A Class of Local Search Based Anytime Algorithms for Continuous Distributed Constraint Optimization Problems

Xin Liao * and Khoi Hoang

Posted Date: 16 January 2024

doi: 10.20944/preprints202401.1158.v1

Keywords: Distributed Constraint Solving; Distributed Constraint Optimization; C-DCOPs

Article

A Class of Local Search Based Anytime Algorithms for Continuous Distributed Constraint Optimization Problems

Xin Liao [1,*] and Khoi Hoang [2]

[1] Southwest University; lxchat@foxmail.com
[2] Washington University in St. Louis, USA; khoi.hoang@wustl.edu
* Correspondence: lxchat@foxmail.com

Abstract: Continuous Distributed Constraint Optimization Problem (C-DCOPs) are a powerful framework to model problems with continuous variables in multi-agent systems. Previous works about C-DCOP algorithms mainly use pseudo-trees to guarantee the anytime property or are without anytime property guarantees. However, there is a risk of privacy leakage in the pseudo-tree due to utility passing between agents. Therefore, based on the basic constraint graph instead of the pseudo-tree, we (i) extend the *Maximum Gain Message* (MGM) algorithm by combining the local search strategy to solve C-DCOPs, named *Continuous MGM* (C-MGM), and it's able to guarantee the monotonicity of the solution quality; (ii) propose a *Parallel C-MGM* (C-PMGM) algorithm to improve the solution quality through parallel random search; and (iii) introduce the differential search into C-PMGM to design a *Parallel Differential Search C-MGM* (C-PDSM) algorithm, which constructs a heuristic method to speed up convergence and improve solution quality. Compared to other anytime C-DCOP algorithms using pseudo-trees, the proposed three algorithms can exhibit better performance in avoiding privacy violations. We theoretically prove that the proposed algorithms are the anytime algorithms and empirically demonstrate that our algorithms outperform the state-of-the-art C-DCOP algorithms.

Keywords: Distributed Constraint Solving; Distributed Constraint Optimization; C-DCOPs

1. Introduction

The *Distributed Constraint Optimization Problems* (DCOPs) [1–4] are an efficient framework to solve the cooperative problems of multi-agent systems. In DCOPs, agents communicate directly or indirectly to coordinate their value assignments of variables to optimize the aggregate constraint utilities, where the variables are discrete and the constraint utilities are presented in tabular form. DCOPs have been successfully applied to model the problems in the real world like distributed meeting scheduling [5,6], sensor networks [7], multi-robot coordination [8,9], smart grid [10,11], and smart home [12,13]. Some improved methods such as [14–17] and some improvements of key metrics like privacy [18–20] have been proposed. In addition, there are some extensions of DCOPs like Asymmetric DCOPs [21–24] and Dynamic DCOPs [25–27].

Over the years, researchers have proposed several different types of algorithms to solve DCOPs, including exact search-based algorithms [28,29], exact inference-based algorithms [2,30], approximate search-based algorithms [31], approximate inference-based algorithms [7], and approximate sampling-based algorithms [32,33]. Although the DCOP algorithms have excellent performance for discrete problems, it is not satisfactory for solving continuous problems, such as target tracking sensor orientation. Therefore, *Continuous DCOPs* (C-DCOPs) [34] were proposed to model problems with continuous variables, and the constraint utilities are functional forms. Correspondingly, researchers proposed new C-DCOP algorithms to deal with the modification of the C-DCOP formulation, including *Continuous Max-Sum* (CMS) [34], *Hybrid CMS* (HCMS) [35], *Bayesian Distributed Pseudo-tree Optimization Procedure* (B-DPOP) [36], *Particle Swarm Based Continuous DCOP*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

(PCD) [37], *Exact Continuous DPOP* (EC-DPOP) [38] and extensions (*Approximate Continuous DPOP* (AC-DPOP), *Clustered AC-DPOP* (CAC-DPOP), and *Continuous DSA* (C-DSA)), *Continuous Cooperative Constraint Approximation* (C-CoCoA) [39], and *Particle Swarm with Local Decision Based Continuous DCOP* (PCD-LD) [40].

The CMS algorithm is a continuous extension of *discrete Max-Sum* (MS) algorithm, which approximates the utility functions as piece-wise linear functions to solve C-DCOPs. Similarly, HCMS improves the solution quality according to the non-linear optimization method based on the MS algorithm. Later, B-DPOP combines the *Bayesian Optimization* framework with DPOP to solve C-DCOPs. After that, Hoang et al. propose three extensions of the inference-based DPOP algorithm and one extension of search-based *Distributed Stochastic Algorithm* (DSA): EC-DPOP, AC-DPOP, CAC-DPOP, and C-DSA, where the first algorithm provides an exact approach to solve general C-DCOPs, and the second and third provide approximate methods. In addition, C-DSA extends DSA to the continuous domain. C-CoCoA is a semi-greedy non-iterative algorithm that can get a high-quality solution with less execution time and overhead.

Unfortunately, none of the above C-DCOP algorithms is an anytime algorithm. An anytime algorithm should have two characteristics: (i) The algorithm can be stopped at any time to output a solution, which means it is iterative; (ii) The algorithm guarantees the monotonicity of the solution quality over time. PCD introduces the *Particle Swarm Optimization* (PSO) algorithm to solve C-DCOPs, and PCD-LD combines local decisions with PCD to improve the solution quality. They are two iterative algorithms and collect all utilities by constructing a *Breadth First Search* (BFS) pseudo-tree to guarantee the monotonicity of the solution quality. Although PCD and PCD-LD are anytime algorithms, the utility passing in the pseudo-tree causes privacy violations [41].

In the wake of the above analysis, we extend *Maximum Gain Message* (MGM) [31] algorithm so that it's able to solve C-DCOPs. Three main reasons prompt us to customize MGM for C-DCOPs as follows:

- We consider that anytime property is an important problem concerned for the C-DCOP algorithm. Since communication may be halted arbitrarily, an algorithm without the anytime property risk being terminated at an unsatisfactory assignment combination. Therefore, an anytime algorithm guarantees lower bounds on performance in anytime environments when given acceptable starting conditions.
- The MGM algorithm is an iterative, search-based algorithm that performs a distributed local search, and it can guarantee the monotonicity of the solution quality through local interactions. In other words, the MGM algorithm is an anytime algorithm.
- Compared with the anytime algorithms using the BFS pseudo-tree, the MGM algorithm solves problems without any restriction on the graph structure. Specifically, MGM uses a basic constraint graph and local interactions to maintain the privacy of agents.

Therefore, we provide three MGM-based C-DCOP algorithms, namely *Continuous MGM* (C-MGM), *Parallel C-MGM* (C-PMGM), and *Parallel Differential Search C-MGM* (C-PDSM). Firstly, C-MGM is an approximation C-DCOP algorithm that is similar to MGM. In C-MGM, each agent compares the gain of its local utility with neighbors to strictly increase the aggregate utility. Then, C-PMGM extends C-MGM and generates a competing solution for each solution to solve C-DCOPs in parallel. Finally, C-PDSM is an improvement based on C-PMGM that uses differential search to update competing solutions. We empirically evaluate the performance of proposed algorithms on benchmark problems, and the results show that our algorithms outperform the state-of-the-art C-DCOP algorithms.

The remainder of this paper is organized as follows. The next section introduces the necessary background, including the DCOP and C-DCOP frameworks, and the standard MGM algorithm. The details of our algorithms are described in Section 3. Then, we present theoretical analysis in Section 4. Afterward, Section 5 reports the empirical evaluation of our algorithms against the state-of-the-art algorithms. Finally, we provide conclusions and future work in Section 6.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

2. Background

In this section, we provide the definition of the DCOP framework. Then, we introduce the C-DCOP framework and provide an example of a C-DCOP. Finally, we describe the MGM algorithm, which is the basis of our proposed algorithms.

2.1. Distributed Constraint Optimization Problems

A *Distributed Constraint Optimization Problem* (DCOP) is a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where

- $\mathbf{A} = \{a_i\}_{i=1}^{n}$ is a set of *agents*, an agent can control one or more variables.
- $\mathbf{X} = \{x_i\}_{i=1}^{m}$ is a set of discrete *variables*, each variable is controlled by one of the agents.
- $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$ is a set of discrete *domains* and each variable $x \in \mathbf{X}$ takes value from the set D_x.
- $\mathbf{F} = \{f_i\}_{i=1}^{l}$ is a set of *utility functions* and each utility function is defined over a set of variables: $f_i \in \mathbf{F} : \prod_{x \in \mathbf{x}^{f_i}} D_x \rightarrow \mathbb{R}$, where the $\mathbf{x}^{f_i} \subseteq \mathbf{X}$ is the *scope* of f_i.
- α : is a *mapping function* that associates each variable to one agent. In this paper, we assume one agent controls only one variable.

In this paper, we assume that one agent only controls one variable, thus $n = m$ and the terms "agent" and "variable" could be used interchangeably. In addition, we assume that all utility functions are binary functions between two variables, thus $f_{ij} \in \mathbf{F} : D_{x_i} \times D_{x_j} \rightarrow \mathbb{R}$ and the $\mathbf{x}^{f_{ij}} = \{x_i, x_j\}$.

A solution of a DCOP is an assignment combination X^* to all variables that are consistent with their respective domains. The goal of the solution is to minimize[1] the aggregate utility as shown in Equation (1), where the aggregate utility is the sum of utilities across all the applicable utility functions.

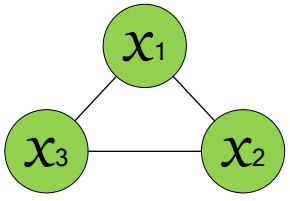
$$X^* = \underset{d_i \in D_{x_i}, d_j \in D_{x_j}}{\arg\min} \sum_{f_{ij} \in \mathbf{F}} f_{ij}(x_i = d_i, x_j = d_j) \tag{1}$$

2.2. Continuous Distributed Constraint Optimization Problems

The *Continuous DCOP* (C-DCOP) framework extends the DCOP framework by modeling the variables as continuous variables. It is defined by a tuple $\langle \mathbf{A}, \mathbf{X}, \mathbf{D}, \mathbf{F}, \alpha \rangle$, where $\mathbf{A}$, $\mathbf{F}$, and α are the same as DCOP. The set of variables $\mathbf{X}$ and the set of domains $\mathbf{D}$ are defined as follows:

- $\mathbf{X} = \{x_i\}_{i=1}^{m}$ is the set of *continuous* variables.
- $\mathbf{D} = \{D_x\}_{x \in \mathbf{X}}$ is the set of *continuous* domains and each *continuous* variable takes any value from the domain $D_x = [LB_x, UB_x]$, where LB and UB represent the lower and upper bounds of the domain, respectively.

The goal of a C-DCOP is the same as Equation (1), where the variables x and domain D_x are continuous. Figure 1 provides an example of a C-DCOP, where Figure 1(a) is a constraint graph with three variables controlled by the agents and three constraints. Figure 1(b) defines the utility functions corresponding to the constraints and gives the domain of variables.



(a) Constraint Graph

$$\forall x_i \in \mathrm{X} : D_{x_i} = [-5, 5]$$
$$f(x_1, x_2) = x_1^3 - x_1^2 + x_2^2 + x_2$$
$$f(x_2, x_3) = x_2^2 - x_3^2 + \ln(2|x_3| + 1)$$
$$f(x_1, x_3) = x_1^3 + \sin(\pi x_1) - \cos(\pi x_3) - x_3$$

(b) Utility Functions

Figure 1. An example of a C-DCOP, where (a) is a constraint graph with four variables, and (b) represents four utility functions.

[1] We are going to consider the minimization in this paper.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

2.3. Maximum Gain Message

The MGM algorithm is an anytime, local search algorithm for solving DCOPs. Its main idea is to guarantee the agent with maximum gain changes its variable through competition between neighboring agents. In this decision method, only one agent can change its value in each local, which can completely avoid the uncertainty of the result caused by the simultaneous assignment of neighboring agents. Therefore, the solution quality improves steadily with iterations.

The framework of MGM is presented in Algorithm 1. Firstly, each agent randomly selects a value and sends it to neighbors (Algorithm 1: Lines 2-3). After receiving these values from neighbors, the agent assumes that the neighbors' values remain unchanged and chooses a new value that maximizes the local gain. Then, the agent sends the local gain to neighbors and simultaneously receives gains from neighbors. If the agent with the maximum gain in the current local, it's assigned the new value, otherwise it remains the chosen random value. Finally, the agent sends the updated value to neighbors and executes the loop (Algorithm 1: Lines 5-12) until the condition is met.

Algorithm 1: Maximum Gain Message Algorithm (MGM)

```
1  foreach agent x_i do
2      value ← Choose a random value;
3      Send value to neighbors;
4  while termination condition not met each agent x_i do
5      Receive neighbors' values;
6      Choose a new value that reduces the local utility most;
7      G_i (gian) ← the number of the local utility reduced by the new value;
8      Send G_i to neighbors;
9      Receive neighbors' G_j;
10     if G_i is less than all G_j of neighbors then
11         Assign the new value;
12         Send value to neighbors;
```

3. Our Algorithms

In this section, we introduce the three proposed algorithms in detail. Let us define a *solution set* $\boldsymbol{S} = \{S_k\}_{k=1}^{K}$ containing K solutions, where $S_k = \{S_k.x_i\}_{i=1}^{n}$ represents *an assignment combination* (i.e. a solution) of n variables controlled by agents. Therefore, an agent x_i holds the value assignments of K solutions on the i^{th} dimension, denoted as $\boldsymbol{S}.x_i = \{S_k.x_i\}_{k=1}^{K}$. In addition, we use $S_k.x_i$ to represent the variable of the k^{th} solution held by agent x_i, and its local utility is $\mathscr{L}_i(S_k, S_k) = \sum_{j \in N_i} f_{ij}(S_k.x_i, S_k.x_j)$, where (k, k) represents two variables on the k^{th} solution and N_i represents the neighbors of agent x_i. For example, we construct a solution set with two solutions for Figure 1 as $\boldsymbol{S} = \{S_1, S_2\}$, where $S_1 = \{S_1.x_1, S_1.x_2, S_1.x_3\}$ and $S_2 = \{S_2.x_1, S_2.x_2, S_2.x_3\}$. For agent x_1, its value assignments are $\boldsymbol{S}.x_1 = \{S_1.x_1, S_2.x_1\}$ and the local utility of $S_1.x_1$ is $\mathscr{L}_i(S_1, S_1)$. It is worth noting that since we consider minimizing the aggregate utility, we use *absolute gain* to represent the local utility reduced, which is a negative value.

3.1. Continuous Maximum Gain Message

Algorithm 2 shows the process of the *Continuous Maximum Gain Message* (C-MGM) algorithm, including three phases: **Initialization**, **Calculation**, and **Comparison**.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

Algorithm 2: Continuous MGM (C-MGM)

1 Initialize parameter: K;
2 $\boldsymbol{S} \leftarrow$ set of K solutions;
3 **foreach** *agent* x_i **do**
4 `init(`$\boldsymbol{S}.x_i$`);`
5 $G_i \leftarrow 0$;
6 $S_r.x_i \leftarrow$ a random value from $\boldsymbol{S}.x_i$;
7 Send **ValueMessage**$(\{i, S_r.x_i\})$ to N_i;
8 **Function** `init(`$\boldsymbol{S}.x_i$`)`**:**
9 **foreach** *solution* $S_k.x_i \in \boldsymbol{S}.x_i$ **do**
10 $S_k.x_i \leftarrow$ a random value from D_{x_i};
11 **while** *termination condition not met each agent* x_i **do**
12 **if** *recevie* **ValueMessage** from $N_{i_j} \in N_i$ **then**
13 **foreach** *solution* $S_k \in \boldsymbol{S}$ **do**
14 $\mathscr{L}_i(S_k, S_r) = \sum_{j \in N_i} f_{ij}(S_k.x_i, S_r.x_j)$;
15 $\mathscr{L}_i(S_r, S_r) = \sum_{j \in N_i} f_{ij}(S_r.x_i, S_r.x_j)$;
16 $G_i = \min\{\mathscr{L}_i(S_k, S_r)\}_{k=1}^{K} - \mathscr{L}_i(S_r, S_r)$;
17 Send **GainMessage**$(\{i, G_i\})$ to N_i;
18 **if** *recevie* **GainMessage** from $N_{i_j} \in N_i$ **then**
19 **if** G_i *is less than all* G_j *and* $G_i < 0$ **then**
20 $S_r.x_i \leftarrow \arg\min_{S_k.x_i} \{\mathscr{L}_i(S_k, S_r)\}_{k=1}^{K}$;
21 **foreach** *solution* $S_k \in \boldsymbol{S}$ **do**
22 $S_k.x_i \leftarrow$ a random value from D_{x_i};
23 **Output:** $S_r.x_i$**;**
24 Send **ValueMessage**$(\{i, S_r.x_i\})$ to N_i;

Initialization: Firstly, C-MGM constructs a solution set for K solutions. Then, each agent assigns a random value from its domain to each solution on the current dimension and generates a variable G_i to record the gain value of agent x_i. Finally, the agent x_i sends **ValueMessage**$(\{i, S_r.x_i\})$ to all neighbors (Algorithm 2: Lines 1-7), where $S_r.x_i$ is a random value in $\boldsymbol{S}.x_i$.

Calculation: After receiving **ValueMessage**$(\{j, S_r.x_j\})$ from neighbors, the agent x_i calculates the local utility for each solution according to its value assignments $\boldsymbol{S}.x_i$, respectively (Algorithm 2: Lines 12-14). We define $\mathscr{L}_i(S_r, S_r)$ to represent the *basic local utility* of agent x_i, which is the local utility assuming that the values of agent x_i and its neighboring agents are not changed, calculated by $\sum_{j \in N_i} f_{ij}(S_r.x_i, S_r.x_j)$. Then, the G_i is assigned a difference value between the minimum and basic local utility (Algorithm 2: Lines 15-16). Finally, the agent x_i sends **GainMessage**$(\{i, G_i\})$ to neighbors.

Comparison: Similarly, the agent x_i enters the comparison phase after receiving **GainMessage**$(\{j, G_j\})$ from each neighbor $N_{i_j} \in N_i$. If the current agent with the minimum gain value among all gain values of neighbors and the gain is less than 0, the agent x_i assigns the value that minimizes local utility to $S_r.x_i$ (Algorithm 2: Lines 18-20). In general, after the size of the domain is determined, the variable of the standard MGM algorithm can take a finite number of values from its domain, but there is an infinite number of values to be selected in the variable of C-MGM. We consider updating the value assignments for each agent to search its domain and avoid falling in the local optimum by adding randomness. Therefore, the agent x_i updates each value $S_k.x_i$ to a random value from its domain D_{x_i} (Algorithm 2: Line 22).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

3.2. Parallel C-MGM

We consider solving K solutions in parallel based on C-MGM. Therefore, we propose the *Parallel C-MGM* (C-PMGM) algorithm as shown in Algorithm 3, which also consists of three phases: **Initialization**, **Calculation**, and **Comparison**.

Algorithm 3: Parallel C-MGM (C-PMGM)

1 Initialize parameter: K;
2 $\boldsymbol{S} \leftarrow$ set of K solutions;
3 $\boldsymbol{C} \leftarrow$ set of K solutions;
4 **foreach** *agent* x_i **do**
5 `init(`$\boldsymbol{S}.x_i$`);`
6 `init(`$\boldsymbol{C}.x_i$`);`
7 $\boldsymbol{G}_i \leftarrow$ set of K gains;
8 **foreach** $G_{k,i} \in \boldsymbol{G}_i$ **do**
9 $G_{k,i} \leftarrow 0$;
10 Send **ValueMessage**$(\{i, \boldsymbol{S}.x_i\})$ to N_i;
11 **while** *termination condition not met each agent* x_i **do**
12 **if** *receive* **ValueMessage** from $N_{i_j} \in N_i$ **then**
13 **foreach** $k \in K$ **do**
14 $\mathscr{L}_i(S_k, S_k) = \sum_{j \in N_i} f_{ij}(S_k.x_i, S_k.x_j)$;
15 $\mathscr{L}_i(C_k, S_k) = \sum_{j \in N_i} f_{ij}(C_k.x_i, S_k.x_j)$;
16 $G_{k,i} = \mathscr{L}_i(C_k, S_k) - \mathscr{L}_i(S_k, S_k)$;
17 Send **GainMessage**$(\{i, \boldsymbol{G}_i\})$ to N_i;
18 **if** *recevie* **GainMessage** from $N_{i_j} \in N_i$ **then**
19 **foreach** $k \in K$ **do**
20 **if** $G_{k,i}$ *is less than all* $G_{k,j}$ *and* $G_{k,i} < 0$ **then**
21 $S_k.x_i = C_k.x_i$;
22 $C_k.x_i \leftarrow$ a random value from D_{x_i};
23 **Output:** $\boldsymbol{S}.x_i$;
24 Send **ValueMessage**$(\{i, \boldsymbol{S}.x_i\})$ to N_i;

Initialization: Except for the similar initialization as C-MGM, C-PMGM additionally generates a *competing solution set* $\boldsymbol{C}$, which is the same as the solution set $\boldsymbol{S}$. In addition, each agent uses a *gain set* $\boldsymbol{G}$ to record the gain values of K solutions (Algorithm 3: Lines 7-9). It is worth noting that the **ValueMessage**$(\{i, \boldsymbol{S}.x_i\})$ sent by agent x_i in C-PMGM to neighbors is the value assignments instead of a random value in $\boldsymbol{S}.x_i$ (Algorithm 3: Line 10).

Calculation: Since the computational overhead of solving K solutions according to the way of C-MGM is expensive, we consider using the values from the corresponding competing solution set to calculate the gains for K solutions (Algorithm 3: Lines 11-16), which means that the indexes of the two solution sets ($\boldsymbol{S}$ and $\boldsymbol{C}$) are consistent. Later, the agent x_i sends **GainMessage**$(\{i, \boldsymbol{G}_i\})$ to neighbors.

Comparison: Similar to C-MGM, the agent x_i in C-PMGM compares K gains with all neighbors. For each gain $G_{k,i}$ in the gain set of agent x_i, if the gain value is the smallest in the current local and less than 0, the value $S_k.x_i$ is assigned to its competing value $C_k.x_i$ (Algorithm 3: Lines 18-21). Since an agent only calculates one local utility for a solution instead of K local utilities (such as *Algorithm 2: Lines 13-14*), the idea is to search as many competing values as possible. Therefore, the agent x_i resets each competing value $C_k.x_i$ to a value from its domain, whether or not the value $S_k.x_i$ is changed.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

3.3. Parallel Differential Search C-MGM

Based on C-PMGM, we construct a heuristic method to guide the exploitation of the competing solution set. Specifically, we extend the update of value assignments in C-PMGM by combining a differential search, named *Parallel Differential Search C-MGM* (C-PDSM). The details of C-PDSM can be found in Algorithm 4.

Algorithm 4: Parallel Differential Search C-MGM (C-PDSM)

1 Initialize parameters: K, ω;
2 $S \leftarrow$ set of K solutions;
3 $C \leftarrow$ set of K solutions;
4 **foreach** *agent* x_i **do**
5 `init(`$S.x_i$`);`
6 `init(`$C.x_i$`);`
7 $G_i \leftarrow$ set of K gains;
8 **foreach** $G_{k,i} \in G_i$ **do**
9 $G_{k,i} \leftarrow 0$;
10 Send **ValueMessage**$(\{i, S.x_i\})$ to N_i;
11 **while** *termination condition not met each agent* x_i **do**
12 **if** *receive* **ValueMessage** from $N_{i_j} \in N_i$ **then**
13 **foreach** $k \in K$ **do**
14 $\mathscr{L}_i(S_k, S_k) = \sum_{j \in N_i} f_{ij}(S_k.x_i, S_k.x_j)$;
15 $\mathscr{L}_i(C_k, S_k) = \sum_{j \in N_i} f_{ij}(C_k.x_i, S_k.x_j)$;
16 $G_{k,i} = \mathscr{L}_i(C_k, S_k) - \mathscr{L}_i(S_k, S_k)$;
17 Send **GainMessage**$(\{i, G_i\})$ to N_i;
18 **if** *recevie* **GainMessage** from $N_{i_j} \in N_i$ **then**
19 **foreach** $k \in K$ **do**
20 **if** $G_{k,i}$ *is less than all* $G_{k,j}$ *and* $G_{k,i} < 0$ **then**
21 $S_k.x_i = C_k.x_i$;
22 `update();`
23 **Output:** $S.x_i$;
24 Send **ValueMessage**$(\{i, S.x_i\})$ to N_i;
25 **Function** `update()`**:**
26 $\Omega \leftarrow \{\}$;
27 **foreach** $G_{k,i} \in G_i$ **do**
28 **if** $G_{k,i} < 0$ **then**
29 $\Omega \cup G_{k,i}$;
30 **foreach** $C_k.x_i \in C.x_i$ **do**
31 **if** $|\Omega| < 2$ **then**
32 $C_k.x_i \leftarrow$ a random value from D_i;
33 **else**
34 Calculate $C_k.x_i$ according to Equation (2);

The difference between C-PDSM and C-PMGM is the update of value assignments (Algorithm 4: Line 22). For a gain set in one agent, the agent filters each absolute gain and adds it to an *absolute gain set* Ω (Algorithm 4: Lines 26-29). Since the differential search method we used requires at least

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

two absolute gains, we provide two ways to update value assignments (Algorithm 4: Lines 30-34): (i) If the number of absolute gains is less than two, the agent update the value assignments of the competing solution set through random search. (ii) Otherwise, the agent updates value assignments by differential search, as shown in Equation (2). Our motivation is that the current value learns the exploitation ability from other values with absolute gain, which can improve the convergence speed of the algorithm.

$$\begin{aligned} C_k.x_i = C_k.x_i + \omega * (C_{\arg\min_k\{\Omega\}}.x_i - C_k.x_i) \\ + \omega * (C_{\arg\max_k\{\Omega\}}.x_i - C_k.x_i) \end{aligned} \tag{2}$$

where ω is a scaling factor $\in [0,2]$, which controls the amplification of the differential variation [42]. The "arg $\min_k\{\Omega\}$" and "arg $\max_k\{\Omega\}$" represent the indexes of competing solution set with the minimum and maximum absolute gain values, respectively.

3.4. An Example of Algorithms

We take Figure 1 as an example to realize the three phases of C-MGN, C-PMGM, and C-PDSM.

In the **Initialization** phase, we assume that the size of the solution set is $K = 2$ and the solution sets of the three algorithms are all $\boldsymbol{S} = \{S_1 = \{3,1,5\}, S_2 = \{-2,2,4\}\}$. Therefore, the value assignments held by agent x_1 are $\boldsymbol{S}.x_1 = \{S_1.x_1 = 3, S_2.x_1 = -2\}$, the value assignments of agent x_2 are $\boldsymbol{S}.x_2 = \{1,2\}$, and the value assignments of agent x_3 are $\boldsymbol{S}.x_3 = \{5,4\}$.

- For C-MGM, the values randomly selected by each agent are $S_r.x_1 = 3$, $S_r.x_2 = 2$, and $S_r.x_3 = 4$.
- For C-PMGM and C-PDSM, we assume the competing solution sets both are $\boldsymbol{C} = \{C_1 = \{-4,-1,0\}, C_2 = \{-3,3,-5\}\}$ and the scaling factor ω in C-PDSM is 1.4.

In the **Calculation** phase, the agent x_i calculates the local utility and gain after receiving the random value $S_r.x_j$ or value assignments $\boldsymbol{S}.x_j$ from neighbors $N_{i_j} \in N_i$. Let's take the agent x_1 as an example for calculation.

- For C-MGM:
  $\mathscr{L}_1(S_1, S_r) = f(3,2) + f(3,4) = 46$
  $\mathscr{L}_1(S_2, S_r) = f(-2,2) + f(-2,4) = -19$
  $\mathscr{L}_1(S_r, S_r) = f(3,2) + f(3,4) = 46$
  $G_1 = -19 - 46 = -65$;
- For C-PMGM and C-PDSM:
  $\mathscr{L}_1(S_1, S_1) = f(3,1) + f(3,5) = 43$
  $\mathscr{L}_1(C_1, S_1) = f(-4,1) + f(-4,5) = -146$
  $G_{1,1} = -146 - 43 = -189$;
  $\mathscr{L}_1(S_2, S_2) = f(-2,2) + f(-2,4) = -19$
  $\mathscr{L}_1(C_2, S_2) = f(-3,2) + f(-3,4) = -62$
  $G_{2,1} = -62 - (-19) = -43$;

In the **Comparison** phase, each agent compares the gains to action. Simultaneously, there are different way to update value assignments in C-MGM, C-PMGM, and C-PDSM. We assume that the agent x_1 with maximum gain among its neighbors, and its actions and updates are as follows:

- For C-MGM:
  $S_r.x_1 = \arg\min_{S_k}\{46, -19\} = S_2.x_1 = -2$;
  $\boldsymbol{S}.x_1 = \{1,-1\}$ (assumed random values);
- For C-PMGM:
  $S_1.x_1 = C_1.x_1 = -4$, $S_2.x_1 = C_2.x_1 = -3$;
  $\boldsymbol{C}.x_1 = \{2,-2\}$ (assumed random values);

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

- For C-PDSM:
  $S_1.x_1 = C_1.x_1 = -4$, $S_2.x_1 = C_2.x_1 = -3$;
  Since both gain values (-189 and -43) are less than 0, $|\Omega| = 2$.
  $C_1.x1 = -4 + 1.4 * [-4 - (-4)] + 1.4 * [-3 - (-4)] = -2.6$
  $C_2.x1 = -3 + 1.4 * [-4 - (-3)] + 1.4 * [-3 - (-3)] = -4.4$;

Finally, C-MGM outputs a solution as $X^* = \{-2, 2, 4\}$. The two solutions output by C-PMGM and C-PDSM are $X_1^* = \{-4, 1, 5\}$ and $X_2^* = \{-3, 2, 4\}$, the difference between C-PMGM and C-PDSM is that the value assignments of agent x_1 in the competing solution set are $\boldsymbol{C}.x_1 = \{2, -2\}$ and $\boldsymbol{C}.x_1 = \{-2.6, -4.4\}$, respectively.

4. Theoretical Analysis

In this section, we prove that C-MGM, C-PMGM, and C-PDSM are three anytime algorithms, and we provide some theoretical properties of our algorithms in terms of communication, computation, and memory. We assume that the graph structure is a binary constraint graph $G = (N, E)$, the number of nodes is $|A|$, and the number of edges in the constraint graph is $|F|$. In addition, we define one iteration as the complete process of the algorithm completing **Calculation** and **Comparison** phase.

Theorem 1. *C-MGM, C-PMGM, and C-PSDM are three anytime algorithms.*

Proof. On the one hand, the three algorithms are iterative, which means they can be terminated at any time and output solutions. On the other hand, we assume that the *aggregate utility* (**U**) of C-MGM at t iteration is $\mathbf{U}_t = \sum_{f_{ij} \in \mathbf{F}} f_{ij}$, which represents the sum of utility on all constraints. When an agent calculates local utility, the same constraint is calculated twice by two neighboring agents, the aggregate utility can be modified as $\mathbf{U}_t = \frac{1}{2} \sum_{x_i \in \mathbf{X}} \mathscr{L}_i$. Therefore, the aggregate utility of the $t+1$ iteration in C-MGM can represent $\mathbf{U}_{t+1} = \frac{1}{2} \sum_{x_i \in \mathbf{X}} \mathscr{L}_i + \sum_{x_i \in \mathcal{X}} G_i$, where $\mathcal{X} \subseteq \mathbf{X}$ represents a set of variables that change their values. In each local, since only one variable with local utility reduced changes its value, we can get that $\sum_{x_i \in \mathcal{X}} G_i < 0$. Therefore, $\mathbf{U}_{t+1} < \mathbf{U}_t$, which means the aggregate utility strictly decreases with increasing iterations. Similarly, C-PMGM and C-PDSM solve K solutions in parallel, the aggregate utilities' change of one solution in them is the same as that of C-MGM. Hence, C-MGM, C-PMGM, and C-PSDM are three anytime algorithms. □

Theorem 2. *The total number of messages of C-MGM, C-PMGM, and C-PDSM with t iterations all are* $4t|F|$.

Proof. For each constraint $f_{ij}(x_i, x_j)$ in one iteration, the agents of the three algorithms need to send value assignments to neighbors in the **Calculation** phase and vice versa, there are two messages per edge. Therefore, the number of messages is $2|F|$ in this phase, where $|F|$ is the number of constraints (edges). In the **Comparison** phase, the agents of the three algorithms share gain or gain sets with neighbors to compare their values, thus the number of messages is also $2|F|$. Finally, the total number of messages of C-MGM, C-PMGM, and C-PDSM with t iterations all are $t * (2|F| + 2|F|) = 4t|F|$. □

Theorem 3. *In one iteration, the message size of C-MGM, C-PMGM, and C-PDSM are respectively* 1, K, *and* K, *where K is the number of solutions.*

Proof. In the **Calculation** phase, the agent in C-MGM sends a random value to its neighbors to calculate the local utility, the message size of the random value is 1. However, the agents in C-PMGM and C-PDSM send value assignments to neighbors. The size of value assignments is K, where K is the number of solutions. In the **Comparison** phase, the size of the gain message in C-MGM is 1, but the size of gain sets in the other two algorithms is K. Hence, since $max\{1, 1\} = 1$, $max\{K, K\} = K$, and $max\{K, K\} = K$, the message size of C-MGM, C-PMGM, and C-PDSM are respectively 1, K, and K in one iteration. □

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

Theorem 4. *In one iteration, the computational overhead of one agent in C-MGM, C-PMGM, and C-PSAM are respectively* $K|N|+1$, $K(2|N|+1)$, *and* $2K(|N|+1)$, *where* K *is the number of solutions and* $|N|$ *is the number of neighbors. Further, the overall computational overhead of C-MGM, C-PMGM, and C-PSAM in one iteration are* $2K|F|+|A|$, $K(4|F|+|A|)$, *and* $K(4|F|+2|A|)$, *respectively.*

Proof. We define the number of neighbors of one agent as $|N|$. For C-MGM, the agent x_i needs to calculate the local utility for K solutions in one iteration, including the basic local utility. In addition, the agent calculates the gain once. Therefore, in one iteration, the computational overhead of one agent in C-MGM is $K|N|+1$. Since $\sum_{x_i \in X} |N_i| = 2|F|$, and the overall computational overhead of C-MGM is $\sum_{x_i \in X} K|N_i|+1 = 2K|F|+|A|$.

For C-PMGM, the agent calculates the basic local utility and *competing local utility* (Algorithm 3: Line 15) for K solutions with neighbors, and the agent calculates the gain set, including K gains. Hence, the computational overhead of one agent in C-PMGM is $2K|N|+K = K(2|N|+1)$, the overall computational overhead is $K(4|F|+|A|)$.

Compared to C-PMGM, an agent of C-PDSM additionally updates the value of K solutions based on C-PMGM. Therefore, the computational overhead of one agent in C-PDSM in one iteration is $2K|N|+K+K = 2K(|N|+1)$, and the overall computational overhead is $K(4|F|+2|A|)$. □

5. Experimental Results

In this section, we empirically evaluate the solution quality of our three algorithms and the state-of-the-art algorithms, PCD, C-CoCoA, and PCD-LD. Firstly, we set initialization parameters for each algorithm. Specially, we evaluate the performance of C-PDSM with different scaling factors to find the most suitable scaling factor for comparison experiments. Then, we compare the solution quality of the six algorithms when the algorithm runs completely. Finally, we use the runtime to evaluate the performance of all algorithms.

Although C-MGM, C-PMGM, and C-PDSM can use any form of the utility function, we use the utility function in the form of $ax^2+bx+cxy+dy+ey^2+f$ to fairly show the results for consistency with researchers' work [38,40], where a, b, c, d, e, and f are random numbers in the range $[-5, 5]$. Furthermore, we set the domain of each variable to $[-50, 50]$ and the number of iterations to 500. It is worth mentioning that we choose the best solutions in C-PMGM and C-PDSM to present comparisons. For each benchmark problem, we independently run each algorithm on 30 different sets of function coefficients (a, b, c, d, e, and f) and take the average solution quality as the experimental result. The experiments are carried out on a computer equipped with Intel(R) Core(TM) i5-10500 CPU, 3.10GHz processor, and 8GB RAM.

We choose three benchmark problems and set their parameters: (i) *Random Graphs:* we use the Erdös-Rényi model [43] to generate random graphs with two different settings: **D1**: Sparse random graphs ($p = 0.1$) and **D2**: dense random graphs ($p = 0.6$), where p represents the probability of connection between two nodes in a graph; (ii) **D3**: *Scale-free Networks:* we refer to the Barabási-Albert (BA) [44] model to generate scale-free networks with an initial set of $m_1 = 10$ connected agents, and at each iteration, a new agent that connected to $m_2 = 3$ other agents is added to the current network; and (iii) **D4**: *Small-world Networks:* we use the Newman-Watts model [45] to provide small-world networks. The number of nearest nodes is set to 8 and the probability of connection adding is set to 0.1.

5.1. Parameter Configuration

In C-MGM, C-PMGM, and C-PDSM, K represents the size of the solution set, which is also the number of solutions that need to be optimized. Considering the previous work of some researchers using the solution set [37,40], we uniformly set K to 1000. For the scaling factor in C-PDSM, we use different scaling factors to compare the solution quality of C-PDSM on sparse random graphs with 100 agents. Figure 2 shows the convergence curves of C-PDSM with different scaling factors, where an obvious parameter setting is $\omega = 1.0$. When $\omega < 1.0$, the convergence speed of C-PDSM becomes

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

slower as ω decreases. Conversely, the solution quality of C-PDSM with different parameters is similar when $\omega > 1.0$. However, we can observe that there is a slight superiority in terms of convergence when $\omega = 1.6$. Therefore, we set ω to 1.6 for C-PDSM. In addition, we set the parameters for competing algorithms according to the references [37,39,40].

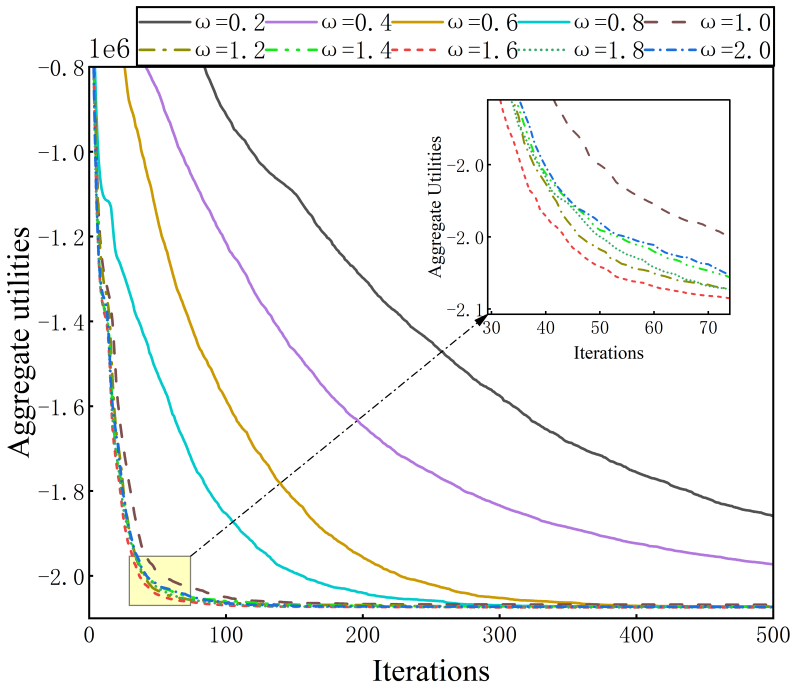


Figure 2. The convergence curves of different scaling factors for C-PDSM.

5.2. Comparison of Solution Quality

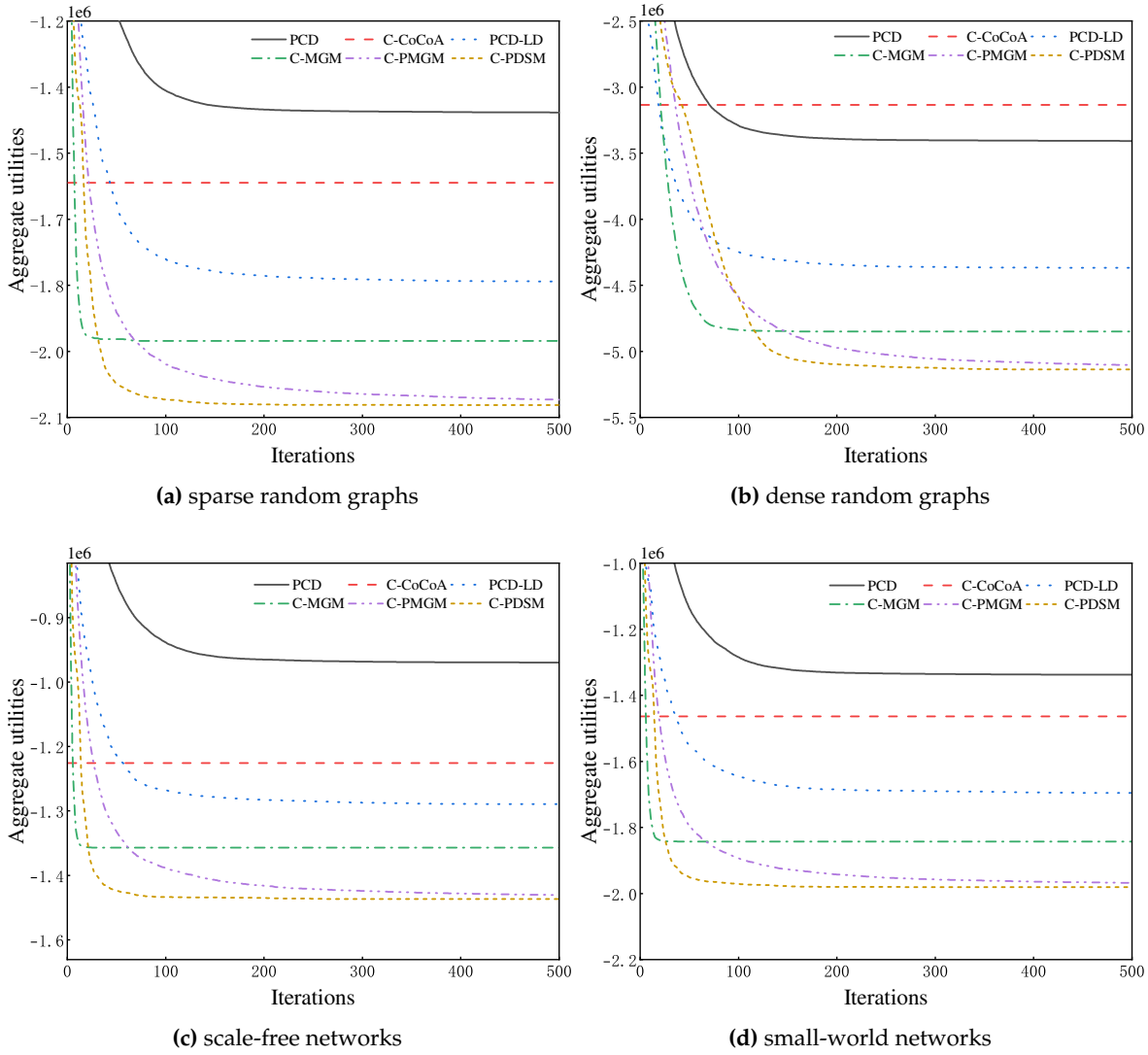
We show convergence curves of our algorithms and competing algorithms on three benchmark problems with 100 agents. Since C-CoCoA is a non-iterative algorithm, we use a straight line to represent its solution quality.

Figure 3a and Figure 3b show the solution quality of the six algorithms on sparse and dense random graphs, respectively. It can be seen that the solution quality of our three algorithms is significantly better than that of competing algorithms. Compared to performance on sparse random graphs, our three algorithms require more iterations to improve the solution quality. The most critical reason is that each agent has more neighbors in the dense configuration. Since only one of the two neighboring agents is allowed to change its value in our algorithms, there are fewer agents to improve their local utilities in one iteration. Therefore, our algorithms converge slower. Nevertheless, the solution quality of the three algorithms is superior to competing algorithms at about 100 iterations.

The convergence curves of the six algorithms on scale-free and small-world networks are presented in Figure 3c and Figure 3d, respectively. The performance of our algorithms is similar to that on sparse random graphs. C-MGM can obtain a high-quality solution and converge after a few iterations, and C-PMGM can improve the solution quality as the number of iterations increases. Compared to C-MGM, C-PDSM can converge faster since it uses heuristics to enhance the exploitation of solutions.

Further, we evaluate the solution quality of the six algorithms on each benchmark problem with 80 and 90 agents, respectively. The detailed results are shown in Table 1. We can observe clearly that the solution quality of C-MGM, C-PMGM, and C-PDSM significantly outperforms that of competing algorithms on different benchmark problems with varying numbers of agents, and C-PDSM has the best solution quality among all algorithms.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1



(a) sparse random graphs

(b) dense random graphs

(c) scale-free networks

(d) small-world networks

Figure 3. The solution quality of the proposed algorithms and competing algorithms on different benchmark problems.

Table 1. The solution quality of the proposed algorithms and competing algorithms on benchmark problems with different numbers of agents.

Number	Type	PCD	C-CcCoA	PCD-LD	C-MGM	C-PMGM	C-PDSM
$\|A\| = 80$	D1	-1042798	-1101989	-1304624	-1366430	-1465110	**-1473604**
	D2	-2585639	-2393154	-3241010	-3529010	-3725781	**-3749835**
	D3	-883870	-979241	-1074588	-1127033	-1213038	**-1220117**
	D4	-1191145	-1273078	-1465341	-1547525	-1650405	**-1660851**
$\|A\| = 90$	D1	-1259013	-1387131	-1576550	-1657470	-1798312	**-1810145**
	D2	-3102676	-2821139	-3769051	-4128793	-4364131	**-4393997**
	D3	-990772	-1120876	-1221272	-1285462	-1379148	**-1386262**
	D4	-1231794	-1332859	-1527462	-1634378	-1748604	**-1761438**
$\|A\| = 100$	D1	-1407602	-1567222	-1791288	-1926485	-2059722	**-2071917**
	D2	-3407684	-3135087	-4367475	-4849739	-5103759	**-5137080**
	D3	-1000479	-1203417	-1286422	-1373971	-1469692	**-1477799**
	D4	-1337766	-1463976	-1695295	-1842810	-1967855	**-1980617**

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 January 2024 doi:10.20944/preprints202401.1158.v1

5.3. Comparison of Algorithms Using Runtime

We compare the best solution quality reached by C-MGM, C-PMGM, C-PDSM, and competing algorithms on each benchmark problem after runtime in 1000 milliseconds (ms).

Table 2 lists the solution quality of algorithms on benchmark problems with different numbers of agents. Since C-MGM uses multiple local utilities to find the maximum gain compared to C-PMGM and C-PDSM, it can quickly find a high-quality solution. In addition, the solution quality of C-MGM outperforms competing algorithms on each benchmark problem except for the comparison with C-CoCoA on dense random graphs, the reason is that C-MGM converges slower due to the increase of neighbors.

Table 2. The solution quality and runtime of the proposed algorithms and competing algorithms on benchmark problems with different numbers of agents, where *time* represents the runtime (ms) when the algorithm reaches its best solution quality.

Number	Type	PCD (*time*)	C-CcCoA (*time*)	PCD-LD (*time*)	C-MGM (*time*)	C-PMGM (*time*)	C-PDSM (*time*)
$\|A\| = 80$	D1	-555861 (960)	-1111394 (136)	-896276 (952)	**-1355370** (978)	-1004208 (956)	1150880 (948)
	D2	-1197702 (931)	**-2332978** (407)	-2157473 (986)	-2200719 (954)	-1280861 (953)	-1706260 (985)
	D3	-467112 (949)	-1024342 (141)	-774797 (959)	**-1127708** (952)	-880713 (927)	-1065422 (941)
	D4	-608475 (956)	-1254330 (157)	-916351 (946)	**-1552279** (976)	-1157349 (940)	-1327593 (944)
$\|A\| = 90$	D1	-619407 (968)	-1380270 (157)	-1073774 (961)	**-1658509** (998)	-1234988 (983)	-1370201 (995)
	D2	-1237887 (907)	**-2829167** (486)	-2339043 (883)	-2339654 (988)	-1345707 (944)	-1728215 (938)
	D3	-513043 (987)	-1103239 (126)	-828157 (949)	**-1274467** (997)	-988866 (949)	-1171402 (985)
	D4	-610568 (980)	-1340294 (157)	-993353 (966)	**-1647306** (972)	-1200160 (965)	-1299064 (950)
$\|A\| = 100$	D1	-684657 (988)	-1533802 (188)	-1138453 (923)	**-1888946** (992)	-1314583 (942)	-1462948 (979)
	D2	-1197062 (935)	**-3176851** (578)	-2445525 (929)	-2501213 (982)	-1346609 (891)	-1764955 (969)
	D3	-531285 (971)	-1204229 (141)	-868545 (987)	**-1366420** (981)	-1009219 (950)	-1298292 (973)
	D4	-637169 (956)	-1504482 (159)	-1135444 (932)	**-1830023** (954)	-1361162 (989)	-1497704 (942)

6. Conclusion and Future Work

Researchers propose C-DCOPs to model distributed problems with continuous variables as an extension of DCOPs. In this paper, we design three algorithms C-MGM, C-PMGM, and C-PDSM for solving C-DCOPs and prove that they are three anytime algorithms. Since the C-MGM algorithm resets value assignments to expend the search space and chooses the maximum gain, it can obtain high-quality solutions with few iterations. The C-PMGM algorithm extends C-MGM by searching multiple solutions in parallel, which can significantly improve the solution quality. Based on C-PMGM, C-PDSM combines differential search to strengthen the exploitation of solutions, which can effectively improve the convergence speed and the solution quality. Extensive experiments on three benchmark problems show that C-MGM, C-PMGM, and C-PDSM outperform the state-of-the-art C-DCOP algorithms. In future work, we plan to improve the convergence speed of the three algorithms on some dense graphs.

Author Contributions: Conceptualization, Xin Liao and Khoi Hoang; methodology, Xin Liao; software, Xin Liao; validation, Xin Liao; formal analysis, Xin Liao; investigation, Xin Liao; resources, Khoi Hoang; data curation, Xin Liao; writing—original draft preparation, Xin Liao; writing—review and editing, Xin Liao; visualization, Khoi Hoang; supervision, Khoi Hoang; project administration, Khoi Hoang; funding acquisition, Khoi Hoang. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The code and benchmark problems for this paper can be found at https://github.com/chattedlx/MGM-extensions.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Modi, P.; Shen, W.M.; Tambe, M.; Yokoo, M. ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence* **2005**, *161*, 149–180.

2. Petcu, A.; Faltings, B. A Scalable Method for Multiagent Constraint Optimization. In Proceedings of the Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005, IJCAI, pp. 266–271.
3. Yeoh, W.; Yokoo, M. Distributed Problem Solving. *AI Magazine* **2012**, *33*, 53–65.
4. Fioretto, F.; Pontelli, E.; Yeoh, W. Distributed Constraint Optimization Problems and Applications: A Survey. *Journal of Artificial Intelligence Research* **2018**, *61*, 623–698.
5. Maheswaran, R.T.; Tambe, M.; Bowring, E.; Pearce, J.P.; Varakantham, P. Taking DCOP to the Real World: Efficient Complete Solutions for Distributed Multi-Event Scheduling. In Proceedings of the Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems, 2004, AAMAS, pp. 310–317.
6. Di, B.; Jennings, N.R. A Contract-based Incentive Mechanism for Distributed Meeting Scheduling: Can Agents Who Value Privacy Tell the Truth? *Autonomous Agents and Multiagent Systems* **2021**, *35*, 35.
7. Farinelli, A.; Rogers, A.; Jennings, N.R. Agent-Based Decentralised Coordination for Sensor Networks Using the Max-sum Algorithms. *Autonomous Agents and Multiagent Systems* **2014**, *28*, 337–380.
8. Zivan, R.; Yedidsion, H.; Okamoto, S.; Glinton, R.; Sycara, K.P. Distributed Constraint Optimization for Teams of Mobile Sensing Agents. *Autonomous Agents and Multiagent Systems* **2015**, *29*, 495–536.
9. Harel, Y.; Roie, Z. Applying DCOP_MST to a Team of Mobile Robots with Directional Sensing Abilities. In Proceedings of the Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems, 2016, AAMAS, pp. 1357–1358.
10. Fioretto, F.; Yeoh, W.; Pontelli, E.; Ma, Y.; Ranade, S.J. A Distributed Constraint Optimization (DCOP) Approach to the Economic Dispatch with Demand Response. In Proceedings of the Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, 2017, AAMAS, pp. 999–1007.
11. Miller, S.; Ramchurn, S.D.; Rogers, A. Optimal Decentralised Dispatch of Embedded Generation in the Smart Grid. In Proceedings of the Proceedings of the 11st International Conference on Autonomous Agents and Multiagent Systems, 2012, AAMAS, pp. 281–288.
12. Rust, P.; Picard, G.; Ramparany, F. Using Message-Passing DCOP Algorithms to Solve Energy-Efficient Smart Environment Configuration Problems. In Proceedings of the Proceedings of the 25th International Joint Conference on Artificial Intelligence, 2016, IJCAI, pp. 468–474.
13. Fioretto, F.; Yeoh, W.; Pontelli, E. A Multiagent System Approach to Scheduling Devices in Smart Homes. In Proceedings of the Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, 2017, AAMAS, pp. 981–989.
14. Chen, Z.; Deng, Y.; Wu, T. An Iterative Refined Max-sum_AD Algorithm via Single-side Value Propagation and Local Search. In Proceedings of the Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, 2017, AAMAS, pp. 195–202.
15. Chen, Z.; Wu, T.; Deng, Y.; Zhang, C. An Ant-based Algorithm to Solve Distributed Constraint Optimization Problems. In Proceedings of the Proceedings of the 32nd AAAI Conference on Artificial Intelligence, 2018, AAAI, pp. 4654–4661.
16. Hoang, K.D.; Fioretto, F.; Yeoh, W.; Pontelli, E.; Zivan, R. A Large Neighboring Search Schema for Multi-agent Optimization. In Proceedings of the Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming, 2018, CP, pp. 688–706.
17. Khan, M.M.; Tran-Thanh, L.; Jennings, N.R. A Generic Domain Pruning Technique for GDL-Based DCOP Algorithms in Cooperative Multi-Agent Systems. In Proceedings of the Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems, 2018, AAMAS, pp. 1595–1603.
18. Grinshpoun, T.; Tassa, A.; Levit, V.; Zivan, R. Privacy Preserving Region Optimal Algorithms for Symmetric and Asymmetric DCOPs. *Artificial Intelligence* **2019**, *266*, 27–50.
19. Tassa, T.; Grinshpoun, T.; Yanai, A. A Privacy Preserving Collusion Secure DCOP Algorithm. In Proceedings of the Proceedings of the 28th International Joint Conference on Artificial Intelligence, 2019, IJCAI, pp. 4774–4780.
20. Tassa, T.; Grinshpoun, T.; Yanai, A. PC-SyncBB: A Privacy Preserving Collusion Secure DCOP Algorithm. *Artificial Intelligence* **2021**, *297*, 103501.
21. Deng, Y.; Chen, Z.; Chen, D.; Zhang, W.; Jiang, X. AsymDPOP: Complete Inference for Asymmetric Distributed Constraint Optimization Problems. In Proceedings of the Proceedings of the 28th International Joint Conference on Artificial Intelligence, 2019, IJCAI, pp. 223–230.

22. van Leeuwen, C.J.; Pawelczak, P. CoCoA: A Non-Iterative Approach to a Local Search (A)DCOP Solver. In Proceedings of the Proceedings of the 35th AAAI Conference on Artificial Intelligence, 2017, AAAI, pp. 3944–3950.
23. Zivan, R.; Parash, T.; Cohen-Lavi, L.; Naveh, Y. Applying Max-sum to Asymmetric Distributed Constraint Optimization Problems. *Autonomous Agents and Multiagent Systems* **2020**, *34*, 1–29.
24. Deng, Y.; Chen, Z.; Chen, D.; Jiang, X.; Li, Q. PT-ISABB: A Hybrid Tree-based Complete Algorithm to Solve Asymmetric Distributed Constraint Optimization Problems. In Proceedings of the Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems, 2019, AAMAS, pp. 1506–1514.
25. Hoang, K.D.; Hou, P.; Fioretto, F.; Yeoh, W.; Zivan, R.; Yokoo, M. Infinite-Horizon Proactive Dynamic DCOPs. In Proceedings of the Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems, 2017, AAMAS, pp. 212–220.
26. Hoang, K.D.; Fioretto, F.; Hou, P.; Yokoo, M.; Yeoh, W.; Zivan, R. Proactive Dynamic Distributed Constraint Optimization. In Proceedings of the Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems, 2016, AAMAS, pp. 597–605.
27. Hoang, K.D.; Fioretto, F.; Hou, P.; Yeoh, W.; Yokoo, M.; Zivan, R. Proactive Dynamic Distributed Constraint Optimization Problems. *Journal Of Artificial Intelligence Research* **2022**, *74*, 179–225.
28. Gutierrez, P.; Meseguer, P.; Yeoh, W. Generalizing ADOPT and BnB-ADOPT. In Proceedings of the Proceedings of the 22th International Joint Conference on Artificial Intelligence, 2011, IJCAI, pp. 554–559.
29. Yeoh, W.; Felner, A.; Koenig, S. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm. *Journal of Artificial Intelligence Research* **2010**, *38*, 85–133.
30. Vinyals, M.; Rodríguez-Aguilar, J.A.; Cerquides, J. Constructing A Unifying Theory of Dynamic Programming DCOP Algorithms via the Generalized Distributive Law. *Autonomous Agents and Multiagent Systems* **2011**, *22*, 439–464.
31. Maheswaran, R.T.; Pearce, J.P.; Tambe, M. Distributed Algorithms for DCOP: A Graphical-Game-Based Approach. In Proceedings of the Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004, PDCS, pp. 432–439.
32. Ottens, B.; Dimitrakakis, C.; Faltings, B. DUCT: An Upper Confidence Bound Approach to Distributed Constraint Optimization Problems. *ACM Transactions on Intelligent Systems and Technology* **2017**, *8*, 69:1–69:27.
33. Nguyen, D.T.; Yeoh, W.; Lau, H.C.; Zivan, R. Distributed Gibbs: A Linear-Space Sampling-Based DCOP Algorithm. *Journal of Artificial Intelligence Research* **2019**, *64*, 705–748.
34. Stranders, R.; Farinelli, A.; Rogers, A.; Jennings, N.R. Decentralised Coordination of Continuously Valued Control Parameters Using the Max-sum Algorithm. In Proceedings of the Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, 2009, AAMAS, pp. 601–608.
35. Voice, T.; Stranders, R.; Rogers, A.; Jennings, N.R. A Hybrid Continuous Max-Sum Algorithm for Decentralised Coordination. In Proceedings of the Proceedings of the 19th European Conference on Artificial Intelligence, 2010, ECAI, pp. 61–66.
36. Fransman, J.; Sijs, J.; Dol, H.; Theunissen, E.; Schutter, B.D. Bayesian-DPOP for Continuous Distributed Constraint Optimization Problems. In Proceedings of the Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems, 2019, AAMAS, pp. 1961–1963.
37. Choudhury, M.; Mahmud, S.; Khan, M.M. A Particle Swarm Based Algorithm for Functional Distributed Constraint Optimization Problems. In Proceedings of the Proceedings of the 34th AAAI Conference on Artificial Intelligence, 2020, AAAI, pp. 7111–7118.
38. Hoang, K.D.; Yeoh, W.; Yokoo, M.; Rabinovich, Z. New Algorithms for Continuous Distributed Constraint Optimization Problems. In Proceedings of the Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, 2020, AAMAS, pp. 502–510.
39. Sarker, A.; Choudhury, M.; Khan, M.M. A Local Search Based Approach to Solve Continuous DCOPs. In Proceedings of the Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems, 2021, AAMAS, pp. 1127–1135.
40. 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*, 2259025.
41. Zivan, R.; Okamoto, S.; Peled, H. Explorative Anytime Local Search for Distributed Constraint Optimization. *Artificial Intelligence* **2014**, *212*, 1–26.

 doi:10.20944/preprints202401.1158.v1

42. Storn, R.; Price, K.V. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization* **1997**, *11*, 341–359.
43. Paul, E.; Rényi, A. On the Evolution of Random Graphs. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* **1960**, *5*, 17–60.
44. Albert-László, B.; Réka, A. Emergence of Scaling in Random Networks. *Science* **1999**, *286*, 509–512.
45. Newman, M.; Watts, D. Renormalization Group Analysis of the Small-World Network Model. *Physics Letters A* **1999**, *263*, 341–346.