Local Search Based Approximate Algorithm for Multi-Objective DCOPs

Maxime Wack[1], Tenda Okimoto[2], Maxime Clement[3], and Katsumi Inoue[4]

[1] Grenoble-INP: ESISAR, Valence, France
[2] Kobe University, Kobe, Japan
[3] The Graduate University for Advanced Studies, Tokyo, Japan
[4] National Institute of Informatics, Tokyo, Japan
{wack.max,forouhard}@gmail.com, tenda@maritime.kobe-u.ac.jp
inoue@nii.ac.jp

Abstract. Many real world optimization problems involve multiple criteria that should be considered separately and optimized simultaneously. A Multi-Objective Distributed Constraint Optimization Problem (MO-DCOP) is the extension of a mono-objective Distributed Constraint Optimization Problem (DCOP). A DCOP is a fundamental problem that can formalize various applications related to multi-agent cooperation. Solving an MO-DCOP is to find the Pareto front which is a set of cost vectors obtained by Pareto optimal solutions. In MO-DCOPs, even if a constraint graph has the simplest tree structure, the size of the Pareto front (the number of Pareto optimal solutions) is often exponential in the number of agents. Since finding all Pareto optimal solutions becomes easily intractable, it is important to consider fast but approximate algorithms. Various sophisticated algorithms have been developed for solving a DCOP and an MO-COP. However, there exists few works on an MO-DCOP. The Bounded Multi-Objective Max-Sum (B-MOMS) algorithm is the first and only existing approximate MO-DCOP algorithm. In this paper, we develop a novel approximate MO-DCOP algorithm called Distributed Iterated Pareto Local Search (DIPLS) and empirically show that DIPLS outperforms the state-of-the-art B-MOMS algorithm.

1 Introduction

A *Distributed Constraint Optimization Problem* (DCOP) [10, 16] is a fundamental problem that can formalize various applications related to multi-agent cooperation. A DCOP consists of a set of agents, each of which needs to decide the value assignment of its variables so that the sum of the resulting costs is minimized. Many application problems in multi-agent systems can be formalized as DCOPs, in particular, distributed resource allocation problems including meeting scheduling [6], sensor networks [5], and synchronization of traffic lights [4].

Many real world optimization problems involve multiple criteria that should be considered separately and optimized simultaneously. A *Multi-Objective Distributed Constraint Optimization Problem* (MO-DCOP) [1, 9, 12] is the extension of a mono-objective DCOP and a Multi-Objective Constraint Optimization

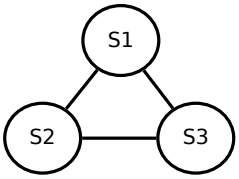
H.K. Dam et al. (Eds.): PRIMA 2014, LNAI 8861, pp. 390–406, 2014.

Problem (MO-COP) [7, 18, 19]. In MO-DCOPs, since trade-offs exist among objectives, there does not generally exist an ideal assignment, which minimizes all objectives simultaneously. Therefore, the "optimal" solution of an MO-DCOP is characterized by using the concept of *Pareto optimality*. An assignment is a Pareto optimal solution if there does not exist another assignment that weakly improves all of the objectives. Solving an MO-DCOP is to find the Pareto front which is a set of cost vectors obtained by all Pareto optimal solutions. Compared to DCOPs and MO-COPs, there exists few works on MO-DCOPs. The Bounded Multi-Objective Max-Sum (B-MOMS) algorithm [1] is the first and only existing approximate MO-DCOP algorithm which is an extension of the bounded max-sum algorithm [17] for solving a mono-objective DCOP. The B-MOMS works on a factor graph. It removes less important edges from a factor graph to make it cycle-free and obtains optimal solutions for the remaining cycle-free graph. A distributed search method with bounded cost vectors [9] is a complete algorithm which can guarantee to find all Pareto optimal solutions. This algorithm is a generalized ADOPT algorithm [10] that performs tree-search and partial dynamic programming. The Multi-Objective L_p-norm based Distributed Pseudo-tree Optimization Procedure (MO-DPOP$_{L_p}$) [12] is an incomplete algorithm which finds a subset of the Pareto front. The MO-DPOP$_{L_p}$ uses a widely used scalarization method and can guarantee to find a set of Pareto optimal solutions but not all.

Since finding all Pareto optimal solutions of MO-DCOPs becomes easily intractable for large-scale problem instances, it is important to consider fast but approximate algorithms. In MO-DCOPs, even if a constraint graph has the simplest tree structure, the number of all Pareto optimal solutions is often exponential (i.e. all assignments are Pareto optimal solutions in the worst case).

In this paper, we develop a novel approximate algorithm called *Distributed Iterated Pareto Local Search* (DIPLS) algorithm for solving an MO-DCOP. This algorithm is the extension of the well-known Pareto Local Search (PLS) [14], and we use it iteratively to generate an approximation of the Pareto front of an MO-DCOP. The PLS is the generalization of the hill-climbing method for optimization problems with multiple criteria. The DIPLS is the extension of this method for MO-DCOPs. In the experiments, we evaluate the performance of DIPLS with different problem settings and show that the local search technique is suitable for solving an MO-DCOP. We also compare DIPLS with the state-of-the-art approximate MO-DCOP algorithm B-MOMS, and empirically show that our proposed algorithm DIPLS outperforms the state-of-the-art B-MOMS.

About application domains of MO-DCOPs, we believe that sensor networks would be a promising area [11]. This problem is a kind of resource allocation problems which is a representative application problem for DCOPs. For example, consider a sensor network in a territory, where each sensor can sense a certain area in this territory. When we consider this problem with multiple criteria, e.g., data management, quality and quantity of observation data, and electrical consumption, this problem can be formalized as an MO-DCOP. Furthermore, when we consider a scheduling problem with several criteria, e.g., working hours, salary, and profit, it can be represented as an MO-DCOP. The other application



s_1	s_2	$cost$	s_2	s_3	$cost$	s_1	s_3	$cost$
a	a	5	a	a	0	a	a	1
a	b	7	a	b	2	a	b	1
b	a	10	b	a	0	b	a	0
b	b	12	b	b	2	b	b	3

Fig. 1. Example of mono-objective DCOP

problem for MO-DCOPs is wireless network of unmanned aerial vehicles [20]. Moreover, we believe that many DCOP application problems (concerned about "privacy") can be represented as MO-DCOPs by considering additional criteria.

The rest of the paper is organized as follows. In the next section, the formalizations of a DCOP and an MO-DCOP are introduced. The following section introduces a new approximate algorithm for MO-DCOPs. Afterwards, we compare our proposed algorithm with the state-of-the-art algorithm for MO-DCOPs. Just before the concluding section, some related works are discussed.

2 Preliminaries

In this section, we briefly describe the formalizations of Distributed Constraint Optimization Problems (DCOPs) and Multi-Objective Distributed Constraint Optimization Problems (MO-DCOPs) which is the extension of a DCOP.

2.1 Distributed Constraint Optimization Problem

A *Distributed Constraint Optimization Problem* (DCOP) [10, 16] is a fundamental problem that can formalize various applications related to multi-agent cooperation. In this paper, we assume all cost values are non-negative. Without loss of generality, we make the following assumptions for simplicity. Relaxing these assumptions to general cases is relatively straightforward:

- Each agent has exactly one variable.
- All constraints are binary.
- Each agent knows all constraints related to its variable.

A DCOP consists of a set of agents, each of which needs to decide the value assignment of its variables so that the sum of the resulting costs is minimized. This problem is defined by a set of agents S, a set of variables X, a set of constraint relations C, and a set of cost functions F. An agent i has its own variable x_i. A variable x_i takes its value from a finite, discrete domain D_i. A constraint relation (i, j) means there exists a constraint relation between x_i and x_j. For x_i and x_j, which have a constraint relation, the cost for an assignment

Table 1. Example of bi-objective DCOP

s_1	s_2	*cost vector*	s_2	s_3	*cost vector*	s_1	s_3	*cost vector*
a	a	(5,2)	a	a	(0,1)	a	a	(1,0)
a	b	(7,1)	a	b	(2,1)	a	b	(1,0)
b	a	(10,3)	b	a	(0,2)	b	a	(0,1)
b	b	(12,0)	b	b	(2,0)	b	b	(3,2)

$\{(x_i, d_i), (x_j, d_j)\}$ is defined by a cost function $f_{i,j}(d_i, d_j) : D_i \times D_j \to \mathbb{R}$. For a value assignment to all variables A, let us denote

$$R(A) = \sum_{(i,j)\in C, \{(x_i,d_i),(x_j,d_j)\}\subseteq A} f_{i,j}(d_i, d_j), \tag{1}$$

where $d_i \in D_i$ and $d_j \in D_j$. Then, an optimal assignment A^* is given as $\arg\min_A R(A)$, i.e., A^* is an assignment that minimizes the sum of the value of all cost functions. A DCOP can be represented using a constraint graph, in which a node corresponds to an agent and an edge represents a constraint.

Definition 1 (Total Ordering among Agents). *A total ordering among agents is a permutation of a sequence of agents $\langle s_1, s_2, ..., s_n\rangle$. We say agent s_{i+1} has higher priority than s_i $(1 \le i \le n-1)$.*

Example 1 (DCOP). Figure 1 shows a DCOP with three agents s_1, s_2 and s_3. Each agent/variable takes its value assignment from a discrete domain $\{a, b\}$. The table shows three cost tables among three agents. The optimal solution of this problem is $\{(s_1, a), (s_2, a), (s_3, a)\}$, and the optimal value is six.

2.2 Multi-objective Distributed Constraint Optimization Problem

A *Multi-Objective Distributed Constraint Optimization Problem* (MO-DCOP) [1, 9, 12] is the extension of a mono-objective DCOP. An MO-DCOP is defined with a set of agents S, a set of variables X, multi-objective constraints $C = \{C^1, \ldots, C^m\}$, i.e., a set of sets of constraint relations, and multi-objective functions $O = \{O^1, \ldots, O^m\}$, i.e., a set of sets of objective functions. For an objective l $(1 \le l \le m)$, a cost function $f^l_{i,j} : D_i \times D_j \to \mathbb{R}$, and a value assignment to all variables A, let us denote

$$R^l(A) = \sum_{(i,j)\in C^l, \{(x_i,d_i),(x_j,d_j)\}\subseteq A} f^l_{i,j}(d_i, d_j), \tag{2}$$

where $d_i \in D_i$ and $d_j \in D_j$. Then, the sum of the values of all cost functions for m objectives is defined by a cost vector, denoted $R(A) = (R^1(A), \ldots, R^m(A))$. Finding an assignment that minimizes all objective functions simultaneously is ideal. However, in general, since trade-offs exist among objectives, there does not exist such an ideal assignment. Therefore, the optimal solution of an MO-DCOP

is characterized using the concept of *Pareto optimality*. Since this possible trade-off between objectives, the size of the Pareto front is exponential in the number of variables, i.e., every possible assignment can be Pareto optimal solution in the worst case. An MO-DCOP can be also represented using a constraint graph.

Definition 2 (Dominance). *For an MO-DCOP and two cost vectors $R(A)$ and $R(A')$, we call that $R(A)$ dominates $R(A')$, denoted by $R(A) \prec R(A')$, iff $R(A)$ is partially less than $R(A')$, i.e., it holds*

- $R^l(A) \leq R^l(A')$ *for all objectives* l, *and*
- *there exists at least one objective* l', *such that* $R^{l'}(A) < R^{l'}(A')$.

Definition 3 (Pareto Optimal Solution). *For an MO-DCOP, an assignment A is said to be the Pareto optimal solution, iff there does not exist another assignment A', such that $R(A') \prec R(A)$.*

Definition 4 (Pareto Front). *For an MO-DCOP, a set of cost vectors obtained by Pareto optimal solutions is said to be the Pareto front. Solving an MO-DCOP is to find the Pareto front.*

Example 2 (MO-DCOP). Table 1 shows a bi-objective DCOP, which is an extension of a DCOP in Figsure 1. Each agent takes its value from a discrete domain $\{a, b\}$. The Pareto optimal solutions of this problem are $\{\{(s_1, a), (s_2, a), (s_3, a)\}$ and $\{(s_1, a), (s_2, b), (s_3, b)\}\}$, and the Pareto front is $\{(6, 3), (10, 1)\}$.

2.3 Local Search

Local search algorithms are one of the most successful method for solving a wide variety of single objective optimization problems. However, it is really easy to adapt this notion to the multi-objective optimization problems. As in the first case, we use the same notion of neighborhood. But we need to redefine the criterion of acceptance for one solution. For the single-objective case, a solution is usually accepted if it is better than the current one, it is important to take into account several objectives. A simple approach may be to use the notion of dominance defined earlier. A solution is now accepted if and only if it is non dominated by another already discovered. All the solutions that are dominated by this new one are then deleted from the set of current solutions. Finally, the obtained set of non-dominated solutions is an approximation of the Pareto Front.

3 Distributed Iterated Pareto Local Search Algorithm

In this section, we develop a novel approximate algorithm called *Distributed Iterated Pareto Local Search* (DIPLS) algorithm for solving an MO-DCOP. This algorithm is the extension of the Pareto Local Search (PLS) [14], and we use it iteratively to generate an approximation of the Pareto front of an MO-DCOP. The PLS is the generalization of the hill-climbing method for optimization problems with multiple criteria. DIPLS is the distributed extension of this method.

Algorithm 1. Distributed Random Solution Generator for s_i

```
1: Required : a fixed total ordering on the agents: ⟨s_1, ..., s_n⟩
2: terminated: false
3: cpa_i: current partial solution
4: c_i: cost vector of cpa_i
5: if i = 1 then
6:    Assigns a random value and compute the cost c_1
7:    Send message (PATH, cpa_1, c_1) to agent s_2
8:    Set terminated true
9: end if
10: while s_i not terminated do
11:    s_i receive message M
12:    if M = (PATH, cpa_{i-1}, c_{i-1}) then
13:       cpa_i ← cpa_{i-1} // Update cpa
14:       Choose a random value and compute c_i of cpa_i
15:       if (i ≠ n) then
16:          Send message (PATH, cpa_i, c_i) to s_{i+1}
17:       else
18:          randomSol ← cpa_n
19:       end if
20:       Set terminated true
21:    end if
22: end while
23: Ensure randomSol : a random solution.
```

The DIPLS uses local search approaches that have been already addressed in DCOPs [2, 3] and also been extended to Multi-Objective Optimization Problems (MOOP) [14, 21]. The basic idea of this algorithm is to try to evolve an initial population generated randomly by the agents, toward the Pareto front. The DIPLS has the following two phases:

Phase 1 : Generate the Initial solutions.
Phase 2 : Use a distributed PLS to evolve non-dominated solutions.

Let us describe phase 1. The initial solutions generation phase is trivial. The agents pick randomly some value for their variable. Then, each agent sends its value to its neighbors in the constraint graph and receives the assignments of its neighbors. The cost vector associated to the solution is computed. Algorithm 1 shows the pseudo-code to be executed by each agent in order to generate a random solution. This algorithm requires a total ordering on agents. Agents, starting by the first, choose randomly the value for their variable (lines 6 and 14) and pass around a single PATH message that includes the current partial assignment to the higher-priority agents and the current associated cost vector (lines 7 and 16). The algorithm stops once all agents assigned a value to their variable, and the process is repeated for each new random solution needed.

In phase 2, the obtained random solutions are iteratively evolved toward the Pareto front using a distributed iterated Pareto local search technique which is an extension of the local search algorithm to the distributed and multi-objective case. This algorithm uses the same notion of neighborhood as in the mono-

Algorithm 2. Distributed Pareto Local Search for s_i

```
1: Require a fixed total ordering on agents: ⟨s_1, ..., s_n⟩
2: listRand: a list of random solutions
3: archive: empty
4: terminated: false
5: if i = n then
6:    archive ← filter listRand by dominance
7:    Broadcast message (ARCHIVE, archive)
8: end if
9: while s_i is not terminated do
10:   s_i receives message M
11:   if M = TERMINATE then
12:      Set terminated true
13:   end if
14:   if M = (ARCHIVE, archive) then
15:      neighbors ← createNeighbors(archive)
16:      Send message (MERGE, neighbors) to s_n
17:   end if
18:   if M = (MERGE, neighbors) then
19:      Merge archive and neighbors
20:      if all merge messages received then
21:         Filter archive by dominance
22:         if new non-dominated solution in archive then
23:            Broadcast message (ARCHIVE, archive)
24:         else
25:            Broadcast message (PF, archive) // Pareto front approximation
26:            Broadcast message (TERMINATE)
27:         end if
28:      end if
29:   end if
30: end while
31: Ensure archive, a Pareto front approximation.
```

objective case. However, the acceptance criterion of the mono-objective local search algorithms needs to be changed to take into account several objectives. The pseudo-code of the distributed Pareto local search algorithm is given in Algorithm 2. This algorithm requires a total ordering on agents and the list of randomly generated solutions, and executes as follows : one agent, the controller (last agent), initially filters the list of random solutions by removing the dominated solutions and adds the non-dominated to an archive (line 6). It then broadcasts an ARCHIVE message that includes the archive (line 7). For each ARCHIVE message received (line 14), agents generates neighbors (line 15) and send MERGE messages including a list of generated neighbors to the controller. For each MERGE message received, the controller adds the received list of neighbors to the archive (line 18-19). After receiving MERGE messages from all the agents (line 20), the controller filters (by dominance) the archive (line 21) and if a new non-dominated solution has been added into the archive, it broadcasts an ARCHIVE message (line 22,23) and the process is repeated until no new non-dominated neighbor can be found starting from a solution of the archive.

Algorithm 3. Create neighbors for s_i

1: **Require** *archive*: a list of solutions
2: **for** each solution s_j in *archive* **do**
3: $neighbor_j \leftarrow$ copy of s_j
4: **for** each value v_k in s_i's domain **do**
5: in $neighbor_j$, s_i assigns v_k to its variable and create $neighbor_{j,k}$
6: Compute cost of $neighbor_{j,k}$
7: Add $neighbor_{j,k}$ to $Neighbors_i$
8: **end for**
9: **end for**
10: Filter by dominance $Neighbors_i$
11: **Ensure** $Neighbors_i$, the list of non-dominated neighbors of s_i.

The algorithm 3 presents the pseudo-code that allows an agent to generate neighbors when it receives an ARCHIVE message. For each solution in the archive, the agent assigns each domain value to its variable and computes the new corresponding cost vector. Each modification of the variable assignment leads to the creation of a neighbor which is added to a list of neighbors (line 1-9). At the end, the agent filters its list of neighbors by dominance and only the non-dominated neighbors will be send via the MERGE message to the controller.

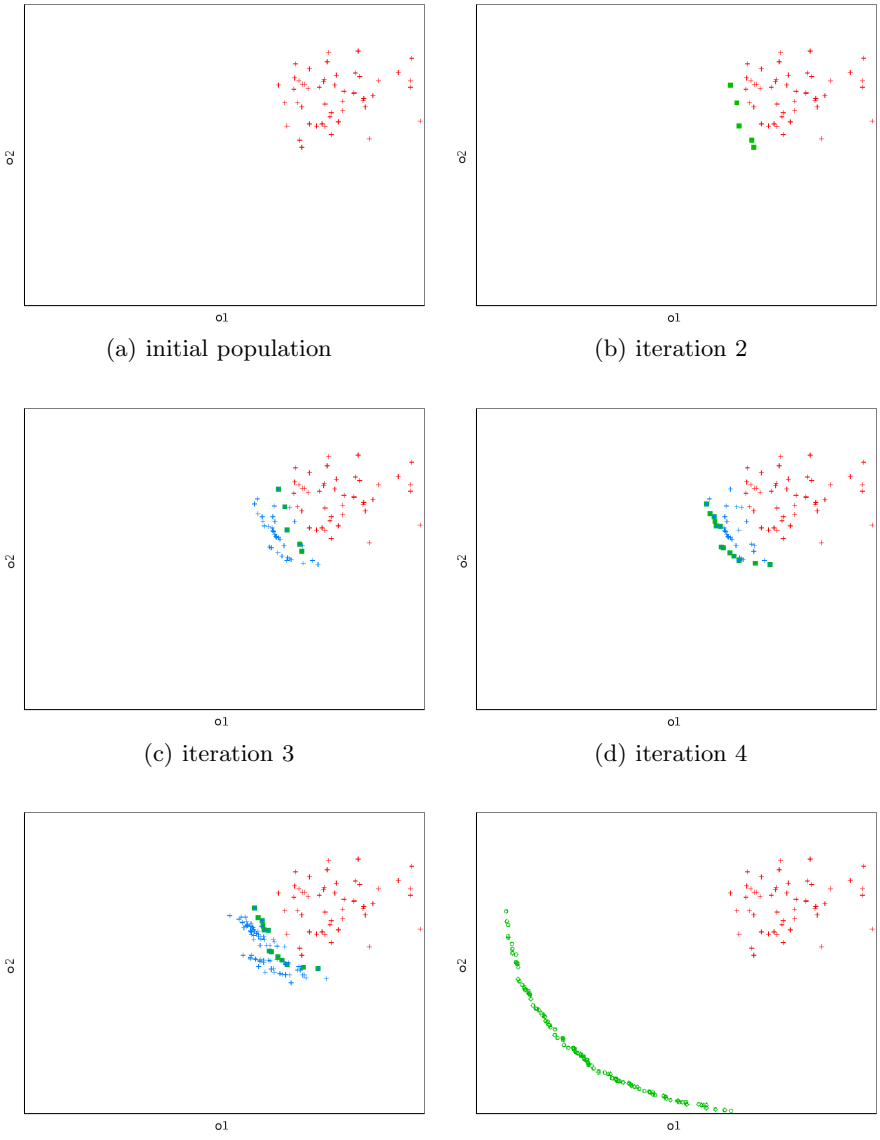
Figure 2 shows the example of the behavior of DIPLS, how it finds the approximation of the Pareto front of an MO-DCOP. It starts with an initial set of solutions (Figure 2(a)). The square points on the figures represent the contents of the ARCHIVE messages sent by the controller to all the agents (Figure 2(b)), while the blue points represent the set of all the generated neighbors sent by each agent to the controller (MERGE messages) (Figure 2(c)). The algorithm is executed iteratively while a new non-dominated solution is found (Figure 2(d)-(e)). At the end, DIPLS provides an approximation of Pareto front (Figure 2(f)).

4 Experimental Evaluation

Experimental Setting

In this section, we compare the performances of DIPLS and the state-of-the-art approximate MO-DCOP algorithm B-MOMS. In our evaluations, we use the following problem instances: the domain size of each variable is two, and the cost values are randomly chosen from the range [0,100] for each objective. We solve bi-objective problem instances. Each data point in a graph represents an average of 100 problem instances. We generate random graphs varying the number of nodes and densities ($\delta \in [0.1, 1.0]$). The density is the constraint tightness of a problem instance by controlling the number of edges as follows. $|E| = \delta \times \frac{1}{2}|S|(|S| - 1)$, where $|S|$ is the number of agents. We implemented these algorithms in Java. All the experiments were carried out on 2.3GHz core with 4GB of RAM.

In order to evaluate the performances of DIPLS and B-MOMS, we define the following three metrics: Let PO be a set of all Pareto optimal solutions of an



(a) initial population (b) iteration 2

(c) iteration 3 (d) iteration 4

(e) iteration 5 (f) Approximation of Pareto front

Fig. 2. Behavior of DIPLS. By Algorithm 1, the agents pick randomly some value for their variable (a). The square points on the figures represent the contents of the ARCHIVE messages (b). The new cross points represent the set of all the generated neighbors (c). DIPLS executes (b) and (c) iteratively while a new non-dominated solution is found (d)-(e). At the end, it provides an approximation of Pareto front (f).

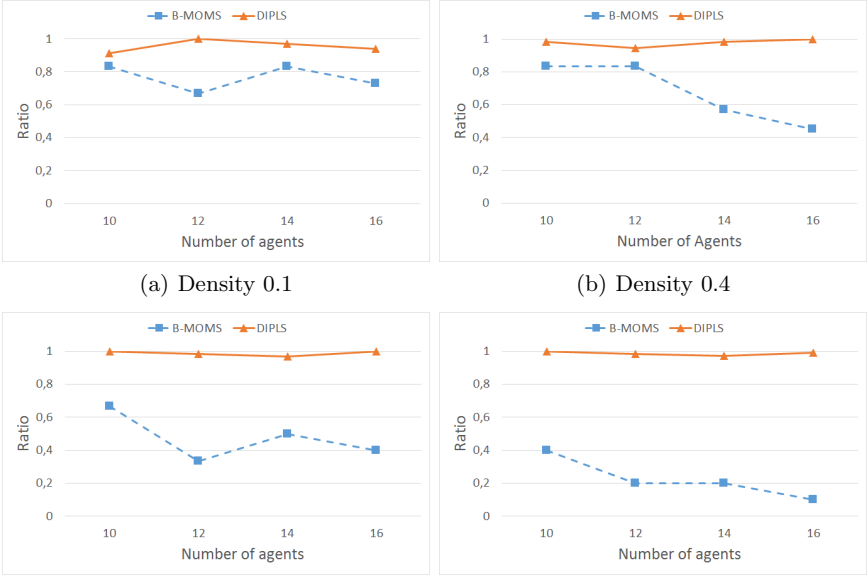
MO-DCOP and $\widetilde{PO}$ be an approximation of PO obtained by DIPLS and B-MOMS. The metric 1 represents the ratio of the Pareto optimal solutions over the set of obtained solutions by DIPLS and B-MOMS. The metric 2 shows the ratio of the obtained Pareto optimal solutions by DIPLS and B-MOMS over the whole set of Pareto optimal solutions of an MO-DCOP. The metric 3 is the required CPU runtime to compute $\widetilde{PO}$.

- Metric 1 $= \dfrac{|\widetilde{PO} \cap PO|}{|\widetilde{PO}|}$.
- Metric 2 $= \dfrac{|\widetilde{PO} \cap PO|}{|PO|}$.
- Metric 3 = runtime to compute $\widetilde{PO}$.

In our experiments, we use the similar setting as in [1]. For metric 1 and 2, it is required to compute a set of all Pareto optimal solutions (PO). To compute the PO of an MO-DCOP, we use a brute-force optimal algorithm like [1]. Since finding all Pareto optimal solutions is exponential in the number of agents ($|S|$), we only report these three metrics for problem instances with $|S| \leq 16$. To go further, we show the quality solutions obtained by DIPLS for 3 and 4 objectives.

Experimental Results (Comparison with B-MOMS)

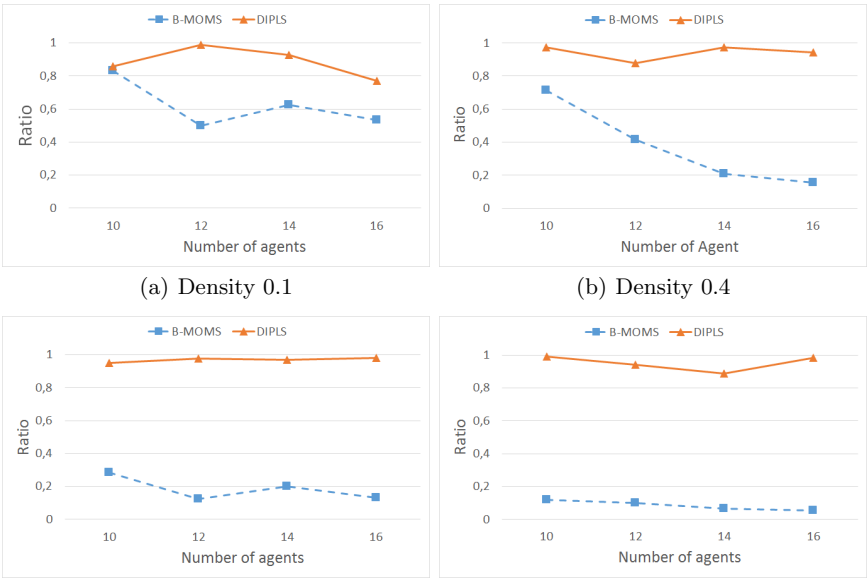
Figure 3 represents the results of metric 1 for constraint graphs with the density 0.1, 0.4, 0.7 and 1.0, varying the number of agents from 10 to 16. The line with



(a) Density 0.1 (b) Density 0.4

(c) Density 0.7 (d) Density 1.0

Fig. 3. Results of metric 1 for DIPLS and B-MOMS



(a) Density 0.1 (b) Density 0.4

(c) Density 0.7 (d) Density 1.0

Fig. 4. Results of metric 2 for DIPLS and B-MOMS

triangle represents the results for our algorithm DIPLS and the line with square represents the results for the state-of-the-art B-MOMS. The x axis shows the number of agents/variables and the y axis represents the results of metric 1, i.e., the ratio of the Pareto optimal solutions over the set of obtained solutions by DIPLS and B-MOMS. We can see that over 90% of all obtained solutions by DIPLS are Pareto optimal solutions, and these results are independent on the densities of constraint graphs (Figure 3(a)-(d)). Additionally, the quality (i.e. the ratio) does not change when the number of agents increases for all densities. When the number of agents is 16 for the density 0.1, the ratio is 0.94 for DIPLS, while it is 0.98 for the density 1.0. On the other hand, for B-MOMS, by increasing the density of the constraint graph, i.e., by increasing the number of constraints in the problem, we can observe that the performances of B-MOMS become worse (Figure 3(a)-(d)). When the number of agents is 16 for the density 0.1, the ratio is 0.74 for B-MOMS, while it is 0.12 for the density 1.0. This can be explained by the number of removed edges in B-MOMS which increases for dense graphs. The experimental results reveal that DIPLS outperforms B-MOMS for metric 1. Furthermore, the performance of DIPLS is not affected by the density of a constraint graph (i.e. the number of constraints) and the number of agents. Also, the difference of the solution quality between DIPLS and B-MOMS becomes larger when the density of a constraint graph and the number of agents increase.

Figure 4 shows the results of metric 2 for DIPLS and B-MOMS. We obtained the similar results as in Figure 3, i.e., DIPLS outperforms B-MOMS for all cases (Figure 4 (a)-(d)). DIPLS can obtain more than 75% over the whole set of Pareto

(a) Density 0.1 (b) Density 0.4

(c) Density 0.7 (d) Density 1.0

Fig. 5. Runtime of DIPLS and B-MOMS

optimal solutions for all cases. On the other hand, B-MOMS can obtain more than 50% of all Pareto optimal solutions for sparse graphs (i.e. constraint graph with low density). However, by increasing the density (i.e. Figure 4 (b)-(d)), the ratio becomes worse and the difference of the results between DIPLS and B-MOMS become larger. When the number of agents is 16 and the density is 0.1, the ratio for DIPLS is 0.77 and the ratio for B-MOMS is 0.54. In the case where density is equal to 0.4, the ratio obtained is 0.93 for DIPLS and 0.17 for B-MOMS. For the density 0.7, it is 0.98 for DIPLS and 0.15 for B-MOMS, and, finally, when the density is 1.0, it is 0.98 for DIPLS and 0.06 that for B-MOMS. The experimental results reveal that DIPLS can obtain more Pareto optimal solutions than B-MOMS. Also, the performance of DIPLS is not affected by the density of a constraint graph and the number of agents as in Figure 3.

Figure 5 shows the results of the average runtime in DIPLS and B-MOMS for constraint graphs with the density 0.1, 0.4, 0.7 and 1.0, varying the number of agents from 10 to 100. In Figure 5(a), the average runtime of DIPLS and B-MOMS increases significantly when the number of agents is upper than 60. We can see the similar results for all densities (see (a)-(d)). Also, when the number of agents is large (more than 60 agents), the average runtime of DIPLS is shorter compared to those for B-MOMS. Additionally, in case the number of agents is smaller than 60, we can see that both results are almost same for most cases, and they are independent from the density. The experimental results for metric

3 reveal that the average runtime in DIPLS is shorter compared to those in B-MOMS for large-scale and complex (high density) problem instances.

In summary, these experimental results reveal that (i) the quality of the obtained solutions by DIPLS is better compared with B-MOMS, (ii) DIPLS can obtain more Pareto optimal solutions than B-MOMS, and (iii) the required runtime of DIPLS is shorter. Also, the differences of these results (i)-(iii) become more significant when we increase the density and the number of agents.

Let us consider why our algorithm DIPLS can obtain better results compared to B-MOMS. This is because B-MOMS obtains an optimal solution for a relaxed problem, i.e., it looses the informations of the original problem by removing some constraints, while DIPLS does not relax the original problem (we never remove the constraints from the graph). If the relaxed problem is not so different from the original problem, the both algorithms can find a better solution quickly.

Experimental Results (Quality Solutions)

In this section, we show the quality solutions obtained by DIPLS for three and four objectives. Table 2 represents the results of the metrics 1 and 2 with three objectives, and Table 3 shows those for four objectives. In both tables, we also show the runtime and the number of all Pareto optimal solutions denoted $\#POS$. In Table 2, we can see that DIPLS can obtain good quality solutions, i.e. the results of metric 1 and 2 are more than 90% for all densities, and also the results are independent on the number of agents (see (a)-(d)). In Table 3, we can see the similar results as in Table 2, i.e., all results of metric 1 and 2 exceed 90% for all cases. These experimental results reveal that the quality solutions obtained by DIPLS do not change by increasing the number of objectives. Furthermore, we observed that the number of Pareto optimal solutions increases when we

Table 2. Results of DIPLS for MO-DCOPs with 3 objectives

(a) Density 0.1

$\#agents$	$metrics1$	$metrics2$	$Runtime$	$\#POS$
10	1.0	0.988	0.008	20
11	1.0	0.997	0.018	39
12	0.999	0.966	0.030	59
13	1.0	1.0	0.057	85
14	1.0	0.970	0.073	90
15	1.0	0.970	0.093	80
16	0.998	0.991	0.410	192

(b) Density 0.4

$\#agents$	$metrics1$	$metrics2$	$Runtime$	$\#POS$
10	1.0	0.989	0.010	13
11	1.0	1.0	0.037	66
12	1.0	0.994	0.035	55
13	1.0	0.986	0.037	41
14	0.974	0.915	0.065	55
15	1.0	0.996	0.167	114
16	0.997	0.997	0.323	161

(c) Density 0.7

$\#agents$	$metrics1$	$metrics2$	$Runtime$	$\#POS$
10	1.0	1.0	0.021	28
11	1.0	0.990	0.028	41
12	1.0	0.995	0.032	31
13	1.0	0.964	0.060	42
14	1.0	0.990	0.254	146
15	1.0	0.965	0.156	83
16	0.998	0.982	0.315	129

(d) Density 1.0

$\#agents$	$metrics1$	$metrics2$	$Runtime$	$\#POS$
10	1.0	1.0	0.025	31
11	1.0	1.0	0.064	74
12	1.0	0.994	0.072	64
13	1.0	0.992	0.120	83
14	1.0	0.985	0.113	64
15	1.0	0.958	0.126	55
16	1.0	1.0	0.430	40

Table 3. Results of DIPLS for MO-DCOPs with 4 objectives

(a) Density 0.1

#agents	*metrics*1	*metrics*2	*Runtime*	*#POS*
10	1.0	1.0	0.089	190
11	1.0	0.978	0.104	46
12	1.0	0.998	0.069	119
13	1.0	1.0	0.109	147
14	1.0	1.0	0.194	187
15	1.0	1.0	1.576	538
16	1.0	0.998	2.322	572

(b) Density 0.4

#agents	*metrics*1	*metrics*2	*Runtime*	*#POS*
10	1.0	1.0	0.045	100
11	1.0	1.0	0.153	208
12	1.0	1.0	0.499	364
13	1.0	1.0	0.220	194
14	1.0	0.990	0.119	107
15	1.0	0.999	1.962	594
16	1.0	0.992	1.585	459

(c) Density 0.7

#agents	*metrics*1	*metrics*2	*Runtime*	*#POS*
10	0.999	0.996	0.043	61
11	1.0	1.0	0.228	208
12	1.0	0.998	0.086	86
13	1.0	1.0	0.140	117
14	1.0	0.993	0.295	178
15	1.0	0.991	0.286	147
16	0.999	0.995	1.268	370

(d) Density 1.0

#agents	*metrics*1	*metrics*2	*Runtime*	*#POS*
10	1.0	1.0	0.082	111
11	1.0	1.0	0.113	119
12	1.0	1.0	0.388	269
13	1.0	0.999	0.664	319
14	1.0	0.990	0.444	213
15	1.0	0.999	1.637	450
16	1.0	0.999	2.890	577

increase the number of objectives. In Table 2 (a), when the number of agents is 16, the number of Pareto optimal solutions ($\#POS$) is 192, while $\#POS$ is 572 for four objectives (Table 3 (a)). In Table 2 (d), in case the number of agents is 16, $\#POS$ is 40, while $\#POS$ is 577 for 4 objectives (Table 3 (d)). The runtime of our algorithm increases, when the number of objectives increases. In Table 2 (a), when the number of agents is 16, the runtime is 0.4, while it is 2.3 for four objectives (Table 3 (a)). In Table 2 (d), in case the number of agents is 16, the runtime is 0.4, while it is 2.8 for four objectives (Table 3 (d)). We consider that this is because the runtime depends on the number of Pareto optimal solutions. For the relationship between the number of objectives and the quality solution, and also the runtime, we will analyze more detailed in our future work.

5 Related Works

The Bounded Multi-Objective Max-Sum (B-MOMS) algorithm [1] is the first and only existing approximate MO-DCOP algorithm which is an extension of the bounded max-sum algorithm [17] for solving a mono-objective DCOPs. The B-MOMS works on a factor graph. It considers the importance of edges and removes less important edges from a factor graph to make it cycle-free, and obtain optimal solutions for the remaining cycle-free graph. For approximate algorithms, providing the bound of a solution is one of the important issues. The B-MOMS can provide the bound of a solution a posteriori, i.e., the error bound is obtained only after we actually run the algorithm and obtain an approximate solution. Having a priori bound, i.e., the error bound is obtained before actually running the algorithm, is desirable, but a posteriori bound is usually more accurate. Compared to B-MOMS, DIPLS cannot guarantee the quality bound.

Various approximate algorithms have been developed for solving a MO-COP, e.g., Multi-Objective Mini-Bucket Elimination (MO-MBE) [18], Multi-objective Best- First AND/OR search algorithm (MO-AOBF) [8], and Multiobjective A* search algorithm (MOA*) [15]. MO-MBE computes a set of lower bounds of MO-COPs. MO-AOBF and MOA* compute a relaxed Pareto front using ϵ-dominance [13]. Most of these approximate algorithms are extension of the representative search and inference based mono-objective COP algorithms. DIPLS is the local search based algorithm, and our experimental results reveal that the local search technique is suitable for solving a MO-DCOP. We consider that this is because of the huge number of Pareto optimal solutions, i.e., small local change has a big chance to obtain the Pareto optimal solution in MO-DCOPs.

6 Conclusion

Many real world optimization problems involve multiple criteria that should be considered separately and optimized simultaneously. An MO-DCOP is a DCOP which involves multiple criteria. In MO-DCOPs, since finding all Pareto optimal solutions is not realistic, it is important to consider fast but approximate algorithms. In this paper, we developed a novel approximate algorithm called Distributed Iterated Pareto Local Search (DIPLS) algorithm. DIPLS use PLS iteratively to generate an approximation of the Pareto front of an MO-DCOP. In the experiments, we evaluated the performance of DIPLS with different problem settings. We compared DIPLS with the state-of-the-art approximate algorithm B-MOMS and empirically showed that DIPLS outperforms B-MOMS. Our experimental results reveal that (i) the quality of the obtained solutions by DIPLS is better compared with B-MOMS, (ii) DIPLS can obtain more Pareto optimal solutions than B-MOMS, and (iii) the required runtime of DIPLS is shorter. Our future works include developing an approximate algorithm which can provide the bound of a solution a priori and a posteriori. Also, we will extend approximate DCOP algorithms for solving an MO-DCOP, and compare the performances of these algorithms with DIPLS. Furthermore, we intend to apply DIPLS on challenging real world problems, e.g., sensor network and scheduling problems.

References

[1] Fave, F.D., Stranders, R., Rogers, A., Jennings, N.: Bounded decentralised coordination over multiple objectives. In: Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, pp. 371–378 (2011)

[2] Fitzpatrick, S., Meertens, L.: An experimental assessment of a stochastic, anytime, decentralized, soft colourer for sparse graphs. In: Steinhöfel, K. (ed.) SAGA 2001. LNCS, vol. 2264, pp. 49–64. Springer, Heidelberg (2001)

[3] Hirayama, K., Yokoo, M.: The distributed breakout algorithms. Artificial Intelligence 161(1-2), 89–115 (2005)

[4] Junges, R., Bazzan, A.: Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In: Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems, pp. 599–606 (2008)
[5] Lesser, V., Ortiz, C., Tambe, M. (eds.): Distributed Sensor Networks: A Multiagent Perspective (Edited book), May 2003. Kluwer Academic Publishers (2003)
[6] Maheswaran, R., Tambe, M., Bowring, E., Pearce, J., Varakantham, P.: Taking dcop to the real world: efficient complete solutions for distributed multi-event scheduling. In: Proceedings of the 3rd International Conference on Autonomous Agents and Multiagent Systems, pp. 310–317 (2004)
[7] Marinescu, R.: Exploiting problem decomposition in multi-objective constraint optimization. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 592–607. Springer, Heidelberg (2009)
[8] Marinescu, R.: Best-first vs. depth-first and/or search for multi-objective constraint optimization. In: Proceedings of the 22nd IEEE International Conference on Tools with Artificial Intelligence, pp. 439–446 (2010)
[9] Matsui, T., Silaghi, M., Hirayama, K., Yokoo, M., Matsuo, H.: Distributed search method with bounded cost vectors on multiple objective dCOPs. In: Rahwan, I., Wobcke, W., Sen, S., Sugawara, T. (eds.) PRIMA 2012. LNCS, vol. 7455, pp. 137–152. Springer, Heidelberg (2012)
[10] Modi, P., Shen, W., Tambe, M., Yokoo, M.: ADOPT: asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence 161(1-2), 149–180 (2005)
[11] Okimoto, T., Ribeiro, T., Clement, M., Inoue, K.: Modeling and algorithm for dynamic multi-objective weighted constraint satisfaction problem. In: Proceedings of the 6th International Conference on Agents and Artificial Intelligence, pp. 420–427 (2014)
[12] Okimoto, T., Schwind, N., Clement, M., Inoue, K.: Lp-norm based algorithm for multi-objective distributed constraint optimization. In: Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems, pp. 1427–1428 (2014)
[13] Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pp. 86–92 (2000)
[14] Paquete, L., Chiarandini, M., Stutzle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study. In: Metaheuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems, pp. 177–200. Springer (2004)
[15] Perny, P., Spanjaard, O.: Near admissible algorithms for multiobjective search. In: Proceedings of the 18th European Conference on Artificial Intelligence, pp. 490–494 (2008)
[16] Petcu, A., Faltings, B.: A scalable method for multiagent constraint optimization. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, pp. 266–271 (2005)
[17] Rogers, A., Farinelli, A., Stranders, R., Jennings, N.: Bounded approximate decentralised coordination via the max-sum algorithm. Artificial Intelligence 175(2), 730–759 (2011)
[18] Rollon, E., Larrosa, J.: Bucket elimination for multiobjective optimization problems. Journal of Heuristics 12(4-5), 307–328 (2006)
[19] Rollon, E., Larrosa, J.: Multi-objective russian doll search. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence, pp. 249–254 (2007)

[20] Sivakumar, A., Tan, C.: UAV swarm coordination using cooperative control for establishing a wireless communications backbone. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 1157–1164 (2010)
[21] Thibaut, L., Jacques, T.: Two-phase pareto local search for the biobjective traveling salesman problem. Journal of Heuristics 16(3), 475–510 (2010)