ITC 1/53
Information Technology and Control
Vol. 53 / No. 1 / 2024
pp.80-97
DOI 10.5755/j01.itc.53.1.33343

An Estimation of Distribution Based Algorithm for Continuous Distributed Constraint Optimization Problems

Received 2023/02/02 | Accepted after revision 2023/08/13

HOW TO CITE: Shi, M., Zhang, P., Liao, X., Xue, Z. (2024). An Estimation of Distribution Based Algorithm for Continuous Distributed Constraint Optimization Problems. *Information Technology and Control*, 53(1), 80-97. https://doi.org/10.5755/j01.itc.53.1.33343

An Estimation of Distribution Based Algorithm for Continuous Distributed Constraint Optimization Problems

Meifeng Shi

College of Computer Science and Engineering, Chongqing University of Technology, Chongqing, China;
Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan;
e-mail: shimf@cqut.edu.cn

Peng Zhang, Xin Liao, Zhijian Xue

College of Computer Science and Engineering, Chongqing University of Technology, Chongqing, China;
e-mails: zp2024ligong@stu.cqut.edu.cn, lxchat@foxmail.com, blade1565@stu.cqut.edu.cn

Corresponding author: zp2024ligong@stu.cqut.edu.cn

Continuous Distributed Constraint Optimization Problem(C-DCOP) is a constraint processing framework for continuous variables problems in multi-agent systems. There is a constraint cost function between two mutually restrictive agents in C-DCOP. The goal of the C-DCOP solving algorithm is to keep the sum of constraint cost functions in an extreme state. In a C-DCOP, each function is defined by a set of continuous variables. At present, some C-DCOP solving algorithms have been proposed, but there are some common problems such as the limitation of constraints cost function form, easy to fall into local optimum, and lack of anytime attribute. Aiming at these thorny problems, we propose a parallel optimization algorithm named Estimation of Distribution Based Algorithm for Continuous Distributed Constraint Optimization Problems (EDA-CD). In EDA-CD, each solution is regarded as an individual, and the distribution of agent value is jointly described by all outstanding individuals. Firstly, all agents cooperate to hold a distributed population. Secondly, each agent calculates the mean and variance of its variables to build probability models in parallel. Finally, the agent evaluates the fitness of samples and updates the probability model through cooperative communication on Breadth First Search (BFS) pseudo-tree. We theoretically prove that EDA-CD is an anytime algorithm. The extensive experimental results on four types of benchmark problems show that the proposed EDA-CD outperforms the state-of-the-art C-DCOP algorithms and has about 20% improvement in solution quality.

KEYWORDS: Estimation of Distribution Algorithm, C-DCOP, Multi-agent System, Breadth First Search Pseudo-tree.

1. Introduction

Distributed Constraint Optimization Problem (DCOP) is a powerful framework used to model and solve complex multi-agent system (MAS) problems. In a MAS, autonomous agents interact with each other to accomplish individual goals or common objectives. DCOP provides a framework for governing the autonomous behavior of these agents [9]. It has been widely used in practical applications, such as meeting scheduling [6], resource allocation [4], missile path plan [26], sensor networks [7], microgrid control [17] and smart homes [10]. The DCOP algorithms can be simply divided into complete algorithms and incomplete algorithms. The complete algorithms aim to provide a global optimal solution, but the computational and memory overhead is expensive, such as ADOPT [18], DPOP [19], and PTFB [15]. Conversely, the incomplete algorithms obtain an approximate solution by reducing computational and memory overhead, such as DSA [27], Max-Sum [8], MGM [16] and ACO_DCOP [3]. DCOP algorithms can also be classified based on their level of centralization into centralized and fully distributed algorithms. Centralized algorithms can prevent unnecessary conflicts among agents, while fully distributed algorithms offer better information privacy and enable agents to interact and coordinate without relying on a central authority. DCOP algorithms can also be categorized into asynchronous and synchronous algorithms. In asynchronous algorithms, agents make decisions based on their local view without specific messages from neighbors. Although minimizing idle time, there is no consistency in local views, requiring action revision. Synchronous algorithms have a systematic search in defined steps, requiring agents to wait for specific messages. Although synchronous methods increase idle time, agents hold a consistent view, which can be preferable over inconsistent views. DCOP algorithms can be further divided into two categories: search-based methods and inference-based strategies. Search-based methods, such as best-first, depth-first, backtracking, and branch-and-bound, explore the state space through systematic search. In contrast, inference-based strategies allow agents to compute aggregated constraint costs from their neighbors, reducing the problem size at each step and propagating the costs to neighboring agents [14].

However, the variables controlled by the agent are continuous in most practical applications, such as rotation angle, activation, and deactivation time of the sensor. Therefore, C-DCOP was proposed to model continuous problems.

Continuous Max-Sum (CMS) [23] is a continuous version of the discrete Max-Sum algorithm which was proposed to solve DCOP, it was proposed to deal with the changes of variable, domain, and constraint cost function. In CMS, the constraint cost function is approximated as a piecewise linear function. However, the limitation of the constraint cost function form makes CMS only applicable to a few practical problems. Hybrid Continuous Max-Sum algorithm (HCMS) [24] obtains a set of approximate solutions by discrete Max-Sum algorithm and improves the quality of approximate solutions by continuous nonlinear optimization method. Since continuous nonlinear optimization methods such as gradient descent require derivative computations, HCMS is difficult to solve non-differentiable problems and cannot guarantee convergence. B-DPOP [12] extends DPOP algorithm by adding Bayesian optimization and Gaussian process models to solve dynamic coordination problems in continuous domains. It converges to optimal solution in fewer sampling iterations, but has high computational complexity requiring significant computing resources and time for larger problems. Exact Continuous DPOP (EC-DPOP), Approximate Continuous DPOP (AC-DPOP), and Clustered AC-DPOP (CAC-DPOP) [13] were proposed to solve C-DCOP, but they generate exponential computation and memory overhead. Moumita [5] proposed the Particle Swarm based C-DCOP (PFD) to reduce the computational and memory overhead. However, PFD has poor search ability and it is easy to fall into local optimum. Amit [20] proposed a non-iterative algorithm for C-DCOP called Continuous Cooperative Constraint Approximation (C-CoCoA). It obtains higher solution quality through semi-greedy local search. Unfortunately, C-CoCoA lacks the anytime attribute and has poor robustness on complex problems. Jeroen proposed the Distributed Bayesian (D-Bay) algorithm, which solves C-DCOP by utilizing Bayesian optimization for adaptive sampling of variables [11].

Based on the above analysis, we proposed an Estimation of Distribution Based Algorithm for Continuous Distributed Constraint Optimization Problems (EDA-CD) to deal with the limitations of existing C-DCOP algorithms. EDA-CD describes the distribution of the solution from each dimension of the solution and approximates the optimal solution by narrowing the distribution interval of each dimension. In EDA-CD, a solution of C-DCOP is defined as a sample, and the global objective function is defined as the fitness of the sample. It is worth mentioning that the computations of fitness use basic operators. Therefore, the evaluation method of fitness makes EDA-CD not limited to constraints cost function form. Our work can be summarized as follows:

- Design a parallel probability model construction method to satisfy the distributed characteristics. Simultaneously, we sample randomly from the probability model to avoid the algorithm falling into the local optimum.
- Design a rank elitist strategy to guarantee the convergence of the proposed algorithm.
- Design the adaptive population size to improve the robustness of the proposed algorithm.
- Prove that the proposed algorithm is an anytime algorithm.
- Provide experimental comparisons between the proposed algorithm and the state-of-the-art C-DCOP algorithms on four types of benchmark problems.

2. Problem Formulation and Background

2.1. Distributed Constraint Optimization Problem

A C-DCOP can be defined as a 5-tuple $< A, X, D, F, \alpha >$, where:

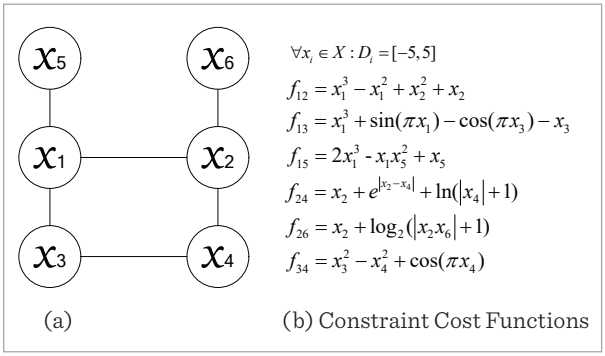
- $A = \{a_1, a_2, ..., a_n\}$ is a set of agents. An agent controls one or more variables.
- $X = \{x_1, x_2, ..., x_m\}$ is a set of continuous variables, where each variable x_i is controlled by agent $a_i \in A$.
- $D = \{D_1, D_2, ..., D_m\}$ is a set of continuous domains. Variable x_i can get any value in $D_i = [LB_i, UB_i]$, where LB_i and UB_i represent the lower and upper bounds of the domain, respectively.
- $F = \{f_1, f_2, ..., f_l\}$ is a set of constraint cost functions. Each $f_i \in F : D_{i1} \times D_{i2} \times ... \times D_{ik} \to R$ specifies the assigned constraint cost function to each combination of $x_{i1}, x_{i2}, ..., x_{ik}$. In this paper, we consider all constraint cost functions are binary.
- $\alpha : X \to A$ is a mapping function to associate each variable $x_j \in X$ to an agent $a_j \in A$. We assume one agent controls only one variable ($n = m$, thus the term "agent" and "variable" could be used interchangeably). Therefore, we use the term agent x_i in the rest of this paper to avoid confusion.

The solution of a C-DCOP is an assignment X^* to all variables that minimizes the sum of all constraint cost functions as shown in Equation 1.

$$X^* = \arg\min_{d_i \in D_i, d_j \in D_j} \sum_{f_{ij} \in F} f_{ij}(d_i, d_j). \quad (1)$$

Figure 1 shows a simple example of C-DCOP, where Figure 1(a) shows a of four variables and each edge represents a constraint cost function defined in Figure 1(b). The domain D_i of x_i is $[-5, 5]$.

Figure 1
An example of C-DCOP



(a) (b) Constraint Cost Functions

2.2. Estimation of Distribution Algorithm

Estimation of Distribution Algorithm (EDA) is a population-based optimization algorithm. In traditional population optimization algorithms, such as genetic algorithms, the population is used to represent a set of candidate solutions to the optimization problem. Each individual in the population has a correspond-

ing fitness value and then performs operations such as selection, crossover, and mutation to simulate natural evolution. However, there are no traditional genetic operations such as crossover and mutation in estimation of distribution algorithm, it updates values by learning and sampling. EDA describes the candidate solution by a probabilistic model. It uses statistical learning methods to establish a probabilistic model describing the solution distribution (Algorithm 1: Lines 3). Each individual calculates its own fitness (Algorithm 1: Lines 5-6). It's sorted by fitness and selects the promising *G* individuals (Algorithm 1: Lines 7-8). Then update the probabilistic model according to the *G* individuals and randomly sample other not selected individuals with a probability model to generate a new population (Algorithm 1: Lines 10-11).

According to the complexity of the probability model and different sampling methods, EDA has developed many different specific implementation methods, but they can all be summarized into two main steps: firstly, randomly generate a lot of solutions, and then select a set of excellent individuals which are used to construct a probability model describing the current solution set using learning. A new population is generated by random sampling from the probability model. Generally, the Monte Carlo method is used to sample the probability model to obtain a new population. The crossover and mutation in the genetic algorithm will destroy the optimized individuals. Genetic algorithm (GA) typically requires complex parameters such as crossover rate, mutation rate, and selection method, while EDA does not, making it easier to use and implement. Additionally, EDA does not require complex crossover and mutation operators to generate new solutions, instead using random sampling and resampling to explore the search space more diversely. EDA models the solution space with a probabilistic model, allowing for more accurate estimation of solution fitness and avoiding limitations in selection operators found in GA. Most importantly, EDA generates new solutions through random sampling and resampling, resulting in a more comprehensive search of the solution space and avoiding premature convergence found in GA.

Algorithm 1: Estimation of Distribution Optimization

1 Generate n -dimensional Samples, S
2 Randomly Initialize variables of each sample
3 calculate the current probabilistic model according to S
4 **while** Termination condition not met **do**
5 **for** each sample $S_k \in S$ **do**
6 calculate the current fitness
7 $S^R \leftarrow Rank(S.fitness)$
8 $S_G^R \leftarrow$ Select the top samples in S^R
9 update the probabilistic model according to S_G^R
10 **if** $S_k \neq S_G^R$
11 update variables according to the probabilistic model

2.3. Breadth First Search Pseudo-tree

BFS pseudo-tree is a commonly used communication structure for DCOP and C-DCOP. The characteristics of BFS pseudo-tree are multi-branch parallel computing, short communication path and time. We briefly introduce the construction, basic terms, notations and concepts of BFS pseudo-tree according to the example in Figure 1. The details of BSF pseudo tree can be found in [2].

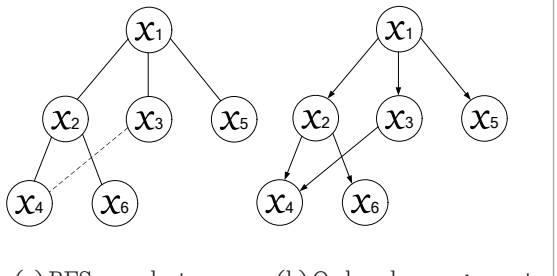
- Firstly, taking agent x_1 as root agent by breadth-first search, and agents x_2, x_3 and x_5 are children of x_1. (Layer: $L_{x_1} = 0$)
- Secondly, the algorithm determines that agent x_4 and x_6 are children of x_2 by traversing the neighbors of x_2. Layer: $L_{x_2} = L_{x_1} + 1 = 1$.
- Next, the algorithm traverses neighbors of x_3 and finds that the x_4 is the child of x_3. Agent x_4 has a Parent (agent x_2), and x_3 is the Pseudo-Parent of x_4. (Layer: $L_{x_3} = L_{x_1} + 1 = 1$).
- Then, the algorithm traverses neighbors of x_5, but the x_5 has no other neighbors except x_1. (Layer: $L_{x_5} = L_{x_1} + 1 = 1$).
- After that, the algorithm traverses x_4.There is a Parent (agent x_2) and a Pseudo-Parent (agent x_3). (Layer: $L_{x_4} = L_{x_2} + 1 = 2$).

Figure 2(a) shows the BFS pseudo-tree constructed by the above steps, the dotted line represents Pseudo-Parent and Pseudo-Child. Solid lines represent Parent and Child. Furthermore, Figure 2(b) shows the ordered BFS pseudo-tree, an agent with a lower layer has higher priority over an agent with a higher

layer. Each agent knows the sets of its higher and lower priority neighbors. In this paper, we use $P_i \subseteq N_i$ and $C_i \subseteq N_i$ to represent the higher and the lower priority neighbor sets of agent x_i, respectively.

In Figure 2(b), agent x_1 is the root agent, and agent x_5, x_4 and x_6 are the leaf agents. The neighbor set of agents x_2 is $N_2 = \{x_1, x_4, x_6\}$, the higher priority neighbor set is $P_2 = \{x_1\}$ and the lower priority neighbor set is $C_2 = \{x_4, x_6\}$.

Figure 2
Pseudo-tree construction (a) and ordered arrangement (b)



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

3. The EDA-CD Algorithm

3.1. Algorithm Introduction

EDA-CD consists of five phases: Initialization, Construction, Evaluation, Update and Sampling. In the Initialization phase, EDA-CD builds a BFS pseudo-tree, initializes parameters and distributed population. In the Construction phase, each agent builds a probability model through the distribution of values. In the Evaluation phase, the agents calculate the fitness of samples in a distributed way. In the Update phase, agents communicate cooperatively to update the probability model. In the Sampling phase, each agent randomly samples from the updated probability model. The specific steps can be found in Algorithm 2.

The **Initialization** Phase. Firstly, EDA-CD constructs an ordered BFS pseudo-tree. Secondly, it initializes three parameters:

- K: The number of samples.
- β: The learning rate, used to update the mean μ and the standard deviation σ.
- G: The number of selected elitist samples.

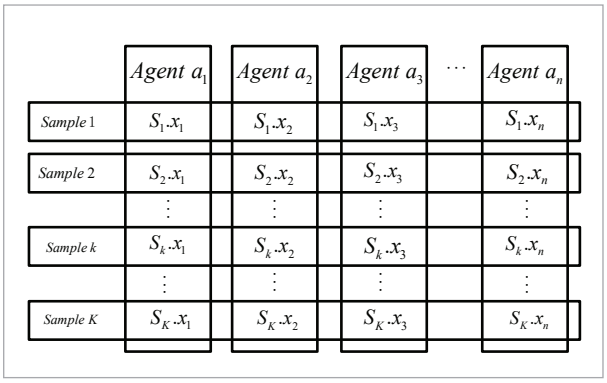
Finally, agents collaboratively generate a distributed population and each agent completes the assignment of its dimension (Algorithm 2: Lines 3-9).

Algorithm 2: The EDA-CD Algorithm

1 Construct BFS pseudo-tree
2 Initialize parameters: K, β, G
3 $S \leftarrow$ Generate K samples
4 **for** each agent x_i **do**
5 **for** each sample $S_k \in S$ **do**
6 $S_k.x_i \leftarrow$ a random value from D_i
7 **end for**
8 send $S.x_i$ to agents in C_i
9 end for
10 **while** Termination condition not met each agent x_i **do**
11 Calculate μ_i^t according to (2)
12 Calculate σ_i^t according to (3)
13 $Messaging(\ )$
14 **if** x_i is a root agent **then**
15 $S^R \leftarrow Rank(S.fitness)$
16 $S_{\_1} \leftarrow Worst(S^R.fitness)$
17 $S_G^R \leftarrow$ Select the top G samples in S^R
18 Send S_G^R and $S_{\_1}$ to agents in C_i
19 **end if**
20 wait until S_G^R and $S_{\_1}$ are received from P_i
21 **if** S_G^R and $S_{\_1}$ are received from P_i **then**
22 Calculate μ_i^{t+1} according to (9)
23 Calculate σ_i^{t+1} according to (10)
24 $S_{|K-G|}.x_i \leftarrow$ Sampling randomly $|K-G|$
25 $S.x_i \leftarrow S_G^R.x_i \cup S_{|K-G|}.x_i$
26 **if** $|C_i| \neq 0$ **then**
27 Send $S.x_i$ to agents in C_i
28 Send S_G^R and $S_{\_1}$ to agents in C_i
29 **end if**
30 **end if**
31 **end while**

Figure 3 shows a distributed population of n agents and K samples. A sample represents a solution of C-DCOP, agent x_i holds one dimension of each sample, S represents a set of samples, named population. We use $S_k.x_n$ to represent the value of the sample k under the dimension of the agent x_n.

Figure 3
Distributed population



The **Construction** Phase. Agents build probability models in parallel at this phase. Specifically, each agent calculates the mean and standard deviation according to Equations 2-3, respectively, (Algorithm 2: Lines 11-12). t represents the number of iterations. Therefore, the probability model of agent x_i is $N\left(\mu_i^t, \left(\sigma_i^t\right)^2\right)$.

$$\mu_i^t = \left(\sum_{k=1}^{K} S_k.x_i\right) / K, \tag{2}$$

$$\sigma_i^t = \sqrt{\left(\sum_{k=1}^{K} (S_k.x_i - \mu_i^t)^2\right) / K}, \tag{3}$$

The **Evaluation** Phase can be described as five steps:

1 When agent x_i receives the value $S.x_j$ sent by the higher priority neighbor $P_{i_j} \in P_i$, agent x_i calculates the partial fitness $S_k.fitness(x_i, x_j)$ with each higher priority neighbor $P_{i_j} \in P_i$ by Equation 4. In addition, agent x_i sends $S_k.fitness(x_i, x_j)$ to the higher neighbor $P_{i_j} \in P_i$ (Procedure 1: Lines 1-6).

$$S_k.fitness\left(x_i, x_j\right) = f_{ij}\left(S_k.x_i, S_k.x_j\right). \tag{4}$$

2 When agent x_i receives the partial fitness sent by the lower priority neighbors $x_j \in C_i$, the agent x_i sums the partial fitness of all the lower priority neighbors as $S_k.fitness[x_i]$ by Equation 5 (Procedure 1: Lines 7-11).

$$S_k.fitness\left(x_i, x_j\right) = f_{ij}\left(S_k.x_i, S_k.x_j\right). \tag{5}$$

3 If the current agent is not the root agent, $S_k.fitness[x_i]$ is sent by the agent x_i to a higher priority neighbor $P_{i_j} \in P_i$ (Procedure 1: Lines 12-14). If the agent x_j is not the root agent, x_j will continue to send until its higher priority neighbor is the root agent.

4 When agent x_j calculates and sends the $S_k.fitness[x_j]$ to the agent in $H_{ji} \in H_j$, all partial fitness of samples are passed to the root agent x_{root}. Therefore, the root agent x_{root} can calculate the complete fitness of each sample by Equation 6.

$$S_k.fitness = \sum_{j \in C_{root}} S_k.fitness[x_j] \tag{6}$$

5 After the root agent obtains the complete fitness of samples, the x_{root} ranks each sample according to the fitness value and is denoted as S^R. In addition,

Procedure 1: *Messaging*( )

```
1  for S.x_j received from P_{i_j} ∈ P do
2   for each sample S_k ∈ S do
3     S_k.fitness(x_i, x_j) ← Cost_ij(S_k.X_i, S_k.X_j)
4   end for
5   Send S.fitness(x_i, x_j) to agent in P_{i_j}
6  end for
7  wait until S.fitness(x_i, x_j) received from agents in C_i
8  if C_i ≠ 0 and S.fitness(x_i, x_j) received from agents in C_i then
9   for each sample S_k ∈ S do
10    S_k.fitness[x_i] ← Σ_{j∈C_i} S.fitness(x_i, x_j)
11  end for
12  if x_i ≠ root then
13   Send S.fitness[x_i] to an P_{i_j} ∈ P_i
14  end for
15 end if
```

the root agent sends the top G samples $\left(S_G^R\right)$ and the worst sample $\left(S_{\_1}\right)$ to the lower priority neighbors C_{root} (Algorithm 2: Lines 14-19). We use the examples in Figure 1 and Figure 2 to explain the above five steps. We use f_{ij} to represent the partial fitness $S_k.fitness\left(x_i, x_j\right)$ between x_i and x_j.

Each agent x_i calculates the partial fitness f_{ij} with each higher priority neighbor $P_{i_j} \in P_i$ and sends f_{ij} to it.

$x_2 : f_{12} \rightarrow x_1$

$x_3 : f_{13} \rightarrow x_1$

$x_4 : f_{24} \rightarrow x_2, f_{34} \rightarrow x_3$

$x_5 : f_{15} \rightarrow x_1$

$x_6 : f_{26} \rightarrow x_2$

Agent x_i sums the partial fitness of lower priority neighbors.

$x_1 : f_{12} + f_{13} + f_{15}$

$x_2 : f_{24} + f_{26}$

$x_3 : f_{34}$

Since agents x_2 and x_3 are not the root agent.

$x_2 : f_{24} + f_{26} \rightarrow x_1$

$x_3 : f_{34} \rightarrow x_1$

Root agent x_1 obtains the completes fitness.

$$x_1 : f_{12} + f_{13} + f_{15} + f_{24} + f_{26} + f_{34} = \sum_{f_{ij} \in F} f_{ij}$$

The **Update** Phase. When the agent receives S_G^R and $S_{\_1}$, x_i calculates the mean and standard deviation of S_G^R by Equations 7- 8, respectively.

$$S_G^{sort}.\overline{x}_i = \left(\sum_{g=1}^{G} S_G^{sort}.x_i\right) / G, \tag{7}$$

$$S_G^R.x_i\left(\sigma\right) = \sqrt{\left(\sum_{g=1}^{G}\left(S_G^{sort}.x_i - S_G^{sort}.\overline{x}_i\right)^2\right) / G}, \tag{8}$$

Then, agent x_i updates the mean and standard deviation [21] of the probability model through by Equations 9-10, where $S_1.x_i$ and $S_2.x_i$ are the best and sub-best samples under the dimension of x_i, respectively (Algorithm 2: Lines 22-23). The updated probability model is $N\left(\mu_i^{t+1}, \left(\sigma_i^{t+1}\right)^2\right)$. β stands for the learning rate.

$$\mu_i^{t+1} = \left(1-\beta\right) * \mu_i^t + \beta * \left(S_1.x_i + S_2.x_i - S_{\_1}.x_i\right), \tag{9}$$

$$\sigma_i^{t+1} = \left(1-\beta\right) * \sigma_i^t + \beta * S_G^R * x_i\left(\sigma\right), \tag{10}$$

The **Sampling** Phase. In the final phase of EDA-CD, agent x_i is randomly sampled from the updated probability model. To maintain population balance, the sampling number is $\left|K-G\right|$. Finally, x_i merges elitist samples $S_G^R.x_i$ and random samples $S_{|K-G|}.x_i$ under one and it as one dimension of a new population (Algorithm 2: Lines 24-25).

3.2. An Example for EDA-CD

In this section, we will use an example to specifically describe the process of EDA-CD. Without considering the influence of the value selection of K and G on solution quality, we set the domain of each agent x_i to $\left[-5,5\right]$, $\beta = 0.01$ $K = 3$ and $G = 6$. It should be noted that in each calculation, we only keep two digits after the decimal point.

Taking Figure 2 as an example, we first **initialize** the parameters of all samples. Table 1 shows the specific values.

Value propagation:

$$X_1.S = \left\{\begin{matrix} X_1.S_1 = -2.60, X_1.S_2 = -3.20, X_1.S_3 = -1.90 \\ X_1.S_4 = -4.00, X_1.S_5 = -3.50, X_1.S_6 = -2.70 \end{matrix}\right\}$$

$$X_2.S = \left\{\begin{matrix} X_2.S_1 = 2.70, X_2.S_2 = 3.20, X_2.S_3 = -2.70 \\ X_2.S_4 = -2.80, X_2.S_5 = 4.20, X_2.S_6 = -4.60 \end{matrix}\right\}$$

$$X_3.S = \left\{\begin{matrix} X_3.S_1 = -4.80, X_3.S_2 = -2.40, X_3.S_3 = -1.00 \\ X_3.S_4 = -0.70, X_3.S_5 = 3.70, X_3.S_6 = 3.60 \end{matrix}\right\}$$

$$X_4.S = \left\{\begin{matrix} X_4.S_1 = -3.40, X_4.S_2 = 1.80, X_4.S_3 = 2.30 \\ X_4.S_4 = -4.10, X_4.S_5 = -4.80, X_4.S_6 = -1.70 \end{matrix}\right\}$$

$$X_5.S = \left\{\begin{matrix} X_5.S_1 = -4.60, X_5.S_2 = -3.00, X_5.S_3 = 1.30 \\ X_5.S_4 = -4.10, X_5.S_5 = -3.90, X_5.S_6 = -1.90 \end{matrix}\right\}$$

$$X_6.S = \left\{\begin{matrix} X_6.S_1 = -2.60, X_6.S_2 = 0.80, X_6.S_3 = 2.80 \\ X_6.S_4 = 3.70, X_6.S_5 = 4.80, X_6.S_6 = -3.40 \end{matrix}\right\}$$

$X_1.S \rightarrow X_2, X_3, X_5$

$X_2.S \rightarrow X_4, X_6$

$X_3.S \rightarrow X_4$

Evaluation

Calculate the mean and standard deviation in each agent:

$$\mu_i^t = \left\{\begin{matrix} \mu_1^t = -2.98, \mu_2^t = 0.00, \mu_3^t = -0.27 \\ \mu_4^t = -1.65, \mu_5^t = -2.70, \mu_6^t = 1.02 \end{matrix}\right\}$$

Table 1
Initialize values of each agent

	Agent a_1	*Agent* a_2	*Agent* a_2	*Agent* a_4	*Agent* a_5	*Agent* a_6
Sample 1	-2.60	2.70	-4.80	-3.40	-4.60	-2.60
Sample 2	-3.20	3.20	-2.40	1.80	-3.00	0.80
Sample 3	-1.90	-2.70	-1.00	2.30	1.30	2.80
Sample 4	-4.00	-2.80	-0.70	-4.10	-4.10	3.70
Sample 5	-3.50	4.20	3.70	-4.80	-3.90	4.80
Sample 6	-2.70	-4.60	3.60	-1.70	-1.90	-3.40

$$\sigma_i^t = \begin{Bmatrix} \sigma_1^t = 0.68, \sigma_2^t = 3.45, \sigma_3^t = 3.07 \\ \sigma_4^t = 2.78, \sigma_5^t = 1.99, \sigma_6^t = 3.09 \end{Bmatrix}$$

Calculate the fitness value between each other:

$$f(S.x_1, S.x_2) = \begin{Bmatrix} -14.35, -29.57, -5.88 \\ -69.36, -33.28, -10.41 \end{Bmatrix}$$

$$f(S.x_1, S.x_3) = \begin{Bmatrix} -12.92, -30.09, -4.55 \\ -64.11, -46.16, -24.40 \end{Bmatrix}$$

$$f(S.x_1, S.x_5) = \begin{Bmatrix} 15.26, -39.74, -9.21 \\ -56.66, -36.41, -31.52 \end{Bmatrix}$$

$$f(S.x_2, S.x_4) = \begin{Bmatrix} 450.04, 8.28, 146.91 \\ 8.10, 8109.04, 14.57 \end{Bmatrix}$$

$$f(S.x_2, S.x_6) = \begin{Bmatrix} 5.70, 5.03, 0.40 \\ 4.37, 8.60, -0.54 \end{Bmatrix}$$

$$f(S.x_3, S.x_4) = \begin{Bmatrix} 11.17, 3.33, -3.70 \\ -15.37, -10.16, 10.66 \end{Bmatrix}$$

The fitness value is propagated to the high priority:

$$f(S.x_3, S.x_4) \to X_3$$
$$f(S.x_2, S.x_6) \to X_2$$
$$f(S.x_2, S.x_4) \to X_2$$
$$f(S.x_1, S.x_5) \to X_1$$
$$f(S.x_1, S.x_3) + f(S.x_3, S.x_4) \to X_1$$
$$f(S.x_1, S.x_2) + f(S.x_2, S.x_6) + f(S.x_2, S.x_4) \to X_1$$

The total fitness value of each sample:

$$S.X.fitness = \begin{Bmatrix} 454.90, -82.76, 123.97 \\ -193.03, 7991.63, -41.64 \end{Bmatrix}$$

Sort the fitness values in ascending order:

$$Rank(S.fitness) = \begin{Bmatrix} -193.03 < -82.76 < -41.64 \\ < 123.97 < 454.90 < 7991.63 \end{Bmatrix}$$

The root agent sends the top G samples $\left(S_G^R\right)$ and the worst sample $\left(S_{\_1}\right)$ to the lower priority neighbors.

C_{root} :

$$S_G^R = \{S_4, S_2, S_6\} = \{-193.03, -82.76, -41.64\}$$
$$S_{\_1} = 7991.63$$
$$S_{\_1}, S_G^R \to C_i$$

Update

Each agent x_i calculates the new mean and standard deviation:

$$S_G^R.\overline{x}_i = \{-3.30, -1.40, 0.17, -1.33, -3.00, 0.37\}$$
$$S_G^R.x_i(\sigma) = \{0.54, 3.33, 2.52, 2.42, 0.90, 2.91\}$$

Then, agent x_i updates the mean and standard deviation of the probability model

$$\mu_i^{t+1} = \{-2.99, -0.04, -0.34, -1.61, -2.65, 1.01\}$$
$$\sigma_i^{t+1} = \{0.68, 0.71, 0.70, 0.70, 0.68, 0.70\}$$

The remaining unselected samples update their values $\left(S_{|K-G|}.x_i\right)$ according to the probability model

$$S_{|K-G|}.x_i = \{S_1.x_i, S_3.x_i, S_5.x_i\}$$

$$S_1.X = \begin{Bmatrix} x_1 = -3.23, x_2 = -1.22, x_3 = 0.77 \\ x_4 = -0.73, x_5 = -3.76, x_6 = 1.61 \end{Bmatrix}$$

$$S_3.X = \begin{Bmatrix} x_1 = -2.86, x_2 = -0.16, x_3 = -0.97 \\ x_4 = -0.79, x_5 = -2.7, x_6 = 0.64 \end{Bmatrix}$$

$$S_5.X = \begin{Bmatrix} x_1 = -3.90, x_2 = 1.23, x_3 = 0.58 \\ x_4 = -1.55, x_5 = -1.5, x_6 = 1.40 \end{Bmatrix}$$

Table 2
Update values of each agent

	Agent a_1	*Agent* a_2	*Agent* a_2	*Agent* a_4	*Agent* a_5	*Agent* a_6
Sample 1	-3.23	-1.22	0.77	-0.73	-3.76	1.61
Sample 2	-3.20	3.20	-2.40	1.80	-3.00	0.80
Sample 3	-2.86	-0.16	-0.97	-0.79	-2.70	0.64
Sample 4	-4.00	-2.80	-0.70	-4.10	-4.10	3.70
Sample 5	-3.90	1.23	0.58	-1.55	-1.50	1.40
Sample 6	-2.70	-4.60	3.60	-1.70	-1.90	-3.40

4. Algorithm Theoretical Analysis

In this section, we define the communication step as CS, which represents the times that an agent communicates with one of its neighbors. In addition, we define the depth of BFS pseudo-tree as D, and the longest path as L. The optimal solution at each iteration is defined as S_{best}.

4.1. Theoretical Proof

Theorem 1. **When** $CS = T + L + D$**, all agents obtain** S_{best} **at** $CS = T$.

Proof of Theorem 1. S_{best} is the sample that has the lowest fitness. In order to calculate the complete fitness of each sample, the root agent needs to wait at the most L communication steps since the longest path of BFS pseudo-tree is L. When the root agent obtains S_{best}, it needs to be passed down to each agent through the BFS pseudo-tree. Since the depth of the pseudo-tree is D, each agent needs to wait at most $T + L + D$, and all agents obtain S_{best} at $CS = T$.

Proposition 1. EDA-CD is an anytime algorithm.

Proof of Proposition 1. When $CS = T + L + D + \delta(\delta > 0)$, each agent obtains S_{best} at $CS = T + \delta$. According to the proposed rank elitist strategy, the optimal solution is retained in the next iteration. S_{best} is updated only after a better solution is searched, only after at $CS = T + L + D + \delta$ will not be worse than $CS = T + \delta$. The solution quality does not decrease over time, and EDA-CD can provide S_{best} at any time. Hence, EDA-CD is an anytime algorithm.

4.2. Complexity Analysis

In this section, we define some parameters as follows:

- K: The number of samples.
- n: The number of agents.
- $|P|$ and $|C|$: The number of higher and lower priority neighbors, respectively. $|N| = |P| + |C|$. Furthermore, we assume the as a complete graph, $|N| \approx n$. A complete iteration is defined as the entire process of construction, evaluation, updating and sampling phase.

The Number of Messages:

- Initialization and sampling phase: Agent x_i sends $S.x_i$ to each lower priority neighbor C_{i_j}, $O(2*|C|)$.
- Evaluation phase: Agent x_i sends $S.fitness(x_i, x_j)$ each higher priority neighbor P_{i_j}, $O(|P|)$. In addition, agent x_i sends $S.fitness[x_j]$ received from $C_{i_{j1}}$ to $P_{i_{j2}}$ once, $O(|1|)$.

The number of messages sent by an agent during an iteration is $O(2*|C| + |P| + 1) = O(|N| + |C| + 1)$. In the worst case, all neighbors are lower priority neighbors, the number of messages by an agent during an iteration is $O(2*n+1) = O(n)$.

The Size of Messages:

In EDA-CD, an agent sends the following four types of messages to neighbors:

- $S.x_i$ and: Containing K samples, $O(2*K*n)$.
- S_G^R: Containing G samples, $O(G*n)$.
- $S_{\_1}$: Containing one sample, $O(1*n)$.

Hence, the size of messages sent by an agent during an iteration is. $O(2*K*n+G*n+1*n)=O(K*n)$.

Computational Complexity

During an iteration, an agent performs the following computation:

- $S.fitness(x_i,x_j)$ with K samples: $O(K*n)$.
- μ_i^t and σ_i^t: $O(2)$.
- μ_i^{t+1} and σ_i^{t+1}: $O(2)$.
- $S_G^R.\overline{x}_i$ and $S_G^R.x_i(\sigma)$: $O(2)$.

Therefore, the computational complexity of an agent is $O(K*n+2*2+2)=O(K*n)$.

5. Experimental Result and Analysis

We verify the performance of the proposed EDA-CD by comparing it with the state-of-the-art C-DCOP solving algorithms on four types of benchmark problems. Although EDA-CD can use any form of function as the constraint cost function, we follow the constrained cost function form $ax^2+bx+cxy+dy+ey^2+f$ in [13] to better show the experimental comparison results, where a,b,c,d,e and f are random numbers in the range $[-5,5]$. We set the domain of each agent x_i to $[-50, 50]$ and the iteration to 500. For each experimental configuration, we independently run the algorithm 30 times and take the average as the experimental result. The number of agents was defined as n in this section.

5.1. Benchmark Problem

- *Random graphs*: We provide two random graph configurations, sparse (density 0.1) and dense (density 0.6), where density represents the probability of any two nodes connected during the construction. We set n from 10 to 100 and the interval to 10.
- *Random trees*: We use random trees as a benchmark problem according to [5]. Firstly, we randomly select a number as the root node from the continuous integer array $[1,n]$ and delete the number. Then, we start breadth first search from the root node, randomly select $m(m\in[1,6])$ subnodes and delete their number. Finally, we repeat the random selection until the length of the array is zero. n is set from 50 to 100 and the interval is 5.
- *Scale-free networks*: We use BA model [1] to generate scale-free networks problems. Firstly, the model generates a connected network of 15 agents. Then, a new agent is connected to the 7 agents in the current network during each iteration. Finally, new agents are added repeatedly until all agents join the network. We set n from 50 to 100 and the interval to 5.
- *Small-world networks*: We use [25] topological model to generate small-world networks. A ring nearest-neighbor coupled network with n nodes, each node is connected to the nearest 3 nodes on both sides. A node and an edge connected are chosen to connect to a random node on the ring with probability p = 0.5 there can be no multi-edges or loops. We set n from 50 to 100 and the interval to 5.

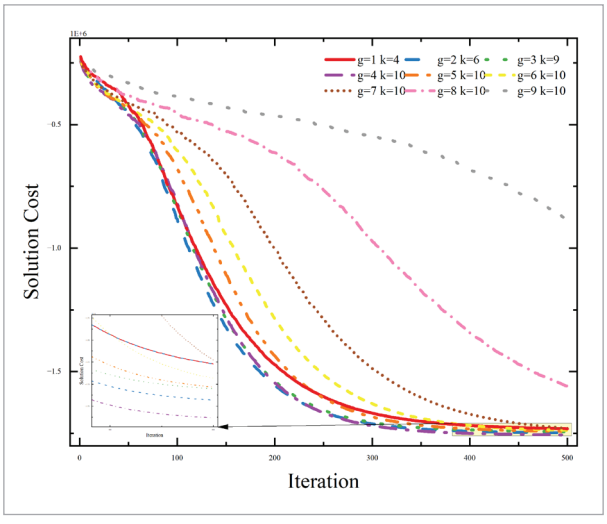
5.2. Fine-tuning Parameters

EDA-CD is a population-based algorithm that has several parameters including the number of samples K, learning rate β, and elitist samples G. According to other C-DCOP algorithms, we generally set the learning rate to 0.01. For the determination of K and G, we set two adaptive values (k,g) and $K=k*n$, $G=g*n$ (n represents the number of agents). Their values range from 1-10 and 1-9, and we determine them by experiments on sparse random graphs.

In this experiment, we divide the value of g into 10 groups (1-9), and each group is matched with k (1-10) values respectively. In each of the groups, we select the solution which quality is the best for comparison in Figure 4.

Figure 4

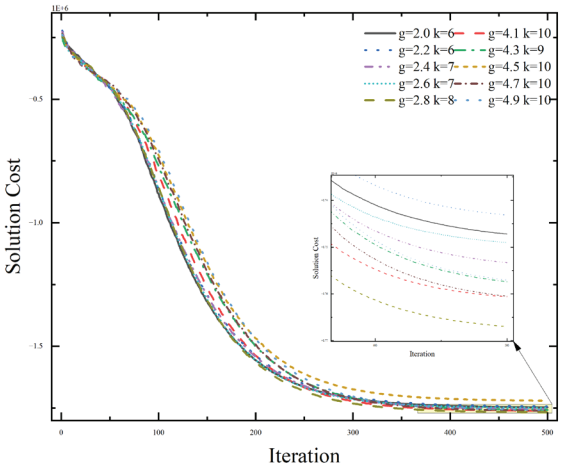
Solution quality of EDA-CD with different adaptive values g and k on sparse random graphs



In Figure 4, We can see that the quality of the solution is the best when g is equal to 4, the solution converges the fastest when g is equal to 2, and the quality of the solution is the second best. Thus, we conducted a set of further experiments. As in the previous experiment, we divided the value of g ($g=2$ and $g=4$) into 10 groups, each g matching 10 groups of k values. In Figure 5, we can see that the solution converges quickly and the quality of the solution is best when $g=2.8$ and $k=8$ from the partial magnification.

Figure 5

Solution quality of EDA-CD with adaptive values $g=2$ and $g=4$ on sparse random graphs



5.3. Experimental Result

We evaluate EDA-CD and its competing algorithms (HCMS, PFD, PFD-LD and C-CoCoA) and the parameters of each algorithm are set as follows:

- HCMS: Based on [13] the number of discrete points is set to 3 and the step size of gradient descent is set to 0.001.
- PFD: According to [5] $w=0.9$, $K=2000$, $c_1=0.9$, $c_2=0.1$, $\max_{sc}=15$ and $\max_{fc}=5$.
- C-CoCoA: Refer to [20] the number of discrete points is set to 3, the step size of gradient descent is set to 0.01 and the number of optimizations is set to 100.
- PFD-LD: According to [22] $w=0.9$, $K=10*n$, $c_1=0.9$, $c_2=0.1$, $\beta=0.1$, $\alpha=0.1$, $\max_{sc}=15$ and $\max_{fc}=5$.
- EDA-CD: $\beta=0.01$, $G=2.8*n$ and $K=2.8*n$.

Due to the limitation that D-Bay can only solve problems with 10 agents, only the results of D-Bay with 10 agents are presented in Figures 6-7. Figure 6(a) shows the solution quality of EDA-CD and competing algorithms on sparse random graphs. It can be seen from Figure 6(a) that the solution quality of EDA-CD is superior to all the competing algorithms in different problem scales. Figure 6(b) shows the Convergence curve. C-CoCoA converges the fastest for it is a non-iterative algorithm, but its quality is not the best. As the number of iterations increases, the solution quality of EDA-CD is better than other algorithms in the end. Although EDA-CD has a slower convergence speed, it has the best quality solution among all algorithms.

Figure 6

Comparison of EDA-CD and its competing algorithms on sparse random graphs

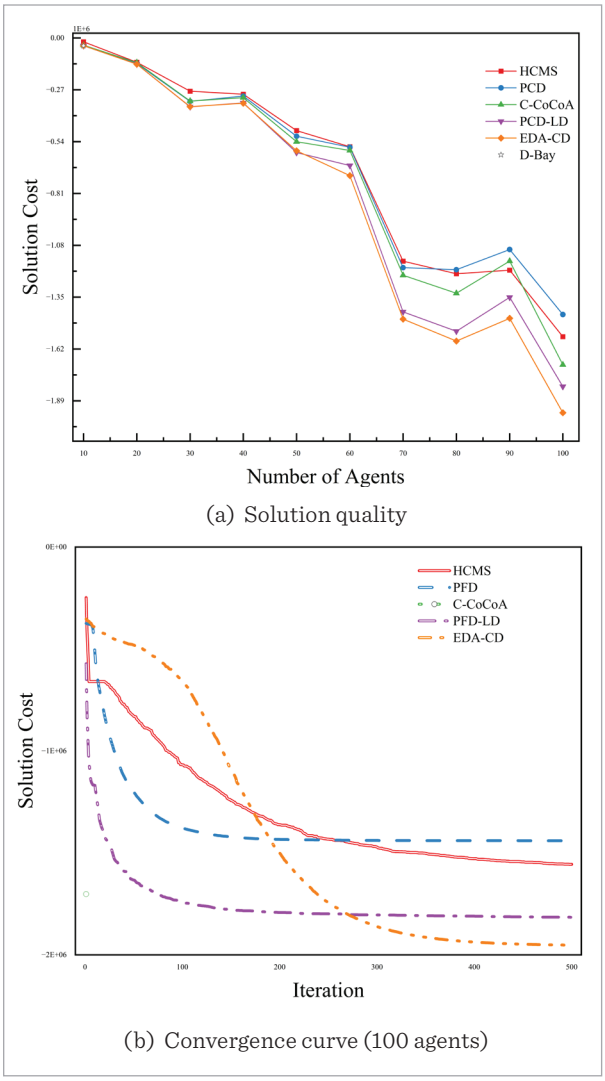
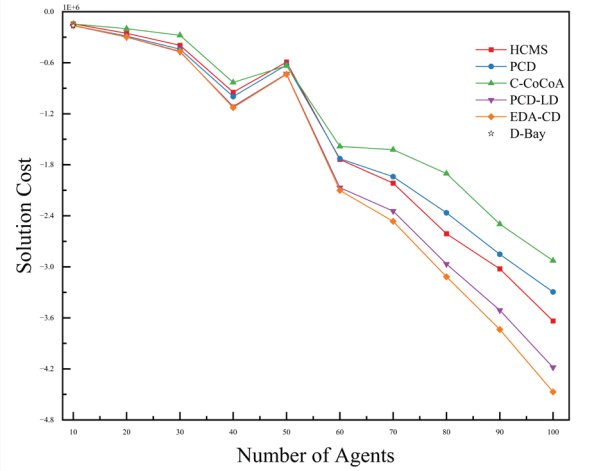


Figure 7 shows the solution quality of EDA-CD and competing algorithms on dense random graphs. Figure 7(a) exhibits that the solution quality of the EDA-CD is better than the other four algorithms under the different number of agents. It can be seen from Figure 7(b) that C-CoCoA is the fastest algorithm to solve the problem, but its solution quality is not good. As the number of iterations increases, HCMS and PFD begin to converge and keep the current optimal solution. After that, EDA-CD continues to optimize and converge to a higher solution quality until the end of the iteration. The convergence speed of EDA-CD is slower than competing algorithms since the sample space of the probability model is large in the early. However, the sample space is constantly approaching the current excellent solution with the update of the probability model, the updated samples are distributed around the optimal solution. Therefore, the solution quality improves over time.

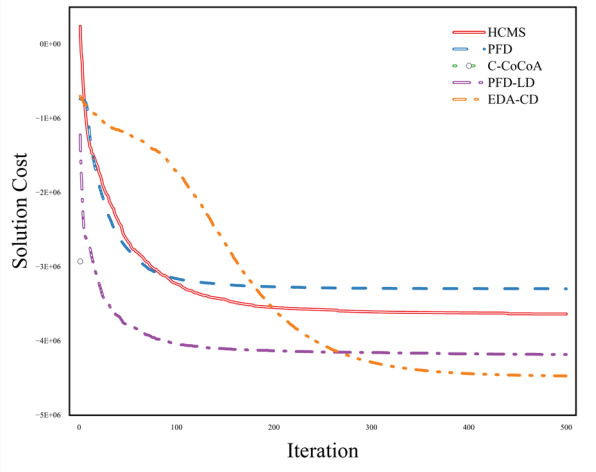
In Figure 8 we can see that the solution quality of EDA-CD is significantly better than HCMS, and PFD. However, the solution quality of EDA-CD is slightly better than C-CoCoA and PFD-LD. The reason is that the nodes in the random trees are less connected and the topological relationship is simple. The semi-greedy local search strategy in C-CoCoA and with local search strategy in PFD-LD perform well.

Figure 7

Comparison of EDA-CD and its competing algorithms on dense random graphs



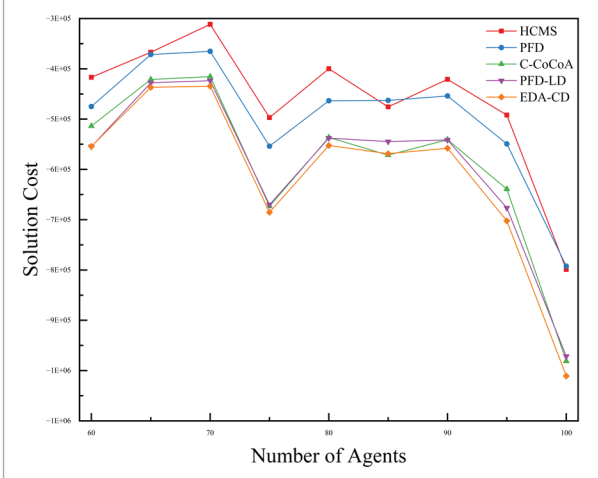
(a) Solution quality



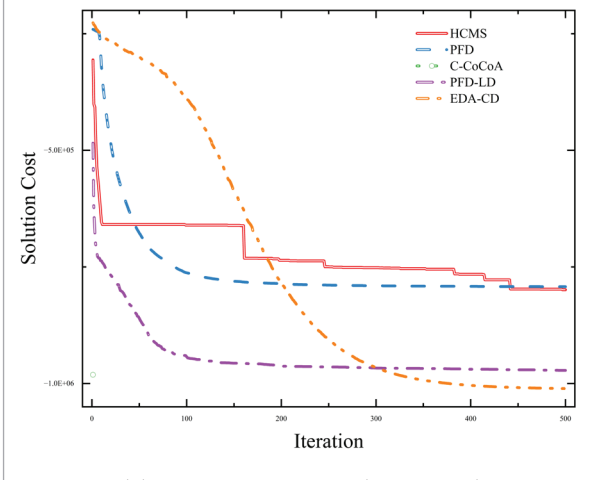
(b) Convergence curve (100 agents)

Figure 8

Comparison of EDA-CD and its competing algorithms on random trees



a) Solution quality

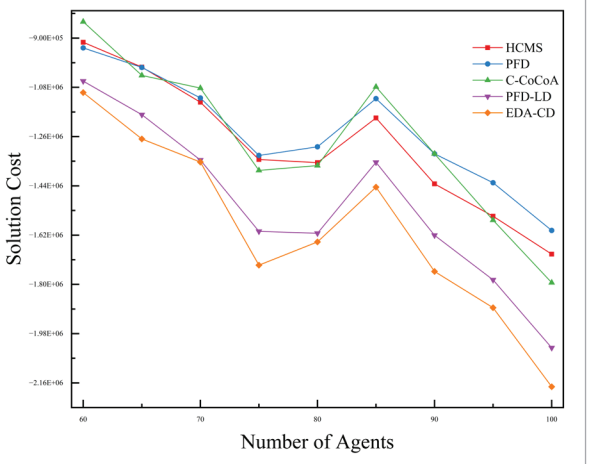


(b) Convergence curve (100 agents)

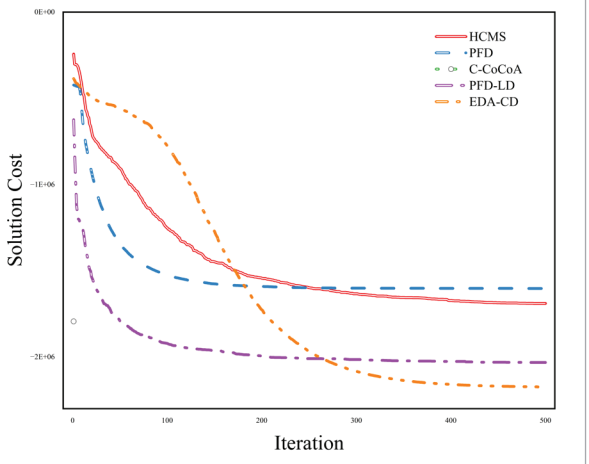
In addition, the solution quality gap between EDA-CD and competing algorithms is more obvious with the increase in the number of agents. Figure 9 presents the solution quality of these four algorithms on scale-free networks. It is obvious that the solving performance of EDA-CD is excellent and the solution quality of EDA-CD is superior to HCMS, PFD, PFD-LD and C-CoCoA on all quantity configurations. Due to the topological relationship of scale-free networks being more complex than random trees. C-CoCoA performs worse than others.

Figure 9

Comparison of EDA-CD and its competing algorithms on scale-free networks



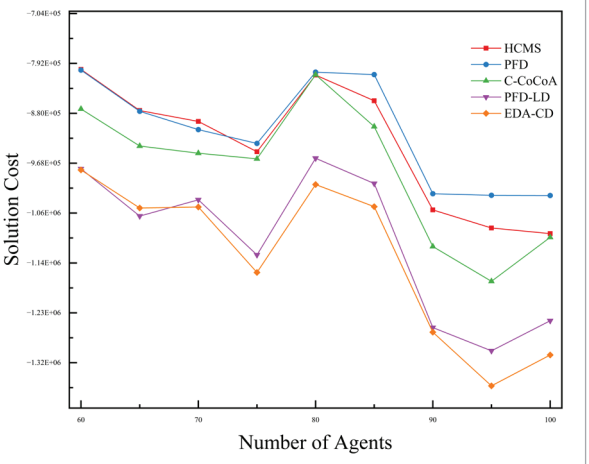
(a) Solution quality



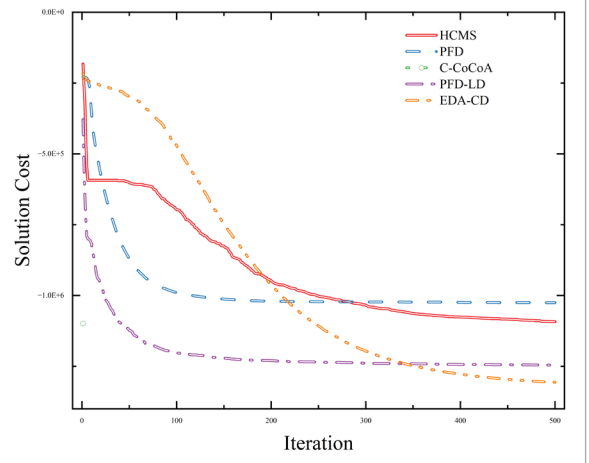
(b) Convergence curve (100 agents)

Figure 10

Comparison of EDA-CD and its competing algorithms on small-world networks



(a) Solution quality



(b) Convergence curve (100 agents)

Figure 10 exhibits the solution quality of EDA-CD and competing algorithms on small-world networks. We can see that the solution quality of EDA-CD is better than the counterparts on all quantity configurations.

Table 3 displays the CPU running time percentages of EDA-CD compared to four competing algorithms on different benchmark problems, with the percentages rounded to two decimal places. It can be seen that EDA-CD is slightly inferior to other algorithms on sparse random graphs, but significantly faster in solving dense random graphs than the other algorithm.

Table 3
The average CPU runtime improvement rates of EDA-CD compared to four competing algorithms

Type of problem	HCMS	PFD	C-CoCoA	PFD-LD
Sparse random graphs	-4.12%	-8.10%	-9.76%	-3.78%
Dense random graphs	**19.73%**	**16.49%**	**21.88%**	**20.67%**
Random trees	**1.74%**	**0.70%**	-7.55%	**8.29%**
Scale-free networks	**3.46%**	**4.76%**	**8.94%**	**9.14%**
Small-world networks	**-2.98%**	-0.12%	-5.94%	**3.76%**

The above experiments indicate that the EDA-CD algorithm has good stability and produces solutions with small fluctuations in quality when solving complex problems. This algorithm is suitable for various types of continuous distributed constraint optimization problems, and can adapt flexibly to different cost function forms. Additionally, by changing the sample generation strategy and parameters, the performance and robustness of the EDA-CD algorithm can be further improved.

6. Statistical Analysis

In order to effectively illustrate the significant superiority of EDA-CD, we use the Wilcoxon signed rank test to analyze 30 independent experiments for each experiment configuration. The steps are as follows:

- We set the significance level $\alpha_{sl} = 0.05$.
- We define the solution quality of EDA-CD and competing algorithms as S_p and S_c. The difference Δ is defined as $\Delta = S_p - S_c$.
- If $\Delta < 0$, $R^+ = R^+ + 1$, $w^+ = w^+ + rank(\Delta)$.
- If $\Delta > 0$, $R^- = R^- + 1$, $w^- = w^- + rank(\Delta)$.
- We determine the value of p based on the distribution of rank sum.

If $p < \alpha_{sl}$, there is a significant difference between EDA-CD and competing algorithms.

If $p > \alpha_{sl}$, there is no significant difference between EDA-CD and competing algorithms.

After statistical analysis of the data, the performance of EDA-CD and its competing algorithms on random graphs is shown in Table 4. We can see that EDA-CD significantly outperforms HCMS and C-CoCoA on all numbers of agents. In the case of a small number of agents, EDA-CD not obviously outperforms PFD and PFD-LD since EDA-CD requires a large number of samples to gain a probability model.

Table 5 is the statistical results on the random tree. C-CoCoA is a semi-greedy search algorithm and the random tree communication structure is simple, so EDA-CD is not obviously excellent in simple problems. EDA-CD is significantly better than other algorithms when they face the same problem.

Tables 6-7 are the statistical results of EDA-CD and the competing algorithms on the scale-free network and the small-world network, respectively. We can see that EDA-CD is significantly better than HCMS and PFD on scale-free networks, and it is significantly better than HCMS, PFD and C-CoCoA on small-world networks.

Table 8 presents the average improvement rates of EDA-CD compared to four competing algorithms on different benchmark problems. We use the improvement rate of each quantity configuration on the benchmark problems to calculate the average improvement rates, and the results are rounded to two decimal places.

It can be seen from Table 8 that EDA-CD is superior to other competitive algorithms and has a good average improvement rate on four benchmark problems. Although C-CoCoA has good performance on the random trees, EDA-CD still has 1.96% average improvement rate compared with C-CoCoA in solution quality. As for the latest PFD-LD, the performance improvement rate of EDA-CD is also at least 5.12%.

Table 4
Wilcoxon signed ranks test results of convergence quality with a level of significance α=0.05 on random graphs

Agent		EDA-CD *vs* HCMS			EDA-CD *vs* PFD			EDA-CD *vs* C-CoCoA			EDA-CD *vs* PFD-LD		
		R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p
10	S	30/0	465/0	0.000	10/17	140/237	0.244	25/5	381/84	0.002	11/19	171/294	0.206
	D	28/2	458/7	**0.000**	10/20	135/330	0.045	29/1	460/5	**0.000**	12/18	95/370	**0.005**
20	S	30/0	465/0	**0.000**	22/8	383/82	**0.002**	17/13	375/90	**0.003**	14/16	213/252	0.688
	D	26/4	439/26	**0.000**	19/10	307/128	0.053	30/0	465/0	**0.000**	19/11	248/217	0.750
30	S	30/0	465/0	**0.000**	27/3	457/8	**0.000**	26/4	454/11	**0.000**	15/15	224/241	0.861
	D	26/4	451/14	**0.000**	17/13	450/15	**0.001**	30/0	465/0	**0.000**	14/16	203/062	0.544
40	S	28/2	458/7	**0.000**	27/3	451/14	**0.000**	23/7	403/62	**0.000**	20/10	257/208	0.614
	D	30/0	465/0	**0.000**	28/2	460/5	**0.000**	30/0	465/0	**0.000**	21/9	303/162	0.147
50	S	30/0	465/0	**0.000**	30/0	465/0	**0.000**	24/6	433/32	**0.000**	10/20	159/306	0.131
	D	30/0	465/0	**0.000**	30/0	465/0	**0.000**	27/3	453/12	**0.000**	20/10	294/171	0.206
60	S	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	26/4	446/19	**0.000**
	D	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	26/4	446/19	**0.000**
70	S	30/0	465/0	**0.000**	30/0	465/0	**0.000**	29/1	464/1	**0.000**	22/8	389/76	**0.001**
	D	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	20/10	387/78	**0.001**
80	S	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	19/11	373/92	**0.004**
	D	29/1	463/2	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	23/7	376/89	**0.003**
90	S	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	27/3	458/7	**0.000**
	D	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	25/5	427/38	**0.000**
100	S	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	28/2	454/11	**0.000**
	D	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	24/6	437/28	**0.000**

Table 5
Wilcoxon signed ranks test results of convergence quality with a level of significance α=0.05 on random tree

Agent	EDA-CD *vs* HCMS			EDA-CD *vs* PFD			EDA-CD *vs* C-CoCoA			EDA-CD *vs* PFD-LD		
	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p
60	30/0	465/0	**0.000**	10/17	140/237	**0.000**	19115	354/111	0.012	16/14	244/221	0.813
65	30/0	465/0	**0.000**	29/1	463/2	**0.000**	22/8	334/131	0.030	19/11	320/145	0.072
70	30/0	465/0	**0.000**	30/0	465/0	**0.000**	26/4	417/48	**0.000**	23/7	403/62	**0.000**
75	30/0	465/0	**0.000**	30/0	465/0	**0.000**	21/9	325/140	0.057	23/7	383/82	**0.002**
80	30/0	465/0	**0.000**	30/0	465/0	**0.000**	19/11	334/136	0.037	24/6	369/960.	**0.005**
85	30/0	465/0	**0.000**	30/0	465/0	**0.000**	12/18	203/262	0.544	24/6	430/35	**0.000**
90	30/0	465/0	**0.000**	30/0	465/0	**0.000**	16/14	308/157	0.120	20/10	360/105	0.009
95	30/0	465/0	**0.000**	30/0	465/0	**0.000**	25/5	445/20	**0.000**	30/0	465/0	**0.000**
100	30/0	465/0	**0.000**	30/0	465/0	**0.000**	16/14	315/150	0.090	27/3	446/19	**0.000**

Table 6
Wilcoxon signed ranks test results of convergence quality with a level of significance α=0.05 on scale-free networks

Agent	EDA-CD *vs* HCMS			EDA-CD *vs* PFD			EDA-CD *vs* C-CoCoA			EDA-CD *vs* PFD-LD		
	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p
60	30/0	465/0	**0.000**	30/0	465/0	**0.000**	19/11	354/111	0.012	16/14	244/221	0.813
65	30/0	465/0	**0.000**	29/1	463/2	**0.000**	22/8	334/131	0.037	19/11	320/145	0.072
70	30/0	465/0	**0.000**	30/0	465/0	**0.000**	26/4	417/48	**0.000**	23/7	403/62	**0.000**
75	30/0	465/0	**0.000**	30/0	465/0	**0.000**	21/9	325/140	0.057	23/7	383/82	**0.002**
80	30/0	465/0	**0.000**	30/0	465/0	**0.000**	19/11	334/131	0.037	24/6	369/96	**0.005**
85	30/0	465/0	**0.000**	30/0	465/0	**0.000**	12/18	203/262	0.544	24/6	430/35	**0.000**
90	30/0	465/0	**0.000**	30/0	465/0	**0.000**	16/14	308/157	0.120	20/10	360/105	0.009
95	30/0	465/0	**0.000**	30/0	465/0	**0.000**	25/5	445/20	**0.000**	30/0	465/0	**0.000**
100	30/0	465/0	**0.000**	30/0	465/0	**0.000**	16/14	315/150	0.090	27/3	446/19	**0.000**

Table 7
Wilcoxon signed ranks test results of convergence quality with a level of significance α=0.05 on small-world networks

Agent	EDA-CD *vs* HCMS			EDA-CD *vs* PFD			EDA-CD *vs* C-CoCoA			EDA-CD *vs* PFD-LD		
	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p	R^+ / R^-	w^+ / w^-	p
60	30/0	465/0	**0.000**	30/0	465/0	**0.000**	24/6	418/48	**0.000**	14/16	240/225	0.877
65	30/0	465/0	**0.000**	30/0	465/0	**0.000**	28/2	457/8	**0.000**	21/9	301/164	0.159
70	30/0	465/0	**0.000**	30/0	465/0	**0.000**	27/3	453/12	**0.000**	17/13	277/188	0.360
75	30/0	465/0	**0.000**	30/0	465/0	**0.000**	29/1	463/2	**0.000**	20/10	340/125	0.027
80	29/1	464/1	**0.000**	30/0	465/0	**0.000**	29/1	464/1	**0.000**	22/8	390/75	**0.001**
85	30/0	465/0	**0.000**	30/0	465/0	**0.000**	29/1	464/1	**0.000**	24/6	387/78	**0.001**
90	30/0	465/0	**0.000**	30/0	465/0	**0.000**	27/3	450/15	**0.001**	18/12	385/80	0.021
95	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	21/9	395/70	**0.001**
100	30/0	465/0	**0.000**	30/0	465/0	**0.000**	30/0	465/0	**0.000**	25/5	431/34	**0.000**

Table 8
The average improvement rates of EDA-CD compared to four competing algorithms

Type of problem	HCMS	PFD	C-CoCoA	PFD-LD
Sparse random graphs	16.98%	15.44%	10.40%	8.40%
Dense random graphs	20.82%	16.57%	36.04%	5.32%
Random trees	24.22%	17.89%	1.96%	5.48%
Scale-free networks	21.76%	25.18%	23.92%	21.40%
Small-world networks	21.15%	21.19%	13.82%	5.12%

7. Conclusion and Future Work

In this paper, EDA-CD is proposed for solving Continuous Distributed Constraint Optimization Problem due to currently C-DCOP solving algorithms are easy to fall into local optimization and the solution quality is poor. In EDA-CD, one solution X^* is multidimensional and each agent completes the assignment of its dimension. To find the best assignment of each dimension, agents construct probability models using elite solutions in parallel to describe the distribution of the current population and then randomly sample the constructed probability models simultaneously to generate offspring solutions, which makes EDA-CD improve the solution quality and avoid falling into the local optimum. EDA-CD is theoretically proven to be an anytime algorithm and extensive experiments demonstrate that EDA-CD is significantly superior to the state-of-art C-DCOP solving algorithms. It is worth noting that EDA-CD provides a novel idea based on the probability model for solving C-DCOP.

While experimental results on four benchmark problems demonstrate EDA-CD's superiority, further evaluation on real-world problems is needed. Future work includes modeling real-world multi-agent systems using the C-DCOP framework and comparing EDA-CD's efficiency with other algorithms in practical problem-solving scenarios.

Acknowledgeent

This work is supported by the Youth Project of Science and Technology Research Program of Chongqing Education Commission of China (No. KJQN202001139), Graduate Education High-Quality Development Action Plan of Chongqing University of Technology (No. gzlcx20233190).

References

1. Barabási, A-L., Albert, R. Emergence of Scaling in Random Networks. Science, 1999, 286, 5439, 509-512. https://doi.org/10.1126/science.286.5439.509
2. Chen, Z., He, Z., He, C. An Improved DPOP Algorithm Based on Breadth First Search Pseudo-Tree for Distributed Constraint Optimization. Applied Intelligence, 2017, 47, 3, 607-623. https://doi.org/10.1007/s10489-017-0905-4
3. Chen, Z., Wu, T., Deng, Y., Zhang, C. An Ant-Based Algorithm to Solve Distributed Constraint Optimization Problems. Proceedings of the AAAI Conference on Artificial Intelligence, 2018, 32, 1. https://doi.org/10.1609/aaai.v32i1.11580
4. Cheng, S., Raja, A., Xie, J. Dynamic Multiagent Load Balancing Using Distributed Constraint Optimization Techniques. Web Intelligence and Agent Systems: An International Journal, 2014, 12, 2, 111-138. https://doi.org/10.3233/WIA-140288
5. Choudhury, M., Mahmud, S., Khan, M. M. A Particle Swarm Based Algorithm for Functional Distributed Constraint Optimization Problems. Proceedings of the AAAI Conference on Artificial Intelligence, 2020, 34, 05, 7111-7118. https://doi.org/10.1609/aaai.v34i05.6198
6. Enembreck, F., André Barthès, J.-P. Distributed Constraint Optimization with MULBS: A Case Study on Collaborative Meeting Scheduling. Journal of Network and Computer Applications, 2012, 35, 1, 164-175. https://doi.org/10.1016/j.jnca.2011.02.016
7. Farinelli, A., Rogers, A., Jennings, N. R. Agent-based Decentralised Coordination for Sensor Networks Using the max-sum Algorithm. Autonomous Agents and Multi-Agent Systems, 2014, 28, 3, 337-380. https://doi.org/10.1007/s10458-013-9225-1
8. Farinelli, A., Rogers, A., Petcu, A., Jennings, N. R. Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm. Seventh International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08) (12/05/08 - 16/05/08), 2008, 639-646.
9. Fioretto, F., Pontelli, E., Yeoh, W. Distributed Constraint Optimization Problems and Applications: A Survey. Journal of Artificial Intelligence Research, 2018, 61, 623-698. https://doi.org/10.1613/jair.5565
10. Fioretto, F., Yeoh, W., Pontelli, E. A Multiagent System Approach to Scheduling Devices in Smart Homes. Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, 2017, 981-989.
11. Fransman, J., Sijs, J., Dol, H., Theunissen, E., De Schutter, B. Distributed Bayesian: A Continuous Distributed Constraint Optimization Problem Solver. Journal

of Artificial Intelligence Research, 2023, 76, 393-433. https://doi.org/10.1613/jair.1.14151

12. Fransman, J., Sijs, J., Dol, H., Theunissen, E., Schutter, BD. Bayesian-DPOP for Continuous Distributed Constraint Optimization Problems. Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, 2019, 1961-1963.
13. Hoang, K. D., Yeoh, W., Yokoo, M., Rabinovich, Z. New Algorithms for Continuous Distributed Constraint Optimization Problems. Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, 2020, 502-510.
14. Leite, A. R., Enembreck, F., Barthès, J.-P. A Distributed Constraint Optimization Problems: Review and perspectives. Expert Systems with Applications, 2014, 41, 11, 5139-5157. https://doi.org/10.1016/j.eswa.2014.02.039
15. Litov, O., Meisels, A. Forward Bounding on Pseudo-trees for DCOPs and ADCOPs. Artificial Intelligence, 2017, 252, 83-99. https://doi.org/10.1016/j.artint.2017.07.003
16. Maheswaran, R. T., Pearce, J. P., Tambe, M. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. Parallel and Distributed Computing Systems (ISCA), 2004.
17. Miller, S., Ramchurn, S. D., Rogers, A. Optimal Decentralised Dispatch of Embedded Generation in the Smart Grid. Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, 2012, 1, 281-288.
18. Modi, P. J., Shen, W.-M., Tambe, M., Yokoo, M. Adopt: Asynchronous Distributed Constraint Optimization with Quality Guarantees. Artificial Intelligence, 2005, 161, 1-2, 149-180. https://doi.org/10.1016/j.artint.2004.09.003
19. Petcu, A., Faltings, B. A Scalable Method for Multiagent Constraint Optimization. Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005, 266-271
20. Sarker, A., Choudhury, M., Khan, M. M. A Local Search Based Approach to Solve Continuous DCOPs. Proceedings of the 20th International Conference on Autonomous Agents and Multi-Agent Systems, 2021, 1127-1135.
21. Sebag, M., Ducoulombier, A. Extending Population-Based Incremental Learning to Continuous Search Spaces. Parallel Problem Solving from Nature - PPSN V, 1998, 1498, 418-427. https://doi.org/10.1007/BFb0056884
22. Shi, M., Liao, X., Chen, Y. A Particle Swarm with Local Decision Algorithm for Functional Distributed Constraint Optimization Problems. International Journal of Pattern Recognition and Artificial Intelligence, 2022, 36, 12, 2259025. https://doi.org/10.1142/S021800142259025X
23. Stranders, R., Farinelli, A., Rogers, A., Jennings, N. R. Decentralised Coordination of Continuously Valued Control Parameters Using the Max-Sum Algorithm. Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, 2009, 1, 601-608.
24. Voice, T., Stranders, R., Rogers, A., Jennings, N. A Hybrid Continuous Max-Sum Algorithm for Decentralised Coordination. 19th European Conference on Artificial Intelligence (16/08/10-20/08/10), 2010, 61-66.
25. Watts, D. J., Strogatz, S. H. Collective Dynamics of 'Small-World' Networks. Nature, 1998, 393, 6684, 440-442. https://doi.org/10.1038/30918
26. Xing-Ming, L., Chang-Feng, X., Ling, W., Fa-Xing, L. Path Planning for Multi-platform Missiles Based on Distributed Constrained Optimization. ACTA ELECTONICA SINICA, 2012, 40, 10, 2068-2072.
27. 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, 2005, 161, 1-2, 55-87. https://doi.org/10.1016/j.artint.2004.10.004