AED: An Anytime Evolutionary DCOP Algorithm

Saaduddin Mahmud
Department of Computer Science and Engineering, University of Dhaka
saadmahmud14@gmail.com

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

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

Long Tran-Thanh
School of Electronics and Computer Science, University of Southampton
ltt08r@ecs.soton.ac.uk

Nicholas R. Jennings
Departments of Computing and Electrical and Electronic Engineering, Imperial College London
n.jennings@imperial.ac.uk

ABSTRACT

Evolutionary optimization is a generic population-based metaheuristic that can be adapted to solve a wide variety of optimization problems and has proven very effective for combinatorial optimization problems. However, the potential of this metaheuristic has not been utilized in Distributed Constraint Optimization Problems (DCOPs), a well-known class of combinatorial optimization problems prevalent in Multi-Agent Systems. In this paper, we present a novel population-based algorithm, Anytime Evolutionary DCOP (AED), that uses evolutionary optimization to solve DCOPs. In AED, the agents cooperatively construct an initial set of random solutions and gradually improve them through a new mechanism that considers an optimistic approximation of local benefits. Moreover, we present a new anytime update mechanism for AED that identifies the best among a distributed set of candidate solutions and notifies all the agents when a new best is found. In our theoretical analysis, we prove that AED is anytime. Finally, we present empirical results indicating AED outperforms the state-of-the-art DCOP algorithms in terms of solution quality.

KEYWORDS

Distributed Problem Solving, DCOPs

ACM Reference Format:
Saaduddin Mahmud, Moumita Choudhury, Md. Mosaddek Khan, Long Tran-Thanh, and Nicholas R. Jennings. 2020. AED: An Anytime Evolutionary DCOP Algorithm. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

1 INTRODUCTION

Distributed Constraint Optimization Problems (DCOPs) are a widely used framework to model constraint handling problems in cooperative Multi-Agent Systems (MAS). In particular, agents in this framework need to coordinate value assignments to their variables in such a way that minimizes constraint violations by optimizing their aggregated costs [28]. This framework has been applied to various areas of multi-agent coordination, including distributed meeting scheduling [20], sensor networks [6][3] and smart grids [8].

Over the last two decades, several algorithms have been proposed to solve DCOPs, and they can be broadly classified into two classes: exact and non-exact. The former always provide an optimal solution of a given DCOP. Among the exact algorithms, SyncBB [12], ADOPT [21], DPOP [24], AFB [11], BnB-ADOPT [27], and PT-FB [18] are widely used. Since solving DCOPs optimally is NP-hard, scalability becomes an issue as the system grows. In contrast, non-exact algorithms compromise some solution quality for scalability. As a consequence, diverse classes of non-exact algorithms have been developed to deal with large-scale DCOPs. Among them, local search based algorithms are generally most inexpensive in terms of computational and communication cost. Some well-known algorithms of this class are DSA [29], MGM & MGM2 [19], and GDBA [22]. Also, in order to further enhance solution quality and incorporate an anytime property in local search based algorithms, the Anytime Local Search (ALS) framework [30] was introduced. While inference based non-exact approaches such as Max-Sum [7][15][14] and Max-Sum_ADVP [31] have also gained attention due to their ability to handle n-ary constraints explicitly and guarantee optimality on acyclic constraint graphical representations of DCOPs. The third class of non-exact approaches that have been developed are sample-based algorithms (e.g. DUCT [23] and PD-Gibbs [25]) in which the cooperative agents sample the search space in a decentralized manner to solve DCOPs.

More recently, a new class of non-exact DCOP algorithms have emerged in the literature through the introduction of a population-based algorithm ACO_DCOP [2]. ACO_DCOP is derived from a centralized population-based approach called Ant Colony Optimization (ACO) [4]. It has been empirically shown that ACO_DCOP produces solutions with better quality than the aforementioned classes of non-exact DCOP algorithms [2]. It is worth noting that although a wide variety of centralized population-based algorithms exist, ACO is the only such method that has been adapted to solve DCOPs. Among the remaining centralized population-based algorithms, a large portion is considered as evolutionary optimization techniques (e.g. Genetic Algorithm [13], Evolutionary Programming [9]). Evolutionary optimization, as a population-based metaheuristic, has proven very effective in solving combinatorial optimization problems such as Traveling Salesman Problem [10], Constraint Satisfaction Problem [26], and many others besides. However, no prior work

Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.

exists that adapts evolutionary optimization techniques to solve DCOPs. Considering the effectiveness of evolutionary optimization techniques in solving combinatorial optimization problems along with the potential of population-based DCOP solver demonstrated by ACO_DCOP motivates us to explore this nascent area.

Against this background, this paper proposes a novel population-based algorithm that uses evolutionary optimization to solve DCOPs. We call this Anytime Evolutionary DCOP (AED). In more detail, AED maintains a set of candidate solutions that are distributed among the agents, and they search for new improved solutions by modifying the candidate solutions. This modification is done through a new mechanism that considers an optimistic approximation of local benefits and utilizes the cooperative nature of the agents. Moreover, we introduce a new anytime update mechanism in order to identify the best among this distributed set of candidate solutions and help the agents to coordinate value assignments to their variables based on the best candidate solution. Our theoretical analysis proves that AED is anytime and empirical evaluation shows its superior solution quality compared to the state-of-the-art non-exact DCOP algorithms.

2 BACKGROUND

In this section, we first describe DCOPs and Evolutionary Optimization in more detail. Then, we discuss challenges that need to be addressed in order to effectively extend evolutionary optimization in the context of DCOPs.

2.1 Distributed Constraint Optimization Problems

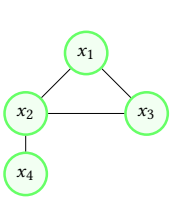
Formally, a DCOP is defined by a tuple $\langle X, D, F, A, \delta \rangle$ [21] where,

- A is a set of agents $\{a_1, a_2, ..., a_n\}$.
- X is a set of discrete variables $\{x_1, x_2, ..., x_m\}$, which are being controlled by the set of agents A.
- D is a set of discrete and finite variable domains $\{D_1, D_2, ..., D_m\}$, where each D_i is a set containing values which may be assigned to its associated variable x_i.
- F is a set of constraints $\{f_1, f_2, ..., f_l\}$, where $f_i \in F$ is a function of a subset of variables $x^i \subseteq X$ defining the relationship among the variables in x^i. Thus, the function $f_i : \times_{x_j \in x^i} D_j \rightarrow R$ denotes the cost for each possible assignment of the variables in x^i.
- $\delta : X \rightarrow A$ is a variable-to-agent mapping function [16] which assigns the control of each variable $x_i \in X$ to an agent of A. Each variable is controlled by a single agent. However, each agent can hold several variables.

Within the framework, the objective of a DCOP algorithm is to produce X^*; a complete assignment that minimizes[1] the aggregated cost of the constraints as shown in Equation 1.

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

For ease of understanding, we assume that each agent controls one variable. Thus, the terms 'variable' and 'agent' are used interchangeably throughout this paper. Figure 1a illustrates a sample DCOP using a constraint graph where each node represents an agent $a_i \in A$ labelled by a variable $x_i \in X$ that it controls and each edge represents a function $f_i \in F$ connecting all $x_j \in x^i$. Figure 1b shows the corresponding cost tables.



(a) A constraint graph

x_2 \ x_1	1	2
1	7	12
2	3	15

x_4 \ x_2	1	2
1	8	4
2	15	6

x_3 \ x_2	1	2
1	2	7
2	11	18

x_3 \ x_1	1	2
1	9	13
2	12	5

(b) Cost tables

Figure 1: Example DCOP

2.2 Evolutionary Optimization

Evolutionary optimization is a generic population-based metaheuristic inspired by biological evolutionary mechanisms such as Selection, Reproduction and Migration. The core mechanism of evolutionary optimization techniques can be summarized in three steps. In the first step, an initial population is generated randomly. A population is a set of 'individuals', each of which is a candidate solution of the corresponding optimization problem. Besides, a fitness function is defined to evaluate the quality of an individual concerning a global objective. The fitness of all the individuals in the initial population is also calculated. In the second step, a subset of the population is selected based on their fitness to reproduce new individuals. This process is known as Selection. In the final step, new individuals are created using the selected subset of the population and their fitness is evaluated. New individuals then replace a subset of old individuals. Evolutionary optimization performs both the second and the third steps iteratively, which results in a gradual improvement in the quality of individuals. An additional step is performed at regular intervals by some parallel/distributed evolutionary optimization models that concurrently maintain multiple sub-populations instead of a single population. In this step, individuals are exchanged between sub-populations. This process is known as Migration, and this interval is known as the Migration Interval.

2.3 Challenges

We need to address the following challenges in order to develop an effective anytime algorithm that uses evolutionary optimization to solve DCOPs:

- **Individual and fitness:** We need to define an individual that represents a solution of a DCOP along with a fitness function to evaluate its quality concerning Equation 1. We also need to provide a method for calculating this fitness function in a distributed manner.
- **Population:** We need to provide a strategy to maintain the population collectively among the agents. Although creating an initial random population is a trivial task for centralized problems, we need to find a distributed method to construct an initial random population for a DCOP.

[1]For a maximization problem argmin is replaced with argmax in Equation 1.

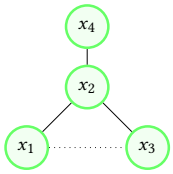


Figure 2: BFS Pseudo-Tree

- **Reproduction mechanism:** In the DCOP framework, information related to the entire problem is not available to any single agent. So it is necessary to design a Reproduction method that can utilize information available to a single agent along with the cooperative nature of the agents.
- **Anytime update mechanism:** We need to design an anytime update mechanism that can successfully perform the following tasks – (i) Identify the best individual in a population that is distributed among the agents. (ii) Notify all the agents when a new best individual is found. (iii) Help coordinate the variable assignment decision based on the best individual in a population.

In the following section, we describe our method that addresses the above challenges.

3 THE AED ALGORITHM

AED is a synchronous iterative algorithm that consists of two phases: Initialization and Optimization. During the former, agents initially order themselves into a pseudo-tree, then initialize the necessary variables and parameters. Finally, they make a random assignment to the variables they control and cooperatively construct the initial population. During the latter phase, agents iteratively improve this initial set of solutions using the cooperation of their neighbours. When an agent detects a better solution, it notifies other agents. Moreover, all the agents synchronously update their assignments based on the best of the individuals reported to them so far. This results in a monotonic improvement of the global objective. Algorithm 1 shows the pseudo-code for AED. For ease of understanding, we show the process of initialization and anytime update separately in Procedure 1 and Procedure 2, respectively. Note that the initialization phase addresses the first two of our challenges, while the optimization phase addresses the rest.

The Initialization Phase of AED consists of two parts; pseudo-tree construction and running INIT (Procedure 1) that initializes the population, parameters and variables (Algorithm 1: Line 1-2). This phase starts by ordering the agents into a Pseudo-Tree. This ordering serves two purposes. It helps in the construction of the initial population and facilitates ANYTIME-UPDATE (Procedure 2) during the optimization phase. Even though either of the BFS or DFS pseudo-tree can be used, AED uses BFS Pseudo-tree[2]. This is because it generally produces a pseudo-tree with smaller height [1], which improves the performance of ANYTIME-UPDATE (see Theoretical Analysis for details). Figure 2 shows an example of a BFS pseudo-tree constructed from the constraint graph shown in Figure 1a having x_4 as the root. Here, the height[3] (i.e. H = 2) of this pseudo-tree is calculated during the time of construction and is maintained by all agents. From this point, N_i refers to the set of neighbours; $C_i \subseteq N_i$ refers to the set of child nodes and PR_i refers to the parent of an agent a_i in the pseudo-tree. For instance, we can see in Figure 2 that $N_2 = \{a_1, a_3, a_4\}$, $C_2 = \{a_1, a_3\}$ and $PR_2 = a_4$ for agent a_2. After the pseudo-tree construction, all the agents synchronously call the procedure INIT (Algorithm 1: Line 2).

Algorithm 1: Anytime Evolutionary DCOP

```
1  Construct pseudo-tree
2  Every agent a_i calls INIT( )
3  while Stop condition not met each agent a_i do
4      P_selected ← Select_rp(|N_i| * ER)
5      P_new ← Partition P_selected into equal size subsets
         {P_new^{n_1}, ..., P_new^{n_|N_i|}}
6      for n_j ∈ N_i do
7          Modify individuals in P_new^{n_j} by Equations 5, 6, 7, 9
8          Send message P_new^{n_j} to n_j
9      for P_received^{n_j} received from n_j ∈ N_i do
10         Modify individuals in P_received^{n_j} by Equations 8, 9
11         Send P_received^{n_j} to n_j
12     for n_j ∈ N_i do
13         Receive P_new^{n_j} back from n_j
14     P_{a_i} ← P_{a_i} ∪ P_new
15     B ← argmin_{I∈P_{a_i}} I.fitness
16     ANYTIME − UPDATE(B)
17     P_{a_i} ← Select_wrp(|N_i| * ER)
18     if Itr = Itr_M + MI then
19         for n_j ∈ N_i do
20             Send Select_wrp(ER) to n_j
21         for P_received^{n_j} received from n_j ∈ N_i do
22             P_{a_i} ← P_{a_i} ∪ P_received^{n_j}
```

INIT starts by initializing all the parameters and variables to their default values[4]. Then each agent a_i sets its variable x_i to a random value from its domain D_i. Lines 3 to 25 of Procedure 1 describe the initial population construction process. In AED, we define population P as a set of individuals that are collectively maintained by all the agents and local population $P_{a_i} \subseteq P$ as the subset of the population maintained by agent a_i. An individual in AED is represented by a complete assignment of variables in X and fitness is calculated using a fitness function shown in Equation 2. This function calculates the aggregated cost of constraints yielded by the assignment. Hence, optimizing this fitness function results in an optimal solution for the corresponding DCOP.

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

Note that a single agent can not calculate the fitness function. Rather it is calculated in parts with the cooperation of all the agents during the construction process. Moreover, the fitness value is added

[2]We suggest using the algorithm described in [1]. Height can be easily calculated by utilizing LAYER information.
[3]Length of the longest path in the pseudo-tree.
[4]AED takes a default value for each of the parameters as input. Default values of the variables are discussed later in this section.

to the representation of an individual because it enables an agent to recalculate the fitness when a new individual is constructed only using local information. We take I = $\{x_1 = 1, x_2 = 2, x_3 = 1, x_4 = 2, fitness = 38\}$ as an example of a complete individual from the DCOP shown in Figure 1. We use dot(.) notation to refer to a specific element of an individual. For example $I.x_1$ refers to x_1 in I. Additionally, we define a Merger operation of two individuals under construction, I_1, I_2 as Merge(I_1, I_2). This operation constructs a new individual I_3 by aggregating the assignments and setting $I_3.fitness = I_1.fitness + I_2.fitness$. We define an extended Merge operation for two ordered sets of individuals S_1 and S_2 as $Merge(S_1, S_2) = \{I_i : Merge(S_1.I_i, S_2.I_i)\}$ where I_i is the i-th individual in a set.

At the beginning of the construction process, each agent a_i sets P_{a_i} to a set of empty individuals[5]. The size of the initial P_{a_i} is defined by parameter IN. Then for each individual $I \in P_{a_i}$, agent a_i makes a random assignment to $I.x_i$. After that each agent a_i executes a merger operation on P_{a_i} with each local population maintained by agents in N_i (Procedure 1: Line 2-8). At this point, an individual $I \in P_{a_i}$ consists of an assignment of variables controlled by a_i, and agents in N_i with fitness set to zero. For example, I = $\{x_1 = 1, x_2 = 2, x_3 = 1, fitness = 0\}$ represents an individual of P_{a_3}. The fitness of each individual is then set to the local cost according to their current assignment (Procedure 1: Line 9-10). Hence, the individual I from the previous example becomes $\{x_1 = 1, x_2 = 2, x_3 = 1, fitness = 20\}$. In the next step, each agent a_i executes a merger operation on P_{a_i} with each local population that is maintained by the agents in C_i. Then each agent a_i sends P_{a_i} to PR_i apart from the root (Procedure 1: Line 11-18). At the end of this step, the local population maintained by the root consists of complete individuals. However, their fitness is twice its actual value since each constraint is calculated twice. Therefore, the root agent at this stage corrects all the fitness values (Procedure 1: Lines 20-21). Finally, the local population of the root agent is distributed through the network so that agents can initialize their local population (Procedure 1: Line 22-25). This concludes the initialization phase and after that, all the agents synchronously start the optimization phase in order to improve this initial population iteratively.

The Optimization Phase of AED consists of five steps, namely Selection, Reproduction, ANYTIME-UPDATE, Reinsertion and Migration. An agent a_i begins an iteration of this phase by selecting individuals from P_{a_i} for the Reproduction step (Algorithm 1: Line 4). Prior to this selection, all the individuals are ranked from $(0, R_{max}]$ based on their relative fitness in the local population P_{a_i}. The rank R_j of an individual $I_j \in P_{a_i}$ is calculated using Equation 3. Here, I_{best} and I_{worst} are the individuals with the lowest and highest fitness in P_{a_i} respectively[6]. We define $Select_{rp}(S)$ as the process of taking a sample with replacement[7] of size S from population P_{a_i} based on the probability calculated using Equation 4. As α increases in Equation 4, the fitness vs. selection probability curve gets steeper. As a consequence, individuals with better fitness get selected more often. In this way, α controls the exploration and exploitation dynamics in the Selection mechanism (See Section 5 for more details).

[5]Individuals with no assignment and fitness set to 0.
[6]For minimization problems, a lower value of fitness is better.
[7]Any individual can be selected more than once.

Procedure 1: INIT( )

1 Initialize algorithm parameters IN, ER, R_{max}, α, β, MI and variables LB, GB, FM, UM
2 $x_i \leftarrow$ random value from D_i
3 $P_{a_i} \leftarrow$Set of empty individuals
4 **for** *Individual* $I \in P_{a_i}$ **do**
5 $I.x_i \leftarrow$a random value from D_i
6 Send P_{a_i} to agents in N_i
7 **for** P_{n_j} *received from* $n_j \in N_i$ **do**
8 $P_{a_i} \leftarrow Merge(P_{a_i}, P_{n_j})$
9 **for** *Individual* $I \in P_{a_i}$ **do**
10 $I.fitness \leftarrow \sum_{n_j \in N_i} Cost_{i,j}(I.x_i, I.x_j)$
11 **if** $|C| = 0$ **then**
12 Send P_{a_i} to PR_i
13 **else**
14 Wait until received P_{c_j} from all $c_j \in C_i$
15 **for** P_{c_j} *received from* $c_j \in C_i$ **do**
16 $P_{a_i} \leftarrow Merge(P_{a_i}, P_{c_j})$
17 **if** $a_i \neq root$ **then**
18 Send P_{a_i} to PR_i
19 **else**
20 **for** *Individual* $I \in P_{a_i}$ **do**
21 $I.cost \leftarrow I.fitness/2$
22 Send P_{a_i} to all agent in C_i
23 **if** *Received* P_{PR_i} *from* PR_i **then**
24 $P_{a_i} \leftarrow P_{PR_i}$
25 Send P_{a_i} to all agent in C_i

For example, assume P_{a_i} consists of 3 individuals I_1, I_2, I_3 with fitness 16, 30, 40 respectively and $R_{max} = 5$. Then Equations 3 and 4 will yield, $P(I_1) = 0.676, P(I_2) = 0.297, P(I_3) = 0.027$ if $\alpha = 1$ and $P(I_1) = 0.92153, P(I_2) = 0.07842, P(I_3) = 0.00005$ if $\alpha = 3$. During this step, each agent a_i selects $|N_i| * ER$ individuals from P_{a_i} which we define as $P_{selected}$.

$$R_j = R_{max} * \frac{|I_{worst}.fitness - I_j.fitness| + 1}{|I_{worst}.fitness - I_{best}.fitness| + 1} \tag{3}$$

$$P(I_j) = \frac{R_j^{\alpha}}{\sum_{I_k \in P_{a_i}} R_k^{\alpha}} \tag{4}$$

Now, lines 5 to 11 of Algorithm 1 illustrate our proposed Reproduction mechanism. Agents start this step by partitioning $P_{selected}$ into $|N_i|$ subsets of size ER. Then each subset is randomly assigned to a unique neighbour. The subset assigned to $n_j \in N_i$ is denoted by $P_{new}^{n_j}$. An agent a_i creates a new individual from each $I \in P_{new}^{n_j}$ with cooperation of neighbour n_j. Initially, agent a_i changes assignment $I.x_i$ by sampling from its domain D_i using Equations 5, 6, 7. Then, $P_{new}^{n_j}$ is sent to n_j. Agent n_j updates its assignment of $I.x_j$ for each $I \in P_{received}^{a_i}$ (i.e. $P_{new}^{n_j}$) using Equation 8. Additionally, both agents a_i and n_j update the fitness of the individual I by adding δ_i and δ_j to I.fitness, respectively. Here, δ_* is calculated using Equation 9 where $I.x_*^{new}$ and $I.x_*^{old}$ are the old and new values of $I.x_*$, respectively.

$$O_{d_i} = \sum_{n_k \in N_i \setminus n_j} Cost_{i,k}(I.x_i, I.x_k) + \min_{d_j \in D_j} Cost_{i,j}(I.x_i, d_j) \quad (5)$$

$$W_{d_i} = O_{max} * \frac{|O_{worst} - O_{d_i}| + 1}{|O_{worst} - O_{best}| + 1} \quad (6)$$

$$P(d_i) = \frac{W_{d_i}^{\beta}}{\sum_{d_k \in D_i} W_{d_k}^{\beta}} \quad (7)$$

$$I.x_j = \underset{d_j \in D_j}{\operatorname{argmin}} \sum_{n_k \in N_j} Cost_{j,k}(d_j, I.x_k) \quad (8)$$

$$\delta_* = \sum_{n_k \in N_*} Cost_{*,k}(I.x_*^{new}, I.x_k) - Cost_{*,k}(I.x_*^{old}, I.x_k) \quad (9)$$

For example, agent a_3 of Figure 1 creates a new individual from $I = \{x_1 = 1, x_2 = 2, x_3 = 2, x_4 = 2, fitness = 49\}$ with the help of neighbour a_2. Here, the domain of agent a_3 and a_2 is $\{1, 2\}$. Initially, agent a_3 calculates P(1) = 0.90 and P(2) = 0.10 using Equation 5, 6, 7 ($\beta = 1$). It then updates $I.x_3$ by sampling this probability distribution. The fitness is also updated by adding δ_i (= -11). Let the updated I be $\{x_1 = 1, x_2 = 2, x_3 = 1, x_4 = 2, fitness = 38\}$, it is then sent to a_2. Based on Equation 8, the new value of $I.x_2$ should be 1. Now, agent a_2 updates $I.x_2$ along with the fitness by adding δ_j (= -16) and sends I back to a_3. Hence, Agent a_3 receives $I = \{x_1 = 1, x_2 = 1, x_3 = 1, x_4 = 2, fitness = 22\}$.

To summarize the Reproduction mechanism, each agent a_i picks a neighbour n_j randomly for each $I \in P_{selected}$. Agent a_i then updates $I.x_i$ by sampling based on the most optimistic cost (i.e. the lowest cost) of the constraint between a_i and n_j and aggregated cost of the remaining local constraints. This cost represents the optimistic local benefit for each domain value. Then n_j sets $I.x_j$ to a value that complements the optimistic change in $I.x_i$ most. The key insight of this mechanism is that it not only takes into account the improvement in fitness that the change in $I.x_i$ will bring but also considers the potential improvement the change in $I.x_j$ will bring. Moreover, note that the parameter β in Equation 7 plays a similar role as parameter α in Equation 3 (See Section 5 for details). After collecting the newly constructed individuals from neighbours they are added to P_{a_i} (Algorithm 1: Line 12-14). Then the best individual B in P_{a_i} is sent for ANYTIME-UPDATE (Algorithm 1: Line 15-16).

To facilitate the anytime update mechanism, each agent maintains four variables LB, GB, FM, UM. LB (Local Best) and GB (Global Best) are initialized to empty individuals with fitness set to infinity. FM and UM are initialized to $\emptyset$. Additionally, GB is stored with a version tag and each agent maintains previous versions of GB having version tags in the range $[Itr - H + 1, Itr]$ (see the Theoretical section for details). Here, Itr refers to the current iteration number. We use GB^j to refer to the latest version of GB with version tag not exceeding j. Ours proposed anytime update mechanism works as follows. Each agent keeps track of two different best, LB and GB. Whenever the fitness of LB becomes less than GB, it has the potential to be the global best solution. So it gets reported to the root through the propagation of a Found message up the pseudo-tree. Since the root gets reports from all the agents, it can identify the true global best solution, and notify all the agents by propagating an Update message down to the pseudo tree. The root also adds the version tag in the Update message to help coordinate variable assignment. Now, ANYTIME-UPDATE starts by keeping LB updated with the best individual B in P_{a_i}. In line 3 of Procedure 2, agents try to identify whether LB is the potential global best. When identified and if the identifying agent is the root, it is the true global best and an Update message UM is constructed. If the agent is not the root, it is a potential global best and a Found message FM is constructed (Procedure 2: Lines 4-8). Each agent forwards the message UM to agents in C_i and the message FM to the PR_i. Upon receiving these messages, an agent takes the following actions:

- If an Update message is received then an agent updates both its GB and LB. Additionally, the agent saves the Update message in UM and sends it to all the agents in C_i during the next iteration (Procedure 2: Lines 12-15).
- If a Found message is received and it is better than LB, only LB is updated. If this remains a potential global best it will be sent to PR_i during next iteration (Procedure 2: Lines 16-17).

An agent a_i then updates the assignment of x_i using $GB^{Itr-H+1}$ (Procedure 2: Lines 18-19). Agents make decisions based on $GB^{Itr-H+1}$ instead of the potentially newer GB^{Itr} so that decisions are made based on the same version of GB. $GB^{Itr-H+1}$ will be same for all agents since it takes at most H iterations for an Update message to propagate to all the agents. For example, assume agent a_1 from Figure 2 finds a potential best individual I at $Itr = 3$. Unless it gets replaced by a better individual, it will reach the root a_4 via agent a_2 through a Found message at $Itr = 4$. Then a_4 constructs an Update message $\{Version : 5, Individual : I\}$ at $Itr = 5$. This message will reach all the agents by $Itr = 6$ and the agents save it as $GB^5 = I$. Finally, at $Itr = 6$ agents assign their variables using $GB^{6-2+1} = GB^5$ which is the best individual found at $Itr = 3$.

Procedure 2: ANYTIME-UPDATE(B)

```
1  if B.fitness < LB.fitness then
2      LB ← B
3  if LB.fitness < GB^Itr.fitness then
4      if a_i = root then
5          GB^itr ← LB
6          UM ← {Version : Itr, Individual : LB}
7      else
8          FM ← {Individual : LB}
9  Send Update Message UM to agents in C_i and Found Message FM to PR_i
10 FM ← ∅
11 UM ← ∅
12 if Received update message M and M ≠ ∅ then
13     GB^{M.Version} ← M.individual
14     LB ←Best between LB and M.individual
15     UM ← M
16 if Received found message M and M ≠ ∅ and
       M.individual.fitness < LB.fitness then
17     LB ← M.individual
18 if Itr >= H then
19     x_i = GB^{Itr−H+1}.x_i
```

After ANYTIME-UPDATE each agent performs Reinsertion, i.e. updates its local population with new individuals. At first each agent adds newly constructed individuals P_{new} to P_{a_i} (Algorithm 1: line 14). After that, each agent a_i updates their P_{a_i} by keeping a sample of size $|N_i| * ER$ and discarding the rest based on their fitness (Algorithm 1: line 17). This sample is taken using $Select_{wrp}(S)$ which is the same as $Select_{rp}(S,)$ except agents sample without replacement[8]. This sampling method keeps the local population P_{a_i} diverse by selecting a unique set of individuals.

Finally, Migration, an essential step of AED, takes place on every MI iteration. We sketch this in lines 18-22 of Algorithm 1. For this step, we define Itr_M as the iteration number when the last Migration occurred. Migration is a simple process of exchanging individuals among the neighbours. In AED, the Reproduction mechanism utilizes local cooperation, so only a subset of variables of an individual change. However, because of Migration, different agents can change a different subset of variables as individuals get to traverse the network through this mechanism. Hence, this step plays an essential role in the optimization process of AED. During this step, an agent a_i selects a sample of size ER using $Select_{wrp}(S)$ for each $n_j \in N_i$ and sends a copy of those individuals to that neighbour. Upon collecting individuals from all the neighbours, an agent, a_i adds them to its local population P_{a_i}. This concludes an iteration of the optimization phase and every step repeats during the subsequent iterations..

4 THEORETICAL ANALYSIS

In this section, we first prove that AED is anytime, that is the quality of solutions found by AED increase monotonically. Then we analyze the complexity of AED in terms of communication, computation and memory requirements.

Lemma 4.1. *At iteration* $\mathbf{i + H}$*, the root agent is aware of the best individual in P at least up to iteration* $\mathbf{i}$.

Proof. Suppose, the best individual up to iteration $\mathbf{i}$ is found at iteration $\mathbf{i'} \leq \mathbf{i}$ by agent $\mathbf{a_x}$ at level $\mathbf{l'}$. Afterwards, one of the following 2 cases will occur at each iteration.

- Case 1. This individual will be reported to the parent of the current agent through a Found message.
- Case 2. This individual gets replaced by a better individual on its way to the root at iteration $\mathbf{i^* > i'}$ by agent $\mathbf{a_y}$ at level $\mathbf{l^*}$.

When only Case 1 occurs, the individual will reach the root at iteration $\mathbf{i' + l' \leq i + H}$ (since l' can be at most H). If Case 2 occurs, the replaced individual will reach the root agent by $\mathbf{i^* + l^*} = \{\mathbf{i^* - (l' - l^*)}\} + \{\mathbf{(l' - l^*) + l^*}\} = \mathbf{i' + l' \leq i + H}$. The same can be shown when the new individual also gets replaced. In either case, at iteration $\mathbf{i+H}$, the root will become aware of the best individual in P up to iteration $\mathbf{i}$ or will become aware of a better individual in P found at iteration $\mathbf{i^* > i}$; meaning the root will be aware of the best individual in P at least up to iteration $\mathbf{i}$. □

Lemma 4.2. *The variable assignment decision made by all the agents at iteration* $\mathbf{i + 2H - 1}$ *yield a global cost equal to the fitness of the best individual in P at least up to iteration* $\mathbf{i}$.

[8]Each individual can be selected at most once.

Proof. At iteration $\mathbf{i + 2H - 1}$, all the agents make decisions about variable assignment using $\mathbf{GB^{i+H}}$. However, $\mathbf{GB^{i+H}}$ is the best individual known to the root up to iteration $\mathbf{i + H}$. We know from Lemma 4.1 that, at iteration $\mathbf{i + H}$, the root is aware of the best individual in P at least up to iteration $\mathbf{i}$. Hence, the fitness of $\mathbf{GB^{i+H}}$ is at least equal to the best individual in P up to iteration $\mathbf{i}$. Hence, at iteration $\mathbf{i + 2H - 1}$, it yields a global cost equal to the fitness of the best individual in P at least up to iteration $\mathbf{i}$. □

Proposition 4.3. *AED is anytime.*

Proof. From Lemma 4.2, the decisions regarding the variable assignments at iterations $\mathbf{i + 2H - 1}$ and $\mathbf{i + 2H - 1 + \delta}$ yields a global cost equal to the fitness of the best individual in P at least up to iterations $\mathbf{i}$ and $\mathbf{i + \delta}$ ($\mathbf{\delta \geq 0}$), respectively. Now, the fitness of the best individual in P up to iteration $\mathbf{i + \delta}$ is at most the fitness at iteration $\mathbf{i}$. So the global cost at iteration $\mathbf{i + \delta}$ is less than or equal to the same cost at iteration $\mathbf{i}$. As a consequence, the quality of the solution monotonically improves as the number of iterations increases. Hence, AED is anytime. □

We now consider algorithm complexity. Assume, $\mathbf{n}$ is the number of agents, $|\mathbf{N}|$ is the number of neighbours and $|\mathbf{D}|$ is the domain size of an agent. In every iteration, an agent sends $2|N|$ messages during the Reproduction step. Additionally, at most $|N|$ messages are passed for each of the ANYTIME-UPDATE and Migration steps. Now, $|N|$ can be at most n (complete graph). Hence, the total number of messages transmitted per agent during an iteration is $O(4|N|) = O(n)$. Since the main component of a message in AED is the set of individuals, the size of a single message can be calculated as the size of an individual multiplied by the number of individuals. During the Reproduction, Migration and ANYTIME-UPDATE steps, at most ER individuals, each of which has size $O(n)$, is sent in a single message. As a result, the size of a single message is $O(ER * n)$. This makes the total message size per agent during an iteration $O(ER * n * n) = O(n^2)$.

Before Reproduction, $|P_{a_i}|$ can be at most $2ER * |N|$ (if Migration occurred in the previous iteration) and Reproduction will add $ER * |N|$ individuals. So the memory requirement per agent is $O(3 * ER * |N| * n) = O(n^2)$. Finally, Reproduction using Equations 5, 6, 7, 8 and 9 requires $|D_i| * |N|$ operations and in total $ER * |N|$ individuals are reproduced during an iteration per agent. Hence, the total computation complexity per agent during an iteration is $O(ER * |N| * |D| * |N|) = O(|D| * n^2)$.

5 EXPERIMENTAL RESULTS

In this section, we empirically evaluate the quality of solutions produced by AED compared to six different state-of-the-art DCOP algorithms. We show that AED asymptotically converges to solutions of quality higher than these six state-of-the-art algorithms. We select these algorithms to represent all four classes of non-exact algorithms. Firstly, among the local search algorithms, we pick DSA (type C, P = 0.8, this value of P yielded the best performance in our settings), MGM2 (with offer probability p = 0.5) and GDBA (N, NM, T; reported to perform the best [22]). Secondly, among the inference-based non-exact algorithms, we compare with Max-Sum_ADVP– as it has empirically shown to perform significantly better than

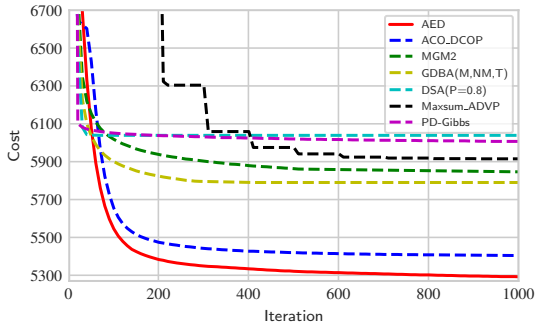


Figure 3: Comparison of AED and the benchmarking algorithms on a sparse configurations of random DCOPs.

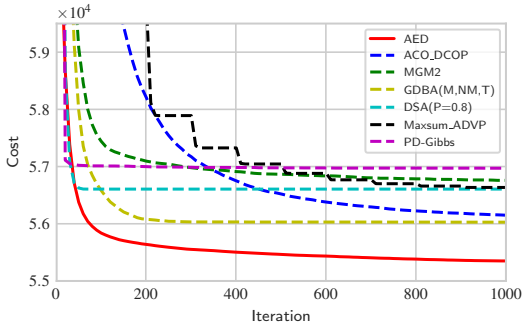


Figure 4: : Comparison of AED and the benchmarking algorithms on a dense configurations of random DCOPs.

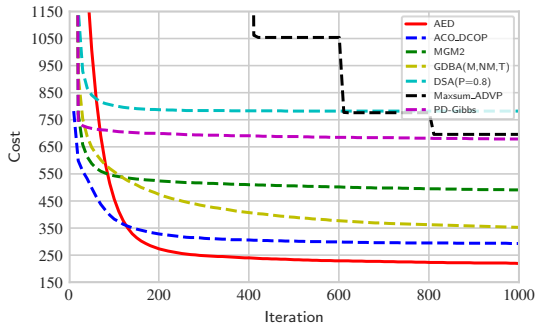


Figure 5: Comparison of AED and the benchmarking algorithms on weighted graph coloring problems.

Max-Sum [31]. We used switching parameter that yielded best result between n and 2n, where n is the number of agents. Thirdly, we consider a sampling-based algorithm, namely PD-Gibbs, which is the only such algorithm that is suitable for large-scale DCOPs [25]. Finally, we compare with ACO_DCOP as it is only available population-based DCOP algorithm. To evaluate ACO_DCOP, we use the same values of the parameters recommended in [2]. We discuss parameter settings of AED[9] in details later in this section. Additionally, we used the ALS framework for non-monotonic algorithms having no anytime update mechanism.

We compare these algorithms on three different benchmarks. We consider random DCOPs for our first benchmark. Specifically, we set the number of agents to 70 and domain size to 10. We use Erdős-Rényi topology (i.e. random graph) to generate the constraint graphs with the value of $p = 0.1$ (i.e. sparse graph) [5]. We then take constraint costs uniformly from the range $[1, 100]$. Our second benchmark is identical to the first setting except the value of $p = 0.6$ (i.e. dense graph). For our last benchmark, we consider weighted graph coloring problems with the number of agents 120, 3 colors per agent, Erdős-Rényi topology with $p = 0.05$ and constraint violation costs are selected uniformly from $[1, 100]$. In all three settings, we run all algorithms on 70 independently generated problems and 30 times on each problem. Moreover, for stopping condition we consider both max-iteration and max-time. For max-iteration, we stop each of the algorithms after the 1000-th iteration. For max-time, we run each algorithm for 4 seconds, 25 seconds and 6 seconds for the aforementioned benchmarks 1, 2 and 3, respectively. In order to conduct these experiments, we use a *GCP-n2-highcpu-64 instance*[10] - a cloud computing service which is publicly accessible at cloud.google.com. It is worth noting that all differences shown in Figures 3, 4, 5 and Table 1 are statistically significant for $p-value < 0.01$.

Figure 3 shows a comparison between AED and the benchmarking algorithms on the sparse random DCOP benchmark after running an equal amount of iteration. On the other hand, the EXP-1 column of Table 1 shows the comparison after running each algorithm an equal amount of time (4 seconds). The closest competitor to AED is ACO_DCOP. Unlike other competitors, both of the population-based algorithms kept on improving the solution until the end of the run due to their superior capability of exploration. However, it can be observed from Table 1 that AED produces 1.7% better solution than ACO_DCOP after running an equal amount of time. In contrast, most of the local search algorithms converge to local optima within 400 iterations (see Figure 3) - with GDBA producing the best performance. After running an equal amount of time, AED outperforms GDBA by a 9% and DSA by 14.9%. Finally, the other two representative algorithms, Max-Sum_ADVP and PD-Gibbs are outperformed by $11.1\% - 13.8\%$ margin. The superiority of AED in this experiment indicates that the Selection method along with the new Reproduction mechanism based on optimistic local benefit achieves a better balance between exploration and exploitation. This helps AED to explore until the end of the run and produce solutions with better quality than the state-of-the-art algorithms.

Figure 4 shows a comparison between AED and other benchmarking algorithms on dense random DCOP benchmark. It clearly shows the advantage of AED over its competitors. To be exact, it outperforms the benchmarking algorithms by a margin of $0.7\% - 3.0\%$ after running an equal amount of time (25 seconds). In this benchmatk, most of the algorithms find results of similar quality with a slight variation. Among the competitors, GDBA outperforms ACO_DCOP by a slight margin till 1000-th iteration. However, after running an equal amount of time, ACO_DCOP manages to produce better solutions and becomes the closest competitor to AED. PD-Gibbs fails to explore much through sampling and converges quickly while producing the most substantial performance difference with AED. It is also worth noting that ACO_DCOP takes 1000

[9]For implementing $Select_{wrp}(.)$ we use Reservoir-sampling algorithm [17]. For performing set operation we use constant time polynomial hashing.
[10]64 Intel Skylake vCPU @ 2.0 GHZ and 58 GB RAM

iterations to produce a similar quality solution that is found by AED at the expense of only 70 iterations.

Table 1: Comparison of AED and the benchmarking algorithms using Max-Time as Stopping condition.

Algorithm	EXP - 1	EXP - 2	EXP - 3
DSA	6076	56799	781
MGM-2	5775	56780	486
GDBA	5770	56051	310
PD-Gibbs	6021	56985	682
MS_ADVP	5877	56786	625
ACO_DCOP	5380	55735	291
AED	**5289**	**55347**	**229**

Figure 5 shows a comparison between AED and the other benchmarking algorithms on weighted graph colouring problems. In this experiment, AED demonstrates its excellent performance by outperforming other algorithms by a significant margin. Among the benchmarking algorithms, ACO_DCOP is the closest but still outperformed by AED by a 27% margin. Among the local search algorithms, GDBA is the most competitive, but AED still finds solutions that are 35% better. Finally, it improves the quality of solutions around 1.73 − 2.4 times over some of its competitors, namely DSA, Max-Sum_ADVP and PD-Gibbs after running an equal amount of time. Through this experiment, it is also evident that AED can also be an effective algorithm for DCSPs

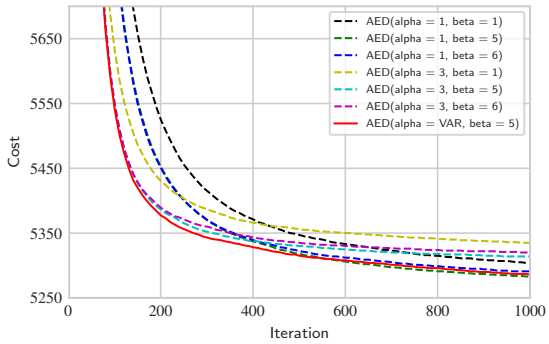


Figure 6: Performance of AED for different α and β on a sparse configurations of random DCOPs.

Now we consider the effects of different parameters on the benchmarks. Firstly, for all three benchmarks, we set IN, which defines the initial population size to 50. A small value of IN will affect the exploration of AED. However, after 50, it does not have any significant effect on the solution quality. Secondly, for all three benchmarks, we set the migration interval, MI to 5. Through the Migration process, individuals get to traverse the network and different agents get to change different variables of an individual. Hence, Migration works as an implicit cooperation mechanism. If the value of MI is set too high, convergence will slow down due to a lack of cooperation. On the other hand, when it is set too low, the population will lack diversity as different sub-population will mix fast. Thirdly, we show the effect of parameter ER on solution

Table 2: Solution Quality & Memory Requirement Per-Agent of AED for different ER value.

ER	Solution Quality			Memory (KB)		
	EXP-1	EXP-2	EXP-3	EXP-1	EXP-2	EXP-3
05	5378	55550	275	39	188	53
10	5346	55450	252	69	367	97
20	5316	**55347**	240	129	**725**	184
40	**5289**	55325	**229**	**248**	1441	**358**
50	5285	55310	223	308	1899	445

quality and memory requirement on Tables 2. ER effectively determines the population size. When it is set too low, exploration will suffer. However, as we increase ER after a certain threshold, it does not improve solution quality by any significant margin. We specifically highlight the different values of ER we used in different benchmarks on Table 2. Notice that even with the small value of $ER = 5$ AED is outperforming the benchmarking algorithms.

Finally, we depict the effect of α and β in Figure 6. While keeping α constant as we increase β, both the solution quality and the convergence rate increase up to a threshold. After that, the convergence rate does not change much but the solution quality starts to suffer. As we increase β, the Reproduction mechanism starts to exploit more than explore. At the threshold value, the balance between exploitation and exploration becomes optimal. After that, when we increase β, the exploration starts to suffer. Hence, this phenomenon occurs. In Figure 6, we observe that this β threshold is 5 (Benchmark 1). For Benchmark 2, we have found this threshold to be 5 and the value is 2 for Benchmark 3. On the other hand, as we increase α, the convergence rate increases but the solution quality decreases. In order to mitigate this problem, we use a variable α. To be precise, in the first 150 iterations, we use $\alpha = 3$. We then use $\alpha = 2$ in the following 150 iterations, and for the rest of the iterations we consider 1 as the value of α. Figure 6 shows that $alpha = VAR$ yields a similar solution quality as $\alpha = 1$; however, the convergence rate is near $\alpha = 3$.

6 CONCLUSIONS

In this paper, we introduce a novel algorithm called AED that effectively uses evolutionary optimization to solve DCOPs. To incorporate the anytime property in AED, we also present a new anytime update mechanism. In our theoretical evaluation, we prove that AED is anytime. Finally, we present empirical results that show that AED outperforms state-of-the-art non-exact algorithms by 1.7% − 14.9% on sparse random DCOPs, 0.7% − 3.0% on dense random DCOPs. More notably, AED produces 0.27 − 2.4 times better solutions on weighted graph colouring problems. These results demonstrate the significance of applying evolutionary optimization techniques in solving DCOPs. In the future, we intend to investigate whether this algorithm can be applied to solve continuous-valued and multi-objective DCOPs.

7 ACKNOWLEDGMENTS

This research is partially supported by the ICT Division of Bangladesh Government and University Grants Commission of Bangladesh.

REFERENCES

[1] Ziyu Chen, Zhen He, and Chen He. 2017. An improved DPOP algorithm based on breadth first search pseudo-tree for distributed constraint optimization. *Applied Intelligence* 47 (2017), 607–623.

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

[3] Moumita Choudhury, Saaduddin Mahmud, and Md. Mosaddek Khan. 2019. A Particle Swarm Based Algorithm for Functional Distributed Constraint Optimization Problems. *ArXiv* abs/1909.06168 (2019).

[4] Marco Dorigo, Mauro Birattari, and Thomas Stützle. 2006. Ant colony optimization: artificial ants as a computational intelligence technique.

[5] Paul Erdős and Alfréd Rényi. 1960. On the evolution of random graphs. *Institute of Mathematics, Hungarian Academy of Sciences* 5 (1960), 17–60.

[6] Alessandro Farinelli, Alex Rogers, and Nick R Jennings. 2014. Agent-based decentralised coordination for sensor networks using the max-sum algorithm. *Autonomous agents and multi-agent systems* 28 (2014), 337–380.

[7] Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R. Jennings. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems*.

[8] Ferdinando Fioretto, William Yeoh, Enrico Pontelli, Ye Ma, and Satishkumar J. Ranade. 2017. A Distributed Constraint Optimization (DCOP) Approach to the Economic Dispatch with Demand Response. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems*.

[9] David B. Fogel. 1966. Artificial Intelligence through Simulated Evolution.

[10] David B. Fogel. 1988. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics* 60 (1988), 139–144.

[11] Amir Gershman, Amnon Meisels, and Roie Zivan. 2009. Asynchronous Forward Bounding for Distributed COPs. *Journal of Artificial Intelligence Research* 34 (2009), 61–88.

[12] Katsutoshi Hirayama and Makoto Yokoo. 1997. Distributed partial constraint satisfaction problem. In *Principles and Practice of Constraint Programming-CP97*. 222–236.

[13] John Henry Holland et al. 1975. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

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

[15] Md Mosaddek Khan, Long Tran-Thanh, Sarvapali D Ramchurn, and Nicholas R Jennings. 2018. Speeding Up GDL-Based Message Passing Algorithms for Large-Scale DCOPs. *Comput. J.* 61 (2018), 1639–1666.

[16] Md. Mosaddek Khan, Long Tran-Thanh, William Yeoh, and Nicholas R. Jennings. 2018. 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 MultiAgent Systems*. 1604–1612.

[17] Kim-Hung Li. 1994. Reservoir-Sampling Algorithms of Time Complexity O(n(1 + Log(N/n))). *ACM Trans. Math. Software* 20 (1994), 481–493.

[18] Omer Litov and Amnon Meisels. 2017. Forward bounding on pseudo-trees for DCOPs and ADCOPs. *Artificial Intelligence* 252 (2017), 83–99.

[19] Rajiv T Maheswaran, Jonathan P Pearce, and Milind Tambe. 2004. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach.. In *Proceedings of the ISCA PDCS*. 432–439.

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

[21] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. 2005. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161 (2005), 149–180.

[22] Steven Okamoto, Roie Zivan, Aviv Nahon, et al. 2016. Distributed Breakout: Beyond Satisfaction.. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*.

[23] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. 2012. DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems. *ACM TIST* 8 (2012), 69:1–69:27.

[24] Adrian Petcu and Boi Faltings. 2005. A Scalable Method for Multiagent Constraint Optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*.

[25] Nguyen Thien, William Yeoh, Hoong Lau, and Roie Zivan. 2019. Distributed Gibbs: A Linear-Space Sampling-Based DCOP Algorithm. *Journal of Artificial Intelligence Research* 64 (2019), 705–748.

[26] Edward PK Tsang and Terry Warwick. 1990. Applying genetic algorithms to constraint satisfaction optimization problems. In *Proceedings of the 9th European Conference on Artificial Intelligence*.

[27] William Yeoh, Ariel Felner, and Sven Koenig. 2008. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. *Journal of Artificial Intelligence Research* 38 (2008), 85–133.

[28] Makoto Yokoo, Edmund H Durfee, Toru Ishida, and Kazuhiro Kuwabara. 1998. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on knowledge and data engineering* 10 (1998), 673–685.

[29] Weixiong Zhang, Guandong Wang, Zhao Xing, and Lars Wittenburg. 2005. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* 161 (2005), 55–87.

[30] Roie Zivan, Steven Okamoto, and Hilla Peled. 2014. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence* 212 (2014), 1–26.

[31] Roie Zivan and Hilla Peled. 2012. Max/min-sum distributed constraint optimization through value propagation on an alternating DAG. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems*.