Noname manuscript No.
(will be inserted by the editor)

An Artificial Bee Colony Based Algorithm for Continuous-DCOPs

K. M. Merajul Arefin · Mashrur Rashik · Saaduddin Mahmud · Md. Mosaddek Khan

Received: date / Accepted: date

Abstract Recently, Continuous Distributed Constraint Optimization Problems (C-DCOPs), an extension of the widely used Distributed Constraint Optimization Problems (DCOPs) framework, has gained interest from the multi-agent systems research community. Unlike the traditional DCOPs, which deals with variables having only discrete values, variables in C-DCOPs can be continuous-valued. So far, among different approaches, population-based algorithms are shown to be most effective for solving C-DCOPs. Considering the potential of population-based approaches, in this paper, we propose a new C-DCOPs solver inspired by a well-known population-based algorithm Artificial Bee Colony, that we call Distributed Artificial Bee Colony algorithm (ABCD). Furthermore, we propose a new exploration mechanism that enhances ABCD to work better in C-DCOPs. Finally, we theoretically prove that ABCD is an anytime algorithm and empirically show our algorithm produces significantly better results than the state-of-the-art C-DCOPs algorithms.

Keywords Distributed Problem Solving · Continuous Distributed Constraint Optimization

K. M. Merajul Arefin
Department of Computer Science and Engineering
University of Dhaka, Dhaka, Bangladesh
E-mail: merajularefin@gmail.com

Mashrur Rashik
Department of Computer Science and Engineering
University of Dhaka, Dhaka, Bangladesh
E-mail: mashrur639@gmail.com

Saaduddin Mahmud
Department of Computer Science and Engineering
University of Dhaka, Dhaka, Bangladesh
E-mail: saadmahmud14@gmail.com

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

1 Introduction

Distributed Constraint Optimization Problems (DCOPs) [14] are a widely used framework for modeling systems where multiple agents cooparate to maximize their aggregated utility. DCOPs have been successfully applied to solve many multi-agent coordination problems, including multi-robot coordination [21], sensor networks [16], smart home automation [6,15], smart grid optimization [13,12], cloud computing applications [8], etc.

Under the DCOPs framework, agents in a system control the decision variables isn't it too early to metion decision variables? used to define the constraint agent utilities. In general, DCOPs assume these variables are discrete and that each constraint utility of the system is in tabular forms. However, several problems such as target tracking sensor orientation [7], sleep scheduling of wireless sensors [10] are better modeled with variables with continuous domains. DCOPs can deal with the continuous-valued variables by discretization. However, for a problem to be tractable, the discretization process must be coarse and sufficiently fine to find high-quality solutions to the problem [16]. To address this issue, [16] suggest a framework called Continuous DCOPs (C-DCOPs), which extends the general DCOPs to operate with variables that take values from a range. Additionally, in C-DCOPs, constraint utilities are specified by functions instead of the tabular form of the traditional DCOPs.

Over the last few years, a number of algorithms have been introduced to solve C-DCOPs [16,17,3,9]. Initially, Continuous Max-Sum (CMS) [16] is proposed to solve C-DCOPs which is an extension of the discrete max-sum algorithm [5]. CMS approximates the utility functions of the system with piecewise linear functions. Then the discrete max-sum algorithm is combined with continuous non-linear optimization methods by Hybrid CMS (HCMS) [17]. Recently, a population-based anytime algorithm, namely PFD [3] is proposed, which has shown better results than state-of-the-art algorithms. However, as the number of agents increases, PFD also consumes more time than its competitor algorithms.

More recently, a variety of exact and non-exact algorithms were introduced by [9]. Specifically, the inference-based DPOP [14] has been expanded to suggest three algorithms: Exact Continuous DPOP (EC-DPOP), Approximate Continuous DPOP (AC-DPOP), and Clustered AC-DPOP (CAC-DPOP). EC-DPOP presents an exact solution where a system's agents are grouped in a tree-structured network. However, this is not a feasible assumption in most problems. AC-DPOP and CAC-DPOP, on the other hand, give approximate solutions to the C-DCOP problem for general constraint graphs. In addition, they develop Continuous DSA (C-DSA) by extending the search based Distributed Stochastic Algorithm (DSA) [20]. As these three approximate algorithms use continuous optimization techniques such as gradient-based optimization, they require derivative calculations and, therefore not suitable for non-differentiable optimization problems. Moreover, recent experiments comparing against other C-DCOP algorithms also show that these algorithms produce approximate solutions of poorer quality [3].

With this background, we propose a new anytime algorithm - Distributed Artificial Bee Colony algorithm for C-DCOPs (ABCD), that is inspired by a well-known Artificial Bee Colony (ABC) algorithm and its recent variants [11,19]. Similar to the ABC algorithm, ABCD is a population-based stochastic algorithm that stores multiple

candidate solutions amongst the agents. It improves the global solution by iteratively adjusting the candidate solutions and discards solutions that do not improve after a certain period. Besides, we propose a novel mechanism that enhances the exploration ability of traditional ABCD. We also theoretically prove that ABCD is an anytime algorithm. Finally, our empirical results show that ABCD provides a better solution than other state-of-the-art algorithms, and our proposed mechanism successfully enhances the quality of solutions found by general ABCD. give some quantitative results

2 Background

In this section, we first describe the C-DCOP framework. We then briefly address the algorithm of the Artificial Bee Colony (ABC) on which our contribution is based.

2.1 Continuous Distributed Constraint Optimization Problems

A Continuous Distributed Constraint Optimization Problem can be defined as a tuple $\langle A, X, D, F, \alpha \rangle$ [16,3] where,

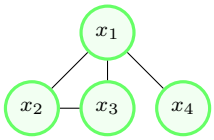
- $A = \{a_i\}_{i=1}^{n}$ is a set of agents.
- $X = \{x_i\}_{i=1}^{m}$ is a set of variable. In this paper, we use the terms "agent" and "variable" interchangeably.
- $D = \{D_i\}_{i=1}^{m}$ is a set of continuous domains for each variable $x_i \in X$. Each variable takes values from the interval $D_i = [LB_i, UB_i]$.
- $F = \{f_i\}_{i=1}^{l}$ is a set of utility functions, 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 utility for that function f_i is defined for every possible value assignment of x^i, that is, $f_i : D_{i_1} \times D_{i_2} \times ... \times D_{i_k} \to \mathbb{R}$ where the arity of the function f_i is k. In this paper, we only considered binary quadratic functions for the sake of simplicity.
- $\alpha : X \to A$ is a mapping function that associates each variable $x_j \in X$ to one agent $a_i \in A$.

The solution of a C-DCOP is an assignment X^* that maximizes the constraint aggregated utility functions as shown in Equation 1.

shouldn't the x below argmax be the same one a

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

Figure 1 shows an example of C-DCOP where Figure 1a depicts the constraint graphs where the connection between each variables is given and variable x_i is controlled by an agent a_i. Figure 1b shows the utility functions defined for each edge in the constraint graph. In this example, every variable $x_i \in X$ have the domain $D_i = [-10, 10]$.



(a) Constraint Graph

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

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

$$f_{13}(x_1, x_3) = e^{\sqrt{x_1^2 + x_3^2}}$$

$$f_{14}(x_1, x_4) = (x_1 + 2x_4 - 7)^2$$

$$f_{23}(x_2, x_3) = x_2^2 + x_3^2 - x_2 x_3$$

(b) Utility Functions

Fig. 1: Example of a C-DCOP

Algorithm 1: Artificial Bee Colony Algorithm

1 $P \leftarrow$ Set of random solutions
2 **repeat**
3 $\quad B \leftarrow$ Search improved solutions near solutions in P
4 $\quad P \leftarrow P \cup B$
5 $\quad C \leftarrow$ Search improved solutions near good solutions of P
6 $\quad P \leftarrow P \cup C$
7 $\quad$ Discard solutions of P that did not improve
8 **until** *Requirements are met*

2.2 Artificial Bee Colony Algorithm

Artificial Bee Colony (ABC) algorithm [11, 19] is a population-based stochastic algorithm to find the minimum or maximum of a multi-dimensional numeric function. ABC is inspired from the behaviour of honey bees on how they search for a better food source in nature. The generalised algorithm has some steps that are provided in Algorithm 1.

Initially, a population P is created with several random solutions (Line 1). Afterwards, for each solution of P, it creates new solutions B (Line 3) and if better solutions are found, P gets updated with B (Lines 4). It also creates new solutions C from particular solutions of P which are selected based on their quality (Line 5) and P gets updated with C if there are better solutions in C (Line 6). Solutions that are not being updated after a certain period in P are replaced with new random solutions (Line 7). These operations are running until the termination criteria are met (Line 8).

However, some challenges that need to be addressed while tailoring ABC for C-DCOPs framework. ABC optimizes a single centralized function, whereas a C-DCOP consists of factored decentralized functions. Moreover, the population generated by the algorithm needs to be stored in a distributed way among the agents. In effect, the aggregated utility of those solutions needs to be calculated in a distributed manner. Furthermore, it is also not trivial to incorporate the anytime property within ABCD. To maintain this property, we need to identify the global best solution within the

whole population and whenever the global best gets updated, a distributed method needs to be devised to propagate that global best to all the agents. In the following section, we describe our algorithm ABCD which addresses all of these challenges.

3 The ABCD Algorithm

In this section, we discuss our proposed algorithm in detail (Algorithm 2). From now on, we are going to use ABCD-C and ABCD-E to denote classical ABCD and our tailored ABCD, respectively. Moreover, the term ABCD will be used to denote both versions in general. To better understand the algorithm, we first introduce the notations used in Algorithm 2. Here, $P = \{P_1, P_2, ..., P_S\}$ is a number of population that are collectively maintained by all agents. Each $P_j = \{P_j^1, P_j^2, ..., P_j^n\}$ is the j-th solution of P where n is the number of agents. Now, each $P^i = \{P_1^i, P_2^i, ..., P_S^i\}$ is the local population maintained by agent a_i where S is the number of solutions in population P. In particular, P_j^i is an object maintained by agent a_i for j-th solution of population P. It has two attributes: a variable x_j that holds the value and $fitness$ that holds the local utility. From now on, we use the terms fitness and utility interchangeably.

I think it will be useful for the

In the case of ABCD-C, the $ROOT$ agent possesses a $count$ variable for each solution in P. In general, $count_j$ holds the number of times j-th solution of population P has been explored and each time it has produced another solution that has failed to produce a better solution. While in ABCD-E, $ROOT$ maintains a set namely $visited$, which has a total of S elements for each solution in the population P. Each element of $visited_u \in visited$ has n boolean values $visited_u.v_i$ for each agent $a_i \in A$. If the value of $visited_u.v_i$ is TRUE, it means that for solution u, an agent a_i has explored its search space. In both the ABCD versions, $ROOT$ also has $prob$ and fit values for each solution which we will discuss later in this section. ABCD-E takes two parameters as inputs: S is the number of solutions in the population that has to be created and M is the number of solutions in E which is called the elite set. ABCD-C maintains an extra parameter named $LIMIT$. $LIMIT$ determines the maximum number of times a solution can be explored. If a solution is explored more than $LIMIT$ times, it will be replaced with a random solution in ABCD-C.

ABCD starts with constructing a pseudo-tree (see [2] for more details) (Line 44). The pseudo-tree is used as a communication structure to transmit messages among the agents. Here, each agent has a unique priority associated with them. To be precise, an agent with a lower depth has higher priority than an agent with higher depth. When two agents have the same depth, their priorities are set randomly. We use N^i to refer agent a_i's neighboring agents and the notations PR^i and CH^i to refer to the parent and children of agent a_i, respectively. The root agent and the leaf agents do not have any parent and children, respectively.

After constructing pseudo-tree, Line 45 executes the INITIALIZATION procedure that initializes the population P using Equation 2. In Equation 2, r_t^i is a random floating number from the range $[0, 1]$ chosen by agent a_i for the t-th solution of population P. $ROOT$ then sets the value for each $visited_u.v_j$ to FALSE (Line 48).

Algorithm 2: ABCD-E algorithm

Input: S - Number of Solution in the Population
M - Number of Solutions in the Elite set

INITIALIZATION( )
while *Termination Criteria not met* **do**
  BUILD( )
  $Q_u^i \leftarrow P_u^i, \forall P_u^i \in P^i$
  if $a_i = ROOT$ **then**
    for $Q_u^i \in Q^i$ **do**
      $a_j \leftarrow$ Random agent from A
      Send Request to a_j to update value for u
      $visited_u.v_j \leftarrow TRUE$
  while *Get Request to update from* $ROOT$ **do**
    $E_l^i \leftarrow$ Random Solution From E^i
    $a_h \leftarrow$ Random agent from $A - \{a_i\}$
    Wait until $E_l^h,\ P_u^h$ is received from a_h
    Calculate Q_u^i from Equation 3
  EVALUATE(Q^i)
  if $a_i = ROOT$ **then**
    Replace P_u^i by Q_u^i in the pseudo tree if $Q_u^i.fitness > P_u^i.fitness, \forall Q_u^i \in Q^i$
    Propagate Q_u as $Gbest$ in the pseudo tree if $Q_u.fitness > Gbest.fitness, \forall Q_u^i \in Q^i$
  Calculate $prob$ for each solution if $a_i = ROOT$
  for $z \leftarrow 1$ **to** S **do**
    if $a_i = ROOT$ **then**
      $P_u^i \leftarrow$ Random solution from P^i according to $prob$
      Propagate u in the pseudo tree to make a copy of u-th solution
      for $E_m^i \in E^i$ **do**
        $a_j \leftarrow$ Random agent from A
        $visited_u.v_j \leftarrow TRUE$
        Send Request to a_j to update value for u and m
    $R_t^i \leftarrow P_u^i, \forall t \in \{1, ..., M\}$
    while *Get Request to update from* $ROOT$ **do**
      $a_h \leftarrow$ Random agent from $A - \{a_i\}$
      $E_l^i \leftarrow$ Random Solution From E^i
      Wait until $E_m^h,\ P_u^h$ from agent a_h
      Calculate R_m^i from Equation 7
    EVALUATE(R^i)
    if $a_i = ROOT$ **then**
      Replace P_u^i by R_t^i in the pseudo tree if $R_t^i.fitness > P_u.fitness, \forall R_t^i \in R^i$
      Propagate R_t as $Gbest$ in the pseudo tree if $R_t^i.fitness > Gbest.fitness, \forall R_t^i \in R^i$
  for $visited_t \in visited$ **do**
    if $a_i = ROOT$ **then**
      if *Every element of* $visited_t$ *is* $TRUE$ **then**
        Set every element of $visited_t$ to FALSE
        Send request to every agent to change P_t^i
    Calculate P_t^i from Equation 2 if request for changing value of P_t^i

Procedure INITIALIZATION

44 Construct Pseudo Tree
45 $P_t^i.x_i \leftarrow$ Set of S values in D_i using Equation 2 $\forall P_t^i \in P^i$
46 **if** $a_i = ROOT$ **then**
47 $\quad Gbest^i.fitness \leftarrow -\infty$
48 $\quad visited_u.v_j \leftarrow FALSE$ where $u \in \{1, ..., S\}$ and $j \in \{1, ..., n\}$

Procedure EVALUATE(W^i)

Input: W^i - Particular population to calculate the aggregated utility
49 Wait until W^j values are received from each agent $a_j \in N^i$
50 **for** $W_k^i \in W^i$ **do**
51 $\quad W_k^i.fitness \leftarrow \sum_{a_j \in N^i} f_{ij}(W_k^i.x_i, W_k^j.x_j)$
52 Wait until all the $fitness$ values are received from each agent $a_j \in CH^i$
53 **for** $W_k^i \in W^i$ **do**
54 $\quad W_k^i.fitness \leftarrow W_k^i.fitness + \sum_{a_j \in CH^i} W_k^j.fitness$
55 **if** $a_i = ROOT$ **then**
56 $\quad$ **for** $W_k^i \in W^i$ **do**
57 $\quad\quad W_k^i.fitness \leftarrow W_k^i.fitness/2$
58 **else**
59 $\quad$ Send $W^i.fitness$ to PR^i

It also initializes the $fitness$ values of $Gbest$ to $-\infty$, as we are searching for a solution with maximum utility (Line 47).

$$P_t^i.x_i = LB_i + r_t^i * (UB_i - LB_i) \tag{2}$$

Now, the procedure EVALUATE is used to calculate the local utility for each object in a decentralized manner and also by using the local utility, we determine the aggregated utility for a particular population W. It first waits for the objects of its neighboring agents $a_j \in N_i$ (Line 49). It then calculates each agent a_i's utility f_{ij} for all of its neighbors a_j. For each function, we pass two values, $W_k^i.x_i$ and $W_k^j.x_j$, to calculate f_{ij}. It aggregates all f_{ij} values and stores it in the $fitness$ variable (Lines 50-51). Then, it waits for $fitness$ values from its child agents $a_j \in CH_i$ (Line 52). After receiving the values, it sums up those values and adds it to its own $fitness$ (Lines 53-54). Any agent other than $ROOT$ sends its $fitness$ values to its parent PR_i (Lines 58-59). This process continues until $ROOT$ receives all the $fitness$ values from its child agents. At this point, the utility of each constrained is doubly counted in the aggregated utility values. This is because the local utility values of both agents in the scope of the constraint are aggregated together. Hence, $ROOT$ divides each aggregated utility by 2 (Lines 55-57).

After the INITIALIZATION phase, ABCD calls the BUILD procedure for population P (Line 3). It first calculates the aggregated utility for the population P (Line 60). Afterwards, $ROOT$ selects M best solutions among them and stores them in a distributed way (Lines 63-64). It also updates $Gbest$ if any of those S solutions

Procedure BUILD

60 EVALUATE(P^i)
61 **if** $a_i = ROOT$ **then**
62 Propagate P_u^i as $Gbest$ in the pseudo tree if $P_u^i.fitness > Gbest.fitness, \forall P_u^i \in P^i$
63 $E \leftarrow$ Set of M best solutions from P
64 Propagate E in the pseudo tree
65 Receive E from $ROOT$ and store the values in E^i
66 $x_i \leftarrow Gbest^i$

have a greater utility than the utility of $Gbest$ (Line 62). Each agent receives the propagation and stores the specific values in E^i (Line 65). Afterwards, each agents constraint variable x_i is set to $Gbest^i$ to achieve the anytime property.

ABCD now moves onto updating each solution in the population P. It first creates a copy of the main P in Q (Line 4). $ROOT$ then chooses a random agent from $a_j \in A$ and sends a request to update Q_u^j, and sets the attribute $visited_u.x_j$ to TRUE (Lines 7-9). In case of ABCD-C, the $ROOT$ agent would increment the value of $count_u$ by 1. Agents who receive the request now update the object. It selects a random solution E_l and another random agent a_h excluding itself (Lines 11-12). It then waits until E_l^h and P_u^h is received from agent a_h (Line 13). It then calculates Q_u^i from Equation 3 where ϕ_u^i and Φ_u^i are two random numbers from $[-0.5, 0.5]$ and $[0, 1]$, respectively (Line 14). Note that often updating a value might take it outside the range of D_i. It uses Equation 4 to get the value inside the range $[LB_i, UB_i]$. After completing the updates, each agent executes $EVALUATE$ for population Q (Line 15). $ROOT$ then checks for a better solution(s). If Q_u is better than P_u, P_u gets replaced by all agents (Line 17). $ROOT$ also tries to update $Gbest$ if there is any better solution than the $Gbest$ (Line 18).

$$Q_u^i.x_i = \frac{1}{2}(E_l^h.x_h + Gbest_i.x_i) + \phi_u^i(P_u^h.x_h - E_l^i.x_i) + \Phi_u^i(P_u^h.x_h - Gbest_i.x_i) \tag{3}$$

$$Q_u^i.x_i = \begin{cases} LB_i, \text{if } Q_u^i.x_i < LB_i \\ UB_i, \text{else if } Q_u^i.x_i > UB_i \\ Q_u^i.x_i, \text{otherwise} \end{cases} \tag{4}$$

$Root$ calculates the probability of each solution being chosen for a re-search. Firstly, Equation 5 converts every value to a positive value because there can also be negative valued fitness in the population (Line 19). While Equation 6 determines the probability of being chosen for any solution

$$P_u^i.fit = \begin{cases} \frac{1}{1+abs(P_u^i.fitness)}, \text{if } P_u^i.fitness < 0 \\ 1 + P_u^i.fitness, \text{otherwise} \end{cases} \tag{5}$$

$$P_u^i.prob = \frac{P_u^i.fit}{\sum_{P_v^i \in P^i} P_v^i.fit} \tag{6}$$

Afterward, ABCD runs an update process S times. Each time $ROOT$ selects a solution from P according to the probabilities it previously calculated (Line 22). It then creates M solutions by changing the selected solution and for that it selects a random agent a_j (Line 25), and set the values for $visited_u.x_j$ to TRUE (Line 26). In the case of ABCD-C, the $ROOT$ agent would increment the value of $count_u$ by 1. Agent a_i will send a request to a_j to update the values given u and m where u denotes the index of the solution in P and m denotes the index of the solution of E (Line 27). Every agent makes M copies of P_u^i for the update process (Line 28). The agents, who receive the request for updating values in R^i, start with selecting another random agent a_h other than itself (Line 30) and E_l^i randomly from E^l (Line 31). It then receives the values E_m^h and P_u^h from agent a_h (Line 32). Equation 7 determines the value of $R_m^i.x_i$ where ϕ_m^i and Φ_m^i are two random numbers from the ranges $[-0.5, 0.5]$ and $[0, 1]$, respectively (Line 33). Equation 4 fits the values inside the domain D_i when some values are outside the range. Following the update process, each agent runs the EVALUATE procedure for R^i (Line 34). $ROOT$ tries to update P_u^i and $Gbest$ with R^i when there is any scope of improvement (Lines 36-37).

Finally, in ABCD-E, $ROOT$ observe each solutions u to check whether every element of $visited_u$ is TRUE or not (Line 40). When TRUE, it means that for u-th solution all agents have explored it. Otherwise, it resets the value of $visited_u$ to FALSE and sends a request to each agent to replace that solution values with random values using Equation 2 (Lines 41-42). While in ABCD-C, $ROOT$ observes each solution u to check whether $count_u$ is greater than $LIMIT$ or not. If it is greater than $LIMIT$, it means that it has been explored at least $LIMIT$ times. Hence, $ROOT$ resets the values of $count_u$ to FALSE and sends a request to every agent to replace that solutions values with random values using Equation 2. Once agents receive the request from $ROOT$, the update operation is executed (Line 43).

$$R_m^i.x_i = \frac{1}{2}(E_m^h.x_h + Gbest_i.x_i) + \phi_m^i(P_u^h.x_h - E_l^i.x_i) + \Phi_m^i(P_u^h.x_h - Gbest_i.x_i) \tag{7}$$

4 Theoretical Analysis

In this section, we prove that both the ABCD-C and ABCD-E algorithm is anytime, that is the quality of solutions found by ABCD increases monotonically. We also evaluate both the time complexity and memory complexity for ABCD.

Proposition 1 *The ABCD algorithm is anytime.*

Proof At the beginning of the iteration $\mathbf{i+1}$, the complete assignment to X is updated according to the $Gbest$ (Algorithm 2: Line 66). Now, $Gbest$ is updated according to the best solution in the population found up to $\mathbf{i}$ (Algorithm 2: Lines 18, 37, and 62). Hence, the utility of $Gbest$ either stays the same or increases. Since X is updated according to Gbest, the global utility of the given C-DCOP instance at iteration $\mathbf{i+1}$ is greater than or equal to the utility at iteration $\mathbf{i}$. As a consequence, the quality of

Table 1: Varying the size of Population (S)

	$S = 20$	$S = 50$	$S = 100$	$S = 200$	$S = 300$
$\|A\| = 40$	621808.35	623668.54	623963.19	624247.30	624297.86
$\|A\| = 60$	1007719.86	1013915.30	1017066.49	1019535.06	1020115.76
$\|A\| = 80$	1063921.75	1067553.42	1069296.47	1071225.38	1071903.99
$\|A\| = 100$	1420278.78	1437155.85	1443268.51	1448638.86	1451822.82

Table 2: Varying the size of Elite Set (M)

	$M = 5$	$M = 10$	$M = 15$	$M = 20$	$M = 25$
$\|A\| = 40$	623812.62	624246.66	624323.55	624336.81	624390.56
$\|A\| = 60$	1440799.10	1460300.46	1461745.54	1463293.99	1463899.43
$\|A\| = 80$	1070036.34	1072747.02	1072886.09	1073526.41	1073734.39
$\|A\| = 100$	1017258.09	1019988.83	1021041.26	1021325.03	1021565.79

the solution monotonically improves as the number of iterations increases. Hence, the ABCD algorithm is anytime.

In terms of complexity, the population that are created in ABCD are stored in a distributed manner. So each agent holds S elements for P, S elements for Q and M elements for R. Hence, in total, each agent has $O(S + S + M) = O(S + M)$ values stored in them. But $ROOT$ holds two extra variables: $visited$ and $prob$. Because of that, $ROOT$ has $O(S * M + S) = O(S * M)$ extra values. Each agent first updates each solution once; and it then updates a solution M times. For this reason, each agent would do $O(S + S * M) = O(S * M)$ operations. As the value of M is very small, this does not create an issue for time consumption.

5 Experimental Results

This section provides empirical results to demonstrate the superiority of ABCD-C and ABCD-E against the current state-of-the-art C-DCOP algorithms, namely AC-DPOP, C-DSA, and PFD. To that end, we first show the effect of population size (S) and elite size (M) on ABCD-E and select the best pair of parameter values. Then we benchmark ABCD, and competing algorithms on Random DCOPs Problems [3] with binary quadratic constraints, i.e. constraints in the form of $ax^2 + bx + dy^2 + ey + fxy + g$. It is worth mentioning, ABCD can operate with functions of any kind. We consider this form of constraint to be consistent with prior works. In each problem, we set each variable's domain to $[-50, 50]$. Furthermore, we consider three different types of constraint graph topology: scale-free graphs, small-world graphs, and random graphs. Finally, we run these benchmarks by varying both the number of agents and constraint density.

The performance of ABCD-E depends on its two parameters, i.e., population size S and elite size M. To demonstrate the effect of population size S on the solution quality, we benchmark ABCD-E on Erdős-Rényi topology[4] with constraint density 0.3. We present the S vs. solution quality of ABCD-E on Table 1. From there, we can see that as we increase S, solution quality quickly improves up to $S = 200$. After that, increasing S does not result in a significant improvement in solution quality. We run a similar experiment for elite size and present the results in Table 2. Like S,

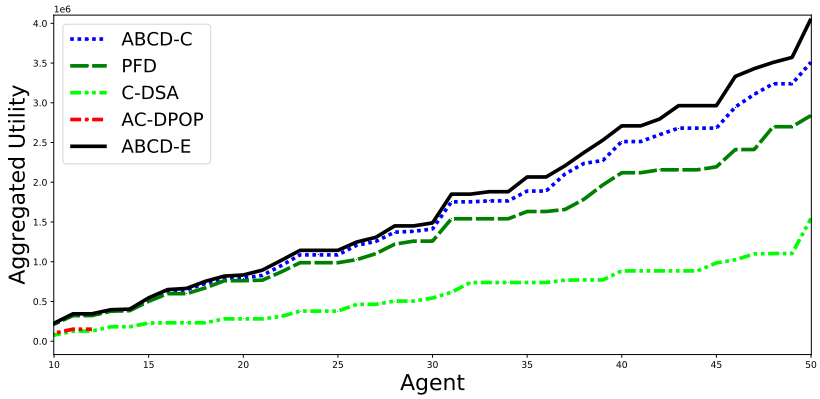


Fig. 2: Random Dense Graph

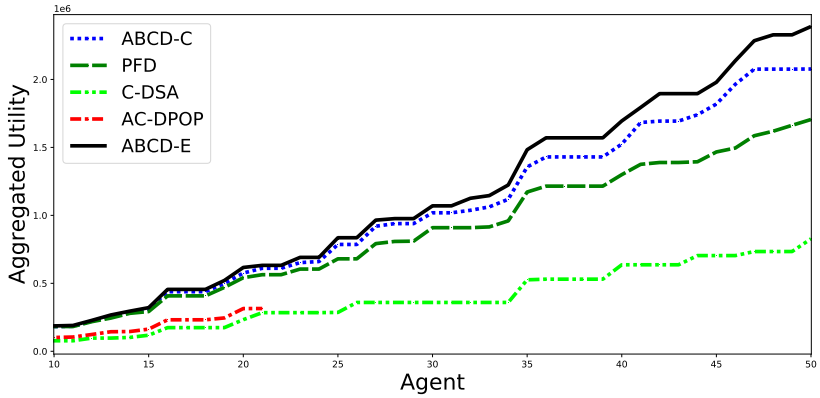


Fig. 3: Random Sparse Graph

increasing M after a certain point does not significantly improve the solution quality. Further, the computational complexity and message numbers of ABCD-E depend on both S and M ($O(S * M)$). Considering this, we select $S = 200$ and $M = 10$. It is worth mentioning that selecting parameter values for ABCD-E is considerably easier than its main competitor algorithm PFD. This is because both S and M are only constrained by available resources, and increasing them results in a consistent improvement of solution quality.

We now compare ABCD-C and ABCD-E with the state-of-the-art C-DCOP algorithms, namely, PFD, AC-DPOP, and C-DSA. We select the parameters for these algorithms as suggested in their corresponding papers. In all settings, we run all algorithms on 50 independently generated problems and 50 times on each problem. We run all the algorithms for an equal amount of time. It is worth noting that all differences shown in Figures 2, 3, 4, and 5 are statistically significant for $p - value < 0.01$.

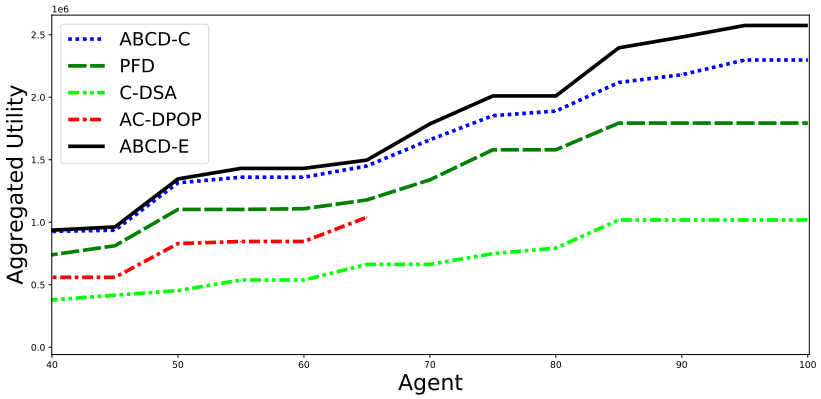


Fig. 4: Scale Free Sparse Graph

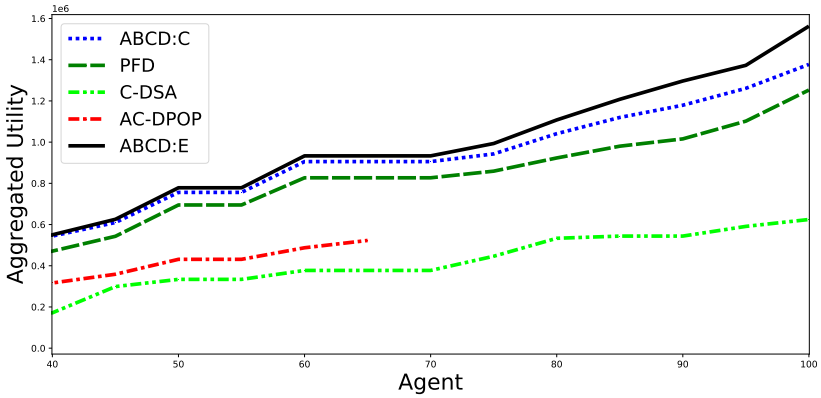


Fig. 5: Small World Sparse Graph

We first consider the random network benchmark. To construct random networks, we use Erdős-Rényi topology [4]. We vary the constraint density from 0.1 to 0.7. In all cases, both ABCD-E and ABCD-C outperform the competing algorithms. We only present the results for density 0.3 (sparse Figure 3) and density 0.7 (dense Figure 2) for space constraints. In sparse settings, ABCD-C produces 22% better solutions than its closest competitor PFD. On the other hand, ABCD-E improves the solutions generated by ABCD-C by 15%. In dense settings, we see a similar trend.

We similarly run the experiment using scale-free network topology. To create scale-free sparse networks, we used Barabasi-Albert [1] topology where the number of edges to connect from a new node to an existing node is 3. We present the result for this setting in Figure 4. We see a similar performance improvement by ABCD-C and ABCD-E as in the random network experiment above. It is worth mentioning a similar trend in performance gain continues as we increase the graph density.

Finally, we run our experiments on small-world networks. To construct small-world networks, we use the Watts-Strogatz topology [18] model where the number of nodes to join with a new node is 3, and the likelihood of rewiring is 0.5. Figure 5 depicts that ABCD-C offers 10% better performance than PFD. Moreover, ABCD-E enhances results generated by ABCD-C by 13%.

These experiments demonstrate both the superiority of ABCD against the competing algorithms and the efficacy of our proposed exploration mechanism (ABCD-E). We see a consistent performance gain over competing algorithms by ABCD under different constraint graph topology, size, and density. We see a similar trend of performance gain when comparing ABCD-E and ABCD-C. The main reason that ABCD-E outperforms ABCD-C is that in ABCD-E, every solution in the population is getting explored by every agent. As different agents possess the ability to modify different parts of the solution due to factored nature of the C-DCOPs, it significantly improves exploration. On the other hand, in ABCD-C, each solution gets explored a limited number of times, and all the agents do not get to improve each solution. As a result, many potential good solutions get prematurely discarded.

6 Conclusions and Future Work

We develop a C-DCOP algorithm, namely ABCD, by tailoring a well-known population-based algorithm (i.e., Artificial Bee Colony). More importantly, we extend ABCD and develop a new approach called ABCD-E that improves ABCD's exploration ability to deliver better performance. We theoretically prove that these are anytime algorithms. Finally, we present empirical results that show that ABCD-E outperforms the state-of-the-art non-exact C-DCOPs algorithms by a notable margin. In the future, we will investigate whether ABCD can be applied to solve multi-objective C-DCOPs.

I think the comclusion sho

References

1. Albert, R., Barabási, A.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1), 47 (2002)
2. Chen, Z., He, Z., He, C.: An improved dpop algorithm based on breadth first search pseudo-tree for distributed constraint optimization. Applied Intelligence **47**(3), 607–623 (2017)
3. Choudhury, M., Mahmud, S., Khan, M., et al.: A particle swarm based algorithm for functional distributed constraint optimization problems. In: AAAI, pp. 7111–7118 (2020)
4. ERDdS, P., R&wi, A.: On random graphs i. Publ. math. debrecen **6**(290-297), 18 (1959)
5. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.: Decentralised coordination of low-power embedded devices using the max-sum algorithm (2008)
6. Fioretto, F., Yeoh, W., Pontelli, E.: A multiagent system approach to scheduling devices in smart homes. In: Workshops at the Thirty-First AAAI Conference on Artificial Intelligence (2017)
7. Fitzpatrick, S., Meetrens, L.: Distributed sensor networks a multiagent perspective, chapter distributed coordination through anarchic optimization (2003)
8. Hoang, K., Wayllace, C., Yeoh, W., Beal, J., Dasgupta, S., Mo, Y., Paulos, A., Schewe, J.: New distributed constraint reasoning algorithms for load balancing in edge computing. In: International Conference on Principles and Practice of Multi-Agent Systems, pp. 69–86. Springer (2019)
9. Hoang, K., Yeoh, W., Yokoo, M., Rabinovich, Z.: New algorithms for continuous distributed constraint optimization problems. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, pp. 502–510 (2020)
10. Hsin, C., Liu, M.: Network coverage using low duty-cycled sensors: random & coordinated sleep algorithms. In: Proceedings of the 3rd international symposium on Information processing in sensor networks, pp. 433–442 (2004)
11. Karaboga, D.: An idea based on honey bee swarm for numerical optimization. Tech. rep., Technical report-tr06, Erciyes university, engineering faculty, computer ... (2005)
12. Kumar, A., Faltings, B., Petcu, A.: Distributed constraint optimization with structured resource constraints (2009)
13. Miller, S., Ramchurn, S., Rogers, A.: Optimal decentralised dispatch of embedded generation in the smart grid (2012)
14. Petcu, A., Faltings, B.: Dpop: A scalable method for multiagent constraint optimization. In: IJCAI 05, CONF, pp. 266–271 (2005)
15. Rust, P., Picard, G., Ramparany, F.: Using message-passing dcop algorithms to solve energy-efficient smart environment configuration problems. In: IJCAI, pp. 468–474 (2016)
16. Stranders, R., Farinelli, A., Rogers, A., Jennings, N.: Decentralised control of continuously valued control parameters using the max-sum algorithm (2009)
17. Voice, T., Stranders, R., Rogers, A., Jennings, N.: A hybrid continuous max-sum algorithm for decentralised coordination. In: ECAI, pp. 61–66 (2010)
18. Watts, D., Strogatz, S.: Collective dynamics of 'small-world'networks. nature **393**(6684), 440–442 (1998)
19. Xiao, S., Wang, W., Wang, H., Tan, D., Wang, Y., Yu, X., Wu, R.: An improved artificial bee colony algorithm based on elite strategy and dimension learning. Mathematics **7**(3), 289 (2019)
20. Zhang, W., Wang, G., Xing, Z., Wittenburg, L.: Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. Artificial Intelligence **161**(1-2), 55–87 (2005)
21. Zivan, R., Yedidsion, H., Okamoto, S., Glinton, R., Sycara, K.: Distributed constraint optimization for teams of mobile sensing agents. Autonomous Agents and Multi-Agent Systems **29**(3), 495–536 (2015)