The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

# An Ant-Based Algorithm to Solve Distributed Constraint Optimization Problems

**Ziyu Chen,[1] Tengfei Wu,[2] Yanchen Deng,[3] Cheng Zhang[1]**
Chongqing University
[1] {chenziyu,bootan}@cqu.edu.cn, [2] wutengfei0404@163.com, [3] dyc941126@126.com

**Abstract**

As an important population-based algorithm, ant colony optimization (ACO) has been successfully applied into various combinatorial optimization problems. However, much existing work in ACO focuses on solving centralized problems. In this paper, we present a novel algorithm that takes the power of ants to solve Distributed Constraint Optimization Problems (DCOPs), called ACO_DCOP. In ACO_DCOP, a new mechanism that captures local benefits is proposed to compute heuristic factors and a new method that considers the cost structure of DCOPs is proposed to compute pheromone deltas appropriately. Moreover, pipelining technique is introduced to make full use of the computational capacity and improve the efficiency. In our theoretical analysis, we prove that ACO_DCOP is an anytime algorithm. Our empirical evaluation indicates that ACO_DCOP is able to find solutions of equal or significantly higher quality than state-of-the-art DCOP algorithms.

## Introduction

Distributed Constraint Satisfaction and Optimization Problems (DCSPs and DCOPs) (Yokoo et al. 1998) are fundamental frameworks for Multi-agent Systems (MAS) in which agents need to coordinate their decisions to find a feasible solution (for DCSPs) or optimize the global objective (for DCOPs). They have been successfully deployed into various real applications including sensor networks (Zhang et al. 2005), task scheduling (Sultanik, Modi, and Regli 2007), power networks (Fioretto et al. 2017), etc.

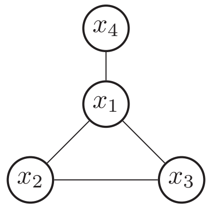
Algorithms for solving DCOPs can be classified into complete algorithms and incomplete algorithms according to whether they guarantee to find the optimal solutions. Typical search based complete algorithms include ADOPT (Modi et al. 2005), BnB-ADOPT (Yeoh, Felner, and Koenig 2008), AFB (Gershman, Meisels, and Zivan 2009) and so on. DPOP (Petcu and Faltings 2005), on the other hand, is an inference based complete algorithm to solve DCOPs by using a dynamic programming technique. Since DCOPs are NP-Hard, considerable research efforts have been made to develop incomplete algorithms. Local search algorithms are the most popular incomplete algorithms including DSA (Zhang et al. 2005), MGM (Maheswaran, Pearce, and Tambe 2004), MGM2 (Maheswaran, Pearce, and Tambe 2004), DSAN (Arshad and Silaghi 2004), GDBA (Okamoto, Zivan, and Nahon 2016), etc. In those algorithms, agents make decisions based on their neighbor states. Recently, several mechanisms such as anytime local search framework (ALS) (Zivan, Okamoto, and Peled 2014), $k-$optimality (Pearce and Tambe 2007) and partial decision scheme (PDS) (Yu et al. 2017) have been proposed to enhance the solution quality of local search algorithms. Max-sum(Farinelli et al. 2008) and its variants (Rogers et al. 2011; Zivan and Peled 2012; Chen, Deng, and Wu 2017) are typical inference based incomplete algorithms, where agents propagate and accumulate utilities through the whole factor graph. Sample based algorithms including DUCT (Ottens, Dimitrakakis, and Faltings 2012) and D-Gibbs (Nguyen, Yeoh, and Lau 2013) are emerging incomplete algorithms, which sample the search space to approximate a function as a product of statistical inference.

Many existing DCOP algorithms derive from centralized single-solution optimization approaches such as hill-climbing algorithm and simulated annealing. Ant colony optimization (Dorigo, Birattari, and Stutzle 2006), as a population-based metaheuristic, has been successfully applied into various NP-Hard problems including traveling salesman problem (TSP) (Dorigo and Gambardella 1997), constraint satisfaction problem (CSP) (Solnon 2002) and many others. However, to the best of our knowledge, the ant solver for DCSP (Semnani and Zamanifar 2012) is the only ant-based algorithm for solving distributed constraint reasoning problems. Unfortunately, since solving DCOPs requires to enumerate all possible assignment combinations to find the optimal solution rather than just find a feasible solution, the ant solver for DCSP cannot directly be applied into solving DCOPs. Considering the difference between DCOPs and DCSPs, we propose a novel ant-based DCOP algorithm in which agents try to keep track of promising areas of the search space by employing pheromone trails as a guide to make wiser decisions.

## Background

### Distributed Constraint Satisfaction and Optimization Problems

DCSPs can be defined by a tuple $\langle A, X, D, C\rangle$ where



(a)

| $x_1 \backslash x_2$ | 0 | 1 |
|---|---|---|
| 0 | 5 | 1 |
| 1 | 2 | 8 |

| $x_1 \backslash x_4$ | 0 | 1 |
|---|---|---|
| 0 | 2 | 1 |
| 1 | 5 | 4 |

| $x_1 \backslash x_3$ | 0 | 1 |
|---|---|---|
| 0 | 4 | 1 |
| 1 | 3 | 5 |

| $x_2 \backslash x_3$ | 0 | 1 |
|---|---|---|
| 0 | 3 | 5 |
| 1 | 2 | 4 |

(b)

Figure 1: A DCOP instance

- $A = \{a_1, a_2, \ldots, a_n\}$ is a set of agents.
- $X = \{x_1, x_2, \ldots, x_m\}$ is a set of variables. Each variable $x_i$ is controlled by one of the agents.
- $D = \{D_1, D_2, \ldots, D_m\}$ is a set of finite variable domains, variable $x_i$ taking a value in $D_i$.
- $C = \{c_1, c_2, \ldots, c_q\}$ is a set of constraints, where a constraint $c_i$ is a predicate that is defined on $D_{i_1} \times D_{i_2} \times \cdots \times D_{i_n}$. The predicate is true iff the assignment of the variables satisfies the constraint.

The goal of DCSPs is to find an assignment to all variables such that all constraints are satisfied. To facilitate understanding, we assume that each agent has a single variable and constraints are binary. Thus, the term "agent" and "variable" can be used interchangeably (i.e., $m = n$). A binary constraint $c_{ij}$ is a constraint involving exactly two variables with the scope $D_i \times D_j$. We also assume that all agents can communicate with every other agent.

DCOPs extend DCSPs by replacing hard constraints with cost functions. Formally, constraints in a DCOP are a set of functions $f_{ij} \in F$, where $f_{ij}$ is any function $f_{ij} : D_i \times D_j \to \mathbb{R}^+$ which denotes how much cost is assigned to each possible combination of values of the involved variables. Without loss of generality, a solution of a DCOP is an assignment to all variables that minimizes the total cost, which is the sum of all constraints:

$$X^* = \underset{d_i \in D_i, d_j \in D_j}{\arg\min} \sum_{f_{ij} \in F} f_{ij}(x_i = d_i, x_j = d_j)$$

A DCOP can be visualized by a *constraint graph* where the vertexes correspond to variables in the DCOP and the edges connect pairs of variables that are constrained. Fig. 1 presents an example of a DCOP problem whose constraint graph and cost matrices are shown in Fig. 1 (a) and Fig. 1 (b).

## Ant Colony Optimization

Ant colony optimization (ACO) is a population-based metaheuristic to solve combinatorial optimization problems, which is inspired by the foraging behavior of ants. Ants communicate with each other by laying pheromone to mark some promising paths that should be followed by the other members of the colony.

As shown in Fig. 2, ACO consists of three parts: constructing solutions, applying local search and updating pheromones. At the beginning of each cycle, several artificial ants construct solutions by traversing a *construction graph* in which each ant constructs one solution by a probabilistic mechanism that is biased by a pheromone factor and a heuristic factor. Specifically, the pheromone factor associating to the quality of solutions represents a posteriori indication of the desirability of the traverse, while the heuristic factor related to the evaluation of the partial solution indicates the a priori desirability of the traverse. After that, an optional local search phase is performed to improve these solutions. Finally, pheromones are updated according to the quality of solutions (i.e., the pheromones associated with good solutions will be increased and the bad ones will be decreased).

Although it has been successfully applied into various NP-Hard problems, ACO cannot be directly applied into distributed constraint reasoning problems since there is no actual path for artificial ants to traverse. Consequently, ants fail to build solutions and pheromone trails also cannot be updated.

**Algorithm 1:** Ant Colony Optimization

```
1 Set parameters, initialize pheromone trails
2 while termination condition is not met do
3     ConstructAntSolutions
4     ApplyLocalSearch (optional)
5     UpdatePheromones
6 end
```

Figure 2: The sketch of ACO

## Ant Solver for DCSP

The ant solver for DCSP overcomes the pathology by using message-passing between agents to simulate the movements of ants. The algorithm starts with converting a constraint graph to a fully-connected construction graph in which the vertexes correspond to agents and each vertex is associated with multi-line of pheromone. Each line of pheromone is related to two values for two agents and is distinguished by the four factors: Source Variable (agent), Source value, Destination Variable (agent), Destination value. Fig. 3 shows a construction graph derived from the constraint graph shown in Fig. 1. Then, agents are ordered by max-static heuristic in which the algorithm first prioritizes agents based on their domain size and assigns higher priority values to the agents that have more neighbors. In each iteration, each agent who has received the assignments from all its higher priority neighbors selects a value for each ant. The value selection is made stochastically according to a probability that depends on a pheromone factor and a heuristic factor. Concretely, the pheromone factor for a value $d_i$ of an agent $a_i$ is the sum of all pheromone lines of $a_i$ that involve $d_i$, while the heuristic factor for $d_i$ is inversely proportional to the number of constraint violations created by $d_i$ between $a_i$ and its higher priority neighbors.

Each agent sends the selected values to its lower priority neighbors after the value selection. If it does not have any lower priority neighbor, it sends the values to the lowest

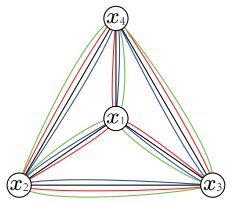


Figure 3: An example of a construction graph in the ant solver for DCSP

priority agent. When receiving the values selected by other agents, the lowest priority agent calculates the cost of each ant. If the cost of the best ant is not zero, then the agent computes the pheromone delta which is defined inversely proportional to the cost of the best ant (i.e., the number of constraint violations) and broadcasts the delta to all of the agents. Otherwise, the solution is found and the algorithm terminates.

When receiving the pheromone delta, an agent will trigger the evaporation phase and updating phase consecutively. The evaporation phase decreases all the pheromone trails uniformly to allow ants to forget bad assignments and then pheromone trails that appear in the best ant path will be updated according to the delta. These procedures are performed until a solution is found or the cycle number is exceeded.

## Proposed Method

### Motivation

A DCOP generally has a larger search space than a DCSP since it requires algorithms to exhaust all possible assignment combinations to find the optimal solution rather than just find a feasible solution in a DCSP. Thus, as an algorithm that is originally designed for solving DCSPs, the ant solver for DCSP cannot solve DCOPs efficiently due to the following facts.

- The calculation of a pheromone factor in an agent does not consider the assignments of the other agents, which makes the agent unable to utilize the assignments of its higher priority neighbors to remove irrelevant pheromone trails and hence prohibits it from computing the desirabilities precisely.
- Heuristic factors in an agent prefer the assignments that do not create any constraint violation against its higher priority neighbors. However, the mechanism greatly limits the search space of an agent in a DCOP since the partial local benefit of an individual agent often conflicts with the global benefit. Thus, a new evaluation mechanism for the total local benefits should be provided to accommodate DCOPs.
- The pheromone delta is inversely proportional to the number of constraint violations of the best ant in the algorithm, which is unsuitable for DCOPs since the cost structure in a DCOP is more complex. Specifically, a constraint in a DCOP is a general-valued function from assignments to costs rather than a predicate in a DCSP. Therefore, a new computation method that is compatible with DCOPs becomes an urgent need.

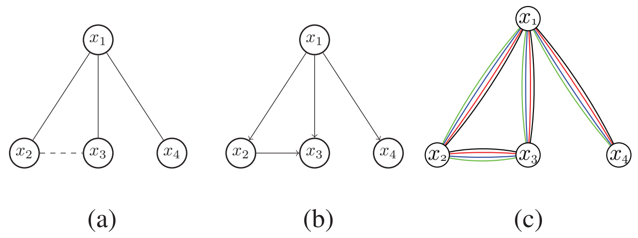


Figure 4: A BFS pseudo-tree, an ordered arrangement and the construction graph

Besides, the algorithm also suffers from large storage overheads since the construction graph is fully connected. The concurrency is another performance bottle-neck of the algorithm since agents need to wait pheromone delta messages to update pheromone trails and start a new cycle.

In fact, compared to local search algorithms, ACO is an effective approach for combinatorial optimization problems since it can keep the balance between local benefits and the global benefit. Although the ant solver for DCSP takes ACO into distributed constraint reasoning problems, its exploration ability is insufficient to solve DCOPs. Thus, we propose a novel algorithm called ACO_DCOP that takes the power of ACO to solve DCOPs.

### ACO_DCOP

ACO_DCOP generally consists of three parts: initialization, constructing solutions and updating. The sketch of the algorithm can be found in Fig. 5.

**Initialization** initiates the algorithm by ordering agents into a BFS pseudo-tree (Chen, He, and He 2017), and then an ordered arrangement (i.e., the message-passing order) is constructed according to the pseudo-tree, where agents with smaller depths are prioritized over the agents with larger depths, agents with the same depth are prioritized by their degrees, and ties are broken alphabetically. In this way, the neighbors of an agent $a_i$ are grouped into the higher priority neighbors $H_i$ and the lower priority neighbors $L_i$. Fig. 4 (a) and (b) show an example of a BFS pseudo-tree of Fig. 1 (a) and the corresponding ordered arrangement in which $x_1$ is the highest priority agent, $x_4$ is the lowest priority agent, and arrows represent the message-passing directions. Then the ordered arrangement is converted to the construction graph by replacing each constraint with multiple pheromone lines. Each pheromone line $\tau_{ij}(d_i, d_j)$ is associated with two values $d_i, d_j$ for agents $a_i, a_j$ and is initialized with $\tau_0$. In this way, storage overheads are much reduced compared to the fully connected construction graph in the ant solver for DCSP. Fig. 4 (c) shows the construction graph of Fig. 4 (b). It is worth mentioning that pheromone lines of a constraint are managed by the agent that has lower priority. As a result, the highest priority agent can only select values randomly since it does not have any pheromone lines.

The initialization procedure of each agent begins with initializing parameters (line 1) where $K$ is the number of ants.

**Algorithm 2:** ACO_DCOP for agent $a_i$

```
1  Initialize Parameters: $\alpha, \beta, \rho, \tau_0, K$
2  StartCycle()
3  foreach $d_i \in D_i$ do
4      calculate $est_i(d_i)$ by formula (3)
   Function StartCycle:
5      foreach ant k do
6          $V_{k,*} \leftarrow \{\}$
7          $V \leftarrow V \cup V_{k,*}$
8      if $a_i$ is the root then
9          foreach ant k do
10             $V_{k,i} \leftarrow$ randomly selects a value from $D_i$
11             $V_{k,*} \leftarrow V_{k,*} \cup V_{k,i}$
12         Sends ValueMessage($\{i, V\}$) to agents in $L_i$
   When received Value(id,recv_V):
13     foreach ant k do
14         $V_{k,*} \leftarrow V_{k,*} \cup recv\_V_{k,*}$
15     if id $\in H_i$ then
16         if $a_i$ has received ValueMessages from all
           higher priority neighbors then
17             foreach ant k do
18                 selects a value for $V_{k,i}$ by formula (5)
19                 $V_{k,*} \leftarrow V_{k,*} \cup V_{k,i}$
20             if $|L_i| \neq 0$ then
21                 Sends ValueMessage($\{i, V\}$) to agents
                   in $L_i$
22             else if $a_i$ is not the lowest priority agent
               then
23                 Sends ValueMessage($\{i, V\}$) to the
                   lowest priority agent
24     if $a_i$ is the lowest priority agent then
25         if each agent has selected values for all ants
           then
26             foreach ant k do
27                 calculates $\Delta_k$ according to $V_{k,*}$ by
                   formula (6)
28                 if cost($V_{k,*}$) < $best\_cost$ then
29                     $best\_cost \leftarrow$ cost($V_{k,*}$)
30                     $v^* \leftarrow V_{k,*}$
31             sends PheromoneMessage($\{V,\Delta,v^*\}$) to all
               agents
   When received Pheromone(recv_V,Δ,v*):
32     $d_i^* \leftarrow v_i^*$
33     foreach ant k do
34         updates pheromone trails according to $\Delta_k$ by
           formula (7)
35     evaporates all pheromone trails by formula (9)
36     UpdateEstimation($recv\_V$)
37     if termination condition is not met then
38         StartCycle()
   Function UpdateEstimation(recv_V):
39     foreach $d_i \in D_i$ do
40         $s \leftarrow 0, c \leftarrow 0$
41         foreach ant k do
42             if $recv\_V_{k,i} = d_i$ then
43                 $s \leftarrow s + \sum_{j \in L_i} \text{cost}_{ij}(d_i, recv\_V_{k,j})$
44                 $c \leftarrow c + 1$
45         if $c \neq 0$ then
46             $avg\_cost \leftarrow s/c$
47             $est_i(d_i) \leftarrow (est_i(d_i) + avg\_cost)/2$
```

Figure 5: The sketch of ACO_DCOP

Then each agent executes **StartCycle** to initialize each ant with an empty set of assignments (line 5 - 7). Here, $V$ is a set of solutions for $K$ ants, $V_{k,*} \in V$ denotes the solution constructed by ant $k$, and $V_{k,i} \in V_{k,*}$ denotes the value selected by agent $a_i$ for ant $k$. If $a_i$ is the root (i.e., $|H_i| = 0$), it selects a value randomly for each ant and sends value messages that consist of its *id* and $V$ to its lower priority neighbors (line 8 - 12).

Take Fig. 4 for an example. Since it is the highest priority agent, $x_1$ selects values randomly for ants. Let's assume there are two ants and $x_1$ selects 0 and 1 for ant 1 and ant 2, respectively. Then $x_1$ sends value message $\{1, \{\{V_{1,1} = 0\}, \{V_{2,1} = 1\}\}\}$ to its lower priority neighbors $x_2, x_3, x_4$.

**Constructing solutions** is a procedure in which each ant gradually builds solutions. When it receives a value message, agent $a_i$ first merges the received solution set for every ant (line 13 - 14). After that, if $a_i$ has received value messages from all its higher priority neighbors, it selects a value $d_i$ for each ant $k$ stochastically with a probability $p_{k,i}(d_i)$ biased by a pheromone factor $\theta_{k,i}(d_i)$ and a heuristic factor $\eta_{k,i}(d_i)$ (line 15 - 19). To allow agents to consider only relevant pheromone trails, pheromone factors in our algorithm depend on the assignments of higher priority neighbors. Formally, a pheromone factor is defined as:

$$\theta_{k,i}(d_i) = \sum_{j \in H_i} \tau_{ij}(d_i, V_{k,j}) \tag{1}$$

Since the partial local benefit of an individual agent often conflicts with the global benefit in DCOPs, a new evaluation mechanism to the local benefit, which includes a cost estimation for lower priority neighbors, is proposed to compute heuristic factors. Formally, a heuristic factor is defined as:

$$\eta_{k,i}(d_i) = \frac{1}{\sum_{j \in H_i} \text{cost}_{ij}(d_i, V_{k,j}) + est_i(d_i) - LB} \tag{2}$$

where $\text{cost}_{ij}(d_i, V_{k,j})$ is the cost that is produced by the constraint $f_{ij}$ when $x_i = d_i$ and $x_j = V_{k,j}$. $est_i(d_i)$ is an estimation to the sum of costs with lower priority neighbors of $a_i$ when $x_i = d_i$, which is periodically updated and initially set to the most optimistic estimation (line 3 - 4). The procedure can be formalized by:

$$est_i(d_i) = \sum_{j \in L_i} \min_{d_j \in D_j} \text{cost}_{ij}(d_i, d_j) \tag{3}$$

$LB$ is a factor to scale up differences of heuristic factors, which is the most optimistic estimation under $est_i$. Formally, $LB$ is defined as:

$$LB = \min_{d_i \in D_i} \Big( \sum_{j \in H_i} \min_{d_j \in D_j} \text{cost}_{ij}(d_i, d_j) + est_i(d_i) \Big) - 1 \tag{4}$$

Given $\theta_{k,i}(d_i)$ and $\eta_{k,i}(d_i)$, the probability for selecting value $d_i$ is defined as:

$$p_{k,i}(d_i) = \frac{\theta_{k,i}(d_i)^\alpha \eta_{k,i}(d_i)^\beta}{\sum_{d_i' \in D_i} \theta_{k,i}(d_i')^\alpha \eta_{k,i}(d_i')^\beta} \tag{5}$$

where $\alpha$ and $\beta$ define the importance of the pheromone factor and the heuristic factor, respectively.

After selecting values, each agent sends value messages to its lower priority neighbors if it has any lower priority neighbors or sends a value message to the lowest priority agent if it does not have any lower priority neighbors and is not the lowest priority agent (line 20 - 23). It is worth noting that the cost of an ant is propagated by value messages. Specifically, each agent accumulates the cost by summing the receiving costs and the cost pertaining to the selected value, and then propagates the cost to lower priority agents by proportioning the cost equally to each outgoing value message to avoid calculating the cost more than once. We omit this in the sketch for space.

For example in Fig. 4, since $x_2$ and $x_4$ have received value messages from $x_1$ and hence satisfied line 15, they now can select value for ants. We will take $x_2$ and ant 1 as a demonstration. Suppose that $\tau_0 = 3, \alpha = 2, \beta = 2$. According to Equations (3) and (4), we have:

$$est_2(0) = 3, est_2(1) = 2$$

$$LB = 2$$

Probabilities of each possible assignment are:

$$p_{1,2}(0) = \frac{3^2 \times 0.167^2}{3^2 \times 0.167^2 + 3^2 \times 1^2} = 0.027$$

$$p_{1,2}(1) = \frac{3^2 \times 1^2}{3^2 \times 0.167^2 + 3^2 \times 1^2} = 0.973$$

Thus, $x_2$ selects 0 with the probability 0.027 and selects 1 with the probability 0.973 for ant 1. After the value selection, $x_2$ sends a value message to its lower priority agent $x_3$, which triggers the value selection procedure of $x_3$. It is noticeable that $x_4$ does not send any value message since it is the lowest priority agent.

**Updating** consists of four parts: calculating pheromone deltas, updating pheromone trails, evaporation and updating estimations. The calculation of pheromone deltas and the updating to the best ant $v^*$ are performed by the lowest priority agent so long as each agent has selected values for all ants (line 24 - 30). Since the cost structure in a DCOP is much more complex than the one in a DCSP, a new computation method that employs the mean cost of all ants as a standard is proposed to produce appropriate deltas. Formally, the delta $\Delta_k \in \Delta$ for ant $k$ is computed by:

$$\Delta_k = 1 - \frac{\text{cost}(V_{k,*}) - best\_cost}{\frac{1}{K}\sum_{k'=1}^{K} \text{cost}(V_{k',*}) - best\_cost} \tag{6}$$

where $\text{cost}(V_{k,*})$ denotes the cost of ant $k$ and $best\_cost$ is the cost of the best solution explored so far. Note that if an ant $k$ performs better than the average level of all ants, $\Delta_k$ is positive and hence corresponding pheromone trails will receive a reward. Otherwise, $\Delta_k$ is negative and corresponding pheromone trails will receive a penalty. It should be noted that ACO_DCOP cannot handle hard constraints (i.e., infinite costs for violated constraints) since it needs solution costs to compute deltas. After computing pheromone deltas, the lowest priority agent sends pheromone messages to all agents (line 31).

When receiving a pheromone message, agent $a_i$ updates the best value $d_i^*$ with the assignment in the best ant (line 32). Then related pheromone trails are updated according to the assignments for ant $k$ and the corresponding pheromone delta $\Delta_k$ (line 33 - 34). That is,

$$\tau_{ij}(V_{k,i}, V_{k,j}) = \tau_{ij}(V_{k,i}, V_{k,j}) + \delta_{k,ij}, \forall j \in H_i \tag{7}$$

where $\delta_{k,ij}$ is the weighted pheromone delta for the pheromone trail between $a_i$ and $a_j$. $\delta_{k,ij}$ is used to alleviate the penalty to the pheromone trail that has a small proportion to the cost of ant $k$. Formally, the weighted pheromone delta of ant $k$ for the selected pheromone trail between $a_i$ and $a_j$ is computed by:

$$\delta_{k,ij} = \begin{cases} \Delta_k & \Delta_k \geq 0 \\ \Delta_k \frac{\lambda \text{cost}_{ij}(V_{k,i}, V_{k,j})}{\text{cost}(V_{k,*})} & \Delta_k < 0 \end{cases} \tag{8}$$

where $\lambda$ is the number of constraints in a DCOP and can be easily counted by ants during the constructing solution procedure.

As the most of ant-based algorithms, an evaporation phase is performed after updating pheromone trails (line 35). Evaporation is a mechanism that allows ants to forget the information they have accumulated to help them to escape from local optima. Concretely, pheromone trails of $a_i$ are updated as:

$$\tau_{ij}(d_i, d_j) = (1 - \rho)\tau_{ij}(d_i, d_j) + \rho\tau_0 \tag{9}$$

where $d_i \in D_i, d_j \in D_j, j \in H_i$, and $0 < \rho < 1$ denotes the evaporation rate. Moreover, pheromone trails are bounded by a maximal value $\tau_{max}$ and a minimal value $\tau_{min}$.

We also include a phase to update the estimation $est_i$ for $a_i$ after the evaporation phase (line 36). Specifically, for each value $d_i \in D_i$, agent $a_i$ calculates the average cost with lower priority neighbors $L_i$ by considering ants whose values for $x_i$ are $d_i$ (line 39 - 46). Then, an estimation is updated by averaging itself and the corresponding average cost (line 47).

Consider agent $x_4$ in Fig. 4. When receiving a value message from $x_3$, the lowest priority agent $x_4$ is ready to compute pheromone deltas. Assume that $V_{1,*} = \{0, 1, 1, 0\}, V_{2,*} = \{1, 0, 0, 1\}$. Then the cost of ant 1 is 8 and the cost of ant 2 is 12. According to Equation (6), the pheromone delta for ant 1 and 2 are:

$$\Delta_1 = 1 - \frac{8-8}{10-8} = 1, \Delta_2 = 1 - \frac{12-8}{10-8} = -1$$

After that, $x_4$ sends the computed deltas ($\Delta = \{1, -1\}$) via pheromone messages to all agents. When receiving the pheromone message, each agent computes weighted pheromone deltas for pheromone trails based on Equation (8). That is:

$$\delta_{1,12} = \delta_{1,13} = \delta_{1,14} = \delta_{1,23} = \Delta_1 = 1$$

$$\delta_{2,12} = -1 \times \frac{4 \times 2}{12} = -0.667, \delta_{2,13} = -1 \times \frac{4 \times 3}{12} = -1$$

| $K$ \ $p_1$ / $n$ | 0.1 | 0.6 |
|---|---|---|
| 50 | 8 | 13 |
| 60 | 10 | 15 |
| 70 | 13 | 20 |

Figure 6: Recommended values for $K$ in different size of random DCOPs

$$\delta_{2,14} = -1 \times \frac{4 \times 4}{12} = -1.333, \delta_{2,23} = -1 \times \frac{4 \times 3}{12} = -1$$

After computing weighted pheromone deltas, each agent updates its pheromone trails by Equation (7) and evaporates its trails by Equation (9). We omit this because of lacking space. To illustrate the updating of estimation, let's consider $x_2$ in Fig. 4. According to function **UpdateEstimation**, we have:

$$est_2(0) = (est_2(0) + \text{cost}_{23}(V_{2,2}, V_{2,3}))/2 = (3+3)/2 = 3$$

$$est_2(1) = (est_2(1) + \text{cost}_{23}(V_{1,2}, V_{1,3}))/2 = (2+4)/2 = 3$$

At this point, one cycle is finished and agents start a next cycle by executing function **StartCycle** (line 37 - 38).

**Pipelining** is introduced to make full use of the computational capacity and reduce the solving time, since agents need to wait the lowest priority agent to send pheromone messages after selecting values in the algorithm.

Rather than waiting for the pheromone message, each agent in our algorithm selects values for ants in every iteration (i.e., the communication step between agents). To do this, the root agent must send value messages to its lower priority neighbors consecutively and each message is tagged with the number of iteration. The other agents group messages by tags and process each group as in the non-pipelining version.

## Theoretical Analysis

In this section, we will prove that our proposed method is an anytime algorithm, i.e., the quality of the solution can only remain the same or increase if more iterations are performed.

**Lemma 1.** *At the t+r iteration, the lowest priority agent holds enough information to calculate costs of all ants in iteration t, where r is a constant that only depends on the problem structure.*

*Proof.* We denote the length of the longest path (i.e., the message-passing path) in the ordered arrangement as $l$ and the length of the longest path to the lowest priority agent as $l'$. Directly from the algorithm, leaf agents (i.e., agents who do not have any lower priority neighbors) will send value messages to the lowest priority agent so long as they select values (line 22 - 23). Thus, $r$ actually depends on the lowest priority agent and the leaf agent who has the longest path from the root. Consider the following cases: 1) $l = l'$ and there are other leaf agents whose lengths of the longest path from the root are equal to $l$; 2) $l = l'$ and there is no other leaf agent whose length of the longest path from the root is equal to $l$; 3) $l > l'$, i.e.,there are some leaf agents whose lengths of the longest path from the root are greater than the one of the lowest priority agent.

Obviously, leaf agents whose lengths of the longest path from the root are $l$ will receive value messages from all their higher priority neighbors after $l$ iterations. Since they don't have any lower priority neighbors, the agents will send value messages to the lowest priority agent, which requires an extra iteration. Thus, the lowest priority agent is able to calculate costs of all ants after $r = l+1$ iterations in the first case. The similar analyses can be applied to the remaining cases and hence $r = l$ in the second case and $r = l+1$ in the third case. Thus, the lemma is proved. □

**Proposition 1.** *ACO_DCOP is an anytime algorithm, i.e., it reports the best result among first $t$ iterations after running for $t + r + 1$ iterations.*

*Proof.* Directly from Lemma 1 and the algorithm, the lowest priority agent can calculate costs of all ants and broadcasts the best ant $v^*$ explored so far via pheromone messages at the $t + r$ iteration (line 26 - 31). When an agent $a_i$ receives the pheromone message at the $t + r + 1$ iteration, it will update its best assignment $d_i^*$ with the corresponding assignment in $v^*$ (line 32). Thus, every agent holds the best assignment found among first $t$ iterations, which concludes the proposition. □

## Complexity Analysis

In every iteration, all agents except the lowest priority agent need to send value messages to their lower priority neighbors (or to the lowest priority agent) and the lowest priority agent needs to broadcast pheromone messages to all agents. Thus, the algorithm requires $n + \lambda + \epsilon - 1$ messages in each iteration, where $\lambda$ is the number of constraints and $\epsilon$ is the number of leaf agents. Since it contains the solutions constructed by all ants, the size of a value message is proportional to the number of agents and the number of ants, namely $O(nK)$. For a pheromone message, it contains the solutions constructed by all ants, the pheromone deltas for all ants and the assignments of the best ant. Thus, the size of a pheromone message is $O((K+1)n + K)$.

Each agent $a_i$ requires $O(w|H_i||D_i|)$ space to store pheromone trails, where $w = \max_{j \in H_i} |D_j|$. The time complexity of $a_i$ mainly depends on the computation of value messages. For each value in $D_i$ and each ant, $a_i$ needs to traverse all its higher priority neighbors to calculate the probability. Thus, $a_i$ requires $O(K|H_i||D_i|)$ operations to compute a value message.

## Empirical Evaluation

We empirically evaluate our proposed method with peer algorithms including DSA ($p = 0.8$), MGM2, DSAN, D-Gibbs, GDBA and the ant solver for DCSP (ACO_DCSP for short) on two type of problems: *random DCOPs* and *weighted graph coloring problems*. For random DCOPs, we set agent number to 70, domain size to 10 and uniformly select cost from $[1, 100]$. Each pair of agents is constrained independently with a probability $p_1 = 0.1$ (for the sparse configuration) or $p_1 = 0.6$ (for the dense configuration).

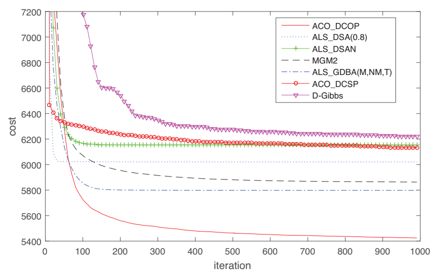


Figure 7: Comparison of ACO_DCOP and the competing algorithms on sparse configuration of random DCOPs

We consider weighted graph coloring problems with 120 agents, 3 available colors for each agent and $p_1 = 0.05$. The cost for each violation is uniformly selected from range 1 to 100. All algorithms are stopped after 1000 iterations. For non-monotonic algorithms, we present the anytime results by using the ALS framework. For ant-based algorithms, we consider the number of ants $K = 13$ for the sparse random DCOPs, $K = 20$ for the dense random DCOPs and weighted graph coloring problems, set $\tau_0 = 3$ for ACO_DCOP and $\tau_0 = 10$ for ACO_DCSP. Moreover, we implement ACO_DCSP as a pipelining version and set $\alpha = 2, \beta = 8, \rho = 0.02$. The experimental results are averaged over 50 independently generated problems that are each solved by each algorithm 30 times.

Our first experiment explores the relationship between the solution quality and ACO parameters including $\alpha$, $\beta$, $\rho$ and $K$. In our experiments, we received good results when $\alpha = 3, \beta = 4$ and $\rho = 0.0025$. For the number of ants, it is closely related to the scale of problems. We vary $K$ from 5 to 25 to evaluate the solution quality under different sizes of random DCOPs and present the recommended values for $K$ in Fig. 6.

Fig. 7 demonstrates the superiority of ACO_DCOP over all the competitors on the sparse configuration of random DCOPs. All differences at 1000 iterations are statistically significant for *p-value*<0.001. Since they do not have any global information, DSA and DSAN converge to poor results very quickly. MGM2 outperforms DSA since it can find a 2-coordinated solution, but it gets trapped in local optima after 400 iterations. GDBA performs slightly better than MGM2 within 150 iterations but it cannot do any further optimization. ACO_DCSP gradually optimizes the solution due to the ability to utilize global information. However, it can be observed that ACO_DCSP eventually produces a poor solution, which indicates that its optimization ability is too weak to solve DCOPs effectively. ACO_DCOP, on the other hand, finds better solutions than all the competing algorithms after 100 iterations and continues to improve slowly in the remaining iterations. At the end of execution, the improvements of ACO_DCOP over all the competing algorithms are 7.5%∼12.8%, which indicates that the heuristic factor and the pheromone factor can collaborate with each other to achieve a better balance between exploration and exploitation.

Fig. 8 shows the comparison of ACO_DCOP and the other

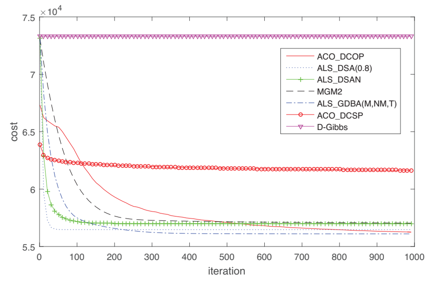


Figure 8: Comparison of ACO_DCOP and the competing algorithms on dense configuration of random DCOPs

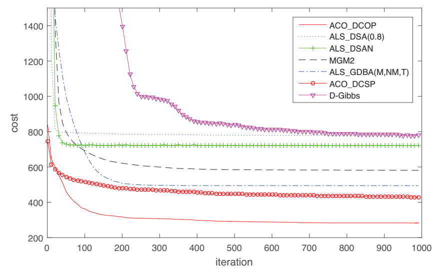


Figure 9: Comparison of ACO_DCOP and the competing algorithms on weighted graph coloring problems

algorithms on the dense configuration of random DCOPs. It can be concluded that ACO_DCOP has advantages over all the competitors other than GDBA about 0.4%∼23.1%, and is slightly inferior to GDBA. Besides, one can easily observe that although both ACO_DCOP and D-Gibbs make decisions stochastically according to probabilities, D-Gibbs cannot perform any optimization and eventually yields a poor result. This is because the probabilities in D-Gibbs are never changed and hence the previous solutions cannot provide any information for further optimization. However, the probabilities in ACO_DCOP are biased by the pheromones, which indicates that the previous solutions can directly feedback to the next optimization and hence enables ACO_DCOP to learn from experience.

Fig. 9 presents the comparison of ACO_DCOP and the competing algorithms on weighted graph coloring problems. It can be seen that ACO_DCSP exhibits more excellent performance over the other algorithms, which is not surprising since ACO_DCSP is originally designed to solve DCSPs. However, it is also noticeable that our proposed method outperforms all the competitors by 34.1%∼63.8%, which indicates that ACO_DCOP is also a very suitable algorithm for DCSPs.

## Conclusion

In this paper, we analyze the restrictions that prohibit ant-based algorithms from solving DCOPs efficiently and introduce a novel ant-based algorithm to solve DCOPs where agents utilize the pheromone factor and the heuristic factor to achieve the balance between exploration and exploitation. The pheromone factor is computed by considering the assignments of higher priority neighbors, while the heuristic factor is computed by a novel evaluation mechanism to the local benefit which includes a cost estimation for lower priority neighbors. We also propose a new method to calculate pheromone delta to cope with the complex cost structure in DCOPs. Finally, we introduce pipelining technique to improve the solving efficiency. Our experimental results show that ACO_DCOP is superior to the competing algorithms.

Also, ACO_DCOP builds a bridge between solving DCOPs and population-based optimization algorithms. We will explore more swarm intelligence approaches to solve DCOPs in our future work. We also attempt to extend the algorithm to solve asymmetric DCOPs.

## Acknowledgment

This research is funded by Chongqing Research Program of Basic Research and Frontier Technology (No. cstc2017jcyjAX0030).

## References

Arshad, M., and Silaghi, M. C. 2004. Distributed simulated annealing. *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems* 112.

Chen, Z.; Deng, Y.; and Wu, T. 2017. An iterative refined max-sum_ad algorithm via single-side value propagation and local search. In *Proc. of the 16th Conference on AAMAS*, 195–202.

Chen, Z.; He, Z.; and He, C. 2017. An improved dpop algorithm based on breadth first search pseudo-tree for distributed constraint optimization. *Applied Intelligence* 1–17.

Dorigo, M., and Gambardella, L. M. 1997. Ant colonies for the travelling salesman problem. *biosystems* 43(2):73–81.

Dorigo, M.; Birattari, M.; and Stutzle, T. 2006. Ant colony optimization. *IEEE computational intelligence magazine* 1(4):28–39.

Farinelli, A.; Rogers, A.; Petcu, A.; and Jennings, N. R. 2008. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. of the 7th Conference on AAMAS*, 639–646.

Fioretto, F.; Yeoh, W.; Pontelli, E.; Ma, Y.; and Ranade, S. J. 2017. A distributed constraint optimization (dcop) approach to the economic dispatch with demand response. In *Proc. of the 16th Conference on AAMAS*, 999–1007.

Gershman, A.; Meisels, A.; and Zivan, R. 2009. Asynchronous forward bounding for distributed cops. *Journal of Artificial Intelligence Research* 34:61–88.

Maheswaran, R. T.; Pearce, J. P.; and Tambe, M. 2004. Distributed algorithms for dcop: A graphical-game-based approach. In *Proceeding of ISCA PDCS'04*, 432–439.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2005. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence* 161(1-2):149–180.

Nguyen, D. T.; Yeoh, W.; and Lau, H. C. 2013. Distributed gibbs: A memory-bounded sampling-based dcop algorithm. In *Proc. of the 12th Conference on AAMAS*, 167–174.

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

Ottens, B.; Dimitrakakis, C.; and Faltings, B. 2012. Duct: An upper confidence bound approach to distributed constraint optimization problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 528–534.

Pearce, J. P., and Tambe, M. 2007. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *Proc. of the 20th IJCAI*, 1446–1451.

Petcu, A., and Faltings, B. 2005. A scalable method for multiagent constraint optimization. In *Proc. of the 19th IJCAI*, 266–271.

Rogers, A.; Farinelli, A.; Stranders, R.; and Jennings, N. R. 2011. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence* 175(2):730–759.

Semnani, S. H., and Zamanifar, K. 2012. The power of ants in solving distributed constraint satisfaction problems. *Applied Soft Computing* 12(2):640–651.

Solnon, C. 2002. Ants can solve constraint satisfaction problems. *IEEE transactions on evolutionary computation* 6(4):347–357.

Sultanik, E. A.; Modi, P. J.; and Regli, W. C. 2007. On modeling multiagent task scheduling as a distributed constraint optimization problem. In *Proc. of the 20th IJCAI*, 1531–1536.

Yeoh, W.; Felner, A.; and Koenig, S. 2008. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. In *Proc. of the 7th Conference on AAMAS*, 591–598.

Yokoo, M.; Durfee, E. H.; Ishida, T.; and Kuwabara, K. 1998. Distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Trans. Data Knowledge Engrg* 673–685.

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

Zhang, W.; Wang, G.; Xing, Z.; and Wittenburg, L. 2005. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial Intelligence* 161(1-2):55–87.

Zivan, R., and Peled, H. 2012. Max/min-sum distributed constraint optimization through value propagation on an alternating dag. In *Proc. of the 11th Conference on AAMAS*, 265–272.

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