Technical Note

A Scalable Algorithm to Solve Distributed Constraint Optimization

Maryam Rahmaninia
Department of Computer Engineering
Ghasre-shirin Branch, IAU
Ghasre-shirin, Iran
ma.rahmaninia@gmail.com

Elnaz Bigdeli
Department of Computer Science
IASBS
Zanjan, Iran
e_bigdeli @iasbs.ac.ir

Mohsen Afsharchi
Department of Computer Engineering
University of Zanjan
Zanjan, Iran
afsharchim@znu.ac.ir

Received: November 18, 2012- Accepted: January 26, 2014

Abstract- Recently, Distributed Constraint Optimization Problems (DCOP) have been drawing a growing body of attention as an important research area in multi agent systems as a large body of real problems can be modeled by them. The primary goal of this research is to design a distributed and effective algorithm to solve DCOP. There are various criteria that measure the efficiency of DCOP algorithms, but the most efficient algorithm for DCOP is the one by which the computation and communication cost is as low as possible and the quality of the solution is high. In this paper, we focus on an approximate DCOP algorithm called DALO (Distributed Asynchronous Local Optimization). Using the main idea of the DALO algorithm, we propose a new algorithm to solve DCOP, which exhibits two important improvements over the DALO algorithm. First we use a sequential partial approach to select a coefficient of leaders to compute the best assignment for agents by which the computation and communication cost decrease in the whole DCOP. The second improvement is an evolutionary approach by which the computation and communication burden for each agent decreases. We present some empirical evidences that show our algorithm performs better than the DALO algorithm.

key words- ***distributed constraint optimization, multi agent system***

I INTRODUCTION

In many practical problems such as resource allocation, planning and scheduling wherein coordinating all entities is not possible, the key goal is to develop decentralized coordination techniques. Multi-agent systems are a popular way to model the complex interactions and coordination required to solve distributed problems [1]. A multi-agent system is a network of agents used to perform distributed computation. Networks of cooperative agents are heterogeneous, and not all agents have direct links to one another. Additionally, information is distributed throughout the network either due to privacy concerns or the impracticality of centralizing. In this network, each agent is an autonomous entity with local information and has the ability to perform an action in cooperative situations in which agents collaborate to achieve a common goal Agents need to coordinate their activities to accomplish their collective goals. Distributed Constraint Optimization (DCOP) is a common formalism for representing multi-agent systems in which agents cooperate to optimize a global objective [2], [3].

DCOP grew out of the field of Constraint Optimization Problem (COP) [4], which was itself built upon work from the field of Constraint Satisfaction Problems (CSP) [5]. Distributed Constraint Satisfaction Problems (DisCSP) were first studied by Yokoo and have recently attracted a growing body of interest [6]. In DisCSPs, constraints simply specify that certain combinations of values are legal or illegal. However, in the more expressive COP and DCOP formalisms, constraints specify how preferable different combinations of values are by associating cost functions with the constraints [3].

Distributed Constraint Optimization (DCOP) has been applied to different domains. DCOPs are able to model the task of scheduling meetings in large organizations, where privacy needs make centralized constraint optimization difficult [7]. DCOPs are also able to model the task of allocating sensor nodes to targets in sensor networks, where the limited communication and computation power of individual sensor nodes makes centralized constraint optimization difficult [9]. Finally, DCOPs are able to model the task of coordinating teams of unmanned vehicles in disaster response scenarios, where the need for rapid local responses makes centralized constraint optimization difficult [10].

There are many algorithms that can solve DCOPs. The central focus of DCOP algorithms is to design an algorithm which has low computational and communicational cost and produces solutions with high quality. It is obvious that to reach more accurate solutions for a DCOP, more computations and communications are required. Most algorithms try to tradeoff among the cost of computation and communication and the quality of solutions. The cost of computation and communication is more crucial in large-scale domains. Considering this point, we propose an algorithm that can scale up in large scale problems. The proposed algorithm is an extended version of the DALO algorithm, which works efficiently in large scale domains. DALO algorithm decomposes a DCOP problem to sub-groups and solves DCOP in each sub-group [11]. This algorithm uses either a *t*-distance or *k*-size criterion to form groups. A *t*-distance or *k*-size criterion creates large-sized groups in dense graphs. On the other side in each sub-group, a complete algorithm is used to compute the best assignment. The computational complexity of complete algorithms is exponential in the number of variables, since distributed constraint optimization is known to be NP-hard [9]. Therefore, the computational cost is very high in groups that are created with a *t*-distance or *k*-size criterion. Consequently, it is hard for complete algorithms to scale up, because the computation burden might increase exponentially as the number of variables increase [11]. The problem gets worse because in DALO algorithm all agents form groups and try to compute the best assignment in their groups. We contribute two important algorithmic advances for the DALO algorithm. The first is, instead of using a complete DCOP solver in each group, we use an evolutionary algorithm to find the solution in each group. The evolutionary algorithm decreases the computational cost by a considerable amount in each group. The second is selecting some leaders to compute the new assignment in each group instead of all leaders. The selection of leaders should be done in a way that the quality of solutions gained by their computation is high. We propose a simple and effective method by which the computation decreases and the quality is acceptable.

The structure of the paper is as follows. In section II, formal definitions of DCOP and *t*-distance optimal and *k*-size optimal solutions are presented. In section III, the DALO algorithm and its main issues are described. The new algorithm is introduced in section IV. An analysis of our new algorithm is presented in V. Experimental results of our new algorithm and its comparison with the DALO algorithm are depicted in section VII. Finally, the conclusion and future works are presented in section VIII.

II. RELATED WORKS

There are two main categories for DCOP algorithms: complete and incomplete. Complete algorithms always find a configuration of variables that optimizes the global objective function. In contrast, incomplete algorithms find semi-optimal solutions. ADOPT (Asynchronous Distributed OPTimization) is the first complete algorithm for DCOP that allows asynchronous concurrent execution and is guaranteed to terminate with the global optimal solution [9]. There has been extensive research that attempted to speed up complete algorithms. Ali et al. introduce a framework of different preprocessing techniques that are based on dynamic programming to speed up ADOPT. Their approach can speed up ADOPT by an order of magnitude [12]. Another study done by Yeoh et al. introduces Branch-and-Bound ADOPT (BnB-ADOPT), a memory-bounded asynchronous DCOP algorithm which changes the search strategy of ADOPT from best first search to depth first branch and bound search. This algorithm is up to one order of magnitude faster than ADOPT on a variety of large DCOP problems [13].

The other complete algorithm for distributed constraint optimization is DPOP (Dynamic Programming Optimization). It works based on dynamic programming. It uses a utility propagation method that is inspired by the sum-product algorithm, which is correct only for tree-shaped constraint networks [3]. Some improvements have already been discovered for the DPOP algorithms. For example, MB-DPOP provides a memory-bounded algorithm that trades off the linear message number of DPOP with polynomial message size [14].

Since DCOP is NP-hard [9], as the scale of application domain increases, complete algorithms exhibit an exponentially increasing coordination overhead. Thus, their use in practical applications,

such as those mentioned above, is severely limited. Therefore, many researchers pay more attention to incomplete algorithms that decreases the computation and communication cost.

Comparing with complete algorithms, incomplete algorithms find semi-optimal solutions and do not guarantee the achievement of global optimal solution. In incomplete algorithms, agents form small groups and then try to optimize solutions within these groups. These algorithms decrease the computation and communication costs. Therefore, these algorithms can be applied to large scale systems and are more robust in dynamic environments. It should be noted that in some domains, the best optimum solution is required and complete algorithms should be used to reach the best solution.

Incomplete algorithms can be categorized in two groups. The first group of incomplete algorithms provide guarantee on the quality of the solutions they compute. The second group of incomplete algorithms do not provide any guarantees on the quality of the solutions. Guarantees on the quality is important in some domains.

Incomplete algorithms such as MGM (Maximum Gain Message) [6] and DSA (Distributed Stochastic Algorithm) [15] are the examples of algorithms that do not provide any guarantee on the quality of solutions. These two algorithms use the maximum gain approach to solve DCOP. In MGM, each agent broadcasts a gain message to all its neighbors and an agent is allowed to act if its gain message is larger than all the gain messages it receives from all its neighbors. For DSA, each agent generates a random number from a uniform distribution on [0, 1] and acts if the number is less than a threshold *p*.

Max-sum as an incomplete algorithm is a very promising technique for DCOP because it provides solutions close to optimality while requiring very limited communication overhead and computation. The max-sum algorithm belongs to the Generalized Distributive Law (GDL) framework. While many incomplete algorithms fail to provide any guarantees on the solution quality in general settings, this algorithm provides guarantees on the quality of solution [16].

In the class of incomplete algorithms, there are algorithms called *k*-optimal. *k*-optimal algorithms guarantee solutions that cannot be improved when any group of *k* or fewer agents change their decision. Many incomplete algorithms, such as MGM [17] and DSA [15] described above, yield *1*-optimal solutions. The other version of MGM1 is MGM2 [18], which provides 2-optimal solutions.

The KOPT algorithm is the only incomplete algorithm that works for arbitrary *k* [19] and provides a guarantee on the quality of solutions. There is a detailed investigation in the field of *k*-optimal algorithms. Pearce and Tambe presented the first known guarantees on solution quality for *k*-optimal solutions [20]. They provided reward and structure-independent guarantees on solution quality for any *k*-optimal DCOP assignment. In addition, they presented tighter guarantee for ring and star graphs in their work. Two main properties of *k*-optimal solutions are introduced in [21]. The first one is the worst case guarantee on the quality of the *k*-optimal solution in a DCOP. The second one is the worst-case guarantee on the number of *k*-optimal solutions that can exist in a DCOP. A close view to the *k*-optimal solution set can be found in [21]. Here, by using coding theory, they provide an upper bound for the *k*-optimal solution set. Furthermore, Vinyals et al. recently introduced the C optimality framework that generalizes both *k*-optimality and *t*-optimality by providing quality guarantees for local optima in regions that can be defined by arbitrary criteria [25].

Another incomplete algorithm for DCOP is DALO, which works based on *t*-distance optimality [11]. In this algorithm, each agent forms a group with other agents with a distance of *t* hops [11], [22]. This algorithm provides a guarantee on the quality of the solution based on the t-distance criterion.

Recent works are extending the whole picture of DCOP to asymmetric world. In asymmetric DCOPs different agents may have different valuations for constraints that they are involved in. This new framework bridges the gap between multi-agent problems which tend to have asymmetric structure and the standard symmetric DCOP model. The benefits of ADCOP model over previous attempts to generalize the DCOP model are discussed in [29]. A very comprehensive review of DCOP could be find in [27].

Our proposed algorithm is an efficient incomplete algorithm that provides a guarantee on the quality of solutions and can be applied to large scale domains, because it has low computation and communication cost.

III. BACKGROUND

In this section, we will provide some basic definitions about DCOP and *t*-distance optimality.

A. Distributed Constraint Optimization (DCOP)

A DCOP is defined by a set of variables $V=\{v_1,\ldots,v_n\}$, a set of discrete finite domains for each variable $D=\{D_1,\ldots,D_n\}$, and a set of constraints $C=\{c_1,\ldots,c_q\}$ between the variables. Each variable is controlled by a separate agent that can communicate with other agents.

Assuming each agent controls a single variable, we will use the terms agent and variable interchangeably. The value generally represents an action that an agent should perform as part of larger team. A joint assignment $A=\{a_1,\ldots,a_n\}$ specifies a value for each variable, in which a_i is the value of agent *i*.

Constraints exist between subsets of these variables that determine costs and rewards to the agent team based on the combinations of values chosen by their respective agents. Constraints convey how the values taken on by different variables affect one another. Variables v_i and v_j are considered neighbors if they share a constraint, and only neighbors can directly communicate with one another. In DCOP formalisms, constraints specify how preferable the different combinations of values are by associating cost functions F_{ij} associated with the constraints. This paper only considers binary constraints, which means that each constraint has two variables. A cost function $F_{ij}: D_i \times D_j \rightarrow R$ takes the values of variables v_j and v_j as an input and returns a value as a non-negative number for a constraint.

The utility of agent *i* for assignment *A* is:

$$U_i(A) = \sum_{v_j} F_{ij}\left(a_i, a_j\right)$$

Where $v_i \leftarrow a_i, v_j \leftarrow a_j, a_i, a_j \in A$ (1)

It means that the utility of the i_{th} agent is the sum of the cost functions of all constraints to which an agent belongs.

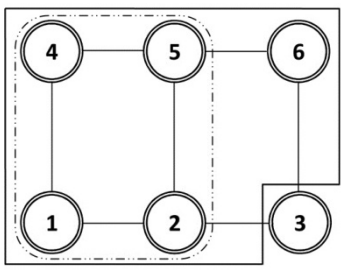
The goal is to choose values for variables such that a given objective function is maximized. The objective function is the sum over a set of cost functions, or valued constraints. Thus, the objective is to maximize:

$$R(A) = \sum_{(v_i, v_j)\in V} F_{ij}\left(a_i, a_j\right)$$

Where $v_i \leftarrow a_i, v_i \leftarrow a_j, a_i, a_j \in A$ (2)

R(A) is the solution quality for the joint assignment *A [9], [23]*.

Constraint reasoning problems are frequently represented diagrammatically, with the variables represented by nodes and the constraints represented as links between nodes in the graph, which is called the constraint graph. The optimal assignment for this DCOP with 6 variables and 7 constraints and an identical cost function is *A = (1,1,1,1,1,1)* (see Figure 1).



Vi	Vj	Fi,j
0	0	0
0	1	1
1	0	0
1	1	2

Fig. 1. The constraint graph of an example DCOP with six binary variables. Each constraint has the same cost function.

B. t-distance Optimality

Definition 1: For two different assignments *A* and A', D(A, A') is:

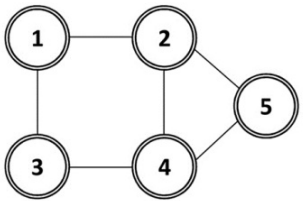
D(A, A') $=\{v_i \in V \mid a_i \neq a_i', v_i \leftarrow a_i \in A, a_i' \in A'\}$ (3)

In other words, D is a deviating group between two assignments *A* and A'.

For example, in Figure 2, for the given assignment (1,1,1,0,0) and (0,0,0,0,0), the deviating group is D((1,1,1,0,0) , (0,0,0,0,0)) = $\{a_1, a_2, a_3\}$. Thus, the distance between the two assignments is the cardinality of the deviating group which is 3.

For a pair of variables v_i and v_j , let $T(v_i , v_j)$ be the shortest distance between them in the constraint graph. Then $\Phi_t(v_i) = \{ v_i | T(v_i , v_j) \leq t, v_i, v_j \in V\}$ denotes a set of variables that can be reached from v_i within *t* hops.

Definition 2: *A* DCOP assignment *A* is *t*-distance optimal if $R(A) \geq R(A')$ for all A', where $D(A, A') \subseteq \Phi_t(v_i)$ for some $v_i \in V$ (see [11], [22]).



Vi	Vj	Fi,j
0	0	1
0	1	0
1	0	0
1	1	2

Fig. 2. An example DCOP with five binary variables. Each constraint has the same cost function.

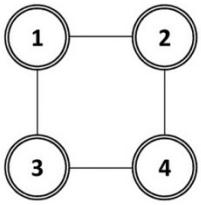
Example: Consider the graph in Figure 2. Given *t = 1*, *1*-distance groups for all variables will be*:* $\Phi_1(v_1) = \{v_1, v_2, v_3\}$, $\Phi_1(v_2) = \{v_1, v_2, v_4, v_5\}$, $\Phi_1(v_3) = \{v_1, v_3, v_4\}$, $\Phi_1(v_4) = \{v_2, v_3, v_4, v_5\}$ and $\Phi_1(v_5) = \{v_2, v_4, v_5\}$. In this example, A = (0, 0, 0, 0, 0) with *R(A) = 6* is 0-distance optimal. Assignment *A* is 0-distance optimal, because if every agent changes its value, then the solution quality will decrease. Assignment *A* is not 1-distance optimal. Moreover, this assignment is not 2-distance optimal, because there is an assignment A'' = (1, 1, 1, 1, 1) with $R(A'') = 12$ for which its utility is more than *A*.

C. k-Optimality

A DCOP assignment A is *k*-optimal if $R(A \geq R(A')$ for all A' for which $|D(A, A')| \leq k$, Where $|D|$ denotes the cardinality of set *D* (see [23]).

Example: Consider the graph in Figure 3 in which $V = \{v_1, v_2, v_3, v_4\}$ are variables with binary domain. In this example, constraints are $c_1 = \{v_1, v_2\}$, $c_2 = \{v_1, v_3\}$, $c_3 = \{v_2, v_4\}$, $c_4 = \{v_3, v_4\}$. $A_1 = \{0, 0, 0, 0\}$ is a *1*-optimal assignment with $R(A_1) = 4$. This assignment is 1-optimal because no unitary change in assignment A_1 improves the reward. But it is not a *2*-optimal

assignment, because if one pair of agents changes their values simultaneously to 1, the solution quality increases. Assignment $A_2 = \{1, 1, 1, 1\}$ with $R(A_2) = 8$ is a *4*-optimal assignment and is also the best assignment.



Vi	Vj	Fi,j
0	0	1
0	1	0
1	0	0
1	1	2

Fig. 3. An example DCOP with four binary variables. Each constraint has the same cost function.

IV. DALO ALGORITHM AND ISSUES

DALO (Distributed Asynchronous Local Optimization) is an algorithm which can compute either *k*-size or *t*-distance optimal solutions for any value of *k* or *t* [22].

The DALO algorithm has three phases. In phase one, agents send initialization messages to nearby agents, to find all of the *k* or *t* groups in the constraint graph and assign each group a unique leader. In phase two, based on the information gathered in the previous phase, all leaders compute a new optimal assignment using a centralized variable elimination algorithm in parallel. In phase three, if the new assignment improves the solution quality, the group leader attempts to set the new assignment. When all the leaders try to set their assignments, there will be conflicts among overlapping groups, which are resolved by an asynchronous locking and commitment protocol.

DALO Issues

Although DALO is an effective algorithm for solving DCOP problems, it suffers from some drawbacks.

Using *k* or *t* as a criterion to create groups may produce groups with a large number of nodes. As it is explained for the DALO algorithm in phase two, a complete algorithm is used to solve DCOP. All group leaders compute new optimal assignments for their groups in parallel. A leader node uses a centralized variable elimination algorithm to solve the sub problem for the local group. The computational complexity of a complete DCOP solver is exponential in its number of variables. By increasing *k* and *t*, the number of agents in a group increases, therefore, using a complete DCOP solver will not be tolerated from a time and space point of view.

The second problem is that all leaders start to calculate the best possible assignment for their group members, but at the end of each round, to avoid conflicts among leader assignments, some of these leaders are chosen and their assignments are set. It means that each agent belongs to different groups, and each agent should finally choose one of the leaders' assignments. Based on the DALO algorithm, if all agents commit to the new assignment, then the leader set the assignment. As it is clear, finding the best assignment in each group needs a large number of messages to be sent and received, which increases the complexity of the algorithm. It is more efficient not to compute new assignments for leaders that do not have set assignments, as this will reduce the computational load.

In this paper, we try to solve these problems. To solve the first problem, we use a genetic approach instead of using a centralized variable elimination algorithm. This approach decreases the computation time substantially. To overcome the second problem, we utilize a partial approach in which only a group of leaders are selected to compute the best assignment instead of all of them. The partial method is an effective method to reduce the communication cost while keeping the solution quality almost the same. Using the partial approach, the number of messages passed among agents decreases by dismissing the leaders whose assignments will be ignored at the end of each round. Consequently, there will be a substantial decrease in computational and communication time in each round.

V. EP-DALO ALGORITHM

In this section, we present our new algorithm called EPDALO (Evolutionary Partial DALO). The EP-DALO algorithm is a scalable algorithm to solve distributed constraint optimization problems.

This algorithm has three phases.

- Initialization: In phase one, each agent sends information to the agents of its group that is constructed with the given t or k.
- Computing the Best Assignment: Computing the new assignment has two steps:
  - Leader Selection: Some leaders are selected to compute a new assignment for their groups based on the sequential partial approach.
  - Computing the best assignment using the genetic algorithm: Using the genetic approach, every agent calculates the best assignment to its group members.
- Set the assignment: In phase three, the conflicts among overlapping groups are resolved, and the new assignment is set.

Phase one is executed once whereas the other phases are repeated until the new assignments show no further improvement.

A. Initialization

In this phase, the leader, which is given t or k starts to construct its group. At first, every agent sends a message containing all its constraints to all agents in its group. Then, it chooses an initial value from its domain and broadcasts it to group members. Consider the constraint graph in Figure 1: the groups of agent 4, which are constructed using $t = 2$ and $k = 2$, are shown by solid and dash lines, consecutively.

B. Computing the Best Assignment

In this phase, the leader of each group calculates the best possible assignment by which the quality is maximized. There are two issues that should be mentioned in this phase.

The first issue is about leaders that have permission to compute new assignments in each round. Based on our discussion in section III, in phase 2 of the DALO algorithm, all leaders calculate a new assignment for their group members whereas some of these assignments will be ignored to avoid conflicts among leader assignments. Hence, it is better to have some leaders not compute new assignments. To this end, the permission to calculate a new assignment is granted to some leaders in each round. The leaders with permission to compute the new assignments are called active leaders. We use the partial sequential approach to select leaders to be activated.

The second issue is about the computation that each leader performs to find the best assignment. A leader node uses a centralized variable elimination algorithm to solve the sub problem for the local group with exponential computational cost. A method with lower computational cost should be used to reduce the computational load. A genetic algorithm is used to find the best solution in each sub-graph.

Therefore, for computing the best assignment at first by using the partial sequential approach, some leaders are selected, and then the selected leaders calculate the best assignment through the genetic approach.

1) Selecting Leaders by Partial Sequential Approach: As it is described in the DALO algorithm, there are *n* different groups for a graph with n agents. Consequently, here are n leaders. We define the index set $L = \{1, 2,\ldots, n\}$. For a given h, the index set L is divided into *h* subsets $S = \{S_1, S_2, \ldots, S_h\}$. Each subset includes the agent's ID which should be activated. In each round, one of the S_i is selected according to a pre-defined approach and the leaders in S_i run the genetic algorithm in a synchronous manner. The main problem is assigning agents to subsets $\{ S_1,\ldots , S_h \}$. The best approach to this end is the one that assigns leaders to subsets that by activation of their leaders solution quality increases in each round. This approach is the most desirable one but in return increases the complexity of algorithm. Leaders' selection can be performed by a sequential approach. This simple method decreases the communication time substantially.

In the sequential approach, S_1 is selected in the first round, S_2 in the second round and S_h in the h_{th} round, and in round $h + 1$, S_i is selected again. Consequently, after h rounds, all subsets are selected exactly once. In this approach, at a given round *r*, one of the subsets $S_1, S_2, \ldots , S_h$ is chosen in a sequential fashion. After *h* rounds each subset has been selected only once.

In this method, we assume that in each round, each leader in a network synchronously checks if:

$$l \,\%\, h = r \,\%\, h \qquad (4)$$

Where *l* is the leader's ID and r is the number of round. Then, this leader computes a new assignment.

As an example, consider the graph in Figure 1. For $h=3$, set S will be:

$$S = \{\{Ag_1, Ag_4\}, \{fAg_2, Ag_5\}, \{Ag_3, Ag_6\}\}$$

Based on the above mentioned approach, in round 1, $\{Ag_1, Ag_4\}$ are selected, in round 2, $\{Ag_2, Ag_5\}$ are selected, and in round 3, $\{ Ag_3, Ag_6\}$ are selected, and the selection process continues in the same fashion. It is obvious that in the end of the h_{th} round, all subsets are selected exactly once. Obviously, different methods can be used in the sequential approach.

2) Calculating the Best Assignment : The Genetic

Genetic Algorithms (GA) are adaptive methods and have *Approach :*In the second phase of our algorithm, each leader starts to find the best assignment for its group. As previously mentioned, a leader node uses a centralized variable elimination algorithm to solve the sub-problem for the local group. The computational time of a complete DCOP solver is exponential. A DCOP is an optimization problem, and nowadays, evolutionary algorithms, particularly Genetic Algorithms (GAs), are considered one of the best known algorithms for solving optimization problems. Instead of using a centralized variable elimination algorithm, we use genetic algorithms to find the best assignment centrally

Genetic Algorithms (GA) are adaptive methods and have been applied to optimization problems in many fields [24]. For the genetic approach, a leader starts out with an initial population of possible solutions called individuals. Each individual is represented as a chromosome using a form of encoding. These chromosomes are evaluated for their fitness. The fittest solutions are those that are more appropriate for the problem. Based on their fitness, certain chromosomes in the population are selected for reproduction. These selected individuals are parents that are manipulated by crossover and mutation to create offspring. The repeated application of these genetic operators on the fittest chromosomes results in an increase in the average fitness of the population over time, and thus result in the identification of improved solutions for the problem under investigation.

Before we apply a genetic algorithm to a distributed constraint optimization problem, we will show how chromosomes are represented and how the fitness of chromosomes are evaluated.

In this paper, each assignment for a group is considered as a chromosome. For group 1 in Figure 2, A= (0, 1, 0) is a possible assignment and also a chromosome in the genetic approach in which a_i indicates the value for the agent *i*. As we described in section II, the value of each agent is chosen from a binary domain. Therefore, each chromosome is a string of binary values indicating an assignment for a group. The fitness of a chromosome is evaluated by a fitness function. The fitness function that is used in this paper to evaluate the fitness of a chromosome is the solution quality, which is defined in section II. Solutions with high quality are considered the fittest ones. Consider the graph in Figure 2: between the two assignments A = (0, 0, 0, 0, 0) with R(A) = 6 and A' =

(1, 1, 1, 1, 1) with R(A') = 12, A' is the fittest solution.

Finding the best assignment using the genetic approach has four phases:

Initialization: In this phase, an initial population is generated. This population is a set of chromosomes. A chromosome is a possible assignment for each group. The chromosomes are equal in size with the group and are a string of binary values. Element *i* in the string indicates the value for $Agent_i$. Consider, each agent chooses a value from a binary domain. Therefore, for a group with *m* agents, there will be 2^m assignments. Generally, the population is generated randomly, and is selected from all possible solutions for a problem. The size of the population is different for each group and depends on the size of the group. It is also considered constant during the execution of the algorithm.

- **Selection**: To generate the next population, a set of solutions is selected from the existing population. Generally, the main criterion for selection is fitness, and individuals that are high fit will be selected to generate the next generation. As mentioned, the fitness of each individual is evaluated by the fitness function, which is the solution quality in this algorithm. According to this criterion, a proportion of the current population with high solution quality will be selected to generate the next population. However, this paper uses the stochastic method. This method presents opportunities for less fit solutions. Two individuals are chosen to act as parents. One of the parents is chosen amongst the best solutions in the population, while the other is randomly chosen from the whole current population.

- **Reproduction**: In this phase, the next population is generated. Two chromosomes are selected to generate new chromosomes. Crossover and mutation are two basic operators of GA that can generate new chromosomes for the next population. The performance of GA depends on these operators. There are many different types of crossover and mutation. Type and implementation of operators depend on the encoding and also on the problem.
  - **Crossover**: We use the single point crossover in our implementation. In single point crossover, one crossover point is selected, the binary string from the beginning of a chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent [26]. Consider the graph in Figure 2. For the group of $Agent_1$, two assignments $A_1 = (0, 1, 0)$, $A_2 = (0, 0, 1)$ are the two parent chromosomes. Using a single point crossover and considering $i = 2$ as a crossover point, the offspring will be $A_3 = (0, 1, 1)$.
  - **Mutation**: A mutation operator simply inverts the value of the chosen chromosome. The mutation operator is used as a slight modification to the offspring. We set mutation probability into 0.007 to mutate values for an individual.

- **Chromosome Evaluation**: In this phase, newly generated offspring are added to the population and then the worst individuals are removed from the population. Depending on whether they are better than the worst individuals in the population, the new offspring may or may not survive to join the new population. Mainly, some solutions with lower utility are selected since using these solutions to generate offspring sometimes leads to high fit chromosomes.

  Population generation is repeated until the algorithm converges to the optimal solution.

A. Set the assignment

Having found the best assignment for a group, the leader attempts to implement the new assignment by sending out requests. This can cause conflicts among overlapping groups, which is resolved by an asynchronous locking and commitment protocol in our approach [11].

VI. PARAMETER ANALYSIS

In the partial sequential approach, based on the description in IV-B1, a subset that consists of leaders' ID is selected in each round. The number of subsets h is an important parameter in this algorithm and influences its computational complexity, solution quality and number of rounds.

In set $S = \{S_1, \ldots, S_h\}$, increasing *h* means increasing the cardinality of S. In a graph with n agents, there are $\left\lceil \frac{n}{h} \right\rceil$ leaders in each subset. Hence, by increasing h, the number of leaders in each subset will be decreased. As it is described, in each round, a subset is selected and if the number of leaders in a subset decreases, the number of leaders that are activated in each round would be few. Therefore, the number of leaders that compute new assignments are few and as a result, the communicational cost decreases. On the other hand, increasing h decreases the solution quality. Increasing *h* decreases both the number of activated leaders in each round. Hence, the number of new assignments that are computed will be decreased, and to reach the specified quality, more number of rounds will be needed.

According to the above description, parameter *h* should be chosen in a way that with few numbers of messages and rounds, the algorithm converges to a solution with high quality. If we consider m as the maximum number of messages passed in each group, the total number of messages that are passed in each round of the DALO algorithm for the whole graph would be m × n. Consequently, the total number of messages after *r* rounds would be:

$$M_{DALO} = n \times m \times r \quad (5)$$

In the EP-DALO algorithm, this amount would be:

$$M_{\text{EP-DALO}} = \frac{n}{h} \times m \times r' \quad (6)$$

Where M is the total number of messages.

We introduce parameter β to compare communicational cost of DALO and EP-DALO with regard to parameter *h*.

$$\beta = \frac{M_{EP-DALO}}{M_{DALO}} \quad (7)$$

Based on our experiment, depending on the structure of the graph, the number of rounds in DALO and EP-DALO are not much different. So, we ignore *r* and r' with a good approximation and consider $\frac{r}{r'} \approx 1$, by sequential approach, the number of messages passed among agents in each round decreases by $\frac{1}{h}$.

An example is given to show how the best value for *h* can be chosen. For a graph with *n = 22*, different values for parameter *h* are chosen to reach a solution with the highest possible quality and a low computational complexity. Figure 4 depicts β vs. *h*. It is verified that *h* should be balanced according to the number of messages, the solution quality and the number of rounds. As it is clear, up to *h=10* in the new algorithm the communicational complexity is decreased, whereas the solution quality is almost the same as the DALO algorithm. For *h > 10*, number of rounds should be increased to reach the quality of the DALO solution.

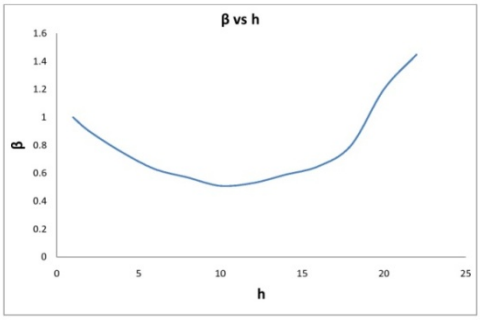


Fig. 4. The best possible value for h is the minimum of the diagram.

VII. EVALUATION METRICS

Different metrics are used to evaluate DCOP algorithms. To evaluate the proposed algorithm, we use four different metrics. The first three metrics are introduced in [22] and [28] and the forth one is used for evaluation in [19] and [23].

These metrics are:

- **Number of Rounds (NR):** The dominant metric for the evaluation of DCOP algorithms is the number of synchronous rounds [30]. A round is defined as one unit of an algorithm progress in which all agents, in parallel, process their incoming messages, perform any required computation, and send their outgoing messages.
- **Communication Load (CL):** In each round, every agent communicates with others due to two reasons. First, to compute the best assignment, each agent needs the constraint of others, therefore, agents send messages containing constraints to other agents. Sending constraints is done only one time in the beginning of the algorithm. Since this kind of communication is not repeated during algorithm execution, we do not consider the time required for this communication in computing the computational load. The second reason is that, at the beginning of each round, each agent sends its value to the members of the groups to which it belongs. Therefore, to solve DCOP, each agent needs to send and receive lots of messages. Algorithms with a low number of messages are considered more applicable.

In our experiments we use two metrics, communication load and communication burden. The total number of messages passed in each round in the whole DCOP is the communication load in the DCOP algorithm and the communication burden for each agent is the number of messages sent by each agent after *R* rounds. e the computational cost in a round, we

- **Computational Cost (CC):** In order to measure use concurrent constraint checks. A constraint check is the act of evaluating a constraint in the problem by comparing the value of one variable to another variable in the problem. Let $cc(v_i, r)$ be the number of constraint checks performed by agent v_i in round *r*. Then, the computation time of round r is defined as:

$$\textit{Computation time in round } r = \max_{v_i \in V} cc(v_i, r) \times t \quad (8)$$

Where t is the time required for one constraint check. The maximum overall agents is used because the agents are conceptually executing in parallel. The length of a round is determined by how long the longest running takes to complete.

The other measure that we use is the total number of constraint checks performed by agent v_i in R rounds.

$$cc(vi) = \sum_{r=1}^{r=R} cc(vi, r) \times t \quad (9)$$

Indeed, this metric shows the computational burden that each agent has to solve DCOP.

- **Solution Quality (SQ):** The other metric that we use in this paper to compare the algorithms is the quality of solution, as introduced in section II.
- **(Gain,#Locked Variables):** The tuple (Gain,#Locked Variables) analyzes the performance of the algorithm on local groups. The gain is the quality of group and the #Locked Variables is the number of variables that are locked to set the new assignment.

All these metrics are utilized to compare DCOP algorithms. The main contribution of this paper is to reduce the algorithms' communication load and computational cost. Since the computation and communication cost are not same in the compared algorithms, the number of rounds is considered identical for both algorithms to compare the algorithms in the same condition. The fourth metric is considered to prove by decreasing CL and CC, the quality of solution is not decreased.

VIII. EXPERIMENTAL RESULT

We considered two different domains for our experiments. The first is the standard graph-coloring

scenario and the second is the domain of the meeting scheduling problem.

In distributed meeting scheduling problem, groups of researchers work in different locations and therefore need to meet to cooperate on a joint project. These researchers have various preferences over the time and place of the meeting. Considering researchers are in different locations, there are costs associated with traveling to meetings. The problem is to maximize the global sum of each researcher's satisfaction with the schedule, while ensuring that the travel costs accrued by researchers do not exceed the local travel budget for their research group.

TABLE I. COMMUNICATIONAL BURDEN FOR EACH AGENT FOR DIFFERENT k,t.

Graph (n;D; t=k)	DALO algorithm	EP-DALO algorithm
(10,0.2,k=2)	88.2	49.3
(10,0.4,k=1)	133	72.2
(10,0.6,k=2)	113.4	58.8
(50,0.2,k=3)	491.62	162.62
(50,0.4,k=5)	1275	427
(50,0.6,k=5)	1984	636
(100,0.2,t=6)	1261	300
(100,0.4,t=4)	423	106
(100,0.6,t=6)	847	186
Average iscrepancy	**(0.69)**	

A graph coloring problem, consists of a graph and a set of colors, and its goal is to assign each vertex a color such that the number of adjacent vertices with the same color is minimized. As a DCOP, there is one agent per vertex that is assigned to decide the associated color.

We performed experiments on different graphs with different structures. Structures of graphs are chosen arbitrary and no special structure is used to prove the efficiency of the proposed algorithm for any structure. Our experiments use different graphs with sizes of $n = 10, 50, 100$. The density of graphs is considered $D = \{0.2, 0.4, 0.6\}$ and parameters k and t for creating groups are considered $t = 1, 2, 3, 4, 5, 6$ and $k = 1, 2, 3, 5, 10, 15$. Each arbitrary graph used in the experiment is shown by a tuple $(n,D, t/k)$. Therefore, the tuple $(50, 0:4, 2)$ is an arbitrary graph with a size of 30 and a density of 0:4 for which groups are created using $t = 2$ or $k = 2$. For each tuple $(n ,D, t/k)$, we create 15 different structures with size of n, density of D, and parameter t/k. The results that are shown for each tuple are the average of the results for 15 different structures. We use the same initial assignment for both algorithms. The stopping criterion used to terminate the running of the algorithms is the number of rounds.

As stated before, the main goal of this paper is to show sthat our algorithm decreases both Computational Cost and Communication Load. To make a fair comparison, we should prove that the other parameters do not change under our new approach. The main concern is the solution's quality. It needs to be proved that by decreasing computation and communication, the quality of the solution will not change too much. It will be shown in the following section that the solution quality dose not change very much in EP-DALO in comparison with DALO. As the experimental results show, EP-DALO decreases the computational cost and communication load and it can be concluded that our proposed algorithm is more efficient than the DALO algorithm.

The results shown in the following are mainly the results of the graph coloring problem. The results of the meeting scheduling problem are depicted in Tables III and IV.

A. *Efficiency*

We present the empirical results from our experiments which used two different incomplete algorithms for DCOP, DALO and EP-DALO. We illustrate that EP-DALO outperforms the DALO algorithm in which groups are created based on parameters t and k. In addition, by comparing EPDALO with DALO, we show that the speed-up

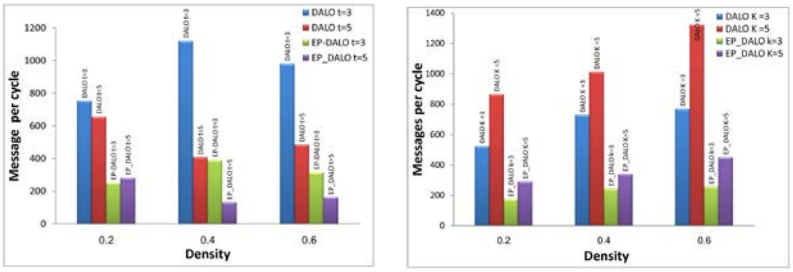


Fig. 5. Average number of messages passed per cycle required to find the optimal solution for graphs with size 50 and different k and t.

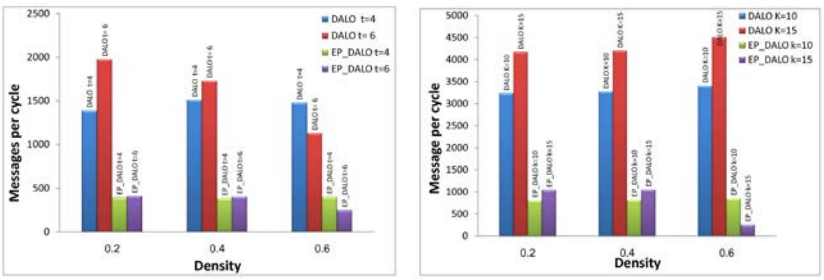


Fig. 6. Average number of messages passed per cycle required to find the optimal solution for graphs with size 100 and different k and *t*.

comes from two sources: a) selecting some leaders to compute the best assignments instead of all them and b) using an evolutionary method that computes the new assignment in a shorter time.

1) Communicational Load (CL): Figures 5 and 6 show the average total number of messages sent by all agents per cycle of execution with DALO and EP-DALO on the graph coloring problem. As the number of agents is increased so is the number of messages sent per cycle. Diagrams show the results for constraint graphs with size *n = 50, 100* and various *t* and *ks*.

Let's consider graphs with *size 50*, density 0.4 and k = 5: the number of messages is *436* for EP-DALO, and DALO have *1372* messages, which show that EP-DALO decreases the number of messages sent per cycle by *68%* (See Figure 7). In addition, EP-DALO algorithm is more efficient for larger t and k in all above mentioned graph sizes. The justification is straightforward. Larger *t* and *k* means each group has a larger number of agents which leads to a larger number of constraints in each group and more communication to find the best assignment. For example, in Figure 6, for graphs with tuple *(100,0.6,t=6)*, the number of messages is *239* by the EP-DALO algorithm and is 1239 by the DALO algorithm. The analysis is the same for various densities in all graphs. By increasing density both the number of constraints in each group and the communicational load increase.

Figure 6 shows how DALO and EP-DALO scale up with an increasing number of agents.

TABLE II. COMPUTATIONAL BURDEN FOR EACH AGENT FOR DIFFERENT t,k.

Graph (n;D; t=k) EP-	DALO algorithm	DALO algorithm
(10,0.2,k=2)	7.56	2.88
(10,0.4,k=1)	9.12	4.37
(10,0.6,k=1)	11.4	4.18
(10,0.6,k=2)	6.44	2.05
(50,0.2,k=3)	4.60	0.72
(50,0.4,k=3)	5.71	1.08
(50,0.6,k=3)	4.95	0.93
(50,0.6,k=5)	8.92	1.29
(100,0.2,t=4)	10.101	2.002
(100,0.2,t=6)	22.11	3.54
(100,0.6,t=4)	34.125	1.18
(100,0.6,t=6)	34.05	4.46
Average Discrepancy	**(0.81)**	

The results in Figures 5-6 show that EP-DALO significantly outperforms both DAL-t and DALO-k on problems with densities 0.4 and 0.6. The decrease by EP-DALO over DALO is 72% on average.

The result for communicational burden is shown in Table I. The computational burden for each agent decreases by *69%* in the EP-DALO algorithm.

2) *Computational Cost (CC):* The EP-DALO algorithm decreases both measures, which are the required time for computation in each round and the time of computation during algorithm execution for each agent. These parameters are measured by elapsed CPU time in each round and after *r* rounds.

The results for graphs with *n = 50, 100* are depicted in Figures 7 and 8. The results in these figures depict the average computational load for R rounds. For example, tuple (100, 0.4, k = 10) has an elapsed CPU time of 7.2 seconds for DALO whereas this parameter for the EP-DALO algorithm is only 4:8 seconds. As the results show, EP-DALO is more efficient by increasing k and t and in dense graphs. The reason is the same as the above mentioned reason in communicational load. The larger number of agents in groups means the larger communication load and computation cost.

As the size of active agents in a group increases, the elapsed CPU time of the DALO algorithm increases exponentially, but with the EP-DALO algorithm, the time increases polynomially. The difference between the result of running two algorithms for *t = 3* and *t = 5* clarifies the efficiency of the algorithm. Consider the tuple (50, 0.2, t = 3, 5) in which the size of the graphs is 50, the density is 0.2, and the distance of 3, 5 is used to create groups. In the graphs with t = 5, the size of groups is more than graphs with t = 3, so computing all possible assignments and finding the best assignment will be complicated. The EP-DALO algorithm will decrease the computational cost by 53% for t = 3 but by 67% for t =5.

The other important point which is clear in the results is that in dense graphs, the proposed algorithm is more efficient. As it is shown in Figure 8, for graphs with size *n = 100* and *t = 6* and density *D = 0.6*, computational cost is decreased by *81%* and for the same graphs with density *D = 0.2*, it is decreased by *64%*.

The computation burden for each agent is an important factor, which is decreased by a considerable amount by our proposed algorithm.

Table II depicts the computation time of each agent after *r* rounds. The result for each tuple is the average of computation time of agents in graphs with size *n*. In the graphs with the tuple (100, 0.2, t = 4), the time spent by an agent for computation after 91 rounds is 10.101 in the DALO algorithm and 2.002 in the EP-DALO algorithm. It is obvious that using EP-DALO decreases the computational burden in considerable amount in each round and after R rounds. According to the results, the computational burden for each agent decreases by 81% in the EP-DALO algorithm in comparison with the DALO algorithm.

TABLE III. RESULTS FOR COMPUTATION TIME FOR DIFFERENT TUPLES USING DALO AND EP-DALO FOR MEETING SCHEDULING DOMAIN

Graph (n;D; t=k)	DALO algorithm EP-	DALO algorithm
(10,0.2,k=1)	10.32	5.45
(10,0.4,k=2)	18.65	9.78
(10,0.6,t=1)	65.97	26.97
(10,0.4,t=2)	163.56	49.97
(50,0.2,k=3)	21.043	13.76
(50,0.4,k=5)	61.076	26.45
(50,0.6,t=3)	418.78	198.97
(50,0.4,t=5)	398.78	165.57
(100,0.2,k=10)	158.98	59.57
(100,0.4,k=15)	204.65	69.897
(100,0.6,t=4)	403.67	140.98
(100,0.6,t=6)	794.98	328.67
Average Discrepancy	(0.59)	

TABLE IV. COMPUTATIONAL BURDEN FOR EACH AGENT FOR DIFFERENT t,k IN MEETING SCHEDULING DOMAIN

Graph $(n, \mathcal{D}, t/k)$	DALO algorithm	EP-DALO algorithm
(10,0.2,k=1)	288.67	113.05
(10,0.4,k=2)	487.45	235.87
(10,0.4,t=1)	507.78	176.8
(10,0.6,t=2)	2346.67	1276.002
(50,0.2,k=3)	674.67	287.3
(50,0.4,k=5)	407.24	148.97
(50,0.4,t=3)	3813.60	982.66
(50,0.6,t=5)	9879.89	5468.01
(100,0.4,k=10)	4683.89	2167.97
(100,0.2,k=15)	8945.91	3295.764
(100,0.4,t=4)	11847.2350	7468.78
(100,0.6,t=6)	28678.34	11385
Average Discrepancy	(0.78)	

B. *Solution Quality (SQ)*

To compare the algorithms, the quality of the solution for DALO and EP-DALO are evaluated. Both algorithms start from a same random initial assignment. The stopping criterion is also defined in a same for both algorithms. The algorithm stops running whenever all groups do not tend to change their assignments because there is no new assignment to increase the utility of groups. We set parameter *t* to *2*, *h* to *3*, and *m* to *5* respectively. Determination of the exact values of h and m is made just by experiment and we set the parameters to the values which have the more desirable results. More discussion related to determination of h and m can be found in [8].

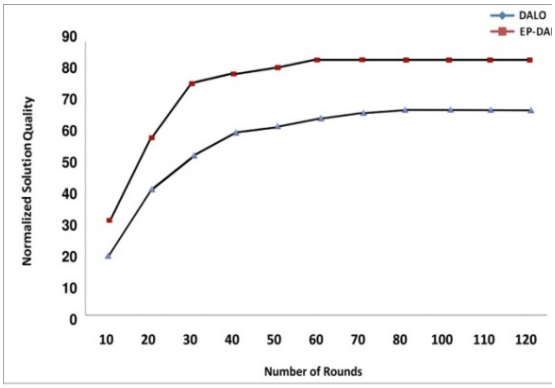


Fig. 9. Solution quality: EP-DALO vs DALO for graphs with density 0.2.

In our first experiment we compare the solution quality of our EP-DALO algorithm and DALO. Obviously, algorithm that achieves a final solution of higher quality in a lower number of rounds is more desirable. Figures 9 and 10 show that the solution quality increases by algorithm in comparison with DALO. For instance in Figure 9 the final solution quality for graphs with *D = 0.2* using DALO is *65*, but using EPDALO the quality in the same graphs is 82. Moreover, after group alteration through EP-DALO, there will be a boost in the solution quality; these increases end in the algorithm convergence to a higher solution quality in lower number of rounds in comparison with DALO.

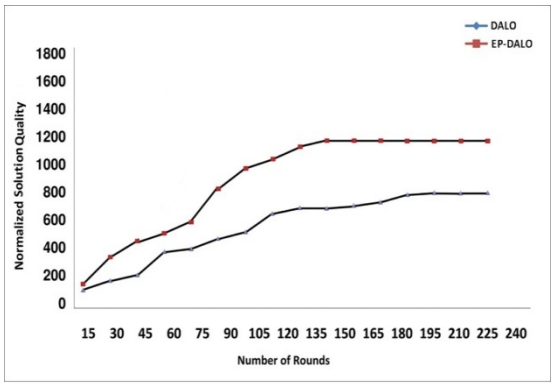


Fig. 10. Solution quality: EP-DALO vs DALO for graphs with density 0.6.

As another example, consider the diagrams in Figure 10, EP-DALO converges after 150 rounds and DALO converges after 195 rounds. The results also show that the EP-DALO algorithm is even more efficient on dense graphs. It is clear that the groups in dense graphs have more number of agents in comparison with sparse graphs. Accordingly the overlap among groups increases and there will be more number of agents which are common among groups. In this case, there will be more number of agents which do not allow a leader to set its assignment by committing to other groups. Overlay, the results in our experiment show that the quality of solution increases *43%* and the number of round decreases *21%*, on average. It can be concluded that using EPDALO algorithm, solutions with higher quality are achieved in a lower number of rounds.

To further understand and compare the performance of EP-DALO and DALO, we provide an analysis on local group changes. In each group, the leader locks some of the variables and if all group members commit to the new assignment, it will be set. By setting the new assignment, the utility of group, which we call it gain, will change. The (Gain,#Locked Variables) pair is used as a metric to compare DCOP algorithms in [11]. It is a proper metric to compare the effect of different group formations in solving DCOPs. The more the number of locks, the more the number of conflicts. Hence, groups with lower number of locks and larger gain are more preferred.

To show the performance of the algorithms, we compare the result for graphs with size 50 and density 0.4. As it is declared in Figure 11 DALO never

achieves a gain larger than 500 and barely locks more than 20 variables. On the other hand, EP-DALO achieves gain 800 by locking more number of variables. For example as it is specified in the Figures 11 and 12 by locking 20 variables DALO achieves gain 500 while EP-DALO can achieve gain 850. In Figure 11, the congestion is on the value 18 which indicates that most of groups locked 18 variables. On the contrary, as it is circled in Figure 12, the congestion is on the value 23. The difference in the number of locked variables is not very much, but the quality improvement is considerable. Hence, by slight increase in the number of variables better solutions are achieved. Our experimentations show hat EP-DALO outperforms DALO both in term of solution quality and the number of rounds that this qualities achieved.

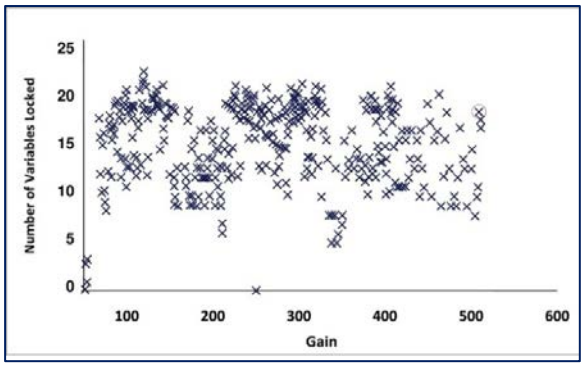


Fig. 11. (Gain,#Locked Variables) for DALO.

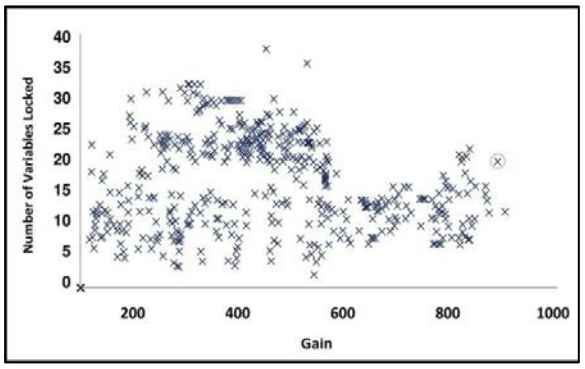


Fig. 12. (Gain,#Locked Variables) for EP-DALO.

VIIII. Conclusion and Future Work

In this paper, we presented a new method for solving DCOPs called EP-DALO. In this algorithm, we use a sequential partial approach to select leaders to compute the new assignments that both prevents the activation of all leaders and decreases the communication cost. Moreover, to compute the new assignments, we use an evolutionary algorithm that decreases the computation in each group. The key features of this algorithm are that the computation and communication cost are very low in whole DCOP and the computation and communication burden of each agent decreases by a considerable amount. Our experiment show that using our new algorithm does not degrade the quality of the solution by decreasing the computation and communication cost. To be more accurate, our algorithm is more accurate on dense graphs. Given these features of our new algorithm, we believe it is a proper candidate for solving DCOP in large scale and real time domains, since the computation and communication cost is the main concern in these domains.

To design an efficient algorithm, all parameters – computation and communication cost and the quality of the solution should be considered altogether. Computing low quality solutions very quickly or computing high quality solutions very slowly is not acceptable in many domains. Therefore, there should be a balance between the time and the quality. The algorithm that tries to create the balance among these parameters is efficient and can be applied to different domains. Our algorithm creates the balance among the above mentioned parameters and creates the solution with acceptable quality at lower costs.

We are planning to extend the algorithm by using dynamic group formation by which the quality of the solution can be increased and the complexity of the algorithm can be decreased.

References

[1] H. J. Levesque, P. R. Cohen, and J. Nunes. On acting together. In AAAI, (1990).

[2] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In 3rd International Joint Conference on Autonomous Agents and Multiagent Systems, pages (438-445), July (2004).

[3] A. Petcu and B. Faltings. A Scalable Method for Multiagent Constraint Optimization. In 9th International Joint Conference on Artificial Intelligence, pages (266-271), Aug. 2005.

[4] M. Silaghi and D. Mitra. Distributed constraint satisfaction and optimization with privacy enforcement. In IAT, 2004.

[5] R.R. Bakker, F. Dikker, F. Tempelman, and P.M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. In IJCAI, 1993.

[6] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. Distributed constraint satisfaction for formalizing distributed problem solving. In International Conference on Distributed Computing Systems, pages 614- 621, 1992.

[7] R. T. Maheswaran, , E. Bowring, J. P. Pearce, P. Varakantham, M. Tambe, Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling. Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS 2004). New York, NY, pp. 310 □ 317.

[8] E.Bigdeli, M.Rahmaninia, and M.Afsharchi. Pkopt: Faster k-optimalsolution for dcop by improving group selection strategy. In the proceeding 22th international conference on tools with Artificial Intelligence (ICTAI). October, July 2010.

[9] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence, 16(12) : 149 □ 180, 2005.

[10] A. Chapman, R. A. Micillo, R. Kota, and N. Jennings. Decentralised dynamic task allocation: A practical game-theoretic approach. In The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS-09), pages 915 □ 922; 2009.

[11] Z. Yin, C. Kiekintveld, A. Kumar, M. Tambe, Local Optimal Solutions for DCOP: New Criteria, Bound, and Algorithm.Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009).

[12] S. Ali, S. Koenig, M. Tambe, Preprocessing techniques for accelerating the DCOP algorithm ADOPT. In AAMAS, 2005.

[13] W. Yeoh, A. Felner and S. Koenig. BnB-ADOPT: An Asynchronous Branch-and-Bound DCOP Algorithm.In Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 591-598, 2008.
[14] A. Petcu, B. Faltings, Mb-dpop: A new memory-bounded algorithm for distributed optimization. In: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI). (2007) 1452-1457.
[15] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz Jr., and M. Tambe, editors, Distributed Sensor Networks: A Multiagent Perspective, pages 257□295. Kluwer Academic Publishers, 2003.
[16] A. Farinelli, A. Rogers, and N. Jennings. Bounded approximate decentralized coordination using the max-sum algorithm. In DCR, 2009.
[17] M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction and optimization problems. In Proc. ICMAS, pages 401 □ 408, 1996.
[18] R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In PDCS, 2004.
[19] H. Katagishi and J.P. Pearce. KOPT: Distributed DCOP algorithm for arbitrary k-optima with monotonically increasing utility. In DCR-07, 2007.
[20] J. P. Pearce, and M. Tambe. Quality Guarantees on k-optimal Solutions for Distributed Constraint Optimization Problems, IJCAI 2007.
[21] J P. Pearce, R T. Maheswaran,Milind Tambe, Solution Sets in DCOPs and Graphical Games: Metrics and Bounds. In AAMAS06, May, 2006.
[22] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe ,Asynchronous Algorithms for Approximate Distributed Constraint Optimization with Quality Bounds. Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010).
[23] J. P. Pearce, M. Tambe, and R. Maheswaran, Solving Multiagent Networks using Distributed Constraint Optimization, AAAI,2007.
[24] D. Beasley, D.R. Bull, R.R. Martin, An overviewof genetic algorithms: Part 1, fundamentals. University Computing 1993;15(2):58-69 Department of Computing Mathematics, University of Cardiff, UK.
[25] M. Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, and M. Bowring. Quality guarantees for region optimal DCOP algorithms. In Proceedings of the Tenth International Conference on Autonomous Agents and Multiagent Systems, pages 133U″ 140, 2011.
[26] M. Mitchell, An introduction to genetic algorithms, MIT Press, 1998.
[27] A. Farinelli, M. Vinyals, A. Rogers and N. R. Jennings, Distributed Constraint Handling and Optimization,MIT Press, 2013.
[28] J. Davin and P. J. Modi. Impact of problem centralization in distributed constraint optimization algorithms. In AAMAS, 2005.
[29] T. Grinshpoun, A. Grubshtein, R. Zivan, A. Netzer, and A. Meisels, Asymmetric Distributed Constraint Optimization Problems. Journal of Artificial Intelligence Research, 47, 613-647, (2013).
[30] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. Knowledge and Data Engineering, 10(5):673-685, 1998.

Maryam Rahmaninia was born in Ghasreshirin, Kermanshah, Iran, In Aug.1985. She received her B.Sc. degree in Computer Science from Shahid Beheshti University, Tehran, Iran, in 2008 and M. Sc. Degree in Computer Science from Institute for Advanced Studies in Basic Sciences university Zanjan, Iran in 2010. Her interests are in neural network, theoretical computer science and multi agent systems.

Elnaz Bigdeli is Ph.D. student at Department of Computer Science & Engineering at the University of Ottawa. She received M.Sc. degree in Computer Science in Institute for Advanced Studies in Basic Science (IASBS) in 2011. She completed her B.Sc. degree in Information Technology Engineering in (IASBS) in 2008. Her research interests lie in the area of machine learning and data mining. In recent years, she has focused on stream data clustering. He has collaborated actively with researchers in other disciplines of computer science, particularly social networks and distributed systems.

Mohsen Afsharchi received his M.Sc. degree in Computer Engineering from the Iran University of Science and Technology in 1996, and Ph.D. in Artificial Intelligence from the University of Calgary, Canada in 2006, respectively. From 1996 to 2002 he was a University Lecturer in the University of Zanjan. Since 2006 he has been with the Computer Engineering Department of the University of Zanjan where he leads the Multi-agent Systems Lab. He is also adjunct Researcher in the Institute for Advanced Studies in Basic Sciennces (IASBS) , Zanjan, Iran. He is currently associate professor and president of the University of Zanjan. His research interests are in Multi-agent Learning, Probabilistic Reasoning and Distributed Constraint Optimization.