PeerJ Computer Science

A local cost simulation-based algorithm to solve distributed constraint optimization problems

Meifeng Shi[1,2], Feipeng Liang[1], Yuan Chen[1] and Ying He[1]

[1] Department of Computer Science and Engineering, Chongqing University of Technology, Chongqing, China
[2] Faculty of Information Science and Electrical Engineering, Kyushu University, Fukuoka, Japan

ABSTRACT

As an important incomplete algorithm for solving Distributed Constraint Optimization Problems (DCOPs), local search algorithms exhibit the advantages of flexibility, high efficiency and high fault tolerance. However, the significant historical values of agents that affect the local cost and global cost are never taken into in existing incomplete algorithms. In this article, a novel Local Cost Simulation-based Algorithm named LCS is presented to exploit the potential of historical values of agents to further enhance the exploration ability of the local search algorithm. In LCS, the Exponential Weighted Moving Average (EWMA) is introduced to simulate the local cost to generate the selection probability of each value. Moreover, populations are constructed for each agent to increase the times of being selected inferior solutions by population optimization and information exchange between populations. We theoretically analyze the feasibility of EWMA and the availability of solution quality improvement. In addition, based on our extensive empirical evaluations, we experimentally demonstrate that LCS outperforms state-of-the-art DCOP incomplete algorithms.

Subjects Agents and Multi-Agent Systems, Optimization Theory and Computation
Keywords DCOPs, LCS, EWMA, Inferior solutions

Submitted 28 October 2022
Accepted 23 February 2023
Published 17 March 2023

Corresponding author
Ying He, hdj@cqut.edu.cn

Academic editor
Bilal Alatas

Additional Information and Declarations can be found on page 21

DOI **10.7717/peerj-cs.1296**

INTRODUCTION

Multi-agent system (MAS) (*Ferber & Weiss, 1999*) is a computing system composed of multiple autonomous agents that can interact with each other. It is an important research area of distributed artificial intelligence. The Distributed Constrained Optimization Problems (DCOPs) (*Modi et al., 2005*) have emerged as one of the main coordination techniques in MAS. It is based on the constrained relationship between agents to make decisions and select a set of values with minimum constraint cost. DCOPs have been successfully applied in real-world coordination tasks, such as sensor networks (*Farinelli, Rogers & Jennings, 2014*), task scheduling (*Sultanik, Modi & Regli, 2007*), power networks (*Fioretto et al., 2017*), *etc.*

Algorithms for DCOPs can be classified into complete algorithms that guarantee to find of the optimal solution and incomplete algorithms that can find the approximate optimal solution in a short time. The complete algorithms can be roughly classified into the search-based algorithm (*Gershman, Meisels & Zivan, 2009*) by traversing the solution space and inference-based algorithm (*Petcu & Faltings, 2005*; *Petcu & Faltings, 2007*) by

performing variable elimination. Complete algorithms based on search mainly include SynchBB (*Hirayama & Yokoo, 1997*), AFB (*Gershman, Meisels & Zivan, 2009*) and ADOPT (*Modi et al., 2005*). Among them, SynchBB is a synchronous algorithm, which uses the idea of branch delimitation to solve DCOP, mainly uses the boundary value to prune the solution space and then compresses the space to find the optimal solution. AFB is an asynchronous algorithm, which is an improved version of SynchBB. In AFB, the agent assigns values to variables sequentially and transmits the partial resolution and the cost of the partial solutions. By receiving the information, the agent asynchronously calculates the upper and lower bounds of the current partial solution, so that it can judge whether backtracking is needed as early as possible, which improves the pruning efficiency of the algorithm. ADOPT adopts a depth-first search tree as a communication structure, and uses a best-first search strategy to perform distributed backtracking. The main representative of the complete algorithm based on inference is DPOP (*Petcu & Faltings, 2005*), which uses pseudo-tree (*Freuder & Quinn, 1985*) as the communication structure, calculates and transmits the utility corresponding to the assignment combination from the bottom to top, and obtains the optimal solution through the elimination operation of the joint utility tables. Since DCOPs are NP-Hard (*Modi et al., 2005*), the computational overheads of complete algorithms will increase exponentially as the scale of the problem increases, which limits their application in practical engineering.The advantage of low computational overheads of incomplete algorithms makes them more popular in large-scale applications.

Incomplete algorithms are generally classified into three categories: local search algorithms, inference-based algorithms, and sampling-based algorithms. Inference-based algorithms mainly include Max-Sum (*Farinelli et al., 2008*), Bounded Max-sum (*Rogers et al., 2011*), Max-sum_ADVP (*Zivan et al., 2017*) and Damped Max-sum (*Cohen & Zivan, 2017*). In Max-sum, the agent optimizes the goal by passing messages in the factor graph (*Aji & McEliece, 2000*) to continuously accumulate beliefs. Bounded Max-sum transforms the original problem into an acyclic graph by removing the relationship edges that have the least impact on the solution quality in the cyclic factor graph. The problem is then solved using the Max-sum algorithm and an approximation ratio is calculated to improve the quality of the solution. Max-sum_ADVP solves the repeated utility problem by adding its own fetch values to the information sent by each variable node and eliminates the assumption of invalid value. Damped Max-sum converts the standard factor graph into an equivalent split-constrained factor graph to control the asymmetry of each constraint, in order to achieve the purpose of balancing exploration and utilization.

Sampling-based algorithms include DUCT (*Ottens, Dimitrakakis & Faltings, 2012*), D-Gibbs (*Nguyen, Yeoh & Lau, 2013*), SD-Gibbs (*Nguyen et al., 2019*) and PD-Gibbs (*Nguyen et al., 2019*) that take statistical samples of the value context to solve DCOPs by constructing another structure. Among them, DUCT applies sampling and confidence bounds to solve DCOP. The main idea of D-Gibbs is to transform DCOP into a maximum likelihood estimation problem in Markov random fields. SD-Gibbs and PD-Gibbs have a linear-space memory requirement and solve some large memory limitation problems.

The local search algorithms are the most popular ones because of their simple logic, flexibility and effectiveness. Local search algorithms mainly include DSA (*Zhang et al.,*

2005), MGM (Maheswaran, Pearce & Tambe, 2004), DSAN (Arshad & Silaghi, 2004), MGM-2 (Maheswaran, Pearce & Tambe, 2006), MGM-3 (Maheswaran, Pearce & Tambe, 2006; Leite, Fabrício & Jean-Paul, 2014) and GDBA (Okamoto, Zivan & Nahon, 2016). DSA records the current local cost according to the values of neighbors and selects the value that can reduce the local cost to the greatest extent according to the probability. MGM considers the local cost of neighbors, and the agent with the largest local gain can change the value. DSAN introduces the dynamic probability idea of the simulated annealing method to jump out of the local optimum. Algorithms such as MGM-2 and MGM-3 use the idea of k-optimal (Bowring et al., 2008), agents cooperate locally to improve the quality of local solutions, but the larger k is, the greater the time complexity will be. GDBA extends DBA (Zhang et al., 2005) to DCOP, stipulates concepts such as effective cost, constraint violation and modified matrix range, and has achieved good results. In the iterative process of the local search algorithm, although the total cost of the solution shows a downward trend, it does not decrease monotonically. After falling to the lowest value, since the agent does not know the current global cost state, the agent will continue to change the value according to the local strategy. Therefore, in order to record the optimal solution in the iterative process and make the algorithm show anytime effect, the Anytime Local Search Framework (Zivan, Okamoto & Peled, 2014) was proposed. The framework relies on a breadth-first tree structure to record the global state. In addition, PDS (Yu et al., 2017) also proposes a partial decision-making framework. It is able to apply local search algorithms and improve the quality of local search algorithm solutions by local decision mechanisms.

Recently, population-based methods and frameworks have emerged and can achieve high-quality solutions. ACO_DCOP (Chen et al., 2018) adapts the traditional Ant Colony Optimization to solve DCOPs by the pheromone mechanism ameliorated. ACO_DCOP defines pheromone as global cost and heuristic information as local cost, and the agent combines the information from both parts to make a decision. A genetic algorithm-based (LSGA) (Chen et al., 2020) framework uses genetic codes to crossover variation the assignment combination.

However, as the iterative trace of the algorithm, the historical values of agents are never considered in the previously mentioned algorithms. To study the influence of the historical values of agents on local search algorithms, a novel Local Cost Simulation-based algorithm named LCS is proposed by combining the Exponentially Weighted Moving Average (EWMA) and population optimization technology, which improves the development of solutions and exploration capabilities. First, EWMA is used to simulate the local cost to generate the selection probability of each value. Then, populations are constructed for each agent to increase the selection times of inferior solutions. Finally, to break out of the local optima to explore other solution spaces, the population interaction is introduced into LCS. We theoretically analyze the feasibility of EWMA, availability of solution quality improvement of the proposed LCS. We also experimentally show the superior performance of LCS over the state-of-the-art DCOPs incomplete algorithms based on our extensive empirical evaluations. Specically, our contributions can be summarized as follows:

(a) We use EWMA to model historical local costs, which provides an effective idea for using historical values to estimate the local cost.

(b) We design a probability-based strategy so that values with lower local cost simulation values have a higher probability of being selected.
(c) We design a population cooperation mechanism to improve the exploitation and exploration ability of LCS.

BACKGROUND & RELATED WORK

Distributed constraint optimization problems

A DCOP can be defined as a tuple (A, X, D, F) (*Chen et al., 2020*) where.

- $A = \{a_1, \ldots, a_n\}$ is a set of agents.
- $X = \{x_1, \ldots, x_m\}$ is a set of variables, where each variable is assigned to an agent.
- $D = \{D_1, \ldots, D_m\}$ is a set of finite variable domains, where the values of the variables are taken from finite domains $D_1, \ldots, D_m$.
- $F = \{f_1, \ldots, f_q\}$ is a set of constraints, where each constraint $f_i : D_{i_1} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}_{\geq 0}$ specifies a non-negative cost for every possible value combination of a set of variables.

For ease of understanding and discussion, we assume that one agent controls only one variable and all constraints are binary. Therefore, in this article, agent and variable can be considered as the same concept and can be replaced with each other. Figure 1 shows a simple DCOP example.

The solution objective of DCOP is to find an assignment X^* that minimizes the sum of constraint costs arising between all agents.

$$X^* = \underset{d_i \in D_i, d_j \in D_j}{\operatorname{argmin}} \sum_{f_{ij} \in F} f_{ij}(x_i = d_i, x_j = d_j). \tag{1}$$

Local search framework for DCOPs

The local search algorithms are typical incomplete algorithms for solving DCOP problems. In each round, an agent sends the message that includes value or other information to neighbors in the constraint graph and receives the messages from neighbors. Then, the agent will select a new value in terms of the messages and decide whether to replace the old one according to different replacement strategies. There are different ways to create messages in different algorithms. In DSA, when the value of neighbors changes, the agent searches its domain to find a value that can reduce local cost, and replaces the old value using the new one with a probability. But in MGM, only the agent with maximal gain among all its neighbors can replace old value. In this article, DSA is used to illustrate this framework.

Table 1 presents the sketch of DSA. Firstly, the agent initializes a value randomly and sends it to all neighbors (line 1–2). Then, the algorithm executes a repeated iterative loop (line 3–9) until the termination condition is met. During the loop, an agent collects the values received from neighbors and selects a new value that reduces the local cost most (line 4–5). Finally, the agent decides to assign the new or the old value according to a probability p (line 7–9). If ($\Delta \geq 0$ and$p \leq$ random()), assign the new value. Else, assign the old value.

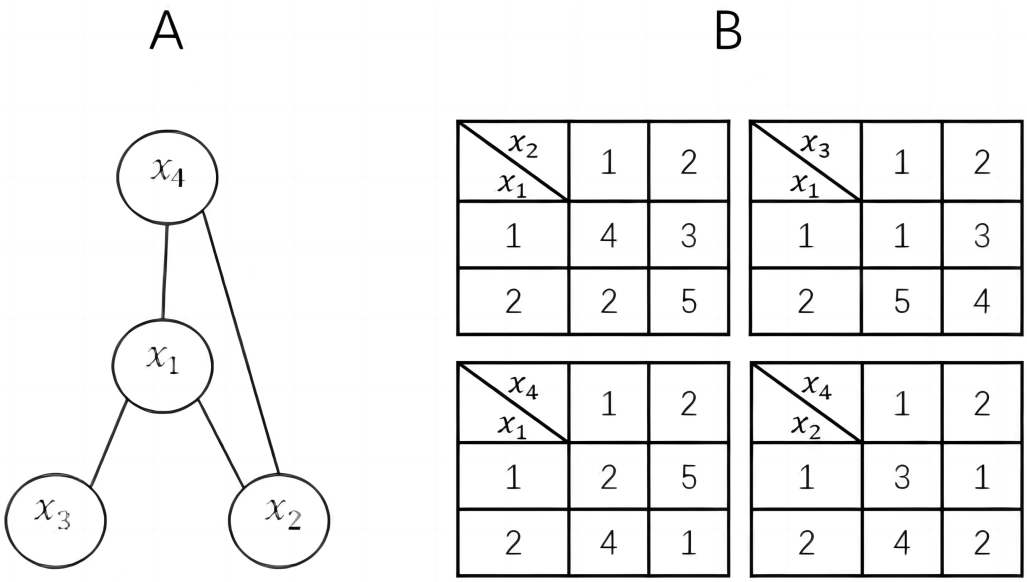

Figure 1 **A DCOP instance.** (A) constraint graph; (B) constraint matrices.
Full-size DOI: 10.7717/peerjcs.1296/fig-1

Related work

Most of incomplete algorithms for DCOPs are context-free (*Deng et al., 2021*). The local search algorithms (*Chapman et al., 2011*) is defined as: at each iteration, agents communicate only the states of the variables under their control to their neighbours on the constraint graph, and that reason about their next state based on the messages received from their neighbours.

In the most typical local search algorithm DSA, an agent sends its value to its neighbors, and at the same time receives the value sent from neighbors. Then, according to the received value, each agent chooses a value from its variable domain that can minimize the local cost to replace the old one with probability. If the the old value is replaced, the agent will resend it to its neighbors. In another typical algorithm, MGM, all agents need to send additional gain messages to their neighbors. Then through the message passing mechanism, only the agent with the largest gain among all neighbors can replace the old value. GDBA extends the DBA algorithm to the general DCOP problem and spans 24 combinations of three design choices. The three designs are strategies to modify the basic cost using multiple weights or additional penalties, definitions of constraint violations such as non-zero cost, non-minimal cost and maximum cost, and modifying the scope of the cost table like entries, rows, columns, or tables during breakthrough. DSAN is an improved version of DSA by applying simulated annealing to make DSA jump out of the local optimum.

The LCS proposed in this article mainly utilizes the historical information that current local search algorithms ignore. The exponential weighted moving average method is used to maintain and update the local cost simulation value, and the population cooperation mechanism is used to fully search the local cost simulation value, thereby improving the quality of the solution.

Table 1 The sketch of DSA.

Algorithm 1 Distributed Stochastic Algorithm

For each agent x_i executes:

1. value $\leftarrow$ *Choose_Random_Value*()
2. send value to neighbors
3. **while** (no termination condition is met)
4. collect neighbors' value
5. select a new value which reduces the local cost most
6. $\Delta \leftarrow$ the number of the local cost reduced by the new value
7. **if** ($\Delta \geq 0 and random() \leq p$) **then**
8. assign the new value
9. send value to neighbors

PROPOSED METHOD

Motivation

In local search algorithms, only two successive rounds of information are taken into account and the historical information from previous rounds is wasted. For the DSA, the current value depends on the local cost difference between the current round and the previous round. That is, the value only depends on changes in a single round of neighbors' values. The local cost is non-increasing in this strategy, and it is easy to fall into a local optimum. MGM follows the main idea of DSA, although the interactive competition between neighbors has been added to determine the value changes. For DSAN (*Arshad & Silaghi, 2004*), only the idea of Simulate Anneal Arithmetic (SAA) is introduced to control the search process to disturb the assignment probability when the algorithm falls into the local optima. The same is true for PDS, LSGA, *etc.* All of them make no use of the series of changes in local cost and its historical records of values.

In fact, for the local search algorithms, the purpose of the agent to change values is generally to reduce the local cost. At the time of local cost decreases, global cost always performs a downward trend as rounds increase. Various improved local search algorithms all focus on the conflict between local cost and global cost because when the global cost decreases while the local costs of some agents increase.

These improvement strategies are committed to making algorithms break away from the monotonic decline of local cost to jump out of the local optima and search for higher-quality solutions. All of these improved algorithms have not taken into account the change records of the local cost. The sum of the local costs of all agents is equal to the global cost of the system. It can be seen that the change of global cost is closely related to that change of the local cost. Suppose that the estimated local cost of each value is known, we can select the lower one to get a lower global cost. There are different local costs and different changes for different values. Thus, it is necessary to estimate the local cost for each value in the domain according to the historical value of agents to guide their assignments. Based on this, we propose a novel algorithm named LCS to solve DCOP based on the idea of local cost simulation.

Table 2 The sketch of LCS.

Algorithm 2 The LCS Algorithm

```
Initialize Parameters: C, P, α, β, γ, ecy
foreach c_k ∈ C do
    foreach d_i ∈ D_i do
        calculate est^0_{c_k}(d_i) by formula (1)
StartCycle()
Function StartCycle:
    foreach c_k ∈ C, p ∈ P do
        V_{c_k,p} ← randomly selects a value from D_i
        V_i ← V_i ∪ V_{c_k,p}
    Sends ValueMessage({i, V_i}) to neighbors
When received Value(m, recv_V_m):
    update local view with (m, recv_V_m)
    foreach c_k,p do
        calculate lc_{c_k,p} by formula (2)
    foreach c_k ∈ C do
        foreach d_i ∈ D_i do
            update est_{c_k}(d_i) by formula (3)
    if (current_cycle % ecy = 0) then
        foreach c_k ∈ C do
            foreach d_i ∈ D_i do
                exchange est_{c_k}(d_i) by formula (4)
    SelectValue()
Function SelectValue:
    clear V_i
    foreach c_k,p do
        V_{c_k,p} ← selects a value by formula (5)
        V_i ← V_i ∪ V_{c_k,p}
    Sends ValueMessage({i, V_i}) to neighbors
```

LCS

LCS consists of two phases: initialization and local cost simulation. We modify the existing ALS framework to make it compatible with the population of LCS. The sketch of LCS can be found in Table 2. Each value in the agent value range has a local cost simulation value that needs to be maintained, and the simulation value is the simulation value of the historical local cost generated when the value in the value range is selected. When the value of a_i is d_i, the local cost simulation value of d_i is denoted as est(d_i).

Definition 1: An individual is a value with a serial number. Individuals with the same serial number in all agents represent a solution. A population is a collection of individuals with its own unique local cost simulation.

Definition 2: $est_{c_k}(d_i)^r$ is the simulation (or estimation) of the local cost that $d_i \in D_i$ will produce. Where $c_k \in C$ is the serial number of the population, and r is the number of simulation updates.

Initialization initiates the simulation and assigns values to individuals. Suppose that there are C populations, each of which has P individuals (C and P are the hyper-parameters). Since the algorithm has not yet started, we create the same initial simulation for all populations. For agent a_i, the initial simulations (estimations) can be calculated by:

$$est^0_{c_k}(d_i) = \sum_{m \in N_i} \max_{d_j \in D_m} cost(d_i, d_j) \tag{2}$$

where $est^0_{c_k}(d_i)$ is the initial estimation for $c_k \in C$. The estimation can simulate previous local costs and be updated according to the change of the local cost. It reflects the quality of d_i and records the change of neighbors' values. Here it represents the worst case for d_i. And N_i is the neighbor of a_i, $cost(d_i, d_j)$ isthe constraint cost generated when a_i takes the value of d_i, and its neighbor a_j takes the value of d_j.

Taking Fig. 1 for an example, we assume that there are two populations, with two individuals in each population respectively. The estimations for x_1 should be calculated by:

$$est^0_{c_1}(1) = est^0_{c_2}(1) = 4+3+5 = 12$$
$$est^0_{c_1}(2) = est^0_{c_2}(2) = 5+5+4 = 14$$

After the estimations are initialized, each agent executes the StartCycle to start message-passing. The agent constructs populations and individuals to save values per round. We use V_i to represent the value set of populations for a_i. At the first round, individual $p \in P$ of population $c_k \in C$ selects an initial value $V_{c_k,p}$ in the domain for itself randomly. After the selection of all individuals, the agent sends the set V_i with values to all neighbors.

For example in Table 3, after initialization, the individuals in each agent are randomly assigned.

Local cost simulation is a procedure to search for solutions. In this phase, the agent communicates continuously with its neighbors, calculates the local cost to update the simulated value, conducts population interaction to better simulate the local cost, and conducts a sufficient search for the potential of the value.

After the agent receives the sets V_m of neighbors, LCS needs to update simulations by local cost. The local cost for each individual can be calculated by:

$$loc_{c_k,p} = \sum_{m \in N_i} cost_{i,j}\left(V_{c_k,p,i}, V_{c_k,p,m}\right) \tag{3}$$

where $loc_{c_k,p}$ is the local cost which is calculated by individual p of population c_k and the same individuals of neighbors.

Continuing the example above, local costs for x_1 can be calculated using:

$$loc_{c_1,p_1} = 3+3+2 = 8, loc_{c_1,p_2} = 4+3+5 = 12$$
$$loc_{c_2,p_1} = 3+1+5 = 9, loc_{c_2,p_2} = 2+4+4 = 10.$$

Table 3 Initialization example for population.

	V_{c_1,p_1}	V_{c_1,p_2}	V_{c_2,p_1}	V_{c_2,p_2}
V_1	1	1	1	2
V_2	2	1	2	1
V_3	2	2	1	2
V_4	1	2	2	1

When the local cost of each individual has been obtained, the agent needs to update the estimation to simulate its change. Considering the temporal relationship between local cost and round, we use the EWMA to simulate the local cost. The updating method of estimation is defined as:

$$\mathrm{est}_{c_k}^{r}(d_i) = \mathrm{est}_{c_k}^{r-1}(d_i) * \beta_{c_k} + \mathrm{loc}_{c_k,p} * \left(1 - \beta_{c_k}\right) \tag{4}$$

where β_{c_k} is the decay rate of population c_k, and r is the number of updates.

Here, we assume that $\beta_{c_1} = 0.5, \beta_{c_2} = 0.6$.$\mathrm{est}_{c_k}^{0}$can be found at formulation 1. The estimations for x_1 can be calculated as:

$$\mathrm{est}_{c_1}^{1}(1) = 12 * 0.5 + 8 * 0.5 = 10$$
$$\mathrm{est}_{c_1}^{2}(1) = 10 * 0.5 + 12 * 0.5 = 11$$
$$\mathrm{est}_{c_2}^{1}(1) = 12 * 0.6 + 9 * 0.4 = 10.8$$
$$\mathrm{est}_{c_2}^{1}(2) = 14 * 0.6 + 10 * 0.4 = 12.4$$

After the update, the simulations of x_1 become: $\mathrm{est}_{c_1}(1) = \mathrm{est}_{c_1}^{2}(1) = 11, \mathrm{est}_{c_1}(2) = \mathrm{est}_{c_1}^{0}(2) = 14, \mathrm{est}_{c_2}(1) = \mathrm{est}_{c_2}^{1}(1) = 10.8, \mathrm{est}_{c_2}(2) = \mathrm{est}_{c_2}^{1}(2) = 12.4$. If there is no superscript on $\mathrm{est}_{c_k}(d_i)$, it means $\mathrm{est}_{c_k}(d_i)$ is the current latest value.

After each population has updated its simulated values, a new estimate of the local cost that is closer to the actual local cost is obtained. At this time, in order to give full play to the advantages of the population, LCS sets up an exchange round, allowing the populations to share the simulated values they have found, which can tell those populations with poor simulated values that they have better values and better directions. to explore. The LCS pre-sets the exchange interval rounds, and every ecy rounds, the agent will exchange the obtained analog values. The purpose of the exchange operation is to share different search directions between populations. The exchange is defined as:

$$\mathrm{est}_{c_k}(d_i) = \mathrm{est}_{c_k}(d_i) * (1 - \gamma) + \min_{c \in C} \mathrm{est}_c(d_i) * \gamma \tag{5}$$

where γ is the learning rate. It can affect the reduction of estimation, by which the population can evolve to the best one.

Since just update, $\min_{c \in C} est_c(1) = est^1_{c_2}(1) = 10.8$, $\min_{c \in C} est_c(2) = est^1_{c_2}(2) = 12.4$. We assume if $\gamma = 0.5$, the exchange values for x_1 can be calculated by:

$$est_{c_1}(1) = 11 * 0.5 + 10.8 * 0.5 = 10.9$$
$$est_{c_1}(2) = 14 * 0.5 + 12.4 * 0.5 = 13.2$$
$$est_{c_2}(1) = 10.8 * 0.5 + 10.8 * 0.5 = 10.8$$
$$est_{c_2}(2) = 12.4 * 0.5 + 12.4 * 0.5 = 12.4$$

After being updated and exchanged, each individual executes a new round to select values. The agent needs to calculate the probability for each population since the estimations in populations are different. Reciprocal of estimation is used to get the probability. The probability for selecting value d_i is defined as:

$$prb_{c_k}(d_i) = \frac{\left(1/est_{c_k}(d_i)\right)^{\alpha}}{\sum_{d'_i \in D_i}\left(1/est_{c_k}(d'_i)\right)^{\alpha}} \tag{6}$$

where α is a parameter that significantly affects the quality of the solution by enhancing the selected probability of d_i. Because the range of estimation for different problems is distinct, it is necessary to set different α. Assume $\alpha = 8$ here, the probability for c_1 of x_1 can be calculated by:

$$prb_{c_1}(1) = \frac{(1/10.9)^8}{(1/10.9)^8 + (1/13.2)^8} = 0.822$$
$$prb_{c_1}(2) = \frac{(1/13.2)^8}{(1/10.9)^8 + (1/13.2)^8} = 0.178.$$

After finishing the new $V_{c_k,p}$ selection, the agent sends the new $V_i$to neighbors again. At this time, one round is completed and the agent repeats the local cost simulation until all rounds are over. Because of the ALS, the optimal solution is $x_1 = 1, x_2 = 2, x_3 = 2, x_4 = 1$, which gets the lowest cost 8 in this round.

THEORETICAL ANALYSIS

Theory

In this section, we prove the feasibility of EWMA, the availability of solution quality improvement, and the superiority of population interaction.

Lemma 1. EWMA will approach the actual local cost as the rounds go infinity.

Proof. Exponential weighted moving average means that the weighted coefficient of value decreases exponentially as time and can simulate the average value of a time series. The closer value is to the current moment, the greater the weighted coefficient is. In LCS, with the increasing of rounds the influence of previous local costs decreases, the average value obtained by EWMA can put less attention on it and focuses on the local costs in recent rounds. EWMA can alter the decay rate β to emphasize different rounds, the expression is:

$$v_t = \beta * v_{t-1} + (1 - \beta) * \theta_t \tag{7}$$

when $v_0 = 0$, there are:

$$v_t = (1-\beta)\left(\theta_t + \beta\theta_{t-1} + \beta^2\theta_{t-2} + \cdots + \beta^{t-1}\theta_1\right). \tag{8}$$

From formula (8), θ corresponds to the local cost, the weighting coefficient corresponding to θ_1 is β^{t-1}. When t is large, the coefficient value is close to 0 and decreases exponentially. The weighting coefficient corresponding to θ_t is 1, and the weight ratio is relatively large. There are multiple populations in the LCS, and different β can be set for the population. Both focus on different stages and control the diversity of searches.

Proposition 1. The quality of solutions improves as the number of message rounds increases before convergence.

Proof. What is the idea of LCS? The relationship between global cost and local cost is:

$$glc = \frac{1}{2}\sum_{a_i \in Agent} loc_{a_i} \tag{9}$$

where glc is the global cost, loc_{a_i} is the local cost of agent a_i. The function of LCS is to approach loc_{a_i}:

$$\lim_{r\to\infty} est^r_{c_k}(d_i) = loc(d_i) \tag{10}$$

We assume that the algorithm converges at the zth round. According to the converse method, we assume that the quality of the algorithm solution decreases as the number of iteration rounds increases. Then the quality of the algorithm solution at round u is less than round u − x, x is the number of rounds in which the algorithm falls into local optimum. To simplify the proof, we assume that $x = 1$. $(1 < x < u < z)$

$$est^u_{c_k}(d_i) - est^{u-1}_{c_k}(d_i) < 0 \tag{11}$$

$$est^{u-1}_{c_k}(d_i) * \beta_{c_k} + loc_{c_k,p} * \left(1-\beta_{c_k}\right) - est^{u-1}_{c_k}(d_i) < 0 \tag{12}$$

$$loc_{c_k,p} * \left(1-\beta_{c_k}\right) - est^{u-1}_{c_k}(d_i) * \left(1-\beta_{c_k}\right) < 0 \tag{13}$$

$$(loc_{c_k,p} - est^{u-1}_{c_k}(d_i)) * \left(1-\beta_{c_k}\right) > 0 \tag{14}$$

where $est^u_{c_k}$ is is the simulated value of the local cost of the uth round, $loc_{c_k,p}$ is the local cost of the corresponding round.

Formula (11) is the inequality obtained from the assumption. Bring formula (4) to get formula (12). Merge similar terms to formula (13) and according to formula (10), the value of $loc_{c_k,p}$ is greater than $est^{u-1}_{c_k}$, then the assumption does not hold. Therefore, the quality of solutions improves as the number of message rounds increases before convergence.

Complexity

In this section, we analyze the complexity of LCS. We define the number of agents $|A| = n$, the number of neighbors of agent x_i is $|N|$.

In the initialization and local cost simulation, only population messages need to be sent per round, so the size of the message is $O(|C| * |P|)$, where $|C|$ is the number of populations and $|P|$ is the number of individuals per population.

The time complexity of LCS mainly depends on the update of simulations and the calculation of value messages. When updating the simulations, it is necessary to calculate

the local cost for each individual. When calculating local cost, one individual requires $O(|N|)$ complexity. So, complexity for all populations requires $O(|C| * |P| * |N|)$. For the calculation of the value message, the agent just needs to traverse the simulations to calculate the probability. The calculation of probability and selection of value can be done sequentially. So, the time complexity of the calculation of the value message is $O(|C| * \max(|P|, |M|))$, where $|M|$ is the length of the domain. In summary, the time complexity of LCS is $O(n)$.

EMPIRICAL EVALUATIONS

Benchmark problems

We empirically evaluate our proposed method with peer algorithms including DSA (type-C and $p = 0.6$), MGM, GDBA, PDS, ACO_DCOP ($K = 19$), Max-sum_ADVP, Damped Max-sum and LSGA. DSA, MGM and GDBA are implemented under the ALS framework. For PDS and LSGA, we choose PDS-DSA and LSGA-DSA ($M = 18$) who perform the best. And LCS is run on ALS framework to record the optimal solution during the iterations.

Our extensive empirical evaluations are benchmarked on four types of problems including random DCOPs, scale-free networks (SFN), weighted graph coloring problems (WGC) and random meeting scheduling problems (*Chen et al., 2020*). The recommended values of α are shown in Table 4. For random DCOPs, we set the agent number to 70 and 120, domain size to 10, choose the constraint uniformly from [1,100], and consider the graph density 0.1 (for sparse) or 0.6 (for dense). For SFN, we set the agent number to 150, domain size to 10, choose the constraint uniformly from [1,100], and consider the problems with $m_1 = 20$, $m_2 = 3$ (for sparse) and $m_1 = 20, m_2 = 10$ (for dense). For WGC, we set the agent number to 120, the available color number to 3, constraint density to 0.05, and choose the constraint uniformly from [1,100]. For random meeting scheduling problems, we set the agent number to 90, the meeting number to 20, the available time-slots to 20, and travel times are randomly selected from 6 to 10. The experimental results take the average of 50 independent problems that each execute 30 times. We run our experiments on a laptop with an Intel Core i7-6700 CPU 2.60 GHz and 8 GB RAM.

Memory footprints & network load

In terms of memory footprints, we set different population sizes to test on random DCOPs and scale-free networks. We take the maximum value of memory footprints for multiple runs of the algorithm. Table 5 shows that population sizes, density and the agent number are all proportional to memory footprints. In terms of network load (*Nguyen et al., 2019*), that is, the amount of information passed around the network, LCS send a polynomial amount of information in each iteration.

Influence of population size

In order to discuss the influence of population size on the performance of the LCS, we ensure that the parameters α, β, γ of LCS are constant and use many different combinations of population sizes of $C = 1, 2, 3, 4, 5, P = 8, 16, 24, 30$. We choose sparse random graphs of 70 agents and the obtained results are shown in Table 6. From the analysis of the results,

Table 4 The recommended value for α.

Problems	α
DCOP_70_0.1	11
DCOP_70_ 0.6	32
DCOP_120_0.1	15
DCOP_120_0.6	43
SFN_150_20_3	10
SFN_150_20_10	20
WGC_120_0.05	2
Meeting scheduling	14

Table 5 The memory footprints of LCS with different population sizes on random DCOPs and SFN (memory footprints(MB)).

C*P	random DCOPs				SFN	
	A=70		A=120		A=150	
	sparse	dense	sparse	dense	sparse	dense
4*8	66.9	165.7	102.5	500.1	154.8	403.8
4*16	68.7	172.8	107.4	539.4	162.9	466.5
4*24	68.9	174.2	122.6	572.6	166.9	498.3
4*30	69.1	190.2	142.6	583.2	175.0	589.6

Table 6 Comparison of different C and P on random DCOPs ($|A| = 70, p = 0.1$) (cost/time (ms)).

P/ C	8	16	24	30
1	5970/858	5481/896	5405/924	5383/932
2	5416/894	5327/949	5300/947	5289/951
3	5331/899	5277/983	5258/1001	5250/1077
4	5287/917	5261/995	5240/1093	5234/1197
5	5269/930	5250/1077	5232/1231	5228/1441

the higher the number of populations C and the number of individuals P is set, the higher the solution quality of the algorithm is, and the running time of the algorithm increases. However, after the solution quality decreases to a certain level, continuing to increase C and P only results in a small gain. At this point, the increasing trend of the algorithm running time becomes larger and the gain ratio obtained is not high. Therefore, according to the results of the table, $C = 4, P = 24$ is chosen to weigh the solution quality and running time.

Influence of population learning rate

In order to discuss the influence of population learning rate on the performance of the LCS, we set different population learning rate γ and the obtained results are shown in Table 7. We still choose sparse random graphs of 70 agents and $\gamma = 0, 0.1, 0.3, 0.5, 0.7, 0.9$. And to make the population emphasize different phases, $\beta = 0.9, 0.8, 0.7, 0.6$ are selected, but you can also try other values. By analyzing the experimental results, when $\gamma = 0$, no

Table 7 Comparison of different on random DCOPs ($|A| = 70, p = 0.1$).

γ	0	0.1	0.3	0.5	0.7	0.9
cost	5302	5270	5262	5249	5240	5245

value exchange between populations was performed at this time. Comparison with the cost of the algorithm after the introduction of the population learning rate γ shows poor results. The effectiveness of the algorithm was improved after the introduction of γ, which confirmed the effectiveness of population cooperation. According to the result, $\gamma = 0.7$ performs the best solution.

Influence of population cooperation mechanism

In this section, we evaluate the influence of the population cooperation mechanism. We call the LCS without population cooperation mechanism as LCS-WPCM. We set the same parameter for both algorithms as $C = 4$, $P = 24$, $\beta = \{0.9, 0.8, 0.7, 0.6\}$ and $\gamma = 0.7$.

Figures 2 and 3 show the convergence curve of LCS-WPCM and LCS on sparse and dense random graphs with 70 agents. From Figs. 2 and 3, we can see that whether in sparse graph or dense graph, population cooperation mechanism can improve the convergence quality of LCS.

Comparisons with the state-of-the-art algorithms

In this section, LCS was compared with the comparison algorithm on the four Benchmark problems. According to the previous analysis, we set the parameters of LCS as $C = 4$, $P = 24$, $\beta = \{0.9, 0.8, 0.7, 0.6\}$ and $\gamma = 0.7$. The Max-sum_ADVP and the Damped Max-sum for our comparison experiments are non-anytime, so we give the experimental results separately.

Figures 4–6 show the comparison between LCS and other algorithms in random DCOPs ($|A| = 70$, $p = 0.1$; $|A| = 70$, $p = 0.6$), and Table 8 gives the comparisons of all other problems. All differences are statistically significant for $p - \text{value} < 1 \times 10^{-36}$. Compared to the other algorithms, LCS improves the solution quality on sparse problems by 2.2% $\sim$ 21.8% and on dense problems by 0.7% $\sim$ 5.7%. By analyzing the decline curve of LCS, it can be found that some more twists and turns are caused by multiple search phases and population interaction. In the initial phase of iteration, LCS decreases more slowly than LSGA-DSA or ACO_DCOP because estimations have not been simulated well. But after a full search, LCS performs better than all other algorithms and cost shows a slow downward trend all the time. LCS has good search ability for both sparse and dense problems, while the other algorithms are biased to only one of them and cannot catch up with the LCS algorithm.

Figures 7–9 show the comparison between LCS and other algorithms in random DCOPs ($|A| = 120$, $p = 0.1$; $|A| = 120$, $p = 0.6$). Compared to the other algorithms, LCS improves the solution quality on sparse problems by 1.5% $\sim$ 19.3% and on dense problems by 0.4% $\sim$ 4.0%. Population-based LSGA-DSA and LCS with population achieved better results than other algorithms, but LSGA-DSA results were not as good as LCS, which proves that the local cost of simulation in DCOP is better than genetic algorithm coding.

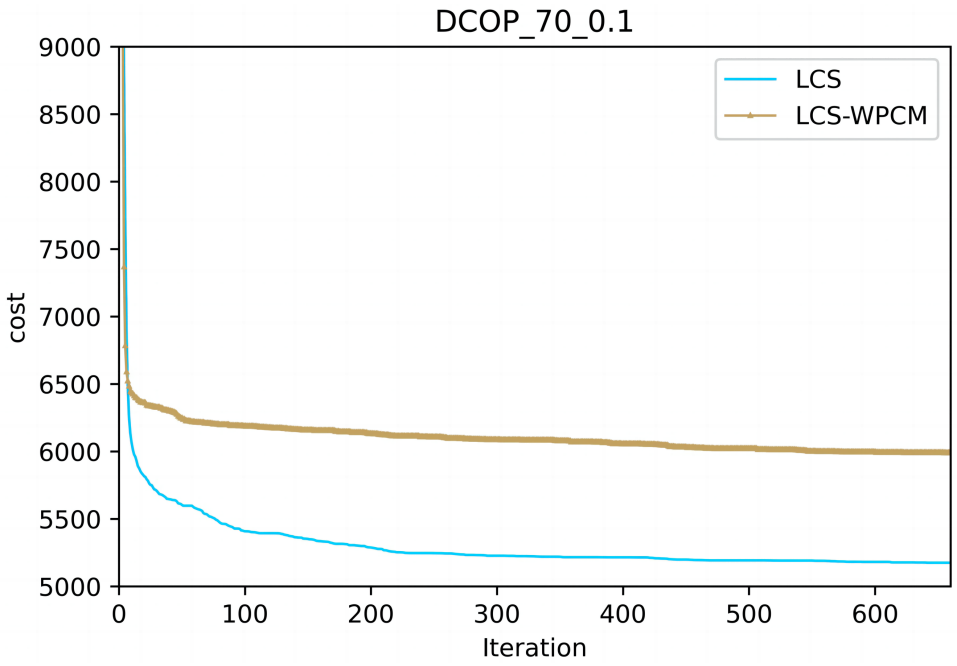

Figure 2 **Sparse random graphs ($|A| = 70, p = 0.1$).**
Full-size DOI: 10.7717/peerjcs.1296/fig-2

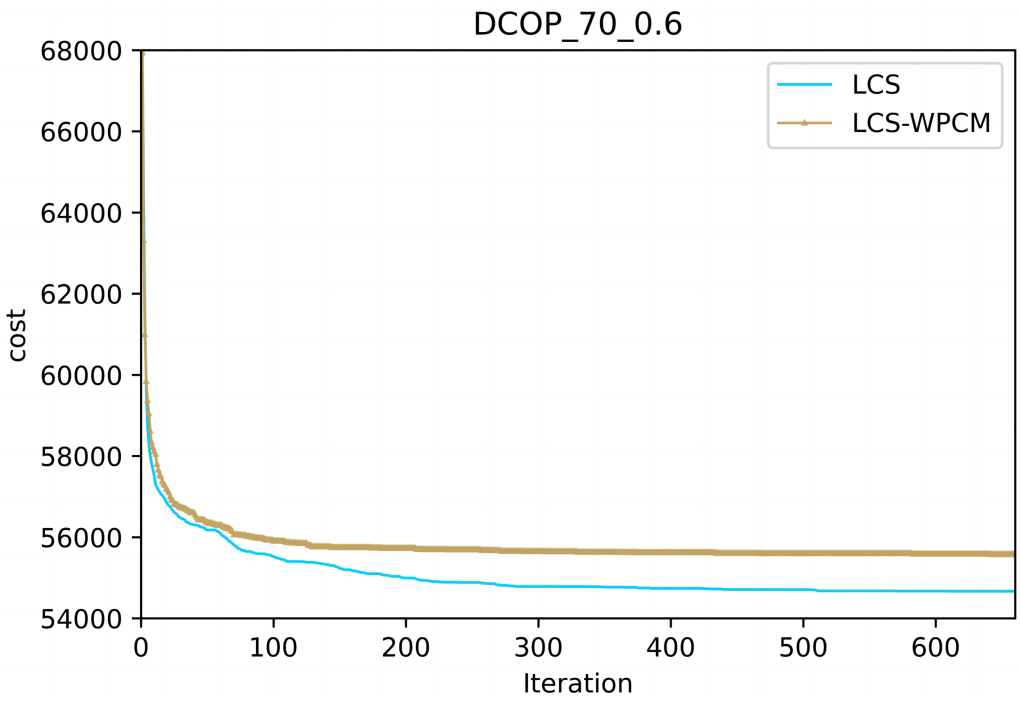

Figure 3 **Dense random graphs ($|A| = 70, p = 0.6$).**
Full-size DOI: 10.7717/peerjcs.1296/fig-3

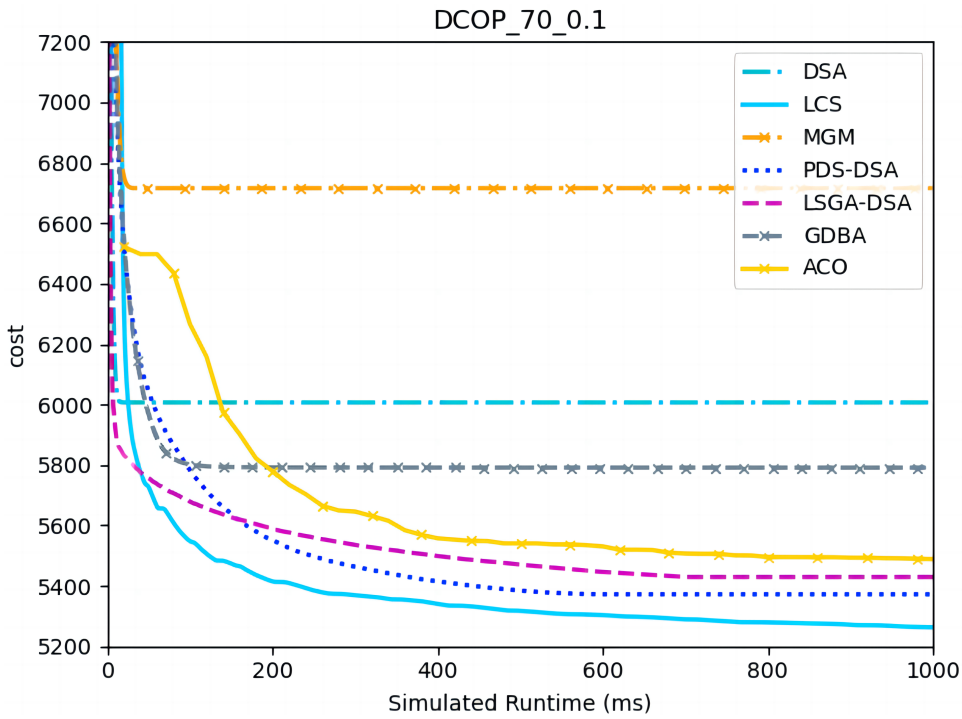

Figure 4 **The cost of LCS, DSA, MGM, GDBA, PDS-DSA, ACO and LSGA-DSA for random DCOPs ($|A| = 70, p = 0.1$).**

Full-size DOI: 10.7717/peerjcs.1296/fig-4

Since the global cost increases with the number of agents and the problem size, the degree of improvement of LCS is lower than before. However, the LCS algorithm still has higher solution quality than other algorithms for sparse and dense problems due to the local cost simulation method and population cooperation with each other.

Figures 10– 12 show the comparison between LCS and other algorithms in scale-free networks($|A| = 150,\ m1 = 20, m2 = 3; |A| = 150,\ m1 = 20, m2 = 10$). From the experimental results it can be seen that the algorithm runs longer on this problem because of its larger size. Compared to the other algorithms, LCS improves the solution quality on sparse problems by 2.1% $\sim$ 26.6% and on dense problems by 1.0% $\sim$ 9.8%. The improvement effect shows that the algorithm improves less on the dense problem than on the sparse problem. The reason for this is that as the number of neighbors connected to the centroids in the scale-free network increases, the simulation accuracy of the local cost is affected and its fluctuation range becomes larger. However, the algorithm preserves the probability of being selected for all values in the value domain, so the effect of this error is improved, which makes the LCS still outperform the other algorithms. Table 8 shows that LCS has good search capability for both sparse and dense problems in different problems, where other algorithms can not catch up with the effect of LCS.

Figures 13 and 14 presents the comparison of LCS with other algorithms on weighted graph coloring problems ($|A| = 120,\ p = 0.05$). Compared to the other algorithms, LCS improves by 38.4% $\sim$ 85.3%, which is a great improvement. In this problem, it can be

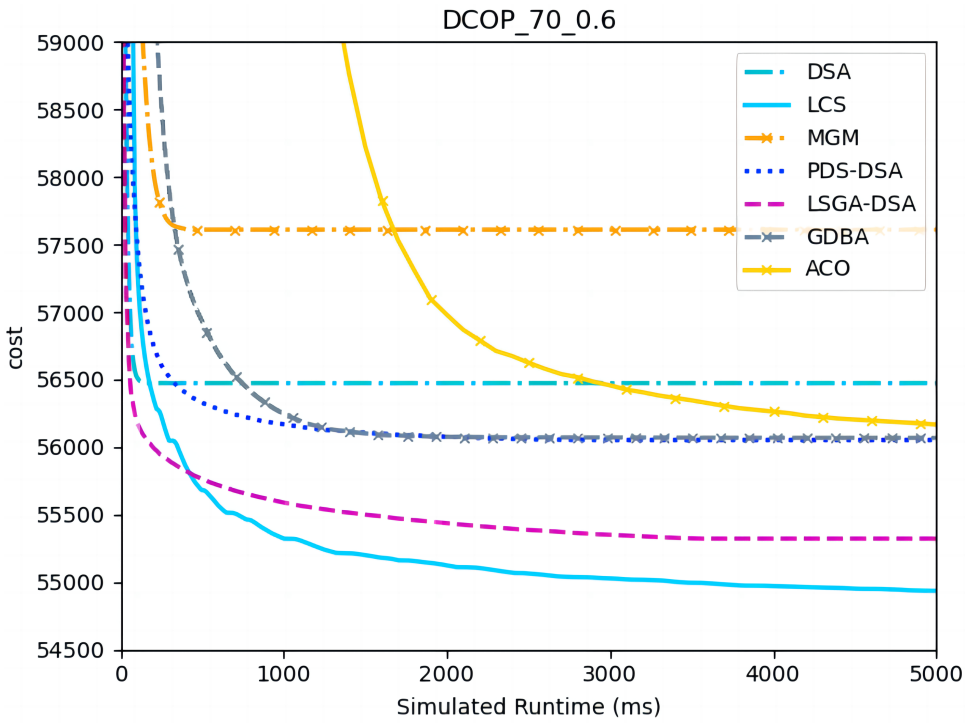

Figure 5 **The cost of LCS, DSA, MGM, GDBA, PDS-DSA, ACO and LSGA-DSA for random DCOPs ($|A| = 70, p = 0.6$).**

Full-size DOI: 10.7717/peerjcs.1296/fig-5

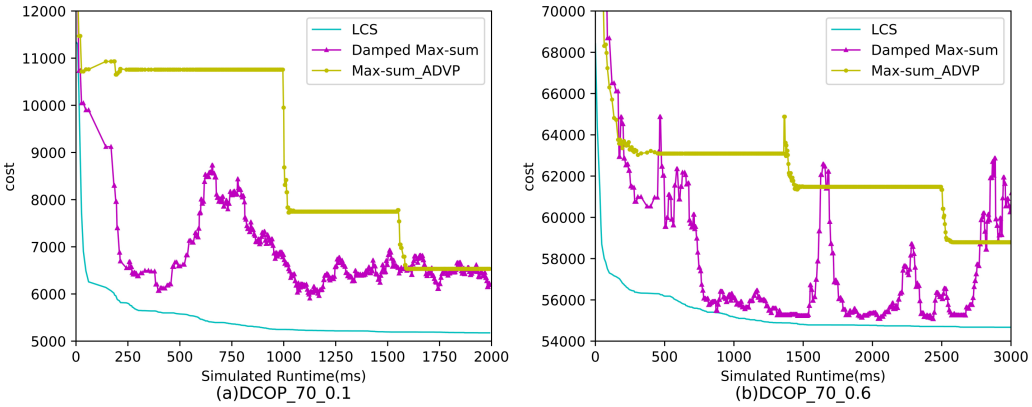

Figure 6 **The cost of LCS, Max-sum_ADVP and Damped Max-sum for random DCOPs ($|A| = 70$).**

Full-size DOI: 10.7717/peerjcs.1296/fig-6

found that all algorithms solve with a lower value of global cost, and the solution of PDS-DSA algorithm has dropped to more than 300, which is already a better solution. However, in contrast to it, LCS drops the solution to about 200, which greatly improves the solution quality. The weighted graph coloring problem favors DCSP (*Yokoo et al.,*

Table 8 Comparison of LCS and the benchmarking algorithms on difference configuration of random DCOPs and SFN.

Problems	random DCOPs				SFN	
	A=70		A=120		A=150	
	sparse	dense	sparse	dense	sparse	dense
DSA	6006	56473	21421	178316	9500	44435
MGM	6715	57609	22615	180505	10978	45729
PDS-DSA	5372	56052	20322	177781	8231	43088
LSGA-DSA	5429	55321	20275	176185	8521	42728
GDBA	5791	56069	21093	177529	9025	43944
ACO_DCOP	5436	55790	–	–	–	–
Max-sum_ADVP	6574	58272	21808	182814	9818	46963
Damped Max-sum	6019	55356	24766	181156	8287	46572
LCS	5257	54922	19970	175461	8059	42322

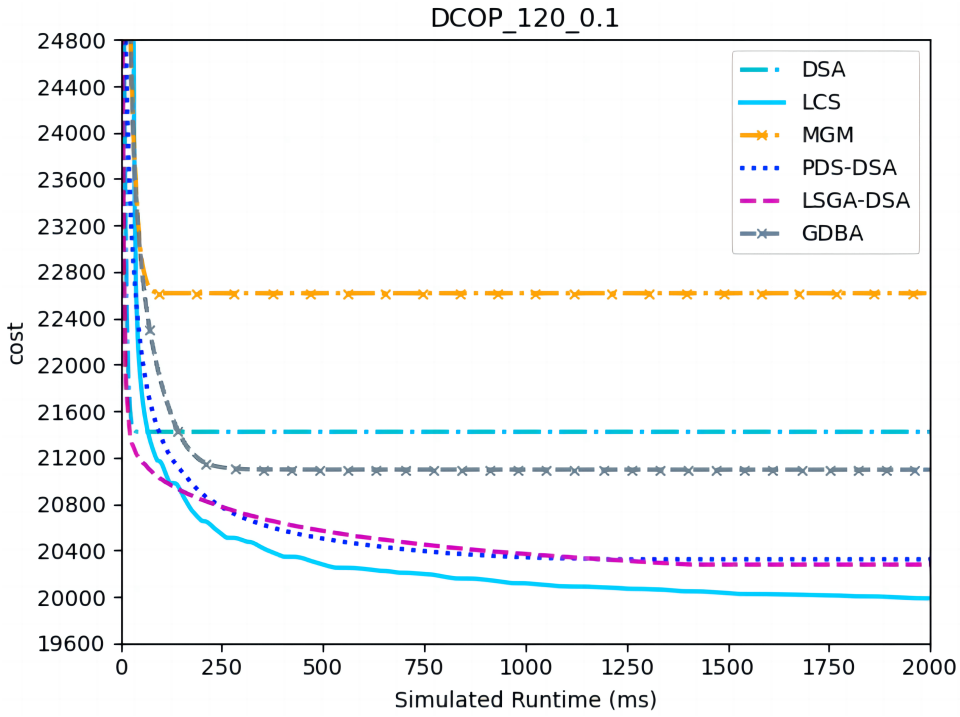

Figure 7 The cost of LCS, DSA, MGM, GDBA, PDS-DSA and LSGA-DSA for random DCOPs ($|A| = 120$, $p = 0.1$).

Full-size DOI: 10.7717/peerjcs.1296/fig-7

1998) similarly, which proves that LCS may be superior in solving similar problems such as DCSP.

Figure 15 presents the comparison of LCS with other algorithms on random meeting scheduling problems($|A| = 90$). We omit the results of MGM and GDBA due to their inferior performances (*Chen et al., 2020*). Compared to the other algorithms, LCS improves

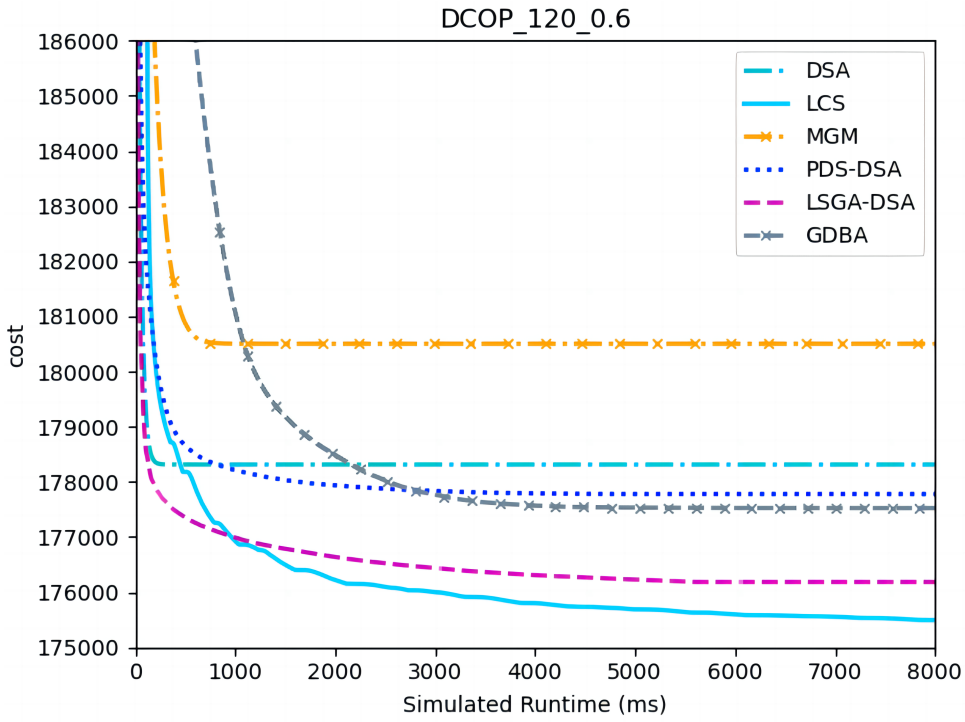

Figure 8 **The cost of LCS, DSA, MGM, GDBA, PDS-DSA and LSGA-DSA for random DCOPs ($|A| = 120$, $p = 0.6$).**

Full-size DOI: 10.7717/peerjcs.1296/fig-8

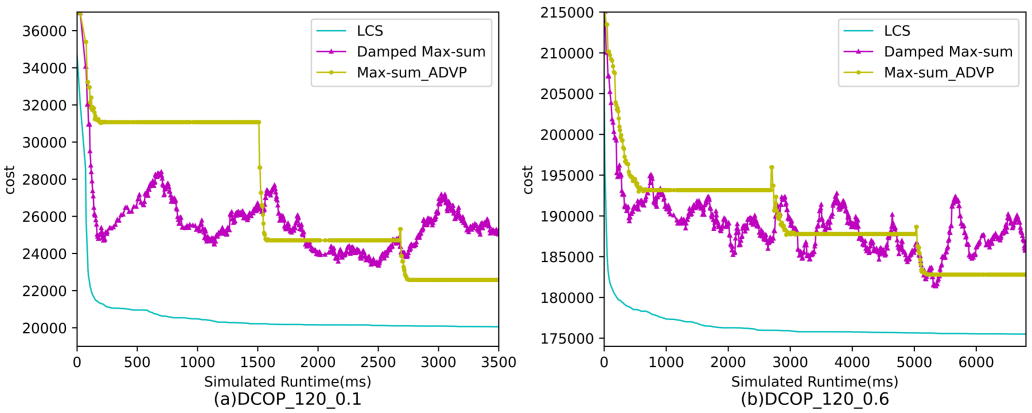

Figure 9 **The cost of LCS, Max-sum_ADVP and Damped Max-sum for random DCOPs ($|A| = 120$).**

Full-size DOI: 10.7717/peerjcs.1296/fig-9

by 0.6% ∼4.2%. In this problem, it can be found that all algorithms solve with a lower value of global cost. Experimental results show that LCS outperforms other algorithms and can improve real-world problems.

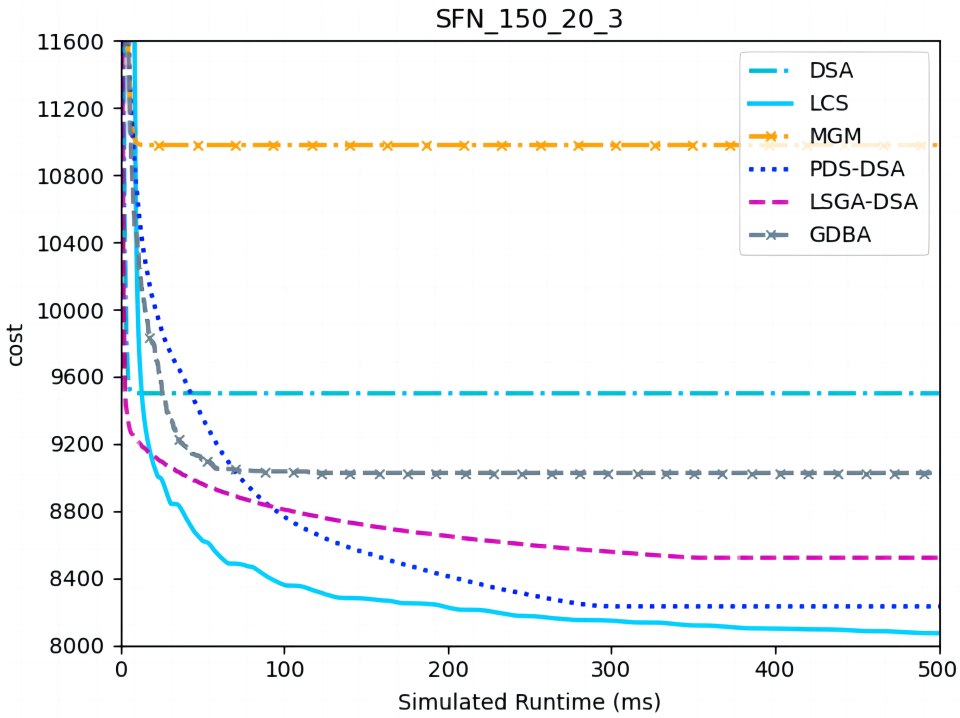

Figure 10 **The cost of LCS, DSA, MGM, GDBA, PDS-DSA and LSGA-DSA for scale-free networks (|A| = 150, m1 = 20, m2 = 3).**

Full-size DOI: 10.7717/peerjcs.1296/fig-10

CONCLUSIONS

The changing of the local cost that can be obtained from historical agent values has a significant influence on the performance of the local search algorithm. LCS is presented in this article to exploit the potential of historical values of agents to further develop the exploration ability of the local search algorithm. The proposed LCS makes up for the vacancy that local cost historical information has never been considered in the local search algorithms. In LCS, the designed EWMA provides an effective scheme for estimating local cost and the population interact mechanism giving a strategy to improve the exploitation and exploration ability of local search algorithms. We theoretically analyze the superiority of LCS from two different aspects. Finally, our experimental results show that LCS is superior to the competing algorithms including the latest local search algorithms and population-based algorithms. In future work, we will explore how to solve the problem of reliance on parameter α and make LCS self-adaptive according to the scale of the problem.

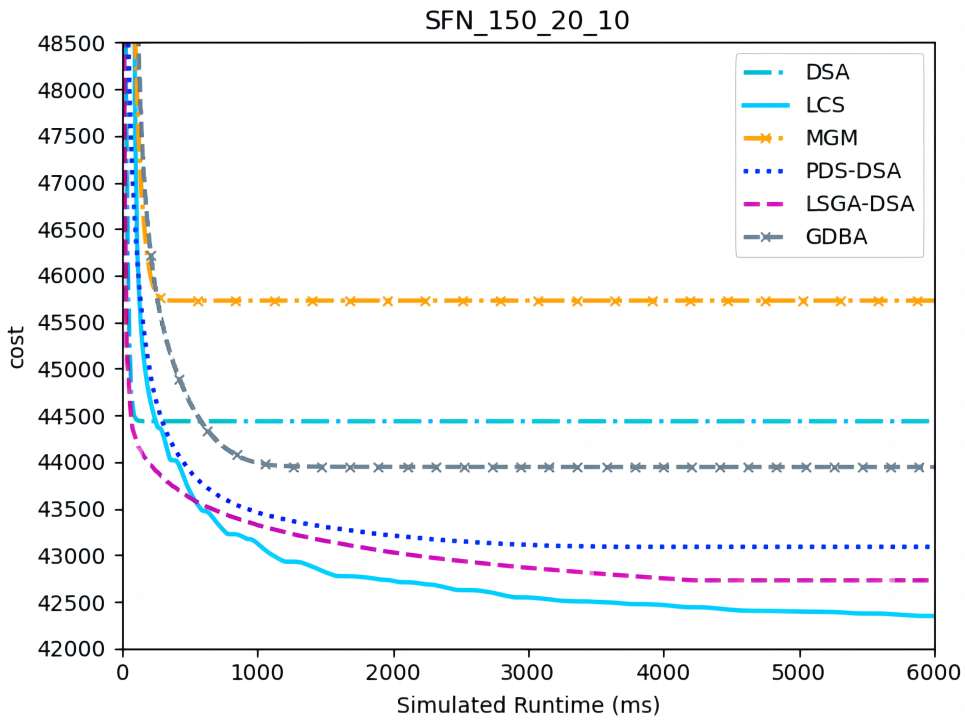

Figure 11 The cost of LCS, DSA, MGM, GDBA, PDS-DSA and LSGA-DSA for scale-free networks (|A| = 150, m1 = 20, m2 = 10).

Full-size DOI: 10.7717/peerjcs.1296/fig-11

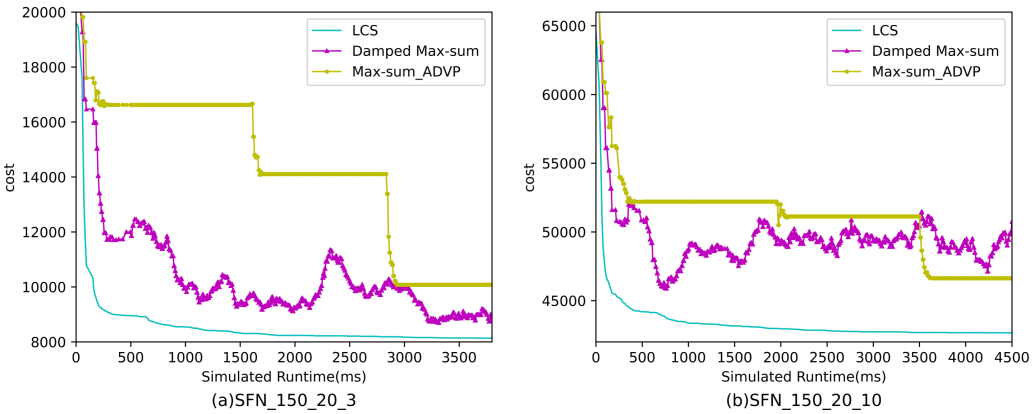

Figure 12 The cost of LCS, Max-sum_ADVP and Damped Max-sum for scale-free networks (|A| = 150).

Full-size DOI: 10.7717/peerjcs.1296/fig-12

ADDITIONAL INFORMATION AND DECLARATIONS

Funding

This work was supported by the Youth Project of Science and Technology Research Program of Chongqing Education Commission of China (No. KJQN202001139), the Chongqing

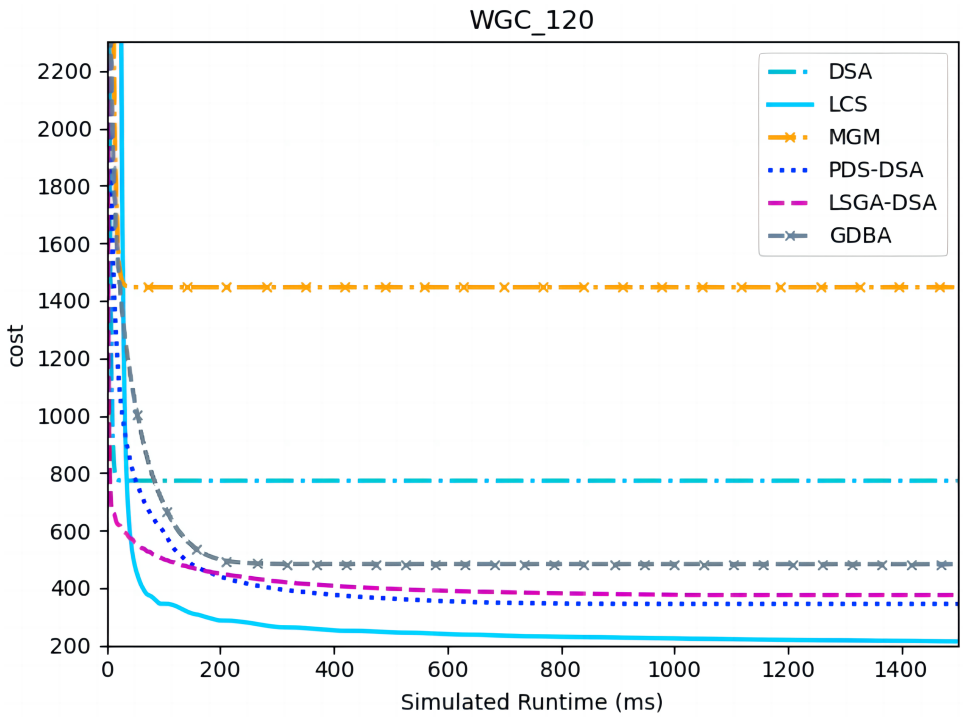

Figure 13 **The cost of LCS, DSA, MGM, GDBA, PDS-DSA and LSGA-DSA for weighted graph coloring problems with 120 agents.**

Full-size DOI: 10.7717/peerjcs.1296/fig-13

Research Program of Basic Research and Frontier Technology (NO. cstc2018jcyjAX0287), and the Scientific Research Foundation of Chongqing University of Technology. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Grant Disclosures

The following grant information was disclosed by the authors:
Youth Project of Science and Technology Research Program of Chongqing Education Commission of China: KJQN202001139.
Chongqing Research Program of Basic Research and Frontier Technology: cstc2018jcyjAX0287.
Scientific Research Foundation of Chongqing University of Technology.

Competing Interests

The authors declare there are no competing interests.

Author Contributions

- Meifeng Shi conceived and designed the experiments, performed the experiments, analyzed the data, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.

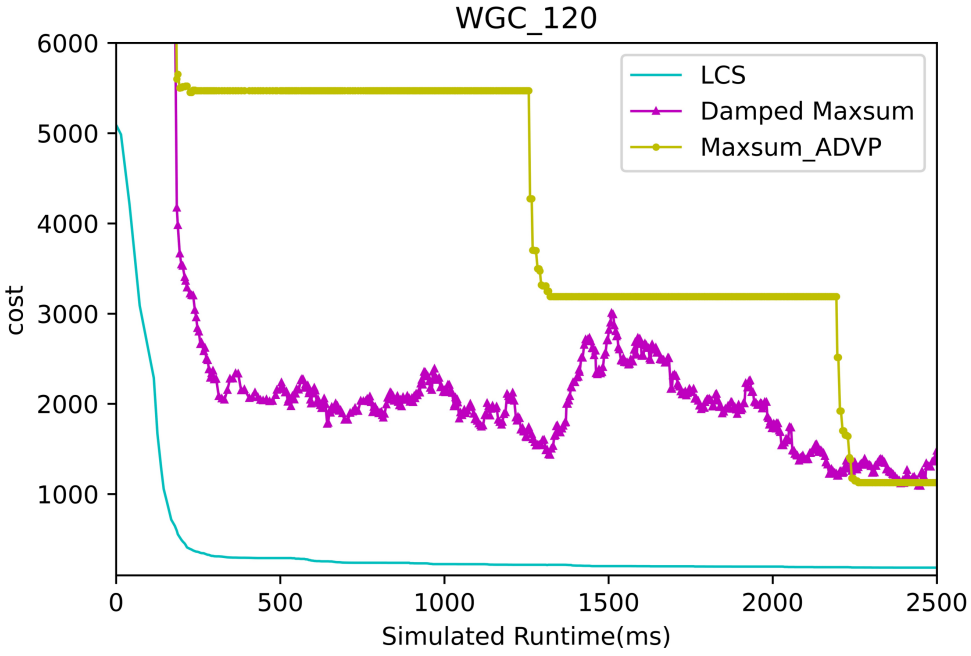

Figure 14 The cost of LCS, Max-sum_ADVP and Damped Max-sum for weighted graph coloring problems with 120 agents.

Full-size DOI: 10.7717/peerjcs.1296/fig-14

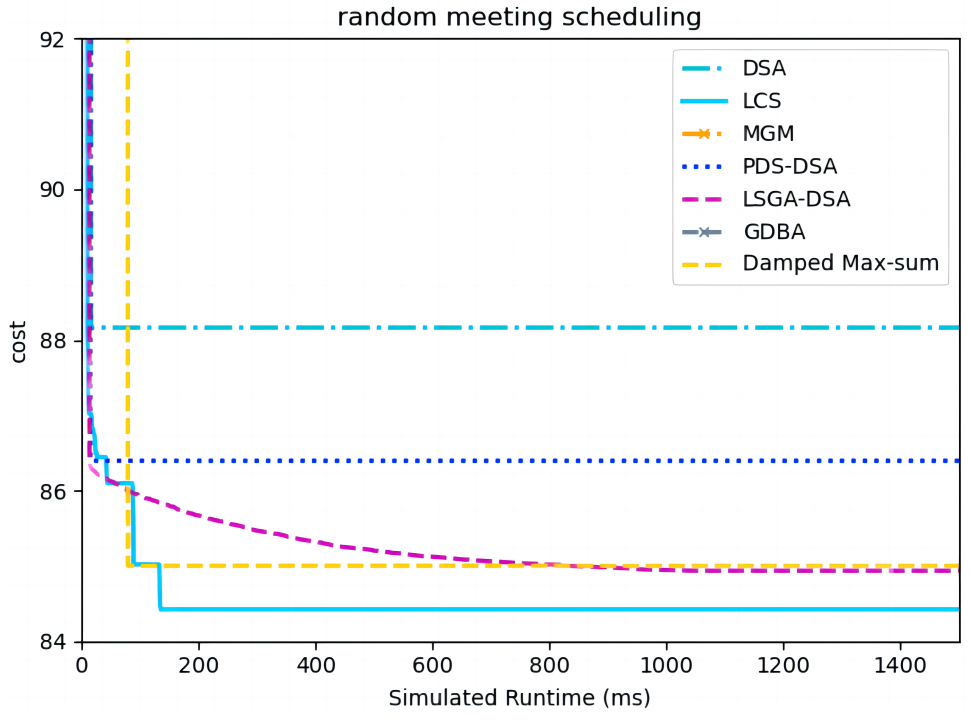

Figure 15 The cost of all algorithms for random meeting scheduling problems with 90 agents.

Full-size DOI: 10.7717/peerjcs.1296/fig-15

- Feipeng Liang performed the experiments, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Yuan Chen analyzed the data, prepared figures and/or tables, and approved the final draft.
- Ying He analyzed the data, authored or reviewed drafts of the article, and approved the final draft.

Data Availability

The following information was supplied regarding data availability:

The code is available in the Supplementary File.

Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.1296#supplemental-information.

REFERENCES

Aji SM, McEliece RJ. 2000. The generalized distributive law. *IEEE Transactions on Information Theory* **46(2)**:325–343 DOI 10.1109/18.825794.

Arshad M, Silaghi MC. 2004. Distributed simulated annealing. In: *Distributed constraint problem solving and reasoning in multi-agent systems, 112*.

Bowring E, Pearce JP, Portway C, Jain M, Tambe M. 2008. On k-optimal distributed constraint optimization algorithms: new bounds and algorithms. In: *Proceedings of the seventh international joint conference on autonomous agents and multiagent systems*, 607–614 DOI 10.1145/1402298.1402309.

Chapman AC, Rogers A, Jennings NR, Leslie DS. 2011. A unifying framework for iterative approximate best-response algorithms for distributed constraint optimization problems1. *The Knowledge Engineering Review* **26(4)**:411–444 DOI 10.1017/S0269888911000178.

Chen Z, Liu L, He J, Yu Z. 2020. A genetic algorithm based framework for local search algorithms for distributed constraint optimization problems. *Autonomous Agents and Multi-Agent Systems* **34**:41 DOI 10.1007/s10458-020-09464-9.

Chen Z, Wu T, Deng Y, Zhang C. 2018. An ant-based algorithm to solve distributed constraint optimization problems. In: *Proc. of the 32th AAAI conference on artificial intelligence*, 4654–4661 DOI 10.1609/aaai.v32i1.11580.

Cohen L, Zivan R. 2017. Max-sum revisited: the real power of damping. In: *International conference on autonomous agents and multiagent systems*. Cham: Springer, 111–124 DOI 10.1007/978-3-319-71679-4_8.

Deng Y, Yu R, Wang X, An B. 2021. Neural regret-matching for distributed constraint optimization problems. In: *IJCAI*, 146–153.

Farinelli A, Rogers A, Jennings NR. 2014. Agent-based decentralised coordination for sensor net-works using the max-sum algorithm. *Autonomous Agents and Multi-Agent Systems* **28(3)**:337–380 DOI 10.1007/s10458-013-9225-1.

Farinelli A, Rogers A, Petcu A, Jennings NR. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: *Proceedings of the 7th conference on AAMAS*, 639–646.

Ferber J, Weiss G. 1999. *Multi-agent systems: an introduction to distributed artificial intelligence*. Reading: Addison-wesley.

Fioretto F, Yeoh W, Pontelli E, Ma Y, Ranade SJ. 2017. A distributed constraint optimization (DCOP) approach to the economic dispatch with demand response. In: *Proceedings of the 16th conference on AAMAS*. 999–1007.

Freuder EC, Quinn MJ. 1985. Taking advantage of stable sets of variables in constraint satisfaction problems. In: *IJCAI*, 1076–1078.

Gershman A, Meisels A, Zivan R. 2009. Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research* **34**:61–88 DOI 10.1613/JAIR.2591.

Hirayama K, Yokoo M. 1997. Distributed partial constraint satisfaction problem. In: *International conference on principles and practice of constraint programming*. Berlin, Heidelberg: Springer, 222–236.

Leite A, Fabrício E, Jean-Paul A. 2014. Distributed constraint optimization problems: review and perspectives. *Expert Systems with Applications* **41(11)**:5139–5157 DOI 10.1016/j.eswa.2014.02.039.

Maheswaran RT, Pearce JP, Tambe M. 2004. Distributed algorithms for dcop: a graphical-game-based approach. In: *Proceedings of the international conference on parallel and distributed computing systems (PDCS)*, 432–439.

Maheswaran RT, Pearce JP, Tambe M. 2006. A family of graphical-game-based algorithms for distributed constraint optimization problems. In: *Coordination of large-scale multiagent systems*. Boston: Springer, 127–146 DOI 10.1007/0-387-27972-5_6.

Modi PJ, Shen WM, Tambe M, Yokoo M. 2005. ADOPT: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* **161(1–2)**:149–180 DOI 10.1016/j.artint.2004.09.003.

Nguyen DT, Yeoh W, Lau HC. 2013. Distributed gibbs: a memory-bounded sampling-based dcop algorithm. In: *Proceedings of the 12th conference on AAMAS*, 167–174.

Nguyen DT, Yeoh W, Lau HC, Zivan R. 2019. Distributed gibbs: a linear-space sampling-based DCOP algorithm. *Journal of Artificial Intelligence Research* **64**:705–748 DOI 10.1613/jair.1.11400.

Okamoto S, Zivan R, Nahon A. 2016. Distributed breakout: beyond satisfaction. In: *Proc. of the 25th IJCAI*, 447–453.

Ottens B, Dimitrakakis C, Faltings B. 2012. DUCT: an upper confidence bound approach to distributed constraint optimization problems. In: *Proceedings of the 26th AAAI conference on artificial intelligence*, 528–534.

Petcu A, Faltings B. 2005. DPOP: a scalable method for multiagent constraint optimization. In: *IJCAI*, 266–271.

Petcu A, Faltings B. 2007. MB-DPOP: a new memory-bounded algorithm for distributed optimization. In: *Proceedings of the 21th IJCAI*, 1452–1457.

Rogers A, Farinelli A, Stranders R, Jennings NR. 2011. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence* **175(2)**:730–759 DOI 10.1016/j.artint.2010.11.001.

Sultanik E, Modi PJ, Regli WC. 2007. On modeling multiagent task scheduling as a distributed constraint optimization problem. In: *Proceedings of the 20th IJCAI*, 1531–1536.

Yokoo M, Durfee EH, Ishida T, Kuwabara K. 1998. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering* **10(5)**:673–685 DOI 10.1109/69.729707.

Yu Z, Chen Z, He J, Deng Y. 2017. A partial decision scheme for local search algorithms for distributed constraint optimization problems. In: *Proceedings of the 16th conference on AAMAS*, 187–194.

Zhang W, Wang G, Xing Z, Wittenburg L. 2005. Distributed stochastic search and distributed breakout: proper-ties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* **161(1–2)**:55–87 DOI 10.1016/j.artint.2004.10.004.

Zivan R, Okamoto S, Peled H. 2014. Explorative anytime local search for distributed constraint optimization. *Artificial Intelligence* **212**:1–26.

Zivan R, Parash T, Cohen L, Peled H, Okamoto S. 2017. Balancing exploration and exploitation in incomplete min/max-sum inference for distributed constraint optimization. *Autonomous Agents and Multi-Agent Systems* **31(5)**:1165–1207 DOI 10.1007/s10458-017-9360-1.