (Lines 44-46). If this is the case, it updates the best position $P_k^i.p_{best}$ and its corresponding fitness value $P_k^i.p_{best}.fitness$ before storing that particle in the set PB (Lines 47-49). The root agent also checks if a better global position has been found (Line 50). If so, it updates the best global position $P^i.g_{best}$ and its corresponding fitness value $P^i.g_{best}.fitness$ before storing that particle in a variable P^* (Lines 51-53). The root agent then sends both PB and P^* in a BEST message to each of its children agents (Line 60).

When each of its children agents receives the BEST message from the root agent, it iterates over all the particles P_k in the set PB, and assigns the positions of those particles as best positions $P_k^i.p_{best}$ of the corresponding particles P_k^i in its local copy (Lines 56-57). Similarly, if a better global particle has been found, it assigns the position of that particle $P^*.x_i$ as its best global position $P^i.g_{best}$ (Lines 58-59). It then propagates both PB and

5. Recall that we consider only binary cost functions in this paper.

Procedure 5: BEST_UPDATE($P^i.fitness$)

```
42 PB ← ∅
43 P* ← ∅
44 if a_i = root then
45     foreach P_k^i ∈ P^i do
46         if P_k^i.fitness < P_k^i.p_best.fitness then
47             P_k^i.p_best ← P_k^i.x_i
48             P_k^i.p_best.fitness ← P_k^i.fitness
49             PB ← PB ∪ {P_k^i}
50         if P_k^i.fitness < P^i.g_best.fitness then
51             P^i.g_best ← P_k^i.x_i
52             P^i.g_best.fitness ← P_k^i.fitness
53             P* ← P_k^i
54 else
55     Wait until BEST(PB, P*) is received from PR_i
56     foreach P_k ∈ PB do
57         P_k^i.p_best ← P_k.x_i
58     if P* ≠ ∅ then
59         P^i.g_best ← P*.x_i
60 Send BEST(PB, P*) to each agent a_j ∈ CH_i
61 return P*
```

Procedure 6: VARIABLE_UPDATE(P^*)

```
62 Calculate s_c and f_c using Equations 8 and 9
63 foreach P_k^i ∈ P^i do
64     if P_k^i = P* then
65         Calculate P_k^i.v_i and P_k^i.x_i using Equations 4 and 6
66     else
67         Calculate P_k^i.v_i and P_k^i.x_i using Equations 5 and 6
68 Send VALUE(P^i.x_i) to each agent a_j ∈ N_i
```

P^* that it received in the BEST message to each of its children agents (Line 60). This process repeats down the pseudo-tree until all agents synchronize their best local and global particles. Finally, the agents increment their cycle counters by one (Line 17).

To update the positions and velocities of the particles, we adapt the update equations used by Guaranteed Convergence PSO (GCPSO) (van den Bergh & Engelbrecht, 2002). At a high level, each agent a_i uses Equations 4 and 5 to update the velocities of the global best particle P^* and other particles $P_k^i \in P^i \setminus \{P^*\}$, respectively, and uses Equation 6 to update the positions of all particles $P_k^i \in P^i$ (Lines 64-67). The revised position is bounded within the acknowledged domain D_i:

$$P^*.v_i^{(t)} = -P^*.x_i^{(t-1)} + P^i.g_{best}^{(t-1)} + wP^*.v_i^{(t-1)} + \rho^{(t)}(1 - 2r_2) \tag{4}$$

$$P_k^i.v_i^{(t)} = wP_k^i.v_i^{(t-1)} + r_1c_1(P_k^i.p_{best}^{(t-1)} - P_k^i.x_i^{(t-1)}) + r_2c_2(P^i.g_{best}^{(t-1)} - P_k^i.x_i^{(t-1)}) \tag{5}$$

$$P_k^i.x_i^{(t)} = P_k^i.x_i^{(t-1)} + P_k^i.v_i^{(t)} \tag{6}$$

In these equations, the superscripts (t) denote the value of the variables at the t^{th} cycle. Here, w, c_1, and c_2 are user-defined input parameters to the algorithm; r_1 and r_2 are two random values that are uniformly sampled from the range $[0, 1]$ by each agent in each cycle; and $\rho^{(t)}$ is defined using Equation 7:

$$\rho^{(t)} = \begin{cases} 2\rho^{(t-1)} & \text{if } s_c^{(t-1)} > max_{s_c} \\ 0.5\rho^{(t-1)} & \text{else if } f_c^{(t-1)} > max_{f_c} \\ \rho^{(t-1)} & \text{otherwise} \end{cases} \tag{7}$$

where max_{s_c} and max_{f_c} are user-defined input parameters of the algorithm; and both s_c and f_c are calculated using Equations 8 and 9, respectively:

$$s_c^{(t)} = \begin{cases} s_c^{(t-1)} + 1 & \text{if } P^{*(t)} \neq P^{*(t-1)} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$f_c^{(t)} = \begin{cases} 0 & \text{if } P^{*(t)} \neq P^{*(t-1)} \\ f_c^{(t-1)} + 1 & \text{otherwise} \end{cases} \tag{9}$$

Intuitively, w represents an inertia weight that defines the influence of the velocity of the previous cycle on the velocity in the current cycle. The constants c_1 and c_2 are called the cognitive and social constants, respectively, in the literature because they affect the terms $P_k^i.p_{best}^{(t-1)} - P_k^i.x_i^{(t-1)}$ and $P^i.g_{best}^{(t-1)} - P_k^i.x_i^{(t-1)}$, which are called cognition and social components, respectively. The cognition component is called such because it considers the particle's own attributes only while the social component is called such because it involves interactions between two particles. Both of the constants c_1 and c_2 define the influence of local and global best positions on the velocity of particles in the current cycle.

The parameter ρ represents the diameter of an area around the global best particle that particles can explore. Its value is determined by the count of consecutive successes s_c and failures f_c. Success is defined when the fitness value of the global best particle improves, and failure is defined when the fitness value remains unchanged. When there are more consecutive successes than a threshold max_{s_c}, the diameter ρ doubles to increase random exploration because the current location of the best particle is promising. On the other hand, when there are more consecutive failures than a threshold max_{f_c}, the diameter ρ is halved to focus the search closer around the location of the best particle.

4.1 Crossover

Although PCD provides reasonable anytime solution quality in several benchmark problems (see details in Section 6.3), the scope for incorporating other genetic operators still exists.

Procedure 7: CROSSOVER

69 Calculate $P_k^i.b_p$ using Equation 10
70 Choose two particle P_a^i and P_b^i.
71 Calculate $P_a^i.v_i$ and $P_a^i.x_i$ using Equations 13 and 11
72 Calculate $P_b^i.v_i$ and $P_b^i.x_i$ using Equations 14 and 12

Hence, in this section, we introduce a new crossover operator that further improves the solution quality of PCD. We refer to this version of PCD with the new crossover operator as PCD_CrossOver. In centralized hybrid PSO models, arithmetic crossover of position and velocity vectors have shown promising results (Lovbjerg, Rasmussen, Krink, et al., 2001). In a centralized scenario, algorithms can execute crossover operations simultaneously for all the variables. But in a distributed scenario, either the agents need to agree in a cooperative crossover execution (Chen, Liu, He, & Yu, 2020) and need to exchange information or the agents can execute crossover operation only for the variables that they hold. In this paper, we follow the latter approach to not incur additional messaging and synchronization overheads. We describe the crossover operation in Procedure 7. Specifically, each agent a_i uses the local fitness value $P_k^i.local\_fitness$ of each particle P_k^i in the evaluation phase (Line 32) to calculate the crossover probability $P_k^i.b_p$ for each particle P_k^i using Equation 10 (Line 69).

$$P_k^i.b_p = \frac{|P_k^i.local\_fitness|}{\sum_{j=1}^{K} |P_j^i.local\_fitness|} \tag{10}$$

Then, each agent a_i selects two random particle P_a^i and P_b^i from its set P^i according to the crossover probabilities (Line 70), and updates their positions using the following crossover operations (Line 71, Line 72):

$$P_a^i.x_i^{(t)} = rP_a^i.x_i^{(t-1)} + (1-r)P_b^i.x_i^{(t-1)} \tag{11}$$

$$P_b^i.x_i^{(t)} = rP_b^i.x_i^{(t-1)} + (1-r)P_a^i.x_i^{(t-1)} \tag{12}$$

where r is a random number from the range $[0, 1]$. If $|P_a.v_i^{(t-1)} + P_b.v_i^{(t-1)}| \neq 0$, then their velocities are also updated using the following crossover operations (Line 71, Line 72). The updated positions are constrained by their domain Di:

$$P_a^i.v_i^{(t)} = \left(\frac{P_a^i.v_i^{(t-1)} + P_b^i.v_i^{(t-1)}}{|P_a^i.v_i^{(t-1)} + P_b^i.v_i^{(t-1)}|} \right) |P_a^i.v_i^{(t-1)}| \tag{13}$$

$$P_b^i.v_i^{(t)} = \left(\frac{P_a^i.v_i^{(t-1)} + P_b^i.v_i^{(t-1)}}{|P_a^i.v_i^{(t-1)} + P_b^i.v_i^{(t-1)}|} \right) |P_b^i.v_i^{(t-1)}| \tag{14}$$

Otherwise, the velocities are updated using the regular update operations described in Equations 4 and 5.

Here we give a real-world example of using a Continuous Distributed Constraint Optimization Problem (C-DCOP) the crossover operator to provide velocity updates in a wireless sensor network (WSN) implementation. Nodes in a WSN work together to collect data from their surroundings and make decisions based on that data. Each node in the network has a collection of variables that indicate the local decisions it can make, such as turning a sensor on or off or modifying its transmission power. When applying a crossover operator for velocity update, two-child nodes can trade some of their decision variables, and the parent node can inherit the combined features of the two children. For example, if one child node has a high transmission power and the other child node has a high battery level, the parent node can inherit both features to optimize its decision-making process. The position update would then update the node's decision variables based on the calculated velocity.

4.2 Example Partial Trace

We now provide a partial trace of our PCD algorithm on the example C-DCOP of Figure 1. Assume that the number of particles $K = 4$. In the **initialization phase**, the agents cooperatively build the BFS pseudo-tree shown in Figure 2, after which each agent a_i is aware of its set of neighboring agents N_i, its set of children agents CH_i, and its parent agent PR_i:

$$N_1 = \{a_2, a_3, a_4\}; CH_1 = \{a_2, a_3, a_4\}; PR_1 = \emptyset \tag{15}$$
$$N_2 = \{a_1\}; CH_2 = \emptyset; PR_2 = a_1 \tag{16}$$
$$N_3 = \{a_1, a_4\}; CH_3 = \emptyset; PR_3 = a_1 \tag{17}$$
$$N_4 = \{a_1, a_3\}; CH_4 = \emptyset; PR_4 = a_1 \tag{18}$$

Each agent a_i then creates a set of particles $P^i = \{P^i_1, P^i_2, P^i_3, P^i_4\}$ and initializes their position and velocity attributes $P^i_k.x_i$ and $P^i_k.v_i$ for all particles $P^i_k \in P^i$. Assume that they are initialized using the assignments below:

$$P_1.X = \{x_1 = -1.0, x_2 = 1.2, x_3 = -2.0, x_4 = 2.0\} \tag{19}$$
$$P_2.X = \{x_1 = -2.0, x_2 = 2.0, x_3 = -1.0, x_4 = 1.0\} \tag{20}$$
$$P_3.X = \{x_1 = 0.0, x_2 = 1.0, x_3 = 2.0, x_4 = -2.0\} \tag{21}$$
$$P_4.X = \{x_1 = 1.1, x_2 = -1.0, x_3 = 1.5, x_4 = 0.5\} \tag{22}$$
$$P_1.V = P_2.V = P_3.V = P_4.V = \{v_1 = 0.0, v_2 = 0.0, v_3 = 0.0, v_4 = 0.0\} \tag{23}$$

Then, each agent a_i sends its position assignments $P^i.x_i$ in a VALUE message to each of its neighboring agents in N_i:

- Agent a_1 sends a VALUE($P^1.x_1$) message to each of its neighboring agents a_2, a_3, and a_4, where $P^1.x_1 = \{P^1_1.x_1, P^1_2.x_1, P^1_3.x_1, P^1_4.x_1\} = \{-1.0, -2.0, 0.0, 1.1\}$.
- Agent a_2 sends a VALUE($P^2.x_2$) message to its neighboring agent a_1, where $P^2.x_2 = \{P^2_1.x_2, P^2_2.x_2, P^2_3.x_2, P^2_4.x_2\} = \{1.2, 2.0, 1.0, -1.0\}$.

Table 2: Local Fitness Scores

	Agent a_1	Agent a_2	Agent a_3	Agent a_4
Particle P_1	-1.44	-0.44	21.00	10.00
Particle P_2	14.00	0.00	12.00	10.00
Particle P_3	-9.00	-1.00	16.00	8.00
Particle P_4	6.64	0.21	7.51	4.92

Table 3: Fitness Scores

	Agent a_1	Agent a_2	Agent a_3	Agent a_4
Particle P_1	14.56	-0.44	21.00	10.00
Particle P_2	18.00	0.00	12.00	10.00
Particle P_3	7.00	-1.00	16.00	8.00
Particle P_4	9.60	0.21	7.51	4.92

- Agent a_3 sends a VALUE($P^3.x_3$) message to each of its neighboring agents a_1 and a_4, where $P^3.x_3 = \{P_1^3.x_3, P_2^3.x_3, P_3^3.x_3, P_4^3.x_3\} = \{-2.0, -1.0, 2.0, 1.5\}$.
- Agent a_4 sends a VALUE($P^4.x_4$) message to each of its neighboring agents a_1 and a_3, where $P^4.x_4 = \{P_1^4.x_4, P_2^4.x_4, P_3^4.x_4, P_4^4.x_4\} = \{2.0, 1.0, -2.0, 0.5\}$.

In the **evaluation phase**, each agent a_i waits for the VALUE messages from its neighboring agents. Upon receiving the VALUE messages from *all* of its neighboring agents, it calculates the local fitness value $P_k^i.local\_fitness$ for each particle $P_k^i \in P^i$. For example, after receiving VALUE($P^1.x_1$) and VALUE($P^3.x_3$) from agents a_1 and a_3, respectively, agent a_4 calculates $P_1^4.local\_fitness$ for particle P_1^4 as follows (see Figure 1 for the set of cost functions of our example C-DCOP):

$$\begin{aligned} P_1^4.local\_fitness &= f_{14}\left(P_1^1.x_1, P_1^4.x_4\right) + f_{34}\left(P_1^3.x_3, P_1^4.x_4\right) && (24)\\ &= 2\left(P_1^1.x_1\right)^2 - 2\left(P_1^4.x_4\right)^2 + \left(P_1^3.x_3\right)^2 + 3\left(P_1^4.x_4\right)^2 && (25)\\ &= 2\left(-1\right)^2 - 2\left(2\right)^2 + \left(-2\right)^2 + 3\left(2\right)^2 && (26)\\ &= 10 && (27)\end{aligned}$$

Table 2 tabulates the values of $P_k^i.local\_fitness$ for each particle P_k^i of each agent a_i.

After computing the local fitness values, since agent a_4 does not have any child agent, it assigns its local fitness value $P_k^4.local\_fitness$ of each particle $P_k^4 \in P^4$ to that particle's regular fitness value $P_k^4.fitness$ and sends that information to its parent agent a_1 in a COST message. Similarly, agents a_2 and a_3 also do the same as they too do not have any child agent. Table 3 tabulates the values of $P_k^i.fitness$ for each particle P_k^i of each agent a_i, and the COST messages sent by the agents are below:

- Agent a_2 sends a COST($P^2.fitness$) message to its parent agent a_1, where $P^2.fitness = \{P_1^2.fitness, P_2^2.fitness, P_3^2.fitness, P_4^2.fitness\} = \{-0.44, 0.00, -1.00, 0.21\}$.

- Agent a_3 sends a COST($P^3.fitness$) message to its parent agent a_1, where $P^3.fitness = \{P_1^3.fitness, P_2^3.fitness, P_3^3.fitness, P_4^3.fitness\} = \{21.00, 12.00, 16.00, 7.51\}$.
- Agent a_4 sends a COST($P^4.fitness$) message to its parent agent a_1, where $P^4.fitness = \{P_1^4.fitness, P_2^4.fitness, P_3^4.fitness, P_4^4.fitness\} = \{10.00, 10.00, 8.00, 4.92\}$.

As the root agent a_1 has children agents, it waits for the COST messages from its children agents. Upon receiving the COST messages from *all* of its children agents, it calculates the fitness value $P_k^1.fitness$ for each particle $P_k^1 \in P^1$. For example, it calculates $P_1^1.fitness$ for particle P_1^1 as follows:

$$P_1^1.fitness = P_1^1.local\_fitness + P_1^2.fitness + P_1^3.fitness + P_1^4.fitness \tag{28}$$
$$= -1.44 + (-0.44) + 21.00 + 10.00 \tag{29}$$
$$= 29.12 \tag{30}$$

As the cost from each cost function in the C-DCOP is doubly counted, the root agent divides its fitness value of each of its particles by two. For example, it updates $P_1^1.fitness$ for particle P_1^1 as follows:

$$P_1^1.fitness = \frac{1}{2}P_1^1.fitness = \frac{1}{2}29.12 = 14.56 \tag{31}$$

In the **update phase**, since this is the first iteration, each particle P_k^1 of the root agent a_1 has a better local position. Thus, the best position $P_k^1.p_{best}$ of each particle P_k^1 is updated to the particle's current position $P_k^1.x_1$ and all the particles are added into the set PB. Similarly, a better global position is found. Thus, the best global position $P^1.g_{best}$ is updated to the position $P_3^1.x_1$ of the best particle P_3^1 and that particle is assigned to the variable P^*. The agent then sends both PB and P^* in a BEST message to its children agents a_2, a_3, and a_4:

- Agent a_1 sends a BEST(PB, P^*) message to its children agents a_2, a_3, and a_4, where $PB = \{P_1^1, P_2^1, P_3^1, P_4^1\}$ and $P^* = P_3^1$.

All non-root agents a_2, a_3, and a_4 wait for the BEST messages from their parent agent a_1. Upon receiving the BEST message, for each particle $P_k \in PB$ in the BEST message, each agent assigns the position $P_k.x_i$ as the best local position $P_k^i.p_{best}$ of the corresponding particle P_k^i. Since PB contains all four particles, the best local positions of all four particles are updated. Similarly, each agent a_i also updates the best global position $P^i.g_{best}$ to the position $P^*.x_i = P_3^i.x_i$ of the best particle P^* in the BEST message. Table 4 tabulates the values of the best local positions $P_k^i.p_{best}$ for each particle P_k^i and the best global position $P^i.g_{best}$ of each agent a_i.

Each agent a_i then increments its cycle counter $t = 1$ before starting the process of updating the positions $P_k^i.x_i$ and velocities $P_k^i.v_i$ of its particles P_k^i. To do so, each agent first calculates s_c, f_c, and ρ using Equations 8, 9, and 7, respectively. In the following, assume that we set $max_{s_c} = 15$, $max_{f_c} = 5$, $w = 0.72$, $c_1 = 1.49$, $c_2 = 1.49$.[6] Since a

6. We discuss the choice of parameter values in detail in Section 6.2

Table 4: $P_k^i.p_{best}$ and $P^i.g_{best}$ of Agent a_i and Particle P_k^i after BEST Update

(a) $P_k^i.p_{best}$ for each Agent a_i

	Agent a_1	Agent a_2	Agent a_3	Agent a_4
Particle P_1	-1.00	1.20	-2.00	2.00
Particle P_2	-2.00	2.00	-1.00	1.00
Particle P_3	0.00	1.00	2.00	-2.00
Particle P_4	1.10	-1.00	1.50	0.50

(b) $P^i.g_{best}$ for each Agent a_i

	Agent a_1	Agent a_2	Agent a_3	Agent a_4
Particle P^*	0.00	1.00	2.00	-2.00

new P^* is found in this cycle, the agent sets $s_c^{(1)} = 1$ (see Equation 8) and $f_c^{(1)} = 0$ (see Equation 9). And since $0 = s_c^{(0)} \not> max_{s_c} = 15$ and $0 = f_c^{(0)} \not> max_{f_c} = 5$, $\rho^{(1)} = \rho^{(0)} = 1$ (see Equation 7), each agent a_i updates the positions $P_k^i.x_i$ and velocities $P_k^i.v_i$ of its particles P_k^i using Equations 4 to 6. For example, agent a_1 updates the position and velocity of its best global particle P_3^1 as follows:

$$P_3^1.v_1^{(t)} = -P_3^1.x_1^{(0)} + P^1.g_{best}^{(0)} + wP_3^1.v_1^{(0)} + \rho^{(1)}(1 - 2r_2) \tag{32}$$
$$= 0 + 0 + 0.72 \cdot 0 + 1(1 - 2 \cdot 0.4) \tag{33}$$
$$= 0.20 \tag{34}$$
$$P_3^1.x_1^{(1)} = P_3^1.x_1^{(0)} + P_3^1.v_1^{(1)} = 0 + 0.2 = 0.20 \tag{35}$$

and the position and velocity of a non-best global particle P_1^1 as follows:

$$P_1^1.v_1^{(1)} = wP_1^1.v_1^{(0)} + r_1c_1(P_1^1.p_{best}^{(0)} - P_1^1.x_1^{(0)}) + r_2c_2(P^1.g_{best}^{(0)} - P_1^1.x_1^{(0)}) \tag{36}$$
$$= 0.72 \cdot 0 + 0.7 \cdot 1.49(-1 - (-1)) + 0.4 \cdot 1.49(0 - (-1)) \tag{37}$$
$$= 0.60 \tag{38}$$
$$P_1^1.x_1^{(1)} = P_1^1.x_1^{(0)} + P_1^1.v_1^{(1)} = -1 + 0.60 = -0.40 \tag{39}$$

Table 5 tabulates the updated position $P_k^i.x_i^{(1)}$ and velocity $P_k^i.v_i^{(1)}$ for each particle P_k^i of each agent a_i.

We now describe an example of the crossover operation in PCD_CrossOver variant. Using the local fitness values calculated in Table 2, each agent a_i calculates its crossover probability $P_k^i.b_p$ for each of its particles P_k^i using Equation 10. For example, agent a_1 calculates $P_1^1.b_p$ for particle P_1^1 as follows:

Table 5: Updated Particle Position and Velocity

	Agent a_1	Agent a_2	Agent a_3	Agent a_4
Particle P_1	$v_1 = 0.60$ $x_1 = -0.40$	$v_2 = 1.19$ $x_2 = -0.81$	$v_3 = 0.20$ $x_3 = 0.20$	$v_4 = -0.66$ $x_4 = 0.44$
Particle P_2	$v_1 = -0.12$ $x_1 = 1.08$	$v_2 = -0.60$ $x_2 = 1.40$	$v_3 = 0.20$ $x_3 = 1.20$	$v_4 = 1.19$ $x_4 = 0.19$
Particle P_3	$v_1 = 2.38$ $x_1 = 0.38$	$v_2 = 1.79$ $x_2 = 0.79$	$v_3 = 0.20$ $x_3 = 2.20$	$v_4 = 0.30$ $x_4 = 1.80$
Particle P_4	$v_1 = -2.38$ $x_1 = -0.38$	$v_2 = -1.79$ $x_2 = -0.79$	$v_3 = 0.20$ $x_3 = -1.80$	$v_4 = -1.49$ $x_4 = -0.99$

Table 6: Crossover Probabilities

	Agent a_1	Agent a_2	Agent a_3	Agent a_4
Particle P_1	0.046	0.267	0.372	0.304
Particle P_2	0.450	0.000	0.212	0.304
Particle P_3	0.290	0.606	0.283	0.243
Particle P_4	0.214	0.127	0.133	0.149

$$P_1^1.b_p = \frac{|-1.44|}{|-1.44| + |14| + |-9| + |6.64|} = 0.046 \tag{40}$$

Table 6 tabulates the crossover probabilities $P_k^i.b_p$ for each particle P_k^i of each agent a_i.

Using these probabilities, each agent a_i selects two random particles P_a^i and P_b^i and update their positions and velocities using Equations 11 to 14. For example, agent a_1 selects particles P_2^1 and P_4^1 and updates their positions as follows:

$$\begin{aligned} P_2^1.x_1^{(1)} &= rP_2^1.x_1^{(0)} + (1-r)P_4^1.x_1^{(0)} && (41)\\ &= 0.3 \cdot -2 + (1-0.3)1.1 && (42)\\ &= 0.17 && (43)\\ P_4^1.x_1^{(1)} &= rP_4^1.x_1^{(0)} + (1-r)P_2^1.x_1^{(0)} && (44)\\ &= 0.3 \cdot 1.1 + (1-0.3)(-2) && (45)\\ &= -1.07 && (46) \end{aligned}$$

where $r = 0.3$. Their velocities are not updated because $|P_2^1.v_1^{(0)} + P_4^1.v_i^{(0)}| = |0+0| = 0$.

5. Theoretical Analysis

In this section, we first prove PCD is an anytime algorithm, that is, the quality of the best solution improves and never degrades over time. We then discuss the complexity of PCD in terms of its communication and memory requirements. In this section, we use the term *iteration* to refer to communication steps, which is the time needed for messages sent by an agent to be received by its neighboring agents.

Lemma 1 *At iteration $t+h$, the root agent a_{root} is aware of the $P^{root}.p_{best}$ and $P^{root}.g_{best}$ up to iteration t, where h is the height of the pseudo-tree.*

Proof. To prove the lemma, we show that, at iteration $t+h$, the root agent has enough information to calculate $P^{root}.p_{best}$ and $P^{root}.g_{best}$ up to iteration t. That is, the root agent can calculate the fitness of each particle. However, the root agent requires the COST messages from all the agents in CH_{root} in order to calculate the fitness of each particle using Equation 2. To get these messages, the root agent has to wait for at most h iterations since the height of the pseudo-tree is h. Consequently, at iteration $t+h$, the root agent is capable of calculating the fitness of each particle up to iteration t. □

Lemma 2 *At iteration $t+2h$, each agent a_i is aware of $P^i.p_{best}$ and $P^i.g_{best}$ up to iteration t, where h is the height of the pseudo-tree.*

Proof. It will take at most h iterations for the BEST message that contains (PB, P^*) from the root agent to reach all other agents since h is the height of the pseudo-tree. Therefore, combining this observation and Lemma 1, it will take at most $t+h+h = t+2h$ iterations for each agent to be aware of $P.p_{best}$ and $P.g_{best}$ up to iteration t. □

Theorem 1 *PCD is an anytime algorithm.*

Proof. From Lemma 2, at iteration $t+2h$ and $t+2h+\delta$, where $\delta \geq 0$, each agent is aware of $P^i.p_{best}$ and $P^i.g_{best}$ up to iterations t and $t+\delta$, respectively. Since $P^i.p_{best}$ and $P^i.g_{best}$ only get updated if a better solution is found, $P^i.p_{best}.fitness$ and $P^i.g_{best}.fitness$ at iteration $t+2h+\delta$ is no larger than $P^i.p_{best}.fitness$ and $P^i.g_{best}.fitness$ at iteration $t+2h$, respectively. In other words, the cost of the solution is monotonically non-increasing over time. Hence, PCD is an anytime algorithm. □

Theorem 2 *For a binary constraint graph $G = (A, E)$, the number of messages of PCD in each cycle is $O(|E| + |A|)$.*

Proof. In each cycle:

- The `INITIALIZATION` procedure requires $2|E|$ messages, where $|E|$ is the number of edges in the constraint graph, since each agent sends a VALUE message to each of its neighbors.
- The `EVALUATION` procedure requires $|A| - 1$ messages since each agent, except for the root agent, sends a COST message to its parent agent.

- The `BEST_UPDATE` procedure requires $|A|-1$ messages since each agent, except for the root agent, receives a BEST message from its parent agent.
- The `VARIABLE_UPDATE` procedure requires $2|E|$ messages since each agent sends a VALUE message to each of its neighbors.

Since each cycle of PCD is composed of either the `INITIALIZATION` or `VARIABLE_UPDATE` procedures – `INITIALIZATION` procedure in the first cycle and `VARIABLE_UPDATE` in subsequent cycles – as well as the `EVALUATION` and `BEST_UPDATE` procedures, the number of messages per cycle is thus $2|E| + |A| - 1 + |A| - 1 = O(|E| + |A|)$. □

Theorem 3 *The total message size complexity per agent of PCD in each cycle is $O(K|A|)$, where K is the total number of particles.*

Proof. Each agent in PCD sends three types of messages: VALUE, COST, and BEST messages. Each of these messages contains a constant amount of information for each of the K particles. Hence, the size of each message is $O(K)$. At each cycle, each agent a_i sends at most $|N_i|$ VALUE messages, one COST message, and $|CH_i|$ BEST messages. Therefore, the total number of messages an agent sends is at most $|N_i| + 1 + |CH_i|$. In the worst case, $|N_i| \approx |A|$ and $|CH_i| \approx |A|$. Consequently, the total number of message sent by an agent is $O(|A| + 1 + |A|) = O(|A|)$. Therefore, in each cycle, the total message size per agent is $O(K|A|)$. □

6. Experimental Results

We now provide our results from empirical evaluations of PCD. First, we study the effect of different parameter choices to fine tune the parameters. This is because the performance of PCD depends on the choice of its various parameter values and whether crossover operations are used to improve the algorithm further. Then, we compare the fine-tuned version of PCD against the state-of-the-art C-DCOP algorithms[7] – namely HCMS (Voice et al., 2010), AC-DPOP (Hoang et al., 2020), and C-DSA (Hoang et al., 2020) – on four benchmark problems – random graphs, random trees, scale-free networks, and random sensor network problems.

6.1 Benchmark Problems

We evaluate PCD on four types of benchmark problems: *Random Graphs*, *Random Trees*, *Scale-Free Networks*, and *Random Sensor Network Problems*.

Random Graphs: We use the Erdős-Rényi topology (Erdős & Rényi, 1960) to construct random graphs. We use two settings for random graphs – sparse, where each pair of nodes in the graph has a probability of 0.2 to have an edge between them, and dense, where the edge probability is 0.6. We set the number of agents $|A|$, which corresponds to the number of nodes in the graph, to 50 and the domain D_i of each agent a_i to $[-50, 50]$. We use binary quadratic functions of the form $ax^2 + bxy + cy^2$ as cost functions, where the coefficients a, b, and c are all randomly chosen between $[-5, 5]$.

7. Although Hoang et al. (Hoang et al., 2020) proposed three non-exact C-DCOP algorithms, we only compare with AC-DPOP and C-DSA here because they are reported to provide the best solutions among the approximate algorithms proposed in their paper.

Random Trees: We follow Hoang et al. (Hoang et al., 2020) and include random trees as one of the benchmark problems since the memory requirement of AC-DPOP is smaller on trees; it is exponential in the tree width of the graph (Hoang et al., 2020). The experimental configurations are similar to the random graph setting, except that we ensure that no cycles are formed.

Scale-Free Networks: In scale-free networks, the problem settings are similar to Random Graphs, except we use the Barabsi-Albert (BA) network topology model (Barabási & Albert, 1999) to generate our constraint networks. To construct the network, initially, we randomly connect a set of agents. We then connect a new agent with a set of m randomly selected existing agents with a probability proportional to the current link numbers. For our problem, we set number of agents $|A| = 100$ and $m = 3$ and the domain D_i of each agent a_i to $[-20, 20]$. Other experimental configurations are same as the random graph setting. The purpose of adding this experiment is to observe the impact of different network topologies (random graph, scale-free and random tree) and larger number of agents on the performance of PCD and PCD_CrossOver.

Random Sensor Network Problems: The fourth benchmark problem is motivated by an example sensor network problem, where the sensors are trying to maximize the signal strength of their radio communication (Nguyen, Yeoh, & Lau, 2012; Nguyen et al., 2019). This problem is motivated by real-world applications where a *meshed communication network* needs to be established by first responders of an emergency rescue operation (e.g., in a city ravaged by an earthquake or a hurricane) or by soldiers in the battlefield. In such a problem, the sensors are arranged in an 8×8 rectangular grid and can make small movements within its cell in the grid. Therefore, the number of agents (i.e., sensors) $|A|$ is 64. The domain D_i of each agent a_i is set to the cell size $[0, 10]$.

The strength of the radio communication between two sensors a_i and a_j is inversely proportional to their squared distance $d(a_i, a_j)^2$ and is subject to interference $\lambda(a_i, a_j)$ from obstacles between the sensors. We thus define the utility $f(a_i, a_j)$ between two sensors (a_i, a_j) as follows:

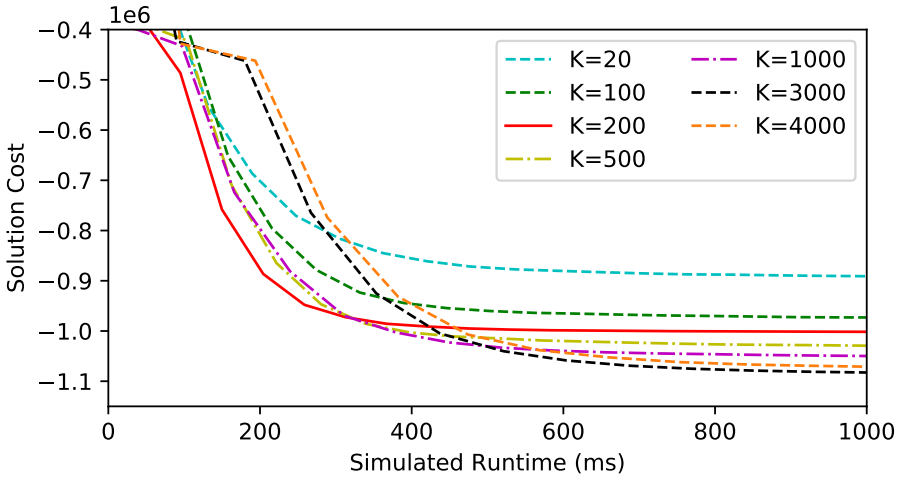
$$f(a_i, a_j) = \frac{C}{d(a_i, a_j)^2 \cdot \lambda(a_i, a_j)} \tag{47}$$

$$\lambda(a_i, a_j) = (r_{x_i} - x_i)^2 + (r_{y_i} - y_i)^2 + (r_{x_j} - x_j)^2 + (r_{y_j} - y_j)^2 + \eta_{ij} \tag{48}$$

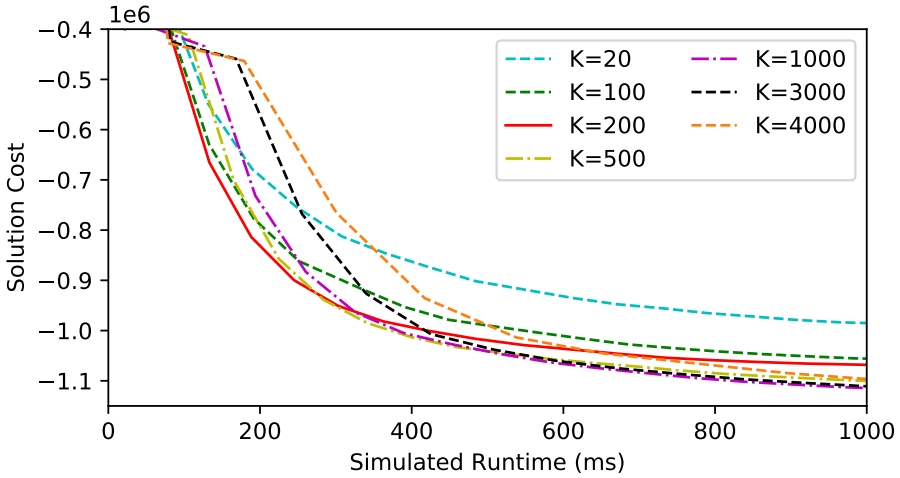
where r_{x_i} and r_{y_i} are random numbers sampled from the domain of agent a_i, η_{ij} is random noise chosen between the range $[1, 10]$ for each pair of sensors a_i and a_j, and C is a constant value of 10,000.

In all of the settings described above, we evaluate all the algorithms on 25 independently-generated problem instances and 20 times on each problem instance. For fairness, we use the simulated runtime metric (Sultanik, Lass, & Regli, 2008) to measure the runtime of the algorithms. The experiments are carried out on a computer with an Intel Core i5-6200U CPU with 2.3GHz processor and 8GB RAM. Our implementation is available in both Java[8] and Python[9].

8. `https://github.com/moumitachoudhury/PCD`
9. `https://github.com/SaminYaser/PCD`



(a) PCD



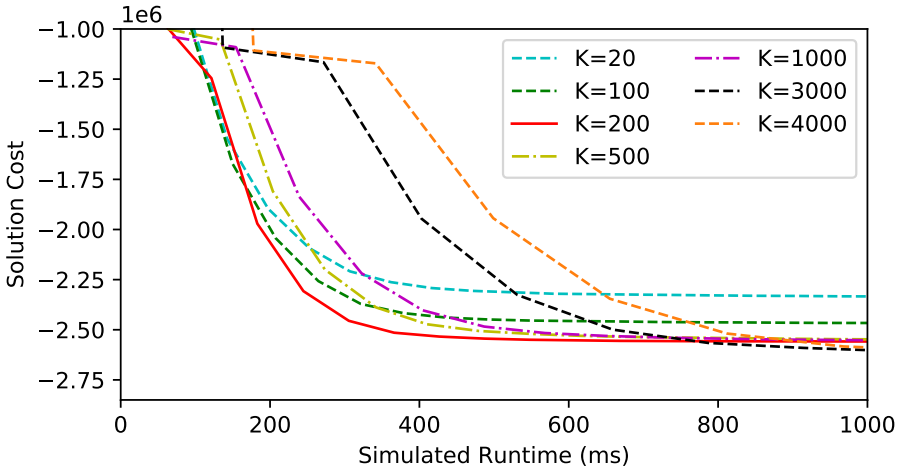
(b) PCD_CrossOver

Figure 3: Solution quality of PCD and PCD_CrossOver with different population sizes K on sparse random graphs.

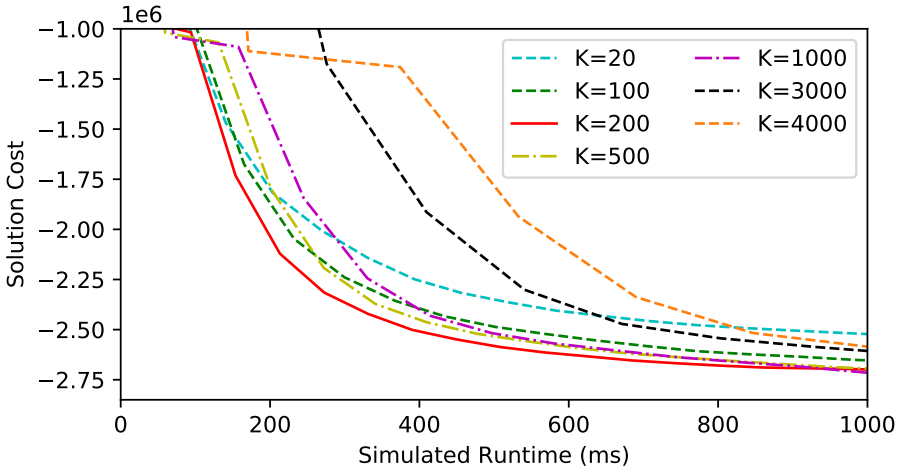
6.2 Fine-tuning Parameters

PCD and its variant PCD_CrossOver have several parameters including the number of particle K, inertia weight w, cognitive constant c_1, social constant c_2, and thresholds max_{f_c} and max_{s_c}. In all our experiments, we follow the recommendations from the literature (van den Bergh & Engelbrecht, 2002) and set $max_{f_c} = 5$ and $max_{s_c} = 15$. We now discuss how we choose the other parameter values.

To determine the value of the number of particles K, we conduct a preliminary experiment, where we compare the quality of solutions found by PCD and PCD_CrossOver for different values of K. Figures 3 and 4 plot the results on sparse and dense random graphs, respectively. In general, the quality of solutions improves with increasing K since a larger



(a) PCD

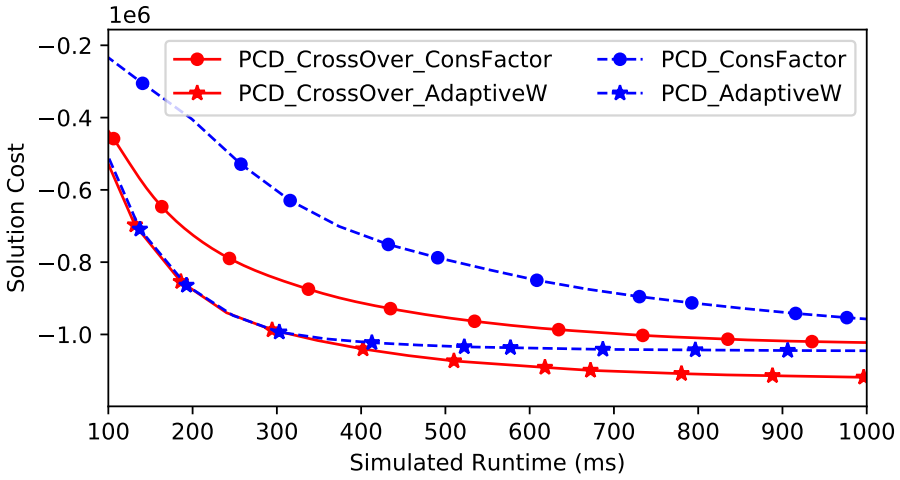


(b) PCD_CrossOver

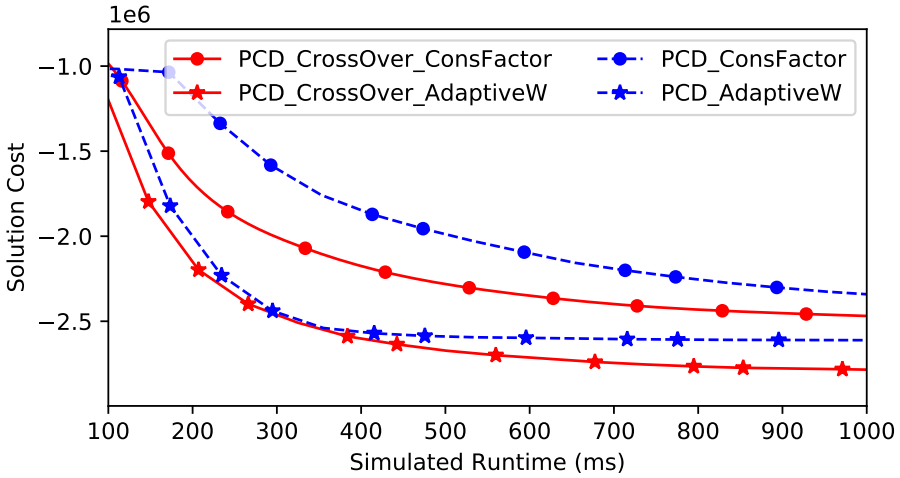
Figure 4: Solution quality of PCD and PCD_CrossOver with different population sizes K on dense random graphs.

population size allows for more diversity in the swarm. However, this comes at the cost for longer runtimes before the algorithms converge. Given these empirical observations, we choose to set $K = 200$ in all our subsequent experiments as it allows the algorithms to converge to reasonably good solutions – it is in fact very close to the best solutions with $K = 4000$ on dense random graphs – as well as converge to them relatively quickly compared to the other values of K. To determine the values of w, c_1, and c_2, we use two design choices:

- **AdaptiveW:** Here, we linearly decrease w using Equation 49:



(a) Sparse graph



(b) Dense graph

Figure 5: Solution quality of PCD and PCD_CrossOver with AdaptiveW or Constriction Factor on random graphs.

$$w = \frac{(w_{max} - w_{min}) \cdot t}{t_{max}} \tag{49}$$

where w_{max} and w_{min} are the maximum and minimum values of w and t_{max} is the maximum number of cycles. To determine the values of w_{max} and w_{min}, we experimented with different pairs of values typically used in experiments of centralized PSO (Shi & Eberhart, 1999, 1998; Carlisle & Dozier, 2000), and found that the best results are when $w_{max} = 1.4$ and $w_{min} = 0.4$. For the c_1 and c_2 values, we set them to $c_1 = c_2 = 1.49$, which have been a popular choice in the centralized PSO model (Eberhart & Shi, 2000; Van den Bergh & Engelbrecht, 2010).

- **Constriction Factor:** Here, we follow the literature (Clerc, 1999), where, instead of using Equation 5 to update the velocity of particles, we use Equation 50 instead:

$$P_k^i.v_i^{(t)} = w(P_k^i.v_i^{(t-1)} + r_1c_1(P_k^i.p_{best}^{(t-1)} - P_k^i.x_i^{(t-1)}) + r_2c_2(P^i.g_{best}^{(t-1)} - P_k^i.x_i^{(t-1)})) \quad (50)$$

$$w = \frac{2}{2 - \phi - \sqrt{(\phi^2 - 4\phi)}} \quad (51)$$

$$\phi = c_1 + c_2 > 4 \quad (52)$$

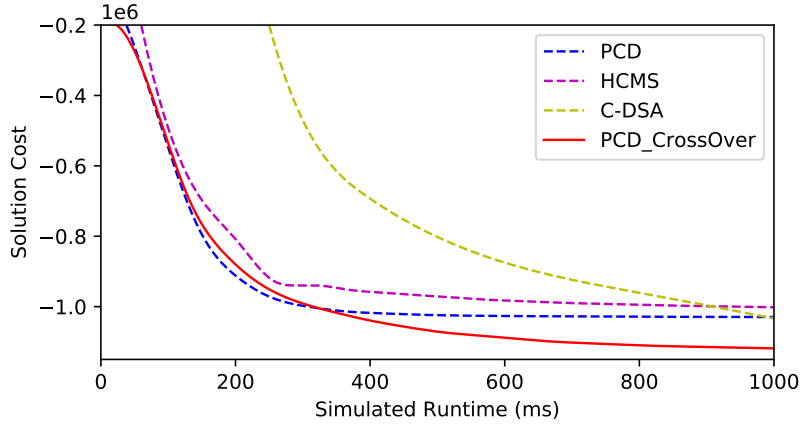
  To satisfy the constraint in Equation 52, we choose $c_1 = c_2 = 2.05$ to get $\phi = 4.1$ and $w = 0.7298$ because this set of parameter values has been proven to be a convergent parameter configuration for the centralized GCPSO model (Van den Bergh & Engelbrecht, 2010).

We then conduct another preliminary experiment, where we compare these two design choices for both PCD and PCD_CrossOver on random graphs. Figure 5 plots the results. It is clear that the parameters values tuned using AdaptiveW result in better solutions compared to when they are tuned using the constriction factor approach for both PCD and PCD_CrossOver and in both sparse and dense random graphs. Therefore, for all subsequent experiments, we use the AdaptiveW approach to tune the w, c_1, and c_2 parameter values.

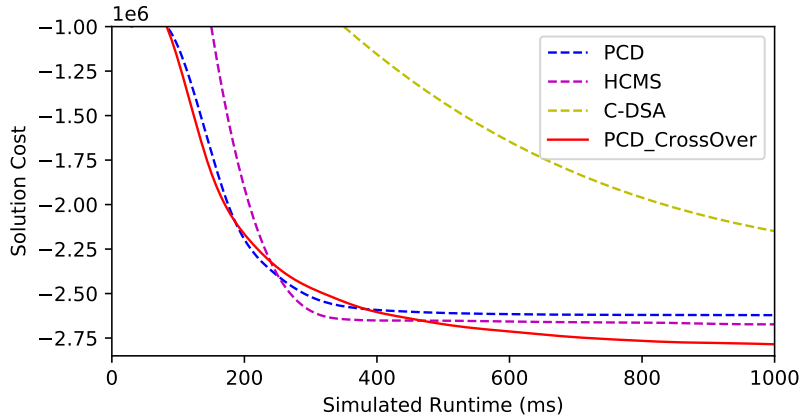
6.3 Comparisons with the State of the Art

In this section, we empirically compare PCD and PCD_CrossOver to existing state-of-the-art C-DCOP algorithms – HCMS (Voice et al., 2010) as well as C-DSA and AC-DPOP (Hoang et al., 2020) on the four benchmark problems described in Section 6.1. We follow the literature (Hoang et al., 2020) to determine the parameter values of HCMS, AC-DPOP, and DSA. Specifically, HCMS and AC-DPOP both maintain 3 discrete points per variable, where each discrete point is chosen randomly from the domain range. AC-DPOP moves each of its points 20 times, where each move is executed by solving a set of gradient equations; and C-DSA uses DSA-B with $p = 0.6$. We set the learning rate of HCMS to 0.01, which is the best value found in our experiments. Moreover, when appropriate we have taken the mean of K parallel runs to compare with our population-based approaches. Finally, it is worth noting that all the differences in the quality of solutions found by the algorithms that we highlight below are statistically significant with p-values that are less than 0.01.

Figure 6 shows the results of our PCD and PCD_CrossOver algorithms as well as the existing HCMS and C-DSA algorithms on random graphs. We omit the results of AC-DPOP because it ran out of memory in this setting. In both sparse and dense random graphs, PCD_CrossOver converges to a better solution compared to all the existing algorithms including PCD. Specifically, PCD_CrossOver improves PCD by 11.7% in sparse graphs and 10.4% in dense graphs, thereby demonstrating the significance of the crossover procedure described in Section 4.1. Moreover, PCD_CrossOver finds solutions that are better than existing algorithms by about $3.0\% - 13.1\%$ on sparse graphs and $7.6\% - 24.1\%$ on dense graphs after one second.



(a) Sparse graph



(b) Dense graph

Figure 6: Solution quality of PCD, PCD_CrossOver, HCMS, and C-DSA on random graphs.

Table 7: Solution quality of PCD, PCD_CrossOver, HCMS, and C-DSA on random graphs with different number of agents.

		PCD_CrossOver	PCD	HCMS	C-DSA
$\|A\| = 30$	$p = 0.2$	-504,082	-469,093	-423,383	-493,730
	$p = 0.6$	-1,166,414	-1,042,611	-1,055,878	-1,096,852
$\|A\| = 50$	$p = 0.2$	-1,151,581	-1,030,328	-974,416	-1,118,344
	$p = 0.6$	-2,897,026	-2,623,534	-2,730,943	-2,334,385
$\|A\| = 70$	$p = 0.2$	-2,129,397	-1,858,646	-2,049,757	-1,636,508
	$p = 0.6$	-5,157,537	-4,494,102	-5,060,671	—

Table 7 shows further comparisons on random graph settings varying the number of agents. For this experiment, we run each algorithm for 1500ms using 30, 50, and 70 agents,

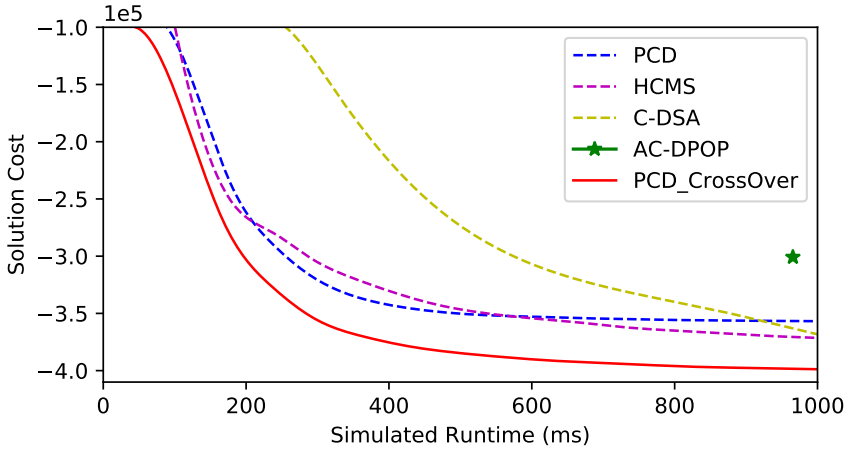


Figure 7: Solution quality of PCD, PCD_CrossOver, HCMS, C-DSA, and AC-DPOP on random trees.

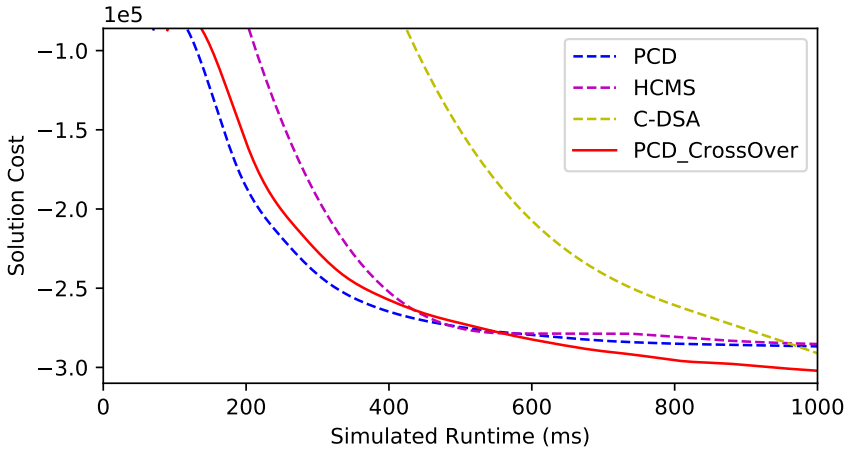


Figure 8: Solution quality of PCD, PCD_CrossOver, HCMS, and C-DSA on Scale-Free Graph.

each with both sparse (edge probability 0.2) and dense (edge probability 0.6) settings. For smaller graphs (i.e., $|A| = 30$), in both sparse and dense settings, the closest competitor of PCD_CrossOver is C-DSA. For larger graphs and sparse settings, the trends remain the same as the smaller instances. However, as the density and graph size increases, C-DSA takes more time than the competing algorithms and HCMS becomes the closest competitor of PCD_CrossOver. In all the settings, PCD_CrossOver 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 PCD_CrossOver in random graph settings.

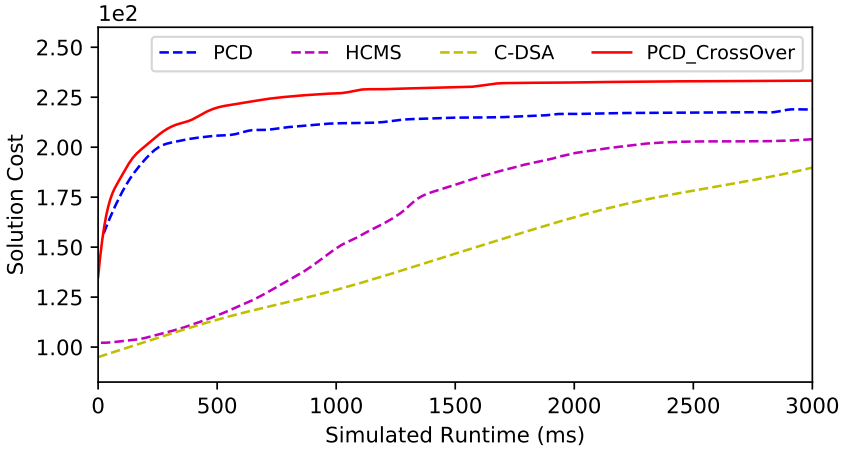


Figure 9: Solution quality of PCD, PCD_CrossOver, HCMS, and C-DSA on random sensor network problems.

Figure 7 shows the results of our PCD and PCD_CrossOver algorithms as well as the existing HCMS, C-DSA, and AC-DPOP algorithms on random trees. The observations and trends from the experiments on random graphs are comparable to random trees as well, where PCD_CrossOver converges to the best solution compared to all other algorithms. Specifically, PCD_CrossOver improves PCD by 16.2% and finds solutions that are better than existing algorithms by about $6.3\% - 8.3\%$ after one second. Figure 8 shows the performance comparison for scale-free networks in a larger graph setting ($|A| = 100$). Although, PCD, C-DSA, and HCMS show similar performances, PCD_CrossOver outperforms the existing algorithms by a margin of 22.28% for HCMS and 14.56% for C-DSA.

In addition, we perform the one-way ANOVA with post-hoc Tukey HSD test for all random graphs, random trees, and random sensor network problems. While we accomplish this, we take into account 4 heuristics (PCD, PCD Crossover, HCMS, and C-DSA) as treatments, each of which shows how well it does in comparison to the ideal mapping. The observed p-value for the one-way ANOVA F-statistic for each experiment is less than 0.05, indicating that one or more treatments are significantly different. Then, using a post-hoc test (Tukey HSD), we find that PCD and PCD Crossover's performance is significantly different from each of the remaining samples on its own (i.e., p 0.01).

Finally, Figure 9 shows the results of our PCD and PCD_CrossOver algorithms as well as the existing HCMS and C-DSA algorithms on random sensor network problems. Similar to random graphs, we omit the results of AC-DPOP because it also ran out of memory in this setting. However, unlike the previous two problem settings, we change the optimization problem from one that minimizes the cost of solutions to one that *maximizes* the cost of solutions to stay consistent with actual sensor networks, where the goal is to maximize signal strengths. The general trends from the previous two problem settings are also applicable here. To be exact, PCD_CrossOver also converges to the best solution compared to all other algorithms. Due to the crossover operator, offspring particles benefits from both parents. This permits a good exploration of the search space between particles, as such, facilitating the attainment of such an outcome. However, interestingly, PCD also finds better solutions

than HCMS and C-DSA in this problem setting. In contrast, they all converge to solutions of similar quality in the previous two problem settings. Specifically, PCD_CrossOver improves PCD by 6.1%, and both PCD_CrossOver and PCD outperform HCMS and C-DSA by about $12.6\% - 17.3\%$ and $6.9\% - 11.9\%$, respectively.

Therefore, the results in these four problem settings clearly demonstrate that PCD_CrossOver finds better solutions than existing state-of-the-art C-DCOP algorithms, highlighting the promise of particle swarm optimization based approaches to solve continuous C-DCOPs.

7. Conclusions and Future Work

Distributed Constraint Optimization Problems (DCOPs) have been used to model a number of multi-agent coordination problems. However, its use of discrete variables prevents it from accurately modeling problems with continuous variables. To overcome this limitation, researchers have proposed *Continuous DCOPs* (C-DCOPs), where the key change is that the variables are now continuous instead of discrete, as well as a number of algorithms to solve them. However, existing C-DCOP algorithms primarily rely on gradient-based optimization methods that require derivative calculations. Consequently, they are not suitable for non-differentiable optimization problems.

To remedy this limitation, we proposed a new approach that generalizes the centralized particle swarm optimization algorithm to a decentralized setting. The new algorithm, called PCD, and its variant that uses crossover operations, called PCD_CrossOver, maintains a set of particles in a decentralized manner, where each particle represents a candidate solution. They iteratively "move" the particles using a series of update equations, which corresponds to updating the solutions maintained over time. Upon termination, the algorithms return the best position over all particles and time steps, which corresponds to the best solution found. We provide theoretical proof for the anytime behavior of our algorithms and show empirical evidence that it outperforms existing state-of-the-art C-DCOP algorithms on four different benchmarks.

In the future, we plan to further investigate the potential of other population-based algorithms, such as *Artificial Bee Colony* (ABC) (Karaboga & Basturk, 2007) and *Cuckoo Search* (CS) (Yang & Deb, 2009), to solve DCOPs and C-DCOPs. Furthermore, we want to study different variants of PSO (e.g., Cooperative PSO (Van den Bergh & Engelbrecht, 2004) and Hybrid PSO (Angeline, 1998)) and the effect of applying other genetic operators that have been proposed over the last few decades to improve the solution quality of PSO. We are keenly interested in the successful deployment of particle swarm-based algorithms for solving real-world distributed constraint optimization problems. However, we recognize the critical importance of ensuring privacy (Kogan, Tassa, & Grinshpoun, 2022; Tassa, Grinshpoun, & Yanai, 2021) in the algorithmic design and implementation. Thus, we believe that further analysis of the privacy-preserving aspect of particle swarm-based algorithms is necessary for their successful application in practical scenarios. In the future, we would like to investigate the effects of different topologies on the performance of our algorithms. We are also interested in exploring whether our algorithms can be generalized to solve multi-objective C-DCOPs, which, to the best of our knowledge, have not yet been explored.

8. Availability of data and materials

The codes used to implement the model described in this paper can be found in the GitHub repository "PCD," which can be accessed via the following links:

1. Java: `https://github.com/moumitachoudhury/PCD`
2. Python: `https://github.com/SaminYaser/PCD`

The codes in these repositories can be used to replicate simulation results similar to those presented in this paper. Furthermore, upon reasonable request, the data presented in this paper may be made directly available to the requester.

9. Credit Authorship Contribution Statement

Moumita Choudhury: Conceptualization, Methodology, Writing – Original Draft, Software, Formal Analysis, Visualization. **Amit Sarker:** Data Curation, Formal Analysis, Validation. **Samin Yaser:** Writing – Review & Editing, Software. **Md. Maruf Al Alif Khan:** Software, Validation. **William Yeoh:** Writing – Original Draft, Writing – Review & Editing, Investigation. **Md. Mosaddek Khan:** Conceptualization, Supervision, Project Administration, Funding acquisition, Writing – Review & Editing.

10. Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

11. Acknowledgements

This research is mainly supported by the ICT Innovation Fund of Bangladesh Government. A preliminary version of this research has appeared previously (Choudhury et al., 2020). This paper provides a more efficient approach and comprehensive description of the algorithm, as well as a broader theoretical and experimental comparison to other cutting-edge C-DCOP algorithms.

References

Abido, M. (2002). Optimal design of power-system stabilizers using particle swarm optimization. *IEEE transactions on energy conversion*, *17*(3), 406–413.

Angeline, P. J. (1998). Using selection to improve particle swarm optimization. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pp. 84–89. IEEE.

Barabási, A.-L., & Albert, R. (1999). Emergence of scaling in random networks. *science*, *286*(5439), 509–512.

Carlisle, A., & Dozier, G. (2000). Adapting particle swarm optimization to dynamic environments. In *International conference on artificial intelligence*, Vol. 1, pp. 429–434. Citeseer.

Chen, Z., He, Z., & He, C. (2017). An improved dpop algorithm based on breadth first search pseudo-tree for distributed constraint optimization. *Applied Intelligence*, *47*(3), 607–623.

Chen, Z., Liu, L., He, J., & Yu, Z. (2020). A genetic algorithm based framework for local search algorithms for distributed constraint optimization problems.. *Auton. Agents Multi Agent Syst.*, *34*(2), 41.

Chen, Z., Wu, T., Deng, Y., & Zhang, C. (2018). An ant-based algorithm to solve distributed constraint optimization problems. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Choudhury, M., Mahmud, S., Khan, M. M., et al. (2020). A particle swarm based algorithm for functional distributed constraint optimization problems.. In *AAAI*, pp. 7111–7118.

Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, Vol. 3, pp. 1951–1957. IEEE.

Dewang, H. S., Mohanty, P. K., & Kundu, S. (2018). A robust path planning for mobile robot using smart particle swarm optimization. *Procedia Computer Science*, *133*, 290–297. International Conference on Robotics and Smart Manufacturing (RoSMa2018).

Eberhart, R. C., & Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512)*, Vol. 1, pp. 84–88. IEEE.

Eberhart, R., & Kennedy, J. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, Vol. 4, pp. 1942–1948. Citeseer.

Erdős, P., & Rényi, A. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, *5*(1), 17–60.

Ever, Y. K. (2017). Using simplified swarm optimization on path planning for intelligent mobile robot. *Procedia Computer Science*, *120*, 83–90. 9th International Conference on Theory and Application of Soft Computing, Computing with Words and Perception, ICSCCW 2017, 22-23 August 2017, Budapest, Hungary.

Farinelli, A., Rogers, A., & Jennings, N. R. (2014). Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous agents and multi-agent systems*, *28*(3), 337–380.

Farinelli, A., Rogers, A., Petcu, A., & Jennings, N. R. (2008). Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international conference on Autonomous agents and multiagent systems*, pp. 639–646. IFAAMAS.

Fitzpatrick, S., & Meetrens, L. (2003). Distributed sensor networks a multiagent perspective, chapter distributed coordination through anarchic optimization..

Ghosh, P., Karmakar, A., Sharma, J., & Phadikar, S. (2019). *CS-PSO based Intrusion Detection System in Cloud Environment: Proceedings of IEMIS 2018, Volume 1*, pp. 261–269.

Hao, Z., Wang, X., & Wang, J. (2022). A study of jamming resource allocation based on a hyperheuristic framework..

Hendrikx, H. (2021). *Accelerated methods for distributed optimization*. Theses, Université Paris sciences et lettres.

Hoang, K. D., Yeoh, W., Yokoo, M., & Rabinovich, Z. (2020). New algorithms for continuous distributed constraint optimization problems. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 502–510.

Hsin, C.-f., & Liu, M. (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*, pp. 433–442. ACM.

Jia, Y.-H., Chen, W.-n., Tianlong, G., Zhang, H., Yuan, H., Lin, Y., Yu, W.-J., & Zhang, J. (2017). A dynamic logistic dispatching system with set-based particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *PP*, 1–15.

Karaboga, D., & Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, *39*(3), 459–471.

Khan, M. M., Tran-Thanh, L., & Jennings, N. R. (2018a). 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)*, pp. 1595–1603. IFAAMAS.

Khan, M. M., Tran-Thanh, L., Yeoh, W., & Jennings, N. R. (2018b). A near-optimal node-to-agent mapping heuristic for gdl-based dcop algorithms in multi-agent systems. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1613–1621. IFAAMAS.

Khan, M. M., Tran-Thanh, L., Ramchurn, S. D., & Jennings, N. R. (2018c). Speeding up gdl-based message passing algorithms for large-scale dcops. *The Computer Journal*, *61*(11), 1639–1666.

Khireddine, A., Larbi, T., Sylia, Z., Gueguen, C., & Lamine, B. (2020). New strategy for resource allocation using pso-pfs hybrid. *International Journal of Wireless and Mobile Computing*, *18*, 175.

Kogan, P., Tassa, T., & Grinshpoun, T. (2022). Privacy preserving dcop solving by mediation. In Dolev, S., Katz, J., & Meisels, A. (Eds.), *Cyber Security, Cryptology, and Machine Learning*, pp. 487–498, Cham. Springer International Publishing.

Litov, O., & Meisels, A. (2017). Forward bounding on pseudo-trees for dcops and adcops. *Artificial Intelligence*, *252*, 83–99.

Lovbjerg, M., Rasmussen, T. K., Krink, T., et al. (2001). Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the genetic and evolutionary computation conference*, Vol. 2001, pp. 469–476. San Francisco, USA.

Maheswaran, R. T., Pearce, J. P., & Tambe, M. (2004). Distributed algorithms for dcop: A graphical-game-based approach.. In *ISCA PDCS*, pp. 432–439.

Mahmud, S., Khan, M. M., Choudhury, M., Tran-Thanh, L., & Jennings, N. R. (2020a). Learning optimal temperature region for solving mixed integer functional dcops. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2628–275.

Mahmud, S., Choudhury, M., Khan, M., Tran-Thanh, L., Jennings, N. R., et al. (2020b). Aed: An anytime evolutionary dcop algorithm. *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, 2020*.

Marinakis, Y., Marinaki, M., & Migdalas, A. (2019). A multi-adaptive particle swarm optimization for the vehicle routing problem with time windows. *Information Sciences*, *481*, 311–329.

Modi, P. J., Shen, W.-M., Tambe, M., & Yokoo, M. (2005). Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, *161*(1-2), 149–180.

Nguyen, D. T., Yeoh, W., & Lau, H. C. (2012). Stochastic dominance in stochastic dcops for risk-sensitive applications. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 257–264.

Nguyen, D. T., Yeoh, W., Lau, H. C., & Zivan, R. (2019). Distributed gibbs: A linear-space sampling-based dcop algorithm. *Journal of Artificial Intelligence Research*, *64*, 705–748.

Petcu, A., & Faltings, B. (2005). A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*.

Rashik, M., Rahman, M. M., Khan, M. M., Mamun-or Rashid, M., Tran-Thanh, L., & Jennings, N. R. (2020). Speeding up distributed pseudo-tree optimization procedures with cross edge consistency to solve dcops. *Applied Intelligence*, 1377–1746.

Sarker, A., Choudhury, M., & Khan, M. M. (2021). A local search based approach to solve continuous dcops. In *Proceedings of the 20th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pp. 1127–1135.

Shi, Y., & Eberhart, R. (1998). A modified particle swarm optimizer. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pp. 69–73. IEEE.

Shi, Y., & Eberhart, R. C. (1999). Empirical study of particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, Vol. 3, pp. 1945–1950. IEEE.

Stranders, R., Farinelli, A., Rogers, A., & Jennings, N. R. (2009). Decentralised coordination of continuously valued control parameters using the max-sum algorithm. In *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pp. 601–608. IFAAMAS.

Sultanik, E., Modi, P. J., & Regli, W. C. (2007). On modeling multiagent task scheduling as a distributed constraint optimization problem.. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 1531–1536.

Sultanik, E. A., Lass, R. N., & Regli, W. C. (2008). Dcopolis: a framework for simulating and deploying distributed constraint reasoning algorithms. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: demo papers*, pp. 1667–1668.

Tassa, T., Grinshpoun, T., & Yanai, A. (2021). Pc-syncbb: A privacy preserving collusion secure dcop algorithm. *Artificial Intelligence*, *297*, 103501.

Tassa, T., Grinshpoun, T., & Zivan, R. (2017). Privacy preserving implementation of the max-sum algorithm and its variants. *Journal of Artificial Intelligence Research*, *59*, 311–349.

van den Bergh, F., & Engelbrecht, A. P. (2002). A new locally convergent particle swarm optimiser. In *Proceedings of the IEEE International conference on systems, man and cybernetics*, Vol. 3, pp. 6–pp. IEEE.

Van den Bergh, F., & Engelbrecht, A. P. (2004). A cooperative approach to particle swarm optimization. *IEEE transactions on evolutionary computation*, *8*(3), 225–239.

Van den Bergh, F., & Engelbrecht, A. P. (2010). A convergence proof for the particle swarm optimiser. *Fundamenta Informaticae*, *105*(4), 341–374.

van Leeuwen, C. J., & Pawelczak, P. (2017). Cocoa: A non-iterative approach to a local search (a) dcop solver. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*.

Voice, T., Stranders, R., Rogers, A., & Jennings, N. R. (2010). A hybrid continuous max-sum algorithm for decentralised coordination.. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pp. 61–66.

Wei, H., Li, S., Jiang, H., Hu, J., & Hu, J. (2018). Hybrid genetic simulated annealing algorithm for improved flow shop scheduling with makespan criterion. *Applied Sciences*, *8*(12).

Yang, X.-S., & Deb, S. (2009). Cuckoo search via lévy flights. In *2009 World congress on nature & biologically inspired computing (NaBIC)*, pp. 210–214. IEEE.

Yedidsion, H., & Zivan, R. (2016). Applying dcop_mst to a team of mobile robots with directional sensing abilities: (extended abstract). In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 1357–1358. IFAAMAS.

Yeoh, W., Felner, A., & Koenig, S. (2010). Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *Journal of Artificial Intelligence Research*, *38*, 85–133.

Yu, H., Gao, Y., Wang, L., & Meng, J. (2020). A hybrid particle swarm optimization algorithm enhanced with nonlinear inertial weight and gaussian mutation for job shop scheduling problems. *Mathematics*, *8*(8).

Zhang, J.-R., Zhang, J., Lok, T.-M., & Lyu, M. R. (2007). A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. *Applied mathematics and computation*, *185*(2), 1026–1037.

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. *Artificial Intelligence*, *161*(1-2), 55–87.