A Particle Swarm Based Algorithm for Functional Distributed Constraint Optimization Problems

Moumita Choudhury,[1] Saaduddin Mahmud,[2] Md. Mosaddek Khan[3]
Department of Computer Science and Engineering, University of Dhaka, Dhaka, Bangladesh
{moumitach22,[1] saadmahmud14[2]}@gmail.com, mosaddek@du.ac.bd[3]

Abstract

Distributed Constraint Optimization Problems (DCOPs) are a widely studied constraint handling framework. The objective of a DCOP algorithm is to optimize a global objective function that can be described as the aggregation of several distributed constraint cost functions. In a DCOP, each of these functions is defined by a set of discrete variables. However, in many applications, such as target tracking or sleep scheduling in sensor networks, continuous valued variables are more suited than the discrete ones. Considering this, Functional DCOPs (F-DCOPs) have been proposed that can explicitly model a problem containing continuous variables. Nevertheless, state-of-the-art F-DCOPs approaches experience onerous memory or computation overhead. To address this issue, we propose a new F-DCOP algorithm, namely Particle Swarm based F-DCOP (PFD), which is inspired by a meta-heuristic, Particle Swarm Optimization (PSO). Although it has been successfully applied to many continuous optimization problems, the potential of PSO has not been utilized in F-DCOPs. To be exact, PFD devises a distributed method of solution construction while significantly reducing the computation and memory requirements. Moreover, we theoretically prove that PFD is an anytime algorithm. Finally, our empirical results indicate that PFD outperforms the state-of-the-art approaches in terms of solution quality and computation overhead.

Introduction

Distributed Constraint Optimization Problems (DCOPs) are an important constraint handling framework of 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. The cost functions can be defined by a set of variables controlled by the corresponding agents. DCOPs have been widely applied to solve a number of multi-agent coordination problems including, multi-agent task scheduling (Sultanik, Modi, and Regli 2007), sensor networks (Farinelli, Rogers, and Jennings 2014), multi-robot coordination (Yedidsion and Zivan 2016).

Over the years, several algorithms have been proposed to solve DCOPs, which includes both exact and non-exact algorithms. Exact algorithms, such as ADOPT (Modi et al. 2005), DPOP (Petcu and Faltings 2005) and PT-FB (Litov and Meisels 2017), are designed in such a way that provide the global optimal solution of a given DCOP. However, exact algorithms experience either or both exponential memory requirements and exponential computational costs as the system grows. On the contrary, non-exact algorithms such as DSA (Zhang et al. 2005), MGM & MGM2 (Maheswaran, Pearce, and Tambe 2004), Max-Sum (Farinelli et al. 2008; Khan, Tran-Thanh, and Jennings 2018), CoCoA (van Leeuwen and Pawelczak 2017), and ACO_DCOP (Chen et al. 2018) compromise some solution quality for scalability.

In general, DCOPs assume that participating agents' variables are discrete. Nevertheless, many real-world applications (e.g. target tracking sensor orientation (Fitzpatrick and Meetrens 2003), sleep scheduling of wireless sensors (Hsin and Liu 2004)) can be best modelled with continuous variables. Therefore, for discrete DCOPs to be able to apply 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 et al. 2009). To overcome this issue, Stranders et al. 2009 have proposed a continuous version of DCOP, which is later referred to as Functional DCOP (F-DCOP) (Hoang et al. 2019). There are two main differences between F-DCOP and DCOP. Firstly, instead of having discrete decision variables, F-DCOP has continuous variables that can take any value between a range. Secondly, the constraint functions are represented in functional forms in F-DCOP rather than in the tabular forms in DCOP.

In order to cope with the modification of the DCOP formulation, Continuous Max-Sum (CMS) has been proposed, which is an extension of the discrete Max-Sum (Stranders et al. 2009). However, this paper approximates the constraint utility functions as piece-wise linear functions which are often not applicable in practice since a handful of real-life applications deals with only piece-wise linear functions. Toward addressing this limiting assumption of CMS, Hybrid

Max-Sum (HCMS) has been proposed in which continuous non-linear optimization methods are combined with the discrete Max Sum algorithm (Voice et al. 2010). However, continuous optimization methods such as gradient-based optimization require derivative calculations, and thus they are not suitable for non-differentiable optimization problems. Hoang et al. 2019 have made the latest contribution to this field. In this paper, authors propose one exact version, Exact Functional DPOP (EF-DPOP), and two approximate versions, Approximate Functional DPOP (AF-DPOP), and Clustered AF-DPOP (CAF-DPOP) of DPOP to solve F-DCOPs. The main limitation of these algorithms is that both AF-DPOP and CAF-DPOP incur exponential memory and computation overhead even though the latter cuts the communication cost by providing a bound on message size.

Against this background, we propose a Particle Swarm Optimization based F-DCOP algorithm that we call PFD. Particle Swarm Optimization (PSO) is a stochastic optimization technique inspired by the social metaphor of bird flocking (Eberhart and Kennedy 1995). It has been successfully applied to many optimization problems such as Function Minimization (Shi and Eberhart 1999), Neural Network Training (Zhang et al. 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 DCOP or F-DCOP. In PFD, agents cooperatively keep a set of particles where each particle represents a candidate solution and iteratively improve the solutions over time. Since PFD requires only primitive mathematical operators such as addition and multiplication, it is less expensive than the gradient-based optimization methods in terms of computation cost and memory requirements. Specifically, we empirically show that PFD finds better solution quality by exploring a large search space compared to the existing F-DCOP solvers.

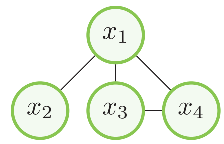
Background and Problem Formulation

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 on the F-DCOP framework, which is the main problem that we want to solve. We then discuss the centralized PSO algorithm and the challenges remain to incorporate PSO with the F-DCOP framework.

Distributed Constraint Optimization Problem

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 $\{a_1, a_2, ..., a_n\}$.
- X is a set of discrete variables $\{x_1, x_2, ..., x_m\}$, where each variable x_j is controlled by one of the agents $a_i \in A$.
- D is a set of discrete domains $\{D_1, D_2, ..., D_m\}$, where each D_i corresponds to the domain of variable x_i.
- F is a set of cost functions $\{f_1, f_2, ..., f_l\}$, where each $f_i \in F$ is defined over a subset $x^i = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}$ of variables X and the cost for the function f_i is defined for every possible value assignment of x^i, that is, f_i: $D_{i_1} \times D_{i_2} \times ... \times D_{i_k} \rightarrow R$ where the arity of the function f_i is k. In this paper, we only consider binary constraints.
- $\alpha : X \rightarrow A$ is a variable to agent mapping function which 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.



(a) Constraint Graph

$$D_i = [-10, 10]$$

$$f(x_1, x_2) = x_1^2 - x_2^2$$

$$f(x_1, x_3) = x_1^2 + 2x_1x_3$$

$$f(x_1, x_4) = 2x_1^2 - 2x_4^2$$

$$f(x_3, x_4) = x_3^2 + 3x_4^2$$

(b) Cost Functions

Figure 1: Example of an F-DCOP.

The solution of a DCOP is an assignment X^* that minimizes the sum of cost functions as shown in Equation 1.

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

Functional Distributed Constraint Optimization Problem

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

- X is the set of continuous variables $\{x_1, x_2, ..., x_m\}$ that are controlled by agents in A.
- D is a set of continuous domains $\{D_1, D_2, ..., D_m\}$, where each variable x_i can take any value between $D_i = [LB_i, UB_i]$ where LB_i and UB_i is the lower and upper bound of a range, respectively.

As discussed in the previous section, a notable difference between F-DCOP and DCOP can be found in the representation of cost function. In DCOP, the cost functions are conventionally represented in tabular form, while in F-DCOP each constraint is represented in the form a function (Hoang et al. 2019). However, the goal remains the same as depicted in Equation 1. Figure 1 presents the example of an F-DCOP where Figure 1a represents a constraint graph with four variables where each variable x_i is controlled by an agent a_i. Each edge in Figure 1a stands for a constraint function and the definition of each function is shown in Figure 1b. In this particular example, each variable x_i can take values from the range $[-10, 10]$.

Algorithm 1: Particle Swarm Optimization

```
1 Generate an n-dimensional population, P
2 Initialize positions and velocities of each particle
3 while Termination condition not met do
4     for each particle P_i ∈ P do
5         calculate current velocity and position
6         if current position < personal best then
7             update personal best
8         if current position < global best then
9             update global best
```

Particle Swarm Optimization

PSO is a population-based optimization[1] technique inspired by the movement of a bird flock or a fish school. 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, initially, each particle is assigned a random position and velocity. 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 personal best position found so far in the search space and the global best position found by the entire swarm (Algorithm 1: Lines 4-5). The combination of the personal best and the global best position ensures that when a better position is found through the search process, the particles will move closer to that position and explore the surrounding search space more thoroughly considering it as a potential solution. The personal best position of each particle and the global best position of the entire population is updated if necessary (Algorithm 1: Lines 6-9). 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 behaviour 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 local optima (van den Bergh and Engelbrecht 2002).

Challenges

The following challenges must be addressed to develop an anytime F-DCOP algorithm that adapts the guaranteed convergence PSO:

- **Particles and Fitness Representation:** We need to define a representation for the particles where each particle represents a solution of the F-DCOP. Moreover, a distributed method for calculating the fitness for each of the particles needs to be devised.

[1]For simplicity, we are going to consider the optimization and minimization interchangeably throughout the paper.

(a) BFS pseudo-tree (b) Ordered arrangement

Figure 2: Pseudo-tree construction and ordered arrangement.

- **Creating the Population:** In centralized optimization problems, creating the initial population is a trivial task. However, in case of F-DCOP, 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 F-DCOP, 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.
- **Maintaining the Anytime Property:** To maintain the anytime property in an F-DCOP model, we need to identify the global best particle and the personal best position for each particle. A distribution method needs to be devised to notify all the agents when a new global best particle or personal best position is found. Finally, a 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 to apply PSO in F-DCOP that addresses the above challenges.

The PFD Algorithm

PFD is a PSO based iterative algorithm consisting of three phases: Initialization, Evaluation, and Update. In the initialization phase, a pseudo-tree is constructed, an initial population is created and parameters are initialized. In the evaluation phase, agents calculate the fitness function for each particle in a distributed way. The update phase keeps track of the best solution found so far and propagates this information to the agents and updates the assignments according to that information. The detailed algorithm can be found in Algorithm 2.

Initialization starts with ordering the agents in a Breadth First Search (BFS) pseudo-tree (Chen, He, and He 2017). The pseudo-tree serves the purpose of defining a message passing order which is used in the Evaluation and Update phase. In the ordered arrangement, an agent with lower depth has higher priority over an agent with higher depth and ties are broken randomly. From the pseudo-tree construction algorithm, each agent knows the lists of its higher and lower priority neighbors. Each agent needs this information to be able to send and receive messages in the later phase of the algorithm. Figure 2(a) and 2(b) illustrates the BFS pseudo-tree and its ordered arrangement of the constraint graph shown in

Algorithm 2: The PFD Algorithm

1 Construct BFS pseudo-tree
2 Initialize parameters: $K, w, c_1, c_2, max_{s_c}, max_{f_c}$
3 $P \leftarrow$ set of K particles
4 **Function** `Init()`:
5   **for** *each particle* $P_k \in P$ **do**
6         $P_k.v_i \leftarrow 0$
7         $P_k.x_i \leftarrow$ a random value from D_i
8   Send $P.x_i$ to agents in L_i
9
10 **for** *each agent* a_i **do**
11   `Init()`
12 **while** *Termination condition not met each agent* a_i **do**
13   **for** $P.x_i$ *received from* $H_{i_j} \in H_i$ **do**
14   **for** *each particle* $P_k \in P$ **do**
15             $P_k.fitness \leftarrow Cost_{i,j}(P_k.x_i, P_k.x_j)$
16   Send $P.fitness$ to agents in H_{i_j}
17   Wait until $P.fitness$ received from all agent in L_i
18   **if** $|L_i| \neq 0$ *and* $P.fitness$ *received from all agent in* L_i **then**
19   **for** *each particle* $P_k \in P$ **do**
20             $P_k.fitness \leftarrow \sum_{j \in L_i} P.fitness$
21   **if** $a_i \neq$ *root* **then**
22   Send $P.fitness$ to an $H_{i_j} \in H_i$
23   **if** $a_i =$ *root* **then**
24   `Update(`$P.fitness$`)`
25   Wait until $P.p_{best}$ and $P.g_{best}$ are received from H_i
26   **if** $P.p_{best}$ *and* $P.g_{best}$ *are received from* H_i **then**
27   Calculate s_c and f_c according to Equations 7, 8
28   **for** *each particle* $P_k \in P$ **do**
29   **if** $P.g_{best} = P_k$ **then**
30   Calculate $P_k.v_i$ and $P_k.x_i$ according to Equations 4, 5
31   **else**
32   Calculate $P_k.v_i$ and $P_k.x_i$ according to Equations 3, 5
33   **if** $|L_i| \neq 0$ **then**
34   Send $P.x_i$ to agents in L_i
35   Send $P.p_{best}$ and $P.g_{best}$ to agents in L_i
36 **Function** `Update(`$P.fitness$`)`:
37     $P.p_{best} \leftarrow \{\}$
38   **for** *each particle* $P_k \in P$ **do**
39   **if** $P_k.fitness < P_k.p_{best}.fitness$ **then**
40             $P_k.p_{best} \leftarrow P_k$
41             $P.p_{best} \leftarrow \{P_k.p_{best}\} \cup P.p_{best}$
42   **if** $P_k.fitness < P.g_{best}.fitness$ **then**
43             $P.g_{best} \leftarrow P_k$
44   Send $P.p_{best}$ and $P.g_{best}$ to agents in L_i
45

Figure 1, respectively. In Figure 2(b), x_1 is the root and the arrows represent the message passing direction of the Initialization and the Update phase. The reverse direction is used for the Evaluation phase. From this point, for an agent a_i, we refer N_i as the set of neighbors, $H_i \subseteq N_i$ and $L_i \subseteq N_i$ as the sets of higher priority and lower priority neighbors of a_i, respectively. For agent x_3[2] of Figure 2(b), $N_3 = \{x_1, x_4\}$, $H_3 = \{x_1\}$ and $L_3 = \{x_4\}$.

PFD requires some parameters as input; one of them is the number of particles, K whose value depends on the specific problem[3]. Each agent then initializes the set of K particles, P. Each particle $P_k \in P$ has a velocity and a position attribute; and each agent only controls the component of the attributes relevant to the variable it holds. The velocity attribute defines the movement directions and the position attribute defines the value of the variable that the agent controls. Then each agent a_i executes **Init** (Algorithm 2: Lines 4-8) and initializes the velocity component, v_i to 0 and position component, x_i to a random value from its domain D_i for each particle P_k. For the example of Figure 2(b), let us assume the number of particles, $K = 2$, and the set of particles, $P = \{P_1, P_2\}$. Here, $P_1.V = P_2.V = \{0, 0, 0, 0\}$ shows the complete assignment for the velocity attribute of two particles and the complete assignment for the position attribute can be shown as, $P_1.X = \{x_1 = -1, x_2 = 0, x_3 = 2, x_4 = 9.5\}, P_2.X = \{x_1 = 3.5, x_2 = 4.9, x_3 = 1, x_4 = 0\}$. We define $P_k.x_i$ and $P_k.v_i$ as the position and velocity component of particle P_k set by the agent a_i. In this example, $P_1.x_3 = 2$ which is the value of variable x_3 of particle P_1 set by the agent a_3. After selecting the value of its variable, each agent shares the particle set $P.x_i$ with its lower priority neighbors L_i. In our example, agent a_3 sends $P.x_3 = \{2, 1\}$ to its lower priority neighbor a_4.

The **Evaluation** phase of PFD starts after the agents receive value assignments from all the higher priority neighbors. Each agent a_i is responsible for calculating the constraint cost associated with each of its higher priority neighbors from H_i. When an agent a_i receives value assignments $P.x_i$ from a higher priority neighbor $H_{i_j} \in H_i$, it calculates the constraint costs between them and sends it to H_{i_j} (Algorithm 2: Lines 13-16). Additionally, each agent except the leaf agents needs to pass the constraint costs upward the pseudo-tree calculated by its corresponding lower priority neighbors, L_i (Algorithm 2: Lines 18-20). The fitness of each particle $P_k.fitness$ is calculated using a fitness function shown in Equation 2, where $P_k.x^i$ represents the assignments of the set of variables x^i. This function calculates the aggregated cost of constraints yielded by the assignment. Note that a single agent can not calculate the complete fitness. Instead, it is calculated in parts with the cooperation of all the agents during the construction process. For the example shown in Figure 1, agent a_4 sends the set of fitness of the particles $\{P_1 = 274.75, P_2 = 1\}$ to a_3 and $\{P_1 = -178.5, P_2 = 24.5\}$ to a_1. Agent a_3 calculates the set of fitness $\{P_1 = -3, P_2 = 19.25\}$ and

[2] a_i and x_i will be used interchangeably throughout the paper.

[3] The rest of parameters and their recommended values for our experiments are discussed later in the text.

sends it to a_1. Furthermore, a_3 receives the fitness from a_4 and passes it to a_1. Similarly, a_2 sends the set of fitness $\{P_1 = 1, P_2 = -11.76\}$ to a_1.

$$P_k.fitness = \sum_{f_i \in F} f_i(P_k.x^i) \quad (2)$$

The **Update** phase consists of two parts: p_{best}, g_{best} update and variable update. We define $P_k.p_{best}$ to be the personal best position achieved so far by each particle and $P.g_{best}$ to be the global best position among all the particles. Since each agent a_i calculates and passes the cost of the constraints to the agents in H_i, the fitness of all the particles propagates to the root. The root agent then sums the fitness values received from the agents in L_{root} for each of the particles, P_k. Then, the root agent checks and updates the $P_k.p_{best}$ for $P_k \in P$ and $P.g_{best}$ for P, and sends the new values to the agents in L_{root} (Algorithm 2: Lines 38-44). When an agent a_i receives $P_k.p_{best}$ and $P.g_{best}$ of the previous iteration, it updates the velocity component $P_k.v_i$ and position component $P_k.x_i$ for $P_k \in P$. To adapt the guaranteed convergence method to PFD, we define two types of update equations for the velocity component. For all the particles except the global best particle, the update equation is shown in Equation 3.

$$\begin{aligned} P_k.v_i(t) = w * P_k.v_i(t-1) + \\ r_1 * c_1 * (P_k.p_{best}(t-1) - P_k.x_i(t-1)) + \\ r_2 * c_2 * (P.g_{best}(t-1) - P_k.x_i(t-1)) \end{aligned} \quad (3)$$

However, when the particle is the global best particle, then from Equation 3, we can see the velocity update will only depend on the term $w * P_k.v_i(t-1)$. Thus, the global best particle will only move away if its inertia weight w and the velocity in the previous iteration $P_k.v_i(t-1)$ are not equal to zero. Otherwise, the global best particle will stop moving and all the particles will eventually follow the global best particle. This phenomenon will lead to the premature convergence of the algorithm. To cope with this issue, we adapt the GCPSO approach (van den Bergh and Engelbrecht 2002) and the velocity update equation for the global best particle is shown in Equation 4.

$$\begin{aligned} P_k.v_i(t) = -P_k.x_i(t-1) + P.g_{best}(t-1) + \\ w * P_k.v_i(t-1) + \rho * (1 - 2r_2) \end{aligned} \quad (4)$$

The position component update equation is the same for all the particles which is defined in Equation 5.

$$P_k.x_i(t) = P_k.x_i(t-1) + P_k.v_i(t) \quad (5)$$

In Equations 3, 4 and 5, $P_k.v_i(t)$ and $P_k.x_i(t)$ refer to the velocity and position components controlled by agent a_i for particle P_k in t^{th} iteration. Here, an iteration refers to a complete round of the Evaluation and Update phase (Algorithm 2: Line 12). Here, w is the inertia weight that defines the influence of current velocity on the updated velocity, r_1 and r_2 are two random values between [0, 1] and c_1, c_2 are two constants. Combinations of c_1 and c_2 define the magnitude of influence that the personal best and the global best have on the updated particle position. In Equation 4, ρ is used to explore a random area near the position of the global best particle. To be precise, ρ defines the diameter of this area that the particles can explore. The value of ρ is adjusted according to Equation 6.

$$\rho(t) = \begin{cases} 1 & t = 0 \\ 2 * \rho(t-1) & s_c(t-1) > max_{s_c} \\ 0.5 * \rho(t-1) & f_c(t-1) > max_{f_c} \\ \rho(t-1) & otherwise \end{cases} \quad (6)$$

In Equation 6, s_c and f_c are the count of consecutive successes and failures, respectively. A success is defined when the global best particle updates its personal best position. Similarly, a failure is defined when the position of the global best particle remains unchanged. The intuition behind changing ρ with each iteration is to reward the random exploration when consecutive successes occur and to penalize when consecutive failures occur. The parameters max_{s_c} and max_{f_c} are the upper bound of s_c and f_c. The following equations define s_c and f_c, respectively.

$$s_c(t) = \begin{cases} s_c(t-1) + 1 & if\ P_G.p_{best}(t) < P.g_{best}(t-1) \\ 0 & otherwise \end{cases} \quad (7)$$

$$f_c(t) = \begin{cases} f_c(t-1) + 1 & if\ P.g_{best}(t) = P.g_{best}(t-1) \\ 0 & otherwise \end{cases} \quad (8)$$

In Equation 7, P_G defines the global best particle of iteration $t-1$. Each agent a_i calculates s_c and f_c according to Equations 7 and 8 after receiving $P_G.p_{best}$ and $P.g_{best}$ from their higher priority neighbors H_i (Algorithm 2: Line 27).

Consider the root agent a_1 in Figure 2. When a_1 receives fitness values from all of its lower priority neighbors, it is ready to calculate the $P.p_{best}$ and $P.g_{best}$. The final updated fitness value, $P.fitness = \{94.25, 33\}$. Since this is the first iteration, $P.p_{best}$ will be current positions of the particles and $P.g_{best}$ will be the position of particle P_2. agent a_1 then propagates this information to the agents in L_1. Then each agent calculates s_c and f_c and updates the values based on Equations 3, 4, and 5.

Theoretical Analysis

In this section, we first prove PFD is an anytime algorithm; that is, the quality of solution improves and never degrades over time. We then discuss AED's complexity in terms of communication, computation, and memory requirements

Lemma 1: At *iteration*[4] $t + d$*, the root is aware of the* $P.p_{best}$ *and* $P.g_{best}$ *up to iteration* t*, where* d *is the longest path in the pseudo-tree starting from the root.*

Proof: In order to prove this lemma, it is sufficient to show that, at iteration $t + d$, the root agent has enough information to calculate $P.p_{best}$ and $P.g_{best}$ 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

[4]For the theoretical analysis section, iteration refers to the required number of communication steps. In one communication step, agents only directly communicate with the neighbors.

all the agents in L_{root} in order to calculate the fitness of each particle using Equation 2. Now, the root agent has to wait for at most d iterations for the cost messages since the length of the longest path in the pseudo-tree is d. In the wake of that, we can infer that at iteration $t+d$, the root agent is capable of calculating the fitness of each particle up to iteration t.

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

Proof: In PFD, for any agent a_i, it will take at most h iterations for the $P.p_{best}$ and $P.g_{best}$ to reach that agent from the root since it is enough to get this message from one of the agents in H_i. Based on the above claim and Lemma 1, it takes at most $t+d+h$ iterations for the $P.p_{best}$ and $P.g_{best}$ up to iteration t to reach all the agents.

Proposition 1: *PFD is an anytime algorithm.*

Proof: From Lemma 2, at iteration $t+d+h$ and $t+d+h+\delta$ ($\delta \geq 0$), each agent is aware of $P.p_{best}$ and $P.g_{best}$ up to iterations t and $t+\delta$, respectively. Since $P.p_{best}$ and $P.g_{best}$ only get updated if a better solution is found, $P.p_{best}$ and $P.g_{best}$ at iteration $t+d+h+\delta \leq$ at iteration $t+d+h$. That is, the solution quality improves monotonically as the number of iteration increases. Hence, PFD is an anytime algorithm.

Complexity Analysis

We define, the total number of agents $|A| = n$ and the total number of neighbors of an agent $a_i \in A$, $|N_i| = |L_i| + |H_i|$. In PFD, an agent sends $|L_i|$ messages during the Initialization and Update phases. Additionally, during the Evaluation phase, an agent sends $|H_i| + 1$ messages. After one round of the Initialization, Evaluation and Update phases, an agent a_i sends $2 * |L_i| + |H_i| = |L_i| + |N_i|$ messages. In the worst case, the graph is complete where $|N_i| = n$. Therefore, the total number of messages sent by an agent a_i is $O(2 * |L_i| + |H_i|) = O(2n)$ in the worst case.

In PFD, each agent sends 3 types of messages; they are $P.x_i$, $P.fitness$ and $P.p_{best}$, $P.g_{best}$ messages. Each of these messages contains the information of K particles, where K is the total number of particles. Hence, the size of each message is $O(K)$. This means, at each iteration the total message size per agent is $O(3 * K * n) = O(K * n)$.

During an iteration, an agent either calculates $P_k.v_i$ and $P_k.x_i$ or $P_k.fitness$ for each of the particle P_k. Hence, the total computation complexity per agent during an iteration is $O(K + K * n) = O(K * n)$.

Experimental Results

In this section, we empirically evaluate the quality of solutions produced by PFD with HCMS and AF-DPOP on two types of graphs: *Random Graphs* and *Random Trees*. However, CMS is not used in comparison because it only works with piece-wise linear functions which is not applicable in most of the real-world applications. Although Hoang et al. 2019 propose three versions of Functional DPOP, we only compare with AF-DPOP here. This is because AF-DPOP is reported to provide the best solution among the approximate algorithms proposed in their paper. For the experimental performance evaluation, binary quadratic functions are used which are of the form $ax^2 + bxy + cy^2$. Note that, although we choose binary quadratic functions for evaluation, PFD is broadly applicable to other classes of problems. The experiments are carried out on a computer with an Intel Core i5-6200U CPU, 2.3 GHz processor and 8 GB RAM. The detailed experimental settings are described below.

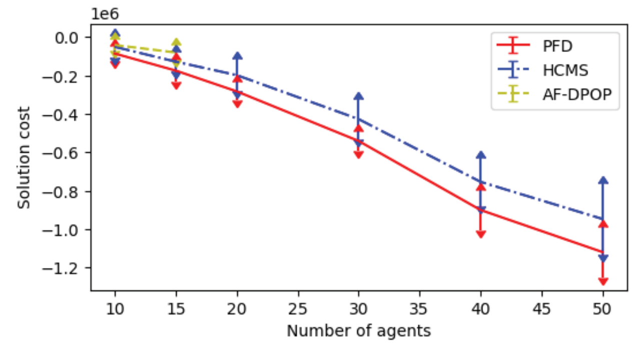


Figure 3: Solution Cost Comparison of PFD and the competing algorithms varying the number of agents (sparse graphs).

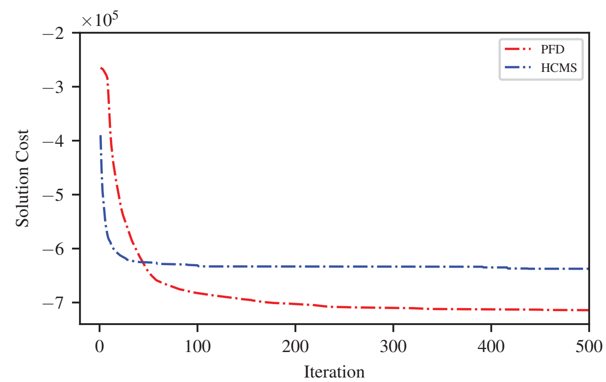


Figure 4: Solution Cost Comparison of PFD and the competing algorithms with iterations (sparse graphs).

Random Graphs: For random graphs, we use three settings− sparse, dense and scale-free. Figure 3 shows the comparison of average costs on Erdős-Rényi topology (Erdős and Rényi 1960) with sparse settings (edge probability 0.2) varying the number of agents. We choose coefficients of the cost functions (a, b, c) randomly between $[-5, 5]$ and set the domains of each agent to $[-50, 50]$. For all the experiments, we set the parameters of PFD, $K = 2000$, $w = 0.9$, $c_1 = 0.9$, $c_2 = 0.1, max_{f_c} = 5$, and $max_{s_c} = 15$. Moreover, we stop both HCMS and PFD after 500 iterations for Figures 3, 5, 6 and 7. Specifically, we choose 3 as the number of discrete points for HCMS and AF-DPOP. The discrete points are chosen randomly between the domain range. The averages are taken over 50 randomly generated problems. Figure 3 shows that PFD performs better than both HCMS and AF-DPOP on average. For $no.\ of\ agents \geq 20$, AF-DPOP run out of memory. Thus, we omit the result of AF-DPOP for

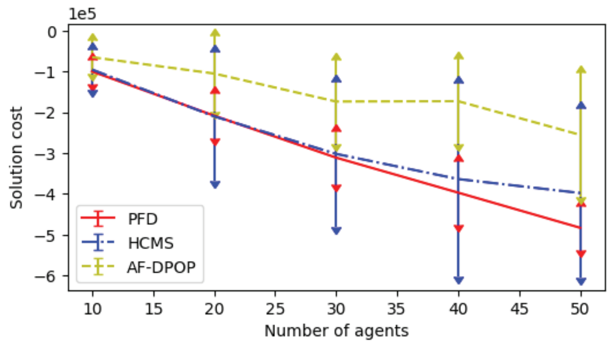


Figure 5: Solution Cost Comparison of PFD and the competing algorithms varying number of agents (scale-free graphs).

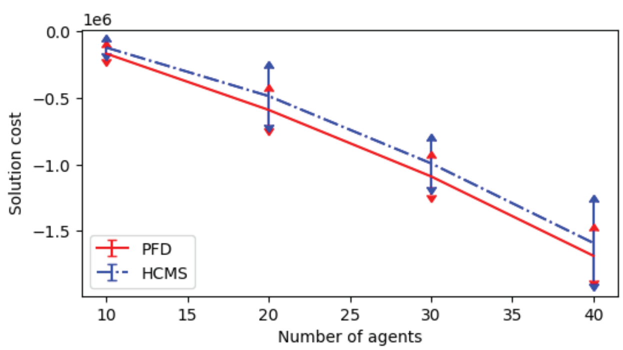


Figure 6: Solution Cost Comparison of PFD and the competing algorithms varying number of agents (dense graphs).

$no.of agents \geq 20$.

Figure 4 shows the comparison between PFD and HCMS on sparse graph settings with increasing number of iterations. We set the number of agents to 50 and the other settings are the same as the above experiment. Furthermore, we stop both algorithms after 500 iterations. Here, HCMS initially performs slightly better than PFD till 50 iterations since the particles of PFD initially start from random positions and require few iterations to move the particles towards the best position. However, PFD outperforms HCMS later and the improvement rate of PFD is steadier than HCMS. Note that, for 50 agents, AF-DPOP runs out of memory in our settings. Hence, we omit the result of AF-DPOP.

To compare with the performance of AF-DPOP on larger graphs, we use scale-free graphs. Figure 5 shows the average cost comparison between the three algorithms with increasing number of agents. PFD shows comparable performance with HCMS up to 30 agents and outperforms HCMS afterwards. Both PFD and HCMS outperforms AF-DPOP. The huge standard deviation of HCMS results into the comparable performance with PFD for smaller agents.

We choose dense graphs as our final random graph settings. Figure 6 shows a comparison between PFD and HCMS on Erdős-Rényi topology with dense settings (edge probability 0.6). PFD shows comparatively better performance than HCMS. Note that AF-DPOP is not used in the dense graph setting due to the huge computation overhead.

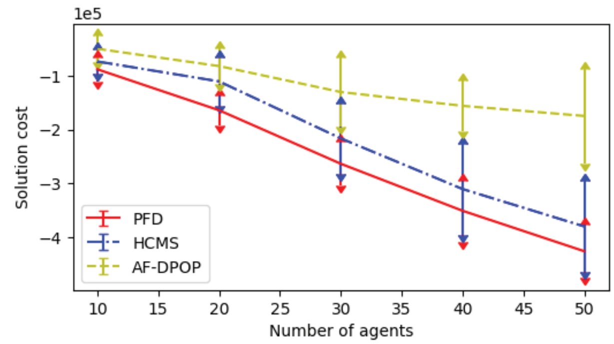


Figure 7: Solution Cost Comparison of PFD and the competing algorithms varying number of agents (random trees).

Random Trees: We use the random tree configuration in our last experimental settings since the memory requirement of AF-DPOP is less on trees. The experimental configurations are similar to the random graph settings. Figure 7 shows comparative results between PFD and the competing algorithms on random trees. The closest competitor of PFD in this setting is HCMS. On an average, PFD outperforms HCMS which in turn outperforms AF-DPOP. When the number of agent is 50, PFD shows better performance than AF-DPOP at a significant level.

Conclusions

In order to model many real-world problems, continuous valued variables are more suitable than discrete valued variables. The F-DCOP framework is a variant of the DCOP framework that can model such problems effectively. To solve F-DCOPs, we propose an anytime algorithm called PFD that is inspired by the Particle Swarm Optimization (PSO) technique. To be precise, PFD devises a new method to calculate and propagate the best particle information across all the agents which influence the swarm to move towards a better solution. We also theoretically prove that our proposed algorithm PFD is anytime. Moreover, the guaranteed convergence version of PSO is tailored in PFD which ensures its convergence to a local optimum. We empirically evaluate our algorithm in a number of settings and compare the results with the state-of-the-art algorithms, HCMS and AF-DPOP. In all of the settings, PFD markedly outperforms its counterparts in terms of solution quality. In the future, we would like to further investigate the potential of PFD on various F-DCOP applications. We also want to explore whether PFD can be extended for multi-objective F-DCOP settings.

Acknowledgments

This research is partially supported by the Information and Communication Technology (ICT) Division, and University Grants Commission (UGC).

References

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

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

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

Eberhart, R., and Kennedy, J. 1995. Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks*, volume 4, 1942–1948. Citeseer.

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

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, 639–646. IFAAMAS.

Farinelli, A.; Rogers, A.; and 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.

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

Hoang, K. D.; Yeoh, W.; Yokoo, M.; and Rabinovich, Z. 2019. New algorithms for functional distributed constraint optimization problems. *arXiv preprint arXiv:1905.13275*.

Hsin, C.-f., and 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*, 433–442. ACM.

Khan, M. M.; Tran-Thanh, L.; and Jennings, N. R. 2018. A generic domain pruning technique for gdl-based dcop algorithms in cooperative multi-agent systems. In *Proceedings of the 17th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 1595–1603. IFAAMAS.

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

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

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

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization.

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

Stranders, R.; Farinelli, A.; Rogers, A.; and 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*, 601–608. IFAAMAS.

Sultanik, E.; Modi, P. J.; and 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*, 1531–1536.

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

van Leeuwen, C. J., and 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.; and Jennings, N. R. 2010. A hybrid continuous max-sum algorithm for decentralised coordination. In *Proceedings of the 19th European Conference on Artificial Intelligence*, 61–66.

Yedidsion, H., and 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*, 1357–1358. IFAAMAS.

Zhang, W.; Wang, G.; Xing, Z.; and 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.

Zhang, J.-R.; Zhang, J.; Lok, T.-M.; and 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.