PAPER Special Section on Frontiers in Agent-based Technology

Distributed Pareto Local Search for Multi-Objective DCOPs

Maxime CLEMENT[†,††a)], ***Nonmember***, **Tenda OKIMOTO**[†††b)], ***Member***, ***and*** **Katsumi INOUE**[†,††c)], ***Nonmember***

SUMMARY Many real world optimization problems involving sets of agents can be modeled as Distributed Constraint Optimization Problems (DCOPs). A DCOP is defined as a set of variables taking values from finite domains, and a set of constraints that yield costs based on the variables' values. Agents are in charge of the variables and must communicate to find a solution minimizing the sum of costs over all constraints. Many applications of DCOPs include multiple criteria. For example, mobile sensor networks must optimize the quality of the measurements and the quality of communication between the agents. This introduces trade-offs between solutions that are compared using the concept of Pareto dominance. Multi-Objective Distributed Constraint Optimization Problems (MO-DCOPs) are used to model such problems where the goal is to find the set of Pareto optimal solutions. This set being exponential in the number of variables, it is important to consider fast approximation algorithms for MO-DCOPs. The bounded multi-objective max-sum (B-MOMS) algorithm is the first and only existing approximation algorithm for MO-DCOPs and is suited for solving a less-constrained problem. In this paper, we propose a novel approximation MO-DCOP algorithm called Distributed Pareto Local Search (DPLS) that uses a local search approach to find an approximation of the set of Pareto optimal solutions. DPLS provides a distributed version of an existing centralized algorithm by complying with the communication limitations and the privacy concerns of multi-agent systems. Experiments on a multi-objective extension of the graph-coloring problem show that DPLS finds significantly better solutions than B-MOMS for problems with medium to high constraint density while requiring a similar runtime.

key words: *multi-objective, distributed constraint optimization problem, pareto local search*

1. Introduction

A *Distributed Constraint Optimization Problem* (DCOP) [1], [2] is a fundamental problem that can formalize various applications related to multi-agent cooperation, e.g., distributed resource allocation problems including meeting scheduling [3], sensor networks [4], and synchronization of traffic lights [5]. A DCOP consists of a set of agents, each deciding the value assignment of its variables so that the sum of the resulting costs is minimized.

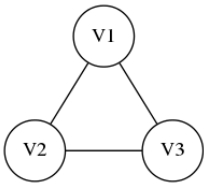
Many real world optimization problems involve multiple criteria that should be considered separately and optimized simultaneously. For example, we can imagine wireless sensor networks where multiple criteria are considered, e.g., data management, quality and quantity of observation data, and electrical consumption, or distributed meeting scheduling problems where we care about total meetings attended, travel times, and the preferences of the participants. In such multi-objective optimization problems, since trade-offs exist among objectives, there generally does not exist an ideal assignment optimizing all objectives simultaneously. Therefore, the optimal solution of a multi-objective problem is characterized by using the concept of *Pareto optimality*. An assignment is a Pareto optimal solution if there does not exist another assignment that can weakly improve all the objectives. We then call *Pareto front* the set of objective vectors generated by the Pareto optimal solutions.

Some frameworks have been introduced for representing multi-objective problems in constraint optimization. In the centralized case, a Multi-Objective Constraint Optimization Problem (MO-COP) [6]–[8] represents the problem of finding the set of Pareto optimal solutions. In the distributed case, a *Multi-Objective Distributed Constraint Optimization Problem* (MO-DCOP) [9]–[11] was proposed as the extension of a mono-objective DCOP. Compared to DCOPs and MO-COPs, there exists only a few works for MO-DCOPs, all of them based on previous works for DCOPs. The Bounded Multi-Objective Max-Sum (B-MOMS) algorithm [9] is an approximation MO-DCOP algorithm extending the bounded max-sum algorithm [12]. A distributed search method with bounded cost vectors [10] generalizes ADOPT [1], a popular DCOP algorithm, and can guarantee to find a single Pareto optimal solution. Finally, the Multi-Objective L_p-norm based Distributed Pseudo-tree Optimization Procedure (MO-DPOP$_{L_p}$) [11] is an incomplete algorithm combining DPOP [2], a dynamic programming DCOP algorithm, with scalarization techniques to find an incomplete set of Pareto optimal solutions.

Since in a multi-objective problem all assignments can produce non-dominated solutions, finding all Pareto optimal solutions becomes easily intractable for large-scale instances. This is why it is important to study fast approaches that can approximate the Pareto front. Moreover, using MO-COP techniques in a distributed setting would require that one agent gathers all information about the problem and performs all computations. However, agents in a distributed system generally care about their privacy and are limited in their communication and computation capabilities, thus requiring an MO-DCOP algorithm.

Manuscript received January 27, 2017.
Manuscript revised June 9, 2017.
Manuscript publicized September 15, 2017.

[†]The authors are with The Graduate University for Advanced Studies, Tokyo, 240–0193 Japan.
[††]The authors are with National Institute of Informatics, Tokyo, 101–8430 Japan.
[†††]The author is with Kobe University, Kobe-shi, 657–8501 Japan.
a) E-mail: maxime-clement@nii.ac.jp
b) E-mail: tenda@maritime.kobe-u.ac.jp
c) E-mail: inoue@nii.ac.jp
DOI: 10.1587/transinf.2016AGP0006



v_1	v_2	$cost$	v_2	v_3	$cost$	v_1	v_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.

In this paper, we develop a novel approximation algorithm called *Distributed Pareto Local Search* (DPLS) algorithm for solving an MO-DCOP. This algorithm is the extension of the well-known Pareto Local Search (PLS) [13] designed for approximating the Pareto front of multi-objective optimization problems. PLS is the generalization of the hill-climbing method for optimization problems with multiple criteria. With the DPLS, we propose an extension of this method for MO-DCOPs. In the experiments, we evaluate the performances of DPLS with different problem settings and show that the local search technique is suitable for solving an MO-DCOP. We also compare DPLS with the state-of-the-art approximation MO-DCOP algorithm B-MOMS, and empirically show that our proposed algorithm DPLS outperforms the state-of-the-art B-MOMS.

The work presented in this paper is a revised and extended version of the conference paper [14]. Compared to the preliminary version, the algorithm proposed in this paper uses an exploration parameter e to adjust the number of assignments whose neighborhood should be explored at each iteration. It also fully respects the assumption that agents can only communicate with each other if their variables share a constraint. Moreover, experimental results were previously shown on random instances but are now performed on a multi-objective extension of graph-coloring.

The rest of the paper is organized as follows. In Sect. 2, the formalism of a DCOP and an MO-DCOP are introduced, as well as the Pareto Local Search. In Sect. 3, we present a new approximation algorithm for MO-DCOPs which extends the Pareto Local Search to a distributed environment. In Sect. 4, we evaluate the performance of our proposed algorithm on instances of multi-objective graph-coloring problem and compare these results with the state-of-the-art approximation algorithm for MO-DCOPs. In Sect. 5, we present the related works before concluding in Sect. 6.

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), and introduce the basic idea behind designing a local search in a multi-objective problem.

2.1 Distributed Constraint Optimization Problem

Definition 1 (DCOP): A *Distributed Constraint Optimization Problem* (DCOP) [2] is defined as a tuple $DCOP = (X, V, \mathcal{D}, C, F)$ where $X = \{x_1, \ldots, x_n\}$ is a set of agents, $V = \{v_1, \ldots, v_n\}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains, $C = \{C_1, \ldots C_c\}$ is a set of constraint relations, and $F = \{f_1, \ldots, f_c\}$ is a set of cost functions. Each agent x_i is in charge of a variable $v_i \in V$ that takes its value from a finite, discrete domain $D_i \in \mathcal{D}$. A constraint relation $C_j \in C, C_j \subset V$ means that there exists a constraint between the variables $v_i \in C_j$. The cost function $f_j : \times_{\forall v_i \in C_j} D_i \to \mathbb{R}_{\geq 0}, f_j \in F$, represents the cost generated by constraint C_j based on the values taken by its variables. An assignment A is a set of values such that $A_i \in D_i$ is the value assigned to variable v_i. When an assignment A is complete, i.e., $|A| = |V|$, we can evaluate its quality by summing the costs of all constraints:

$$R(A) = \sum_{C_j \in C} f_j(A)$$

The solution of a DCOP is an optimal assignment A^o minimizing the sum of all cost functions, given as $A^o \in \arg\min_A R(A)$.

A DCOP can be represented using a hypergraph, called a *constraint hypergraph*, in which nodes represent variables and edges represent constraints.

Example 1 (DCOP): Figure 1 shows a DCOP with three variables v_1, v_2 and v_3. Each variable takes its value assignment from a discrete domain $D_1 = D_2 = D_3 = \{a, b\}$. The table shows three cost tables among three variables. The optimal solution of this problem is $A^o = \{(v_1, a), (v_2, a), (v_3, a)\}$, with an optimal cost $R(A^o) = 6$.

2.2 Multi-Objective Distributed Constraint Optimization Problem

Definition 2 (Multi-Objective DCOP): A *Multi-Objective Distributed Constraint Optimization Problem* (MO-DCOP) [9]–[11] is defined as a tuple $MO\text{-}DCOP = (X, V, \mathcal{D}, C, \mathcal{F})$ where $X = \{x_1, \ldots, x_n\}$ is a set of agents, $V = \{v_1, \ldots, v_n\}$ is a set of variables, $\mathcal{D} = \{D_1, \ldots, D_n\}$ is a set of domains, $C = \{C_1, \ldots C_c\}$ is a set of constraint relations, and $\mathcal{F} = \{F_1, \ldots, F_m\}$ is a set of sets of cost functions. For an objective l $(1 \leq l \leq m)$, a cost function $f_j^l : \times_{\forall v_i \in C_j} D_i \to \mathbb{R}_{\geq 0}, f_j^l \in F_l$, represents the cost generated by constraint C_j for the objective l. We can now represent the quality of a complete assignment A by a cost vector $R(A) = (R^1(A), \ldots, R^m(A))$ where

$$R^l(A) = \sum_{C_j \in C} f_j^l(A)$$

Table 1 Example of bi-objective DCOP.

v_1	v_2	*cost vector*	v_2	v_3	*cost vector*	v_1	v_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)

Algorithm 1 Pareto Local Search

1: **Input**: $\mathcal{A}_0$ an initial set of assignments
2: $\mathcal{A} \leftarrow ND(\mathcal{A}_0)$
3: *archive* $\leftarrow \mathcal{A}$
4: **while** *archive* $\neq \emptyset$ **do**
5: $A \leftarrow$ random assignment in *archive*
6: **for** each $A' \in \mathcal{N}(A)$ **do**
7: $\mathcal{A} \leftarrow ND(\mathcal{A} \cup \{A'\})$
8: **end for**
9: explored(A) $\leftarrow$ true
10: *archive* $\leftarrow \{A \in \mathcal{A} | \neg explored(A)\}$
11: **end while**
12: **Output**: $\mathcal{A}$, an approximation of the Pareto front

Finding an assignment that minimizes all objective functions simultaneously is ideal. However, trade-offs can exist among the different objectives and there usually does not exist such an ideal assignment. Therefore, optimal solutions of an MO-DCOP are characterized using the concept of *Pareto optimality*.

Definition 3 (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 4 (Pareto optimal solution): For an MO-DCOP, an assignment A is said to be a *Pareto optimal solution*, iff there does not exist another assignment A', such that $R(A') \prec R(A)$.

Definition 5 (Pareto Front): For an MO-DCOP, the *Pareto front* is the set of cost vectors obtained by the Pareto optimal solutions.

Solving an MO-DCOP consists in finding its Pareto front whose size is, in the worst case, exponential in the number of variables.

Example 2 (MO-DCOP): Table 1 shows an MO-DCOP with two objectives, extending the DCOP of Example 1. Each variable takes its value from a discrete domain $\{a, b\}$. The Pareto optimal solutions of this problem are $\{\{(v_1, a), (v_2, a), (v_3, a)\}$ and $\{(v_1, a), (v_2, b), (v_3, b)\}\}$, and the Pareto front is $\{(6, 3), (10, 1)\}$.

2.3 Pareto Local Search

The idea of a local search is to iteratively improve a solution by exploring its neighborhood. The best solution within this neighborhood is kept and its own neighborhood is in turn explored. This is repeated until no improvement can be found in the neighborhood of the current best solution. In the mono-objective case, a total ordering of the solutions exists, making the selection of the improving solution straightforward. However, in the multi-objective case, multiple solutions can offer an improvement when using the dominance relation, requiring new dominance-based algorithms.

For centralized Multi-Objective Optimization Problems, the Pareto Local Search (PLS) [13] was proposed as an extension of the standard local search from the mono-objective case to the multi-objective case. Algorithm 1 shows the PLS framework, which can be used to centrally solve an MO-DCOP. PLS takes as input an initial set of assignments $\mathcal{A}_0$ which forms the initial set of non-dominated assignments $\mathcal{A}$ (line 1) to be returned at the end of the search. At each iteration, an assignment A is randomly taken from the *archive* (line 1), the set of assignments in $\mathcal{A}$ not yet explored. Using an operator ND, which returns all the non-dominated solutions in a given set, we add all non-dominated neighbors of A to $\mathcal{A}$ (line 1) and consider A to now be explored. PLS terminates once the *archive* is empty, meaning that all non-dominated assignments in $\mathcal{A}$ have been explored.

Similarly to a mono-objective local search, PLS requires to define a neighborhood operator $\mathcal{N}(A)$ (line 1) that generates the neighborhood of A by applying small changes to it. Common neighborhood operators change the value of one variable at a time or swaps values between two variables. Compared to a mono-objective local search where only one local optimal solution need to be considered at a time, PLS requires to maintain a set of locally optimal solutions (denoted by $\mathcal{A}$ in the algorithm).

In the next section, we propose to distribute the PLS in order to solve MO-DCOPs in a distributed fashion.

3. Distributed Pareto Local Search Algorithm

In this section, we propose a novel approximation algorithm for MO-DCOPs called *Distributed Pareto Local Search* (DPLS). This algorithm extends the Pareto Local Search (PLS) [13] presented in Sect. 2.3, from a centralized to a distributed setting.

To adapt the PLS for distributed settings, we assume that: (i) an agent only knows the cost functions involving its own variable, and (ii) two agents can only communicate if a constraint exists between their variables. To comply with the first assumption, we suppose that an agent x_i only knows its local cost functions which include variable v_i. To comply with the second assumption, agents are ordered in a *pseudo-tree*, a popular graph structure for DCOP algorithms [15]. In

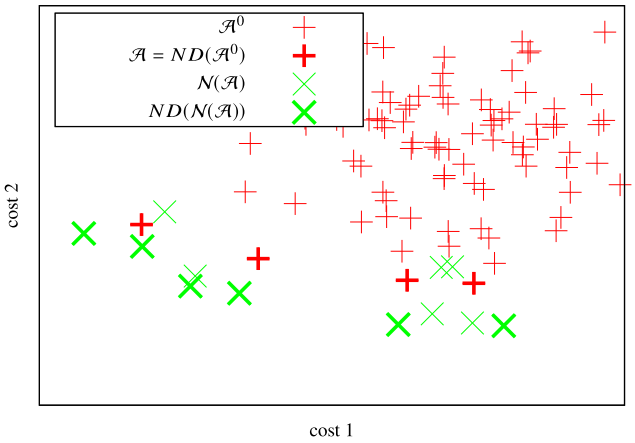


Fig. 2 Example of the first iteration of DPLS

a pseudo-tree, all variables sharing a constraint are required to be part of a same path between the root and a leaf. Such structure can be obtained using a depth-first traversal of the constraint hypergraph. Each agent x_i can only send or receive messages to or from its parent $parent_i$, and children $children_i$, in the pseudo-tree.

The DPLS has two phases. In the first phase, an initial set $\mathcal{A}^0$ of solutions is generated. In the second phase, a local search is performed in a distributed fashion, maintaining a set of non-dominated solutions $\mathcal{A}$, initialized using $\mathcal{A}^0$. At each iteration, the neighborhoods of some solutions in $\mathcal{A}$ are explored in an attempt to find new non-dominated solutions. Figure 2 shows an example of the first iteration of DPLS. From a random set of solutions ($\mathcal{A}^0$), only the non-dominated solutions $\mathcal{A}$ are considered. Their neighborhood $\mathcal{N}(\mathcal{A})$ is explored, offering new non-dominated solutions. To perform this search in a distributed way, each agent x_i uses a local operator $\mathcal{N}_i$ such that $\mathcal{N}_i(A)$ generates the set of assignments that differs from A only by the value of variable v_i. At each iteration, the root sends the assignments to explore down the pseudo-tree. Each agent then computes its corresponding local neighborhood and sends the new non-dominated solutions up the tree. To limit the number and size of messages, an agent waits for the solutions from its children before sending its own neighborhood. In addition, it compares its own neighborhood with the ones received from its children and discards all solutions that are dominated. The iteration finishes once the root has received neighborhoods from all its children and updated the set of optimal solutions accordingly. The assignments explored at each iteration are randomly selected within the set of *unexplored* non-dominated solutions and a parameter e is used to adjust the number of solutions selected. Increasing this number can lead to faster convergence, decreasing the number of messages used by DPLS but increasing their size. This can be an important adjustment in distributed systems depending on the quality of the communication network.

3.1 Algorithm

Algorithm 2 shows the pseudo-code for generating a random solution. Starting from the leaf agents in the pseudo-tree, the idea of this algorithm is to propagate a partial assignment pA up the pseudo-tree, along with its cost. Each agent x_i starts by selecting a random value from its variable domain

Algorithm 2 Distributed Generation of a Solution for x_i

```
1:  pA_i ← random value in D_i
2:  r_i ← 0
3:  for each x_j ∈ children_i do
4:      x_i receives (pA_j, r_j) from x_j
5:      pA_i ← pA_i ∪ pA_j
6:      r_i ← r_i ⊕ r_j
7:  end for
8:  r_i ← r_i ⊕ R_i(pA_i)
9:  send (pA_i, r_i) to parent_i
10: Output: a random solution pA (at root agent)
```

Algorithm 3 Distributed Pareto Local Search for x_i

```
1:  Input: A_0 an initial set of assignments; e the maximum
    number of assignments to explore at each iteration
2:  A ← ND(A_0)
3:  if x_i is root then
4:      archive ← A
5:      subarchive ← selectSubset(archive, e)
6:      send (subarchive) to each x_j ∈ children_i
7:      neighbors_i ← ND(N_i(subarchive))
8:  end if
9:  while ¬terminated_i do
10:     x_i receives message M
11:     if M = (terminate) then
12:         terminated_i ← true
13:         send (terminate) to each x_j ∈ children_i
14:     end if
15:     if M = (subarchive) then
16:         send (subarchive) to each x_j ∈ children_i
17:         neighbors_i ← ND(N_i(subarchive))
18:         if x_i is leaf then
19:             send (neighbors_i) to parent_i
20:         end if
21:     end if
22:     if M = (neighbors_j) then
23:         neighbors_i ← neighbors_i ∪ neighbors_j
24:         if x_i received all neighbors_j, ∀x_j ∈ children_j then
25:             neighbors_i ← ND(neighbors_i)
26:             send (neighbors_i) to parent_i
27:             if x_i is root then
28:                 explored(A) ← true, ∀A ∈ subarchive
29:                 A ← ND(A ∪ neighbors_i)
30:                 archive ← {A ∈ A|¬explored(A)}
31:                 if archive = ∅ then
32:                     terminated_i ← true
33:                     send (terminate) to each x_j ∈ children_i
34:                 else
35:                     subarchive ← selectSubset(archive, e)
36:                     send (subarchive) to each x_j ∈ children_i
37:                     neighbors_i ← ND(N_i(subarchive))
38:                 end if
39:             end if
40:         end if
41:     end if
42: end while
43: Output: A, an approximation of the Pareto front
```

D_i (line 2) that is added to the partial assignments received from its children (line 2). A partial cost r_i is computed by adding the costs received from the children r_j (line 2) with the local cost generated by the partial assignment $R_i(pA)$ (line 2). Due to the requirement that variables sharing a constraint are part of a same path between the root of the pseudo-tree and a leaf agent, an agent x_i can compute a local cost vector $R_i(pA_i)$ corresponding to the costs generated by all the constraints involving x_i and its descendants. This partial cost, along with the partial assignment, is sent to the parent or, in the case of the root agent, the assignment is complete and we know its corresponding cost vector.

Algorithm 3 shows the pseudo-code of the search phase. Similarly to the centralized PLS, we maintain a set of non-dominated solutions $\mathcal{A}$ from which we consider *archive*, the set of solutions in $\mathcal{A}$ not yet explored. $\mathcal{A}$ and *archive* are initialized with the set of non-dominated solutions generated using Algorithm 2. At each iteration, the root selects a subset *subarchive* of size e from the *archive* (line 3). This *subarchive* is sent down the pseudo-tree and corresponding non-dominated neighbors are sent back up the tree. Upon reaching the root, the new non-dominated solutions are added to $\mathcal{A}$ and the *archive* is updated accordingly. This is repeated until all solutions in $\mathcal{A}$ have been explored.

To perform the search, three messages are used. To propagate the assignments to explore, we use *subarchive* messages (line 3). The search is started when the root agent sends the first *subarchive* to explore. Upon receiving this message, an agent first forwards it to all its children before computing its local neighborhood (line 3). If the agent is a leaf in the pseudo-tree, it can directly send this neighborhood to his parent. The *neighbors* message (line 3) is used to bring the neighborhood of each agent back to the root. An agent receives this message from its children, combining the received neighborhood with its local one (line 3). Once the neighborhood of each children have been received, the resulting set of non-dominated solutions are sent to the parent node (line 3). In the case of the root agent, we now have the complete set of non-dominated solutions of the current *subarchive*. We consider assignments in the *subarchive* to now be explored (line 3), and the newly found solutions are added to $\mathcal{A}$ (line 3). A new *archive* (line 3) and *subarchive* (line 3) are generated, and a new iteration is started by sending the *subarchive* to the children. If at this point the *subarchive* is empty, the search terminates and $\mathcal{A}$ contains the non-dominated solutions encountered during the search. To end the algorithm, the root sends *terminate* messages (line 3) that are propagated by every agent down the pseudo-tree (line 3).

3.2 Properties

Property 1 (Termination): DPLS terminates when all encountered non-dominated solutions ($\mathcal{A}$) have been explored, i.e., we searched their neighborhood for new solutions. This means that in the worst case, DPLS terminates once the whole search space has been explored.

Proof 1: DPLS terminates once the *archive*, the set of assignments in $\mathcal{A}$ that are not yet explored, becomes empty. Since a problem has a finite number of possible assignments and at each iteration some assignments in $\mathcal{A}$ become explored, the *archive* eventually becomes empty and DPLS terminates.

Property 2 (Anytime): DPLS is anytime, i.e., at the end of each iteration the root knows all the non-dominated solutions encountered so far (maintained in $\mathcal{A}$).

Proof 2: At each iteration, we explore the neighborhoods $\mathcal{N}(A), \forall A \in$ *subarchive*. After that, non-dominated solutions within the neighborhoods are added to $\mathcal{A}$ and all assignments in the *subarchive* are considered *explored*. Since a solution in $\mathcal{A}$ is discarded only if it is dominated by another solution, we can guarantee the following at the end of each iteration: $\mathcal{A} = ND(\bigcup_{A|explored(A)} \mathcal{N}(A))$.

4. Experimental Evaluation

In this section, we evaluate the performances of DPLS for various settings and compare it with B-MOMS [9], the state-of-the-art approximation algorithm for MO-DCOP.

4.1 Experimental Setting

In our evaluations, we consider an extended graph-coloring problem originally proposed for evaluating the performances of B-MOMS. While this is an abstract problem, it is used in real applications such as scheduling problems [16] and wireless sensor networks [17]. In this problem, each agent x_i owns a variable v_i taking its value from the domain $D_i = \{1, 2, 3\}$. Three cost functions corresponding to three objectives are defined:

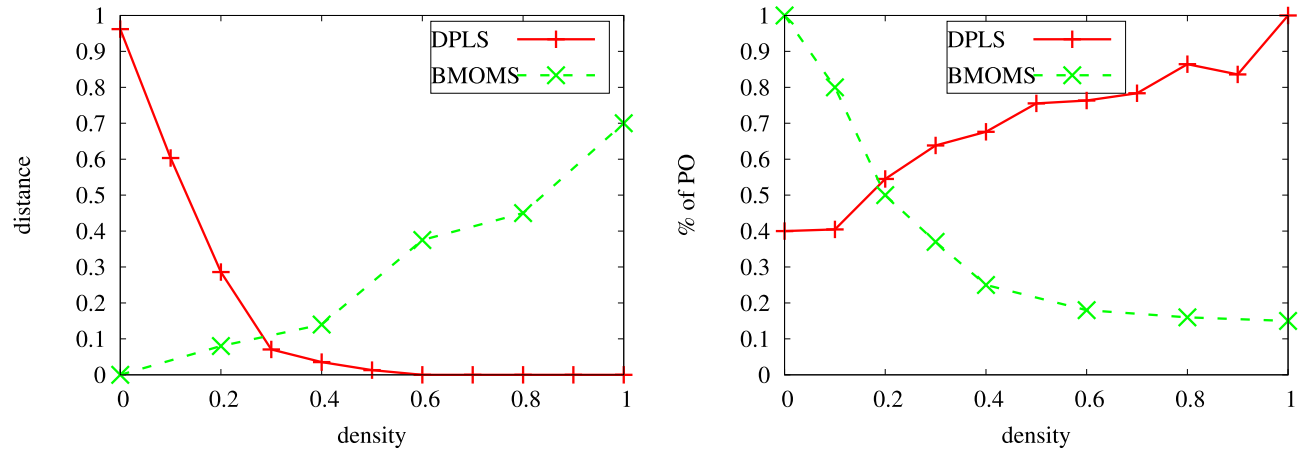
$$f^1(v_i, v_j) = \begin{cases} 0 & v_i \neq v_j \\ 1 & v_i = v_j \end{cases}, f^2(v_i, v_j) = \begin{cases} 0 & \text{if } |v_i - v_j| = 1 \\ 1 & \text{otherwise} \end{cases},$$

$$f^3(v_i, v_j) = \begin{cases} 0 & \text{if } i < j \text{ and } v_j < v_j \\ 1 & \text{otherwise} \end{cases}$$

The first objective represents the common graph-coloring conflict function. The second objective aims towards having a distance of one between the values of the variables. The third objective considers that variables with higher indexes should take higher values.

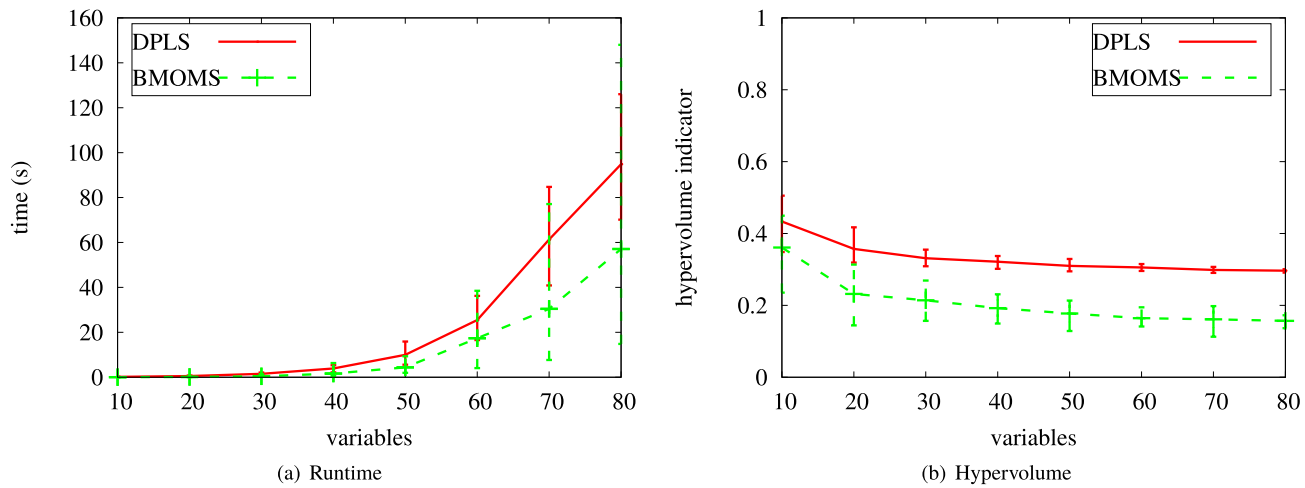
To produce our instances, random connected graphs were generated by specifying $|V|$ the number of variables and δ the density of the graph. When $\delta = 0$, the number of edges in the graph is equal to $|V| - 1$ and when $\delta = 1$, the number of edges is equal to $\frac{|V|}{2}(|V| - 1)$.

In order to evaluate the performances of DPLS and compare them with B-MOMS, we use three different metrics, namely *minimum Euclidean distance*, *fraction of*



(a) Min. distance from a Pareto optimal solution (b) Fraction of PO found

Fig. 3 Results for DPLS and B-MOMS on multi-objective graph-coloring instances with $|V| = 14$.



(a) Runtime (b) Hypervolume

Fig. 4 Results with $\delta = 0.5$ and varying $|V|$.

Pareto optimal solutions found and the *hypervolume indicator*. Let PO be a set of all Pareto optimal solutions of an MO-DCOP and $\widetilde{PO}$ be an approximation of PO obtained by DPLS and B-MOMS. The *minimum Euclidean distance* measure the minimum distance between solutions $A \in \widetilde{PO}$ and an optimal solution $A^o \in PO$. The *fraction of Pareto optimal solutions found* is expressed as: $\frac{|\widetilde{PO} \cap PO|}{|PO|}$. The *hypervolume indicator* [18] is a measure of the volume of the objective space covered by a set of solutions. This last measure has the advantage of not requiring the exact set of Pareto optimal solutions, requiring instead a reference point to define the range of the objective space. For the multi-objective graph-coloring problem, the cost of each objective is bounded by the number of constraints in the problem and we use these bounds as reference point.

To compute PO, we use a brute-force optimal algorithm whose complexity is exponential in the number of variables ($|V|$). We thus could not compute PO for large instances and only report the *distance* and *fraction of Pareto optimal solutions* for problem instances with $|V| = 14$. For $|V| > 14$, we use the *hypervolume indicator* to measure the ratio of optimal solution space covered by $\widetilde{PO}$. We consider as optimal solution space the hypervolume ranging from the reference point to the utopia point where all objectives are equal to 0.

For all experiments, DPLS was started with 100 randomly generated assignments and each iteration explored the neighborhood of one assignment ($e = 1$).

4.2 Experimental Results

Figure 3 shows the results obtained with B-MOMS and DPLS over 40 multi-objective graph-coloring instances with $|V| = 14$ and varying the density. Figure 3 (a) shows the minimum Euclidean distance between $\widetilde{PO}$ and PO, and Fig. 3 (b) shows the ratio of Pareto optimal solutions found by B-MOMS and DPLS. At low densities, DPLS easily fall into a local optima where changing the value of a single variable does not improve the solution cost. This causes the

quality of DPLS to vary a lot depending on the instances, sometimes finding the full Pareto front and other times not finding a single Pareto optimal solution. This leads, for density $\delta < 0.2$, to finding an average of only around 40% of all Pareto optimal solutions. B-MOMS, which is exact for $\delta = 0$, can still provide 80% of the Pareto solutions for $\delta = 0.1$. When density increases and each variable is involved in more constraints, the chances for improvements using local search also increases and DPLS finds more than 60% of all optimal solutions when density is above 0.3. B-MOMS, which relies on removing edges from the constraint hypergraph, suffers a great decline in solution quality when the density increases, finding less than 25% of PO for $\delta > 0.4$.

We now consider results obtained when varying $|V|$ between 10 and 80 with density $\delta = 0.5$. Figure 4 shows the average runtime and hypervolume indicator obtained over 40 instances. We first notice that the runtime of both B-MOMS and DPLS increases exponentially with the number of variables, following the increase in the size of the search space. Regarding the quality of the solutions obtained, we see that for quite dense graphs, DPLS always finds better approximations, with the gap between the two algorithms increasing from a 5% difference for $|V| = 10$ to a 20% difference for $|V| = 80$.

In summary, we showed that DPLS is able to find significantly better solutions than B-MOMS on problem of medium and high density, finding more optimal solutions for density $\delta > 0.25$. For reference, graph-coloring instances based on geographical constraints can have a density of up to $\delta = 0.63$. While DPLS is slightly slower than B-MOMS, both algorithms share a similar time complexity that is exponential in the number of variables. DPLS however has the advantage of being anytime.

5. Related Works

The DPLS uses local search approaches that have been already addressed in DCOPs [19] and also been extended to Multi-Objective Optimization Problems (MOOP) [13]. Compared to the centralized Pareto Local Search (PLS), DPLS adds an exploration parameter e allowing to adjust between the strategy of PLS consisting in exploring one solution per iteration, and exploring all non-dominated solutions [20].

For MO-DCOPs, the Bounded Multi-Objective Max-Sum (B-MOMS) algorithm [9] is the first and only existing approximation algorithm. It is an extension of the Bounded Max-Sum algorithm [12] designed for mono-objective DCOPs. B-MOMS works on a factor graph where each constraint and each variable are nodes. Max-Sum is a dynamic programming algorithm that can find the optimal solution of a DCOP if its factor graph is a tree (no cycle). In the Bounded Max-Sum, a bounding phase is used to remove the least important edges of the graph and obtain a factor graph without any cycle. Based on the edges removed, an approximation ratio is provided to guarantee the quality of the solutions obtained. In B-MOMS, both the bounding phase and the Max-Sum phase are adapted to solve MO-DCOPs, offering a complete algorithm on cycle-less graphs, and providing a posteriori quality guarantee otherwise. B-MOMS was recently used to solve the real problem of managing water resources systems [21]. Compared to this algorithm, DPLS cannot provide guarantee on the quality of its solutions. However, B-MOMS is not anytime and requires to wait the end of the solving before any solution can be used.

MO-ADOPT [10] is the other existing algorithm designed specifically for MO-DCOPs. It extends ADOPT [1], a popular search algorithm for DCOPs, and guarantees to find a Pareto optimal solution. However, it searches for a single solution and not for a set of solution or the whole Pareto front. In DPLS, we propose to find an approximation of the Pareto front, which often involve finding a large set of solutions.

For MO-COP, various approximation algorithms have been developed, e.g., Multi-Objective Mini-Bucket Elimination (MO-MBE) [6], Multi-objective Best-First AND/OR search algorithm (MO-AOBF) [22], and multi-objective A* search algorithm (MOA*) [23]. Most of these algorithms are extension of existing search and inference based mono-objective COP algorithms. In comparison, DPLS is a local search based algorithm, and our experiments revealed that a local search technique is suitable for solving an MO-DCOP.

6. Conclusion

In MO-DCOPs, since finding all Pareto optimal solutions is not realistic, it is important to consider fast approximation algorithms. In this paper, we developed a novel approximation algorithm called Distributed Pareto Local Search (DPLS) algorithm. DPLS uses a local search approach to generate an approximation of the Pareto front of an MO-DCOP. In the experiments, we evaluated the performance of DPLS on randomly generated multi-objective graph-coloring instances and compared the obtained results with B-MOMS, the state-of-the-art approximation algorithm for MO-DCOPs. Our experiments showed that DPLS finds better approximations than B-MOMS in problems with medium or high density.

As future works, we plan on further studying several aspects of the DPLS. First, we want to experiment with the anytime property of DPLS and study ways to improve this property, by using specific exploration strategies for example. This ability to stop the algorithm at any time with the assurance to have the best solutions encountered so far is very important when the available processing time is not known in advance. Another aspect we want to study is how to adapt some neighborhood operators to a distributed environment. Some existing operators are very successful on specific problems and could be interesting to use for the DPLS. However, these operators can involve several variables and using them in distributed settings is not trivial as they can produce overlapping local neighborhoods, requiring addi-

tional coordination between agents. Similarly, exploration and restart strategies should be implemented in distributed settings as they proved to be quite successful for the centralized PLS [24]. Another method that should be studied for DPLS is the Queued PLS [25] that delays the removal of dominated solution from the archive as it can improve the approximation found by the algorithm. Furthermore, we intend to apply DPLS to challenging real world problems, e.g., sensor network and scheduling problems.

References

[1] P.J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo, "ADOPT: Asynchronous distributed constraint optimization with quality guarantees," Artif. Intell., vol.161, no.1-2, pp.149–180, 2005.
[2] A. Petcu and B. Faltings, "A scalable method for multiagent constraint optimization," Proc. 19th International Joint Conference on Artificial Intelligence, pp.266–271, 2005.
[3] R. Maheswaran, M. Tambe, E. Bowring, J. Pearce, and P. Varakantham, "Taking DCOP to the real world: Efficient complete solutions for distributed multi-event scheduling," Proc. 3rd International Conference on Autonomous Agents and Multiagent Systems, pp.310–317, 2004.
[4] V. Lesser, C. Ortiz, and M. Tambe, eds., Distributed Sensor Networks: A Multiagent Perspective, Kluwer Academic Publishers, 2003.
[5] R. Junges and A. Bazzan, "Evaluating the performance of DCOP algorithms in a real world, dynamic problem," Proc. 7th International Conference on Autonomous Agents and Multiagent Systems, pp.599–606, 2008.
[6] E. Rollón and J. Larrosa, "Bucket elimination for multiobjective optimization problems," J. Heuristics., vol.12, no.4-5, pp.307–328, 2006.
[7] E. Rollón and J. Larrosa, "Multi-objective russian doll search," Proc. 22nd AAAI Conference on Artificial Intelligence, pp.249–254, 2007.
[8] R. Marinescu, "Exploiting problem decomposition in multi-objective constraint optimization," Proc. 15th International Conference on Principles and Practice of Constraint Programming, pp.592–607, 2009.
[9] F.D. Fave, R. Stranders, A. Rogers, and N. Jennings, "Bounded decentralised coordination over multiple objectives," Proc. 10th International Conference on Autonomous Agents and Multiagent Systems, pp.371–378, 2011.
[10] T. Matsui, M. Silaghi, K. Hirayama, M. Yokoo, and H. Matsuo, "Distributed search method with bounded cost vectors on multiple objective DCOPs," Proc. 15th International Conference on Principles and Practice of Multi-Agent Systems, pp.137–152, 2012.
[11] T. Okimoto, N. Schwind, M. Clement, and K. Inoue, "Lp-norm based algorithm for multi-objective distributed constraint optimization," Proc. 13th International Conference on Autonomous Agents and Multiagent Systems, pp.1427–1428, 2014.
[12] A. Rogers, A. Farinelli, R. Stranders, and N. Jennings, "Bounded approximate decentralised coordination via the max-sum algorithm," Artif. Intell., vol.175, no.2, pp.730–759, 2011.
[13] L. Paquete, M. Chiarandini, and T. Stützle, "Pareto local optimum sets in the bi-objective traveling salesman problem: An experimental study," Metaheuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems, pp.177–200, Springer Verlag, 2004.
[14] M. Wack, T. Okimoto, M. Clement, and K. Inoue, "Local search based approximate algorithm for multi-objective DCOPs," International Conference on Principles and Practice of Multi-Agent Systems, pp.390–406, Springer, 2014.
[15] T. Schiex, H. Fargier, and G. Verfaillie, "Valued constraint satisfaction problems: Hard and easy problems," Proc. 14th International Joint Conference on Artificial Intelligence, pp.631–639, 1995.
[16] F.T. Leighton, "A graph coloring algorithm for large scheduling problems," J. Res. Nat. Bur. Stand., vol.84, no.6, pp.489–506, 1979.
[17] W. Zhang, G. Wang, Z. Xing, and L. Wittenburg, "Distributed stochastic search and distributed breakout: properties, comparison and applications to constraint optimization problems in sensor networks," Artif. Intell., vol.161, no.1-2, pp.55–87, 2005.
[18] C.M. Fonseca, L. Paquete, and M. López-Ibáñez, "An improved dimension-sweep algorithm for the hypervolume indicator," Proc. 2006 Congress on Evolutionary Computation (CEC 2006), pp.1157–1163, IEEE Press, Piscataway, NJ, July 2006.
[19] K. Hirayama and M. Yokoo, "The distributed breakout algorithms," Artif. Intell., vol.161, no.1-2, pp.89–115, 2005.
[20] E. Angel, E. Bampis, and L. Gourvès, "Approximating the pareto curve with local search for the bicriteria tsp (1, 2) problem," Theor. Comput. Sci., vol.310, no.1-3, pp.135–146, 2004.
[21] F. Amigoni, A. Castelletti, and M. Giuliani, "Modeling the management of water resources systems using multi-objective DCOPs," Proc. 2015 International Conference on Autonomous Agents and Multiagent Systems, pp.821–829, International Foundation for Autonomous Agents and Multiagent Systems, 2015.
[22] R. Marinescu, "Best-first vs. depth-first and/or search for multi-objective constraint optimization," Proc. 22nd IEEE International Conference on Tools with Artificial Intelligence, pp.439–446, 2010.
[23] P. Perny and O. Spanjaard, "Near admissible algorithms for multi-objective search," Proc. 18th European Conference on Artificial Intelligence, pp.490–494, 2008.
[24] M.M. Drugan and D. Thierens, "Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies," J. Heuristics., vol.18, no.5, pp.727–766, 2012.
[25] M. Inja, C. Kooijman, M. de Waard, D.M. Roijers, and S. Whiteson, "Queued pareto local search for multi-objective optimization," Proc. 13th International Conference on Parallel Problem Solving from Nature, pp.589–599, 2014.

Maxime Clement Maxime Clement is a PhD student at The Graduate School for Advanced Studies (SOKENDAI) since 2014. He got his Master degree in Artificial Intelligence from Pierre and Marie Curie University in 2013. His research interests include Artificial Intelligence (AI), Multi-Agent Systems (MAS), and Decision Making.

Tenda Okimoto Tenda Okimoto is an Associate Professor at Kobe university since 2014. Before joining Kobe university, he worked as an assistant professor in Transdisciplinary Research Integration Center (TRIC) / National Institute of Informatics (NII). He got his PhD degree (Informatics) from Kyushu University in 2012. His research interests include Artificial Intelligence (AI), Game Theory, Operations Research (OR), and Disaster Research.

Katsumi Inoue Katsumi Inoue is a Professor at Principles of Informatics Research Division, National Institute of Informatics (NII) since 2004. He is also a Professor of Department of Informatics, School of Multidisciplinary Sciences, SOKENDAI (The Graduate University for Advanced Studies) since 2005, and is a Specially Appointed Professor, Department of Computer Science, School of Computing, Tokyo Institute of Technology since 2015. He received the Doctor of Engineering degree from Kyoto University (1993). His research interests include Artificial Intelligence, Logic Programming, and Computer Science in general.