

Anytime Local Search for Distributed Constraint Optimization

(Short Paper)

Roie Zivan
Department of Industrial Engineering and Management
Ben-Gurion University of the Negev
Beer-Sheva, 84105, Israel
{zivanr}@bgu.ac.il

ABSTRACT

Most former studies of Distributed Constraint Optimization Problems (DisCOPs) search considered only complete search algorithms, which are practical only for relatively small problems. Distributed local search algorithms can be used for solving DisCOPs. However, because of the differences between the global evaluation of a system's state and the private evaluation of states by agents, agents are unaware of the global best state which is explored by the algorithm. Previous attempts to use local search algorithms for solving DisCOPs reported the state held by the system at the termination of the algorithm, which was not necessarily the best state explored.

A general framework for implementing distributed local search algorithms for DisCOPs is proposed. The proposed framework makes use of a *BFS*-tree in order to accumulate the costs of the system's state in its different steps and to propagate the detection of a new best step when it is found. The resulting framework enhances local search algorithms for DisCOPs with the *anytime* property. The proposed framework does not require additional network load. Agents are required to hold a small (linear) additional space (beside the requirements of the algorithm in use). The proposed framework preserves privacy at a higher level than complete DisCOP algorithms which make use of a pseudo-tree (*ADOPT*, *DPOP*).

1. INTRODUCTION

The Distributed Constraint Optimization Problem (*DisCOP*) is a general model for distributed problem solving that has a wide range of applications in Multi-Agent Systems and has generated significant interest from researchers [4, 5, 6, 8].

A number of complete algorithms were proposed in the last few years for solving DisCOPs [1, 5, 6]. While the completeness of these algorithms is an advantage in the sense that they guarantee to report the optimal solution, this is also a drawback since in order to validate that an acquired solution is optimal they must traverse the entire search space. This drawback limits the use of these algorithms to relatively small problems.

In the case of centralized optimization problems, local search techniques are used when the problems are too large to perform a complete search. Traditionally, local search algorithms maintain a complete assignment for the problem and use a goal function in or-

der to evaluate this assignment. Different methods which balance between exploration and exploitation are used to improve the current assignment of the algorithm [7]. An important feature of most local search algorithms is that they hold the best assignment that was found throughout the search. This makes them *anytime* algorithms, i.e., the quality of the solution can only increase or remain the same if more iterations of the algorithm are performed [9]. This feature cannot be applied in a distributed environment where agents are only aware of the cost of their own assignment (and maybe their neighbors too) but no one actually knows when a good global solution is obtained.

Several local search algorithms were proposed for Distributed Constraints *Satisfaction* problems. Most of them apply a synchronous framework which in each synchronous step of the algorithm agents propagate their assignments to all their neighbors in the constraint graph, collect all the assignments of their neighbors, and decide whether to change their assignment [8].

Such local search algorithms are applicable for *DisCOPs*. However, they cannot report the best solution they traverse (i.e., they are not *anytime* algorithms [9]). The reason is that in contrast to a satisfaction scenario where in the global optimal state all agents are also in a local (e.g., private) optimal state, in a distributed optimization scenario agents might have a local (private) assignment to their variables which is not optimal but the system is in an optimal (global) state.

In [8], *DSA* and *DBA* are evaluated solving sensor network DisCOPs. Apparently these algorithms perform well on this application even without a pure *anytime* property. The algorithms were compared by evaluating the state held by agents at the end of the run. Such evaluation limits the chances of local search algorithms which implement an exploring heuristic to be successful.

In order to implement local *anytime* search algorithms which follow the same general structure of *DSA* and *DBA* for Distributed Optimization problems, the global result of every synchronous iteration must be calculated and the best solution must be stored. A trivial solution would be to centralize in every iteration the costs calculated by all agents to a single agent which will inform the other agents each time a solution which improves the results on all previous solutions is obtained. However, this method has drawbacks both in the increase in the number of messages and in the violation of privacy caused from the need to inform a single agent (not necessarily a neighbor of all agents) of the quality of all other agents' states in each of the iterations of the algorithm.

The present paper proposes a general framework for enhancing local search algorithms which follow the general synchronous structure of *DSA* and *DBA* and solve DisCOPs with the *anytime* property. In the proposed framework the quality of each state is

DSA

1. $value \leftarrow \text{ChooseRandomValue}()$
2. **while** (no termination condition is met)
3. send value to neighbors
4. collect neighbors' values
5. **if** (ReplacementDecision())
6. select and assign the next value

Figure 1: Standard DSA.

accumulated via a *Breadth First Search tree (BFS-tree)* structure. Agents receive the information about the quality of the recent states of the algorithm from their *sons* in the *BFS-tree*, calculate the resulting quality including their own contribution according to the goal function, and pass it to their parents. The *root agent* makes the final calculation of the cost for each state and propagates down the tree the index number of the most successful state. When the search is terminated, all agents hold the assignment of the best state according to the global goal function.

In order to produce the best state out of m steps, the algorithm must run $m + (2 * h)$ synchronous steps where h is the height of the tree used. Since the only requirement of the tree is to maintain a parent route from every agent to the root agent, the tree can be a *BFS-tree* and its height h is expected to be small. The proposed framework does not require agents to send any messages beside the messages sent by the original algorithm. The space requirements for each agent are $O(h)$. In terms of privacy, the proposed framework preserves a higher level of privacy than state-of-the-art algorithms for solving *DisCOPs* which use a pseudo-tree

2. LOCAL SEARCH FOR DISCOPS

¹ The general design of local search algorithms for Distributed Constraint Problems is synchronous. In each step of the algorithm an agent sends its assignment to all its neighbors in the constraint network and receives the assignment of all its neighbors. For lack of space we only present one algorithm that applies to this general framework, the *Distributed Stochastic Algorithm (DSA)*. It is presented following the recent version of [8].²

The basic idea of the *DSA* algorithm is simple. After an initial step in which agents pick some value for their variable (random according to [8]), agents perform a sequence of steps until some termination condition is met. In each step, an agent sends its value assignment to its neighbors in the constraints graph and receives the assignments of its neighbors.³ After collecting the assignments of all its neighbors, an agent decides whether to keep its value assignment or to change it using a stochastic strategy (see [8] for details on the possible strategies and the difference in the resulting performance). A sketch of *DSA* is presented in Figure 1.

3. ANYTIME FRAMEWORK

Local search algorithms combine exploration and exploitation properties in order to converge to local minimas, and escape from them in order to explore other parts of the search space. When a centralized constraints optimization local search algorithm is performed, the quality of the states of the algorithm are completely

¹A detailed description of Distributed Constraint Optimization was left out for lack of space. A detailed description can be found in [1]. We assume that agents are aware only of their own topology (constraints that they are involved in personally and privately hold).

²In our description we consider an improvement a decrease in the number of violated constraints (as in Max-CSPs).

³In this paper we follow the general definition of a *DisCOP* and a *DisCSP* which does not include a synchronization mechanism. If such a mechanism exists, agents in *DSA* can send value messages only in steps in which they change their assignments.

DSA_DisCOP

1. $height \leftarrow$ height in the *BFS-tree*
2. $dist \leftarrow$ distance from root
3. $best \leftarrow null$
4. $best_index \leftarrow null$
5. $current_step \leftarrow 0$
6. **if** (root)
7. $best_cost \leftarrow \infty$
8. $value_current \leftarrow \text{ChooseRandomValue}()$
9. **while** ($current_step < (m + dist + height)$)
10. send value and $cost_i$ to parent
11. send value to non tree neighbors
12. send value and $best_index$ to sons
13. collect neighbors' values
14. $cost_i \leftarrow \text{CalculateStepCost}(current_step - height)$
15. **if**(root)
16. **if**($cost_i < best_cost$)
17. $best_cost \leftarrow cost_i$
18. $best \leftarrow value_i$
19. $best_index \leftarrow i$
20. **if** (message from parent includes a new $best_index j$)
21. $best \leftarrow value_j$
22. $best_index \leftarrow j$
23. **if** (ReplacementDecision())
24. select and assign the next value
25. delete $value_i(current_step - (2 * dist))$
26. delete cost of step ($current_step - height$)
27. $current_step ++$
28. **for** (1 to $dist + height$)
29. receive message from parent
30. **if** (message from parent includes a new $best_index j$)
31. $best \leftarrow value_j$
32. $best_index \leftarrow j$
33. send $best_index$ to sons

Figure 2: DSA in the *ALS_DisCOP* framework.

known and therefore there is no difficulty in holding the best state which was explored. In a distributed constraint optimization problem, agents are only aware of their own private state (the violated constraints which they are involved in and their costs) and, thus, a state which can seem to have high quality to a single agent might be of low global quality and vice versa.

We propose a framework that will enhance *DisCOP* local search algorithms with the *anytime* property. In the proposed *Anytime Local Search* framework, *ALS_DisCOP*, a tree is used as in *ADOPT* [5] and *DPOP* [6]. In contrast to *ADOPT* and *DPOP* that require the use of a pseudo-tree, the only requirement in *ALS_DisCOP* is that every agent has a parent route to the root agent. Thus, a *Breadth First Search (BFS)* tree on the constraint graph can be used. The *BFS-tree* structure is used in order to accumulate the cost of agents in the different states during the execution of the algorithm. Each agent calculates the cost of the sub-tree it is a root of in the *BFS-tree* and passes it to its parents. The root agent calculates the complete cost of each state and if it is found to be the best state found so far, propagates its index to the rest of the agents. Each agent A_i is required to hold its assignments in the last $2 * d_i$ steps where d_i is the length of the route of parents in the *BFS-tree* from A_i to the root agent and is bounded by the height of the *BFS-tree* (h).

In each step of the algorithm an agent collects from its sons in the *BFS-tree* the calculation of the cost of the sub-tree of which they are the root. When it receives the costs for a step j from all its sons, it adds its own cost for the state in step j and sends the result to its parent. When the root agent receives the calculations of the cost of step i from all its sons, it calculates the global state cost. If it is better than the best state found so far, in the next step it will inform all its sons that the state in step j is the best state found so far. Agents which are informed of the new best step store

their assignment in that step as the best assignment and pass the information about the best index to their sons in the next step. After every synchronous step the agents can delete the information stored about any of the steps which were not the best and are not of the last $2 * d$ steps. When the algorithm is terminated, the agents must perform another h steps (again, h is the height of the *BFS*-tree) in which they do not replace their assignment to make sure that all the agents are aware of the same index of the best step.

The structure of the framework is homogeneous for all algorithms with a distributed synchronous local search general structure (such as *DSA* and *DBA*). It is interleaved in the algorithm execution as follows:

1. In the initialization phase, besides choosing a random value for the variable, agents initialize the parameters which are used by the framework. The root initializes an extra integer variable to hold the cost of the best step.
2. In order to get the best out of m steps of the algorithm, $m+h$ steps are performed (h is the height of the *BFS*-tree). This is required so all the information needed for the root agent to calculate the cost of the m steps will reach it (see Section 4.1).
3. After values are exchanged, each agent calculates the cost of the state according to its height. An agent with height h_i calculates the cost of the state in which its sub-tree was in h_i steps ago. The root agent checks if the cost it calculated is smaller than the best cost found so far and if so saves its information. All other agents check if the best index received from their parent is new. If so they save the information (index and assignment) of the step with the corresponding index.
4. Before the step is over, the agent deletes the information that has become redundant. This includes the information on the cost which it passed to its parent on this step and the assignment of the step which its index should have been received on this step in case it was found to be better than previous steps by the root agent.
5. On the next step, the value message an agent sends to its parent will include the cost calculation it had performed in this step and the messages to its sons will include the index of the best step it knows of.
6. When the termination condition of the algorithm is met, the agents perform additional h steps in which only the best index is propagated down the tree. This way, if the last step cost calculated by the root agent is found to be best, its propagation to all agents is completed. Furthermore, by performing these steps, the possibility that different agents hold assignments of their best steps which belong to different steps is prevented (see Section 4.1).

The code for *DSA* in the *ALS_DisCOP* framework is presented in Figure 2⁴.

4. PROPERTIES OF ALS_DISCOP

ALS_DisCOP is a framework for implementing local search algorithms for *DisCOPs*. Regardless of the algorithm being used, the *ALS_DisCOP* framework offers properties which ensure the preservice of the algorithms' behavior.

⁴We assume the existence of a *BFS* tree when the algorithm begins.

4.1 Anytime property

The main goal of the *ALS_DisCOP* framework is to enhance a distributed local search algorithm with the *anytime* property (i.e. that the cost of the solution held by the algorithm at the end of the run would monotonically decrease if the algorithm is allowed to run for additional iterations [9]). In order to prove that *ALS_DisCOP* is an *anytime* framework for distributed local search algorithms, we first prove the following Lemma:

LEMMA 1. *When the algorithm terminates, all the assignments of the best state held by all the agents in the system are assignments they held in the same step (in other words the best_index of all agents is equal).*

proof: The proof for this Lemma derives directly from the last part of the framework which includes h steps in which messages including only the *best_index* are passed. Since no new *best_index* is found by the root in these steps and h steps are enough for all the agents in the *BFS*-tree to receive a new *best_index*, then even if a *best_index* was found in the last (standard) step of the algorithm its propagation is completed. □

Next, we prove that at the end of the run of $m + h$ iterations (synchronous steps of the algorithm in the framework), the index of the step with the best state which was held by the system in the m first steps of the algorithm is held by the root agent. To this end we present the following Lemma (the simple proof was left out for lack of space):

LEMMA 2. *At the $i+h$ step, the root agent holds all the needed information for calculating the quality of state i .*

The *anytime* property of the *ALS_DisCOP* framework derives directly from the Lemmas 1 and 2. h steps after each step is performed, the root agent holds the information needed to evaluate its quality and can propagate its index to all other agents in case it is the best. Since we require that agents hold the value assignments of the last $2 * dist$ (*dist* is the length in a rout of parents of an agent from the root which is at most h , see Section 3) steps they can get the update on the index of the best step before they delete the relevant assignment and hold it until they receive another update. Thus, after $m + h$ steps, the root agent holds the index of the step with the best state among the first m steps and according to Lemma 1, at the end of the algorithm run, all agents hold the value assignment of the step which was found by the root to be the best. If the algorithm is run for k more steps (the termination condition is met after $m + k + h$ steps), in the k steps performed after the first m steps, either a better solution is found or the same solution which was found best in the first m steps is reported. □

4.2 Performance analysis

Distributed algorithms are commonly measured in terms of time for completion and network load [3]. When considering a local search algorithm for *DisCOPs*, since it is not complete and the agents cannot determine that they are in an optimal state, there is no point in measuring time. We note that in the proposed framework, in order to get the best state among m steps the algorithm needs to run for $m + (2 * h)$ steps. However, the tree which is used by *ALS_DisCOP* has different requirements than the pseudo-trees which are used by complete algorithms (like *ADOPT* and *DPOP*). In a pseudo-tree which is used in complete algorithms, constrained agents must be on the same parent route (in every binary constraint, one of the constrained agents is an ancestor of the other). This requirement makes the pseudo-tree less effective when the constraint graph is dense. In contrast, the only requirement in *ALS_DisCOP* is that every agent has a parent route to the root

agent. Thus, a *Breadth First Search (BFS)* tree can be used. Since a *BFS*-tree includes the shortest route from each node to the root, the height of the resulting tree (especially when the constraint graph is dense) is expected to be small.

In terms of network load, the *ALS_DisCOP* framework does not require any additional messages. In standard local search algorithms (such as *DSA* and *DBA*), agents at each step send at least one message to each of their neighbors. The *ALS_DisCOP* framework does not require more. The additional information that agents add to messages is constant (costs of steps or best indexes).

In terms of space, an agent i is required to hold the value assignments of the last $2*dist$ steps (again $dist$ is the distance in tree arcs of an agent from the root) and to hold the cost of the last $height$ states (where $height$ is the height of the agent in the *BFS*-tree). This results in an $O(h)$ additional space requirement (linear in the worst case) for each agent.

4.3 Privacy of *ALS_DisCOP*

When a complete algorithm is being performed, agents must be able to announce that parts of the search space were already scanned and do not include a better solution than the best solution which was already found by the algorithm. This is done by declaring lower bound costs on partial assignments (*Nogoods*). While this report of costs of partial assignments is necessary for the algorithm's completeness, it is also a drawback when it comes to privacy. In a naive algorithm like synchronous *B&B*, agents are required to reveal their assignments to non-neighboring agents. In the case of algorithms which exploit the structure of the constraint network, such as *ADOPT* and *DPOP*, agents report costs with respect to their contexts, which means, for example in *ADOPT*, that a parent of a leaf agent is informed not only of the lowest cost of any of the leaf's possible value assignments but of the assignments of agents with which it is constrained and caused this cost.

In a local search algorithm, partial assignments can be revisited and therefore a cost of a state must be reported, but the context which caused this cost can remain concealed. The trade-off that is commonly considered between complete search and local search is time versus completeness. It turns out that there is another trade-off to consider, which is the privacy loss required by a complete algorithm which is not required in a local search algorithm.

In addition to the information which the local search algorithm requires agents to exchange between them, *ALS_DisCOP* requires that each agent will pass the cost of a state in the sub-tree of which it is the root. As in other algorithms which use a tree (as *ADOPT* and *DPOP*), the main problem in *ALS_DisCOP* is with the information passed by leaves in the tree to their parents [2].

When a non-leaf agent A_j passes the cost of its sub-tree to its parent, the parent does not know how many sons A_j has and the contribution of each of these agents to the reported cost. On the other hand, when a leaf agent reports a cost, its parent knows that it is the cost of a single agent (the leaf itself). However, agents are not aware of the system's topology except for their own neighbors. So in fact, even though the parent of a leaf receives its cost in every step of the algorithm, the parent does not know how many neighbors of its leaf son has in the constraint network and which constraints were violated, therefore the privacy violation is minor.

5. CONCLUSIONS

DisCOPs are hard optimization problems which require exhaustive search. Therefore, complete search algorithms are limited for solving relatively small problems.

Distributed local search algorithms were proposed for Distributed CSPs and were applied for DisCOPs [8]. However, these algo-

gorithms failed to report the best state traversed by the algorithm.

In order to enhance local search algorithms for DisCOPs with the *anytime* property, a general framework for performing distributed local search algorithms in DisCOPs was proposed in this paper. In the proposed framework, agents use a *BFS*-tree structure in order to accumulate the costs of a state of the system to the root agent which compares the cost of each state with the cost of the best state found so far and propagates the index of a new best state, once it is found, to all other agents. At the end of the run, the agents hold the best state which was traversed by the algorithm.

Apart from a small number of idle steps at the end of the run of the algorithm (two times the height of the *BFS*-tree), the framework does not require any additional slowdown in the performance of the algorithm. In contrast to complete algorithms which use a pseudo-tree, the tree used in *ALS_DisCOP* can be a *Breadth First Search (BFS)* tree. Thus, the height of the tree is expected to be small. In terms of network load, the only messages used in the *ALS_DisCOP* framework are the algorithm's messages (i.e., no additional messages are required by the framework). Agents are required to use small (linear in the worst case) additional space. In terms of privacy, *ALS_DisCOP* preserves a higher level of privacy than complete *DisCOP* algorithms which use a pseudo-tree, since an agent is not required to reveal to its neighbor with which of the other agents it is constrained (who are its other neighbors in the constraints graph).

We hope that this paper will encourage studies of distributed local search algorithms for DisCOPs which will combine stochastic and systematic methods for exploration and exploit the special properties of a distributed optimization problem.

6. REFERENCES

- [1] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward-bounding for distributed constraints optimization. In *Proc. ECAI-06*, pages 103–107, August 2006.
- [2] R. Greenstadt, B. J. Grosz, and M. D. Smith. Ssdpop: Improving the privacy of dcopt with secret sharing, distributed constraint reasoning workshop (dcr), providence, rhode island, september 2007. In *Distributed Constraint Reasoning Workshop (DCR), CP-07*, Providence, RI, USA, September 2006.
- [3] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Series, 1997.
- [4] R. Mailer and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proc. AAMAS-2004*, pages 438–445, July 2004.
- [5] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimization with quality guarantees. *Artificial Intelligence*, 161:1-2:149–180, January 2005.
- [6] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.
- [7] A. Schaerf. Local search techniques for large high-school timetabling problems. *IEEE Transactions on Systems, Man, and Cybernetics— Part A: Systems and Humans*, 29(4):368–377, 1999.
- [8] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparison and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:1-2:55–88, January 2005.
- [9] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.