Balanced Exploitation and Exploration for Max-sum Distributed Constraint Optimization

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

Abstract. Distributed Constraint Optimization Problems (DCOPs) are NP-hard and therefore most recent studies consider incomplete (local) search algorithms for solving them. Specifically, the Max-sum algorithm has drawn attention in recent years and has been applied to a number of realistic applications. Unfortunately, in many cases Max-sum does not converge. When problems include cycles of various sizes in the factor graph upon which Max-sum performs, the algorithm does not converge and the states that it visits are of low quality.
In this paper we advance the research on incomplete search for DCOPs by: (1) Proposing a version of the Max-sum algorithm that operates on an alternating directed acyclic graph (Max-sum_AD), which guarantees convergence. (2) Proposing exploration methods that allow the algorithm to escape the high quality state to which it converges. Our empirical study reveals the improvement in performance of the proposed exploitive algorithm when combined with exploration methods, compared with the performance of the standard Max-sum algorithm.

1 Introduction

The Distributed Constraint Optimization Problem (DCOP) 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 [7, 8, 13, 10].

A number of studies on DCOPs presented complete algorithms [7, 8, 4]. However, since DCOPs are NP-hard, there is a growing interest in the last few years in local (incomplete) DCOP algorithms [6, 13, 14, 11, 12]. Although local search does not guarantee that the obtained solution is optimal, it is applicable for large problems and compatible with real time applications.

The general design of the state of the art local search algorithms for DCOPs is synchronous. In each step of the algorithm an agent sends her assignment to all her neighbors in the constraint network and receives the assignment of all her neighbors. They differ in the method agents use to decide whether to replace their current value assignments to their variables, e.g., in the max gain messages algorithm (MGM) [6], the agent that can improve her state the most in her neighborhood replaces her assignment. A stochastic decision whether to replace an assignment is made by agents in the distributed stochastic algorithm (DSA) [13].

An incomplete algorithm that does not follow the standard structure of distributed local search algorithms and has drawn much attention recently is the Max-sum algorithm [3]. In contrast to standard local search algorithms, agents in Max-sum do not propagate assignments but rather calculated utilities (or costs) for each possible value assignment to their neighboring agents' variables. The general structure of the algorithm is exploitive, i.e., the agents attempt to compute the best costs/utilities for possible value assignments according to their own problem data and recent information they received via messages from their neighbors.

The growing interest in the Max-sum algorithm in recent years included its use for solving DCOPs representing various multi-agent applications, e.g., sensor systems [11] and task allocation for rescue teams in disaster areas [9]. In addition, a method for approximating the distance of the solution found by Max-sum from the optimal solution for a given problem was proposed [2]. This version required the elimination of some of the problem's constraints in order to reduce the DCOP to a tree structured problem which can be solved in polynomial time. Then, the sum of the worst costs for all eliminated constraints serves as the bound on the approximation of the optimal solution.

Previous studies have revealed that Max-sum does not always converge to a solution [3]. In fact, in some of the cases where it does not converge, it traverses states with low quality solutions and thus, at the end of the run the solution reported is of poor quality. This pathology occurs when the constraint graph of the problem includes cycles of various sizes. Unfortunately, many DCOPs which were investigated in previous studies are dense and indeed include such cycles (e.g., [7, 4]). Our experimental study revealed that for random problems, for a variety of density parameters from as low as 10%, Max-sum does not converge.

An attempt to cope with the in-convergence of Max-sum was proposed in [3]. It included the union of groups of agents to clusters of adjacent agents represented by a single agent in the cluster. The constraints between the agents in the cluster were aggregated and held by the agent representing the cluster. Thus, it required that some constraints would be revealed in a preprocessing phase to agents which are not included in the constraints (the constraint between agents A_1 and A_2 is revealed to agent A_3). The amount of information that is aggregated is not limited and in dense problems can result in a single agent holding a large part of the problem's constraints (partial centralization). In this work we avoid such an aggregation of the problem's data in a pre-processing phase and propose algorithms and methods that solve the original DCOP (as the standard Max-sum algorithm does).

In this paper we contribute to the understanding of incomplete search for DCOPs by:

1. Proposing a new version of the Max-sum algorithm which uses an alternating directed acyclic graph (DAG). The proposed algorithm (Max-sum_AD) avoids cycles by performing iterations of the algorithm in which messages are sent according to a predefined order. In order not to ignore constraints of the DCOP, after a number of iterations which guarantees the convergence of the algorithm, the order from which the direction of the DAG is derived is reversed. Then, the algorithm is performed on the reversed DAG until it converges again. We prove that the maximal number of iterations in a single direction required for the algorithm to converge is equal

to the longest path in the DAG, l (linear in the worst case). Thus, by performing l iterations in each direction we converge to a solution after considering all the constraints in the DCOP.

2. Proposing exploration heuristic methods for Max-sum_AD. The proposed methods allow the algorithm to converge to different solutions of high quality. By using the algorithm within the anytime framework, proposed for local search on DCOPs in [14], we can select the best among these solutions to be reported by the algorithm at the end of its run. To best of our knowledge, no exploration methods were proposed for Max-sum to date. Thus, we are the first to balance between exploration and exploitation of the Max-sum algorithm. Our empirical study demonstrates the success of this balanced performance in comparison with the standard Max-sum algorithm.

The rest of this paper is organized as follows: DCOPs are presented in Section 2. Section 3 presents the standard Max-sum algorithm. The Max-sum_AD algorithm is presented in Section 4. Section 5 presents exploration methods for the Max-sum_AD algorithm. Section 6 includes an evaluation of the proposed algorithm and exploration methods. Our conclusions are presented in Section 7.

2 Distributed Constraint Optimization

A $DCOP$ is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $A_1, A_2, ..., A_n$. $\mathcal{X}$ is a finite set of variables X_1,X_2,...,X_m. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains D_1, D_2,...,D_m. Each domain D_i contains the finite set of values which can be assigned to variable X_i. We denote an assignment of value $d \in D_i$ to X_i by an ordered pair $\langle X_i, d \rangle$. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary constraint* refers to exactly two variables and is of the form $C_{ij} : D_i \times D_j \rightarrow \mathbb{R}^+ \cup \{0\}$. A *binary DCOP* is a DCOP in which all constraints are binary. A *partial assignment* (PA) is a set of value assignments to variables, in which each variable appears at most once. *vars(PA)* is the set of all variables that appear in PA, $vars(PA) = \{X_i \mid \exists d \in D_i \wedge \langle X_i, d \rangle \in PA\}$. A constraint $C \in \mathcal{R}$ of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$ is *applicable* to PA if $X_{i_1}, X_{i_2}, \ldots, X_{i_k} \in vars(PA)$. The *cost of a partial assignment* PA is the sum of all applicable constraints to PA over the assignments in PA. A *full assignment* is a partial assignment that includes all the variables ($vars(PA) = \mathcal{X}$). A *solution* is a full assignment of minimal cost.

3 Standard Max-sum

The Max-Sum algorithm [3] operates on a *factor graph* which is a bipartite graph in which the nodes represent variables and constraints [1]. Each node representing a variable of the original DCOP is connected to all function-nodes that represent constraints

[1] We preserve the terminology of [3] and call constraint representing nodes in the factor graph "function nodes".

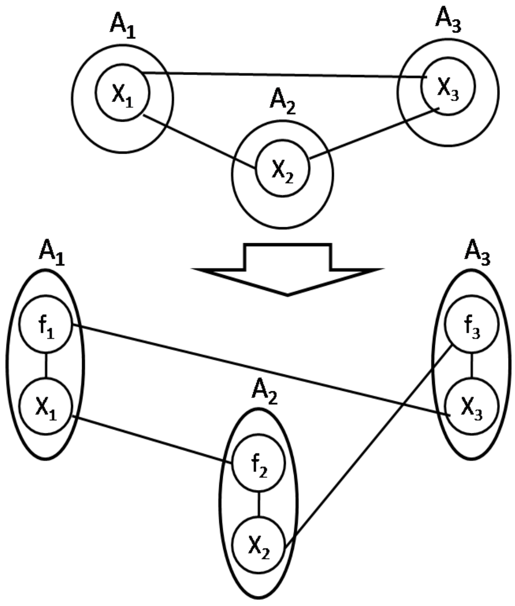


Fig. 1. Transformation of a DCOP to a factor graph

which it is involved in. Similarly, a function-node is connected to all variable-nodes that represent variables in the original DCOP which are included in the constraint it represents. Agents in Max-sum perform the roles of different nodes in the factor graph. We will assume that each agent takes the role of the variable-nodes which represent her own variables and for each function-node, one of the agents who's variable is involved in the constraint it represents, performs its role. Variable-nodes and function-nodes are considered as "agents" in Max-sum, i.e., they can send messages, read messages and perform computation.

Figure 1 demonstrates the transformation of a DCOP to a factor graph. On the top we have a DCOP with three agents, each holding a single variable. All variables are connected by binary constraints. On the bottom we have a factor graph. Each agent takes the role of the node representing her own variable and the role of one of the function-nodes representing a constraint it is involved in.

Figure 2 presents a sketch of the Max-sum algorithm. The code for variable-nodes and function-nodes is similar apart from the computation of the content of messages to be sent. For variable-nodes only data received from neighbors is considered. In messages sent by function-nodes the content is produced considering data received from neighbors and the original constraint represented by the function-node.

It remains to describe the process of the production of messages by the factor graph nodes. A message sent from a variable-node representing variable x to a function-node f at iteration i includes for each of the values $d \in D_x$, the sum of costs/utilities for this value she received from all function neighbors apart from f in iteration $i-1$. Formally,

```
Max-sum (node $n$)
1. $N_n \leftarrow$ all of $n$'s neighboring nodes
2. while (no termination condition is met)
3.     collect messages from $N_n$
4.     for each $n' \in N_n$
5.       if ($n$ is a variable-node)
6.         produce message $m_{n'}$
               using messages from $N_n \setminus \{n'\}$
7.       if ($n$ is a function-node)
8.         produce message $m_{n'}$
               using constraint and messages from $N_n \setminus \{n'\}$
9.       send $m_{n'}$ to $n'$
```

Fig. 2. Standard Max-sum.

for value $d \in D_x$ the message will include: $\sum_{f' \in F_x, f' \neq f} cost(f'.d)$, where F_x is the set of function-node neighbors of variable x and $cost(f'.d)$ is the cost/utility for value d included in the message received from f' in iteration $i-1$.

A message sent from a function-node f to a variable-node x in iteration i includes for each possible value $d \in D_x$ the best (minimal in a minimization problem, maximal in a maximization problem) cost/utility that can be achieved from any combination of assignments to the variables involved in f apart from x and the assignment of value d to variable x. Formally, in a minimization problem, the message from f to x includes for each value $d \in D_x$: $min_{ass_{-x}} cost(\langle x, d\rangle, ass_{-x})$, where ass_{-x} is a possible combination of assignments to variables involved in f not including x. The cost of an assignment $a = (\langle x, d\rangle, ass_{-x})$ is: $f(a) + \sum_{x' \in X_f, x' \neq x} cost(x'.d')$. Where $f(a)$ is the original cost in the constraint represented by f for the assignment a and $cost(x'.d')$ is the cost which was received in the message sent from node-variable x' in iteration $i-1$, for the value d' which is assigned to x' in a.

While the selection of value assignments to variables is not a part of the Max-sum algorithm, we need to describe how the solution is selected at the end of the run. Each variable selects the value assignment which received the best (lowest for a minimization problem and highest for a maximization problem) sum of costs/utilities included in the messages which were received most recently from its neighboring function-nodes. Formally, in a minimization problem, for variable x we select the value $\hat{d} \in D_x$ as follows: $\hat{d} = min_{d \in D_x} \sum_{f \in F_x} cost(f.d)$. Notice that the same information used by the variable-node to select the content of the messages it sends is used for selecting its assignment.

4 Max-sum with an Alternating DAG (Max-sum_AD)

In this section we propose a version of the Max-sum algorithm which guarantees convergence without eliminating constraints of the original DCOP. We will discuss exploration methods for this version of the algorithm in the next section.

In order to guarantee the convergence of the algorithm we need to avoid the pathology described in [3], caused by cycles of various sizes in the factor graph. To this end we select an order on all nodes in the factor graph. For example, we can order nodes according to the indexes of agents performing their role in the algorithm. A node who's role is performed by agent A_i is ordered before a node who's role is performed by agent A_j if $i < j$. For variable and function nodes held by the same agent, we can determine (without loss of generality) that a variable-node is ordered before function-nodes held by the same agent (and not the other way around). Then, we perform the algorithm for l iterations allowing nodes to send messages only to nodes which are "after" them according to this order (in the case of ordering by indexes, send messages only to agents with larger indexes than their own). After l iterations in this direction, the order is reversed and messages are sent for the next l iterations only in the opposite direction (e.g., to agents with lower indexes) . In each direction the Max-sum algorithm is performed as described in Section3 with the exception of the restriction on the messages. Thus, in every calculation of a message sent by, for example, variable-node x to function-node f, all of the most recent messages x received from its neighboring functions $f' \in F_x$, $f' \neq f$ are considered. However, for neighbors which are before x according to the current order, the most recent messages were received following the previous iteration, while from neighboring function-nodes which are after x according to the current order, the last messages were received before the last alternation of directions.

The resulting algorithm Max-sum_AD has messages sent according to a directed acyclic graph (DAG) which is determined by the current order. Each time the order changes we get a DAG on which messages on each edge of the graph are sent only in a single direction.

```
**Max-sum_AD (node $n$)**
1.  $o \leftarrow$ select an order on all nodes in the factor graph
2.  $direct\_changes \leftarrow 0$
3.  $N_n \leftarrow$ all of $n$'s neighboring nodes
4.  while (no termination condition is met)
5.      if ($direct\_changes$ is even)
6.          $current\_order \leftarrow o$
7.      else
8.          $current\_order \leftarrow reverse(o)$
9.          $N_{prev\_n} \leftarrow \{\hat{n} \in N_n :$
                $\hat{n}$ is before $n$ in $current\_order\}$
10.         $N_{follow\_n} \leftarrow N_n \setminus N_{prev\_n}$
11.     for($k$ iterations)
12.         collect messages from $N_{prev\_n}$
13.         for each $n' \in N_{follow\_n}$
14.             if ($n$ is a variable-node)
15.                 produce message $m'_n$ using
                        messages from $N_n \setminus \{n'\}$
16.             if ($n$ is a function-node)
17.                 produce message $m_{n'}$ using constraint
                        and messages received from $N_n \setminus \{n'\}$
18.             send $m_{n'}$ to $n'$
19.     $direct\_changes \leftarrow direct\_changes + 1$
```

Fig. 3. Max-sum_AD.

Figure 3 presents a sketch of the Max-sum_AD algorithm. It deffer's from standard Max-sum in the selection of directions and the disjoint sets of neighbors from whom the nodes receive messages and to whom they send messages (lines 5 - 10). Keeping track of the number of direction changes allows us to determine the current direction and act accordingly (lines 2, 5 and 19).
Next we prove the convergence of Max-sum_AD.

Lemma 1 *For any node n in the factor graph, if l' is the longest path in the DAG from some other node to n, then after l' iterations in the same direction, the content of the messages n receives does not change until the next change of direction.*

Proof: We prove by induction on l'. For $l' = 0$, node n does not receive messages from any other node as long as the direction does not change. We assume the correction of the Lemma for any length l' of a path shorter than the longest path in the DAG, l. If we denote the last node in the path whose length is equal to l by n', then according to the assumption, all the neighbors that are sending messages to n' after $l - 1$ iterations receive messages with the same content in all the following iterations with the same $current\_order$. Thus, after $l - 1$ iterations the data they use to produce the content of the messages they send is fixed. Therefore, in the following iterations they will send the same messages to node n'. $\square$.

An immediate corollary from Lemma 1 is that agents will not change their assignment selection for their variable after l iterations in the same direction until the direction is alternated, since the information used by variable-nodes for selecting an assignment is the same information they use for generating messages to function-nodes (see Section 3). Thus, the algorithm converges to a single complete assignment. The decision to escape it by changing direction is an algorithmic decision. Notice that after the first alternation of direction, although we send messages only in a single direction, the data passed in the last messages which were received before the change in direction is used for the calculation of the content of messages to be sent. Thus, all the constraints of the problem are considered.

5 Exploration Methods for Max-sum_AD

The Max-sum_AD algorithm presented above converges in linear time. After performing l iterations in each direction (where l as before is the length of the longest path in the DAG) we allow each of the constraints in the problem to be considered in the final selection of assignments, i.e., the algorithm is completely exploitive and converges to a solution after considering all the DCOP constraints . This process is deterministic. If after the second direction change we set all costs/utilities in the messages received most recently to zero, and perform l iterations according to the initial order and l in the reversed order, we will converge to the same solution. We will refer to this exploitive version of Max-sum_AD in which after every even change of directions we set all costs/utilities to zero as $plain$. Next, we propose exploration methods which can allow the Max-sum_AD algorithm to continue the search for a better solution.

1. In the first, instead of setting the costs/utilities to zero after even direction changes we continue to accumulate them as in standard Max-sum. We will refer to this method as $standard$.
2. The second method selects a random number of iterations to be performed in each direction. After each change of direction, a random number $1 \leq l' \leq k$ is selected uniformly and the algorithm is performed for l' iterations in one direction before a new l' is selected for the number of iterations to be performed in the reversed direction. We denote this method by *Random Number of Iterations Selection* (RNIS). The range k should allow convergence in some cases and avoid them in others. In our experiments we used $k = n$ where n was the number of nodes in the factor graph. There exists multiple methods for a random number selection in distributed systems (e.g., [1]). Specifically in Max-sum_AD we can have a single agent select the random numbers of iterations for future rounds and propagate this selection to all other agents via a BFS tree on the DAG as in the anytime framework proposed in [14].
3. The third method handles an unlucky selection of the order on the factor graph nodes which determines the DAG the algorithm uses. It selects a random order and performs the algorithm in both directions on this order before selecting an order again. We denote this method by *Random Order Selection* (ROS). In our implementation we selected an agent to be "first" in the order randomly and all the other agents were ordered according to their indexes following this agent (e.g., in a problem with 10 agents, if agent A_5 is selected to be first, the order is $A_5, A_6, ..., A10, A1, ..., A4$). We determined that for nodes which their role is performed by the same agent, variable-nodes come before function-nodes in the order.

6 Experimental Evaluation

We present a set of experiments that demonstrate the advantage of the proposed Max-sum_AD algorithm when combined with the proposed exploration methods, over the Max-sum algorithm.

The experiments were performed on minimization random DCOPs in which each agent holds a single variable. Each variable had five values in its domain. The network of constraints in each of the experiments, was generated randomly by selecting the probability p_1 of a constraint among any pair of agents/variables. The cost of any pair of assignments of values to a constrained pair of variables was selected uniformly between 1 and 10. Such uniform random DCOPs with constraint networks of n variables, k values in each domain, a constraint density of p_1 and a bounded range of costs/utilities are commonly used in experimental evaluations of centralized and distributed algorithms for solving constraint optimization problems [5, 4]. Other experimental evaluations of DCOPs include random max graph coloring problems [7, 13, 3], which are a subclass of random generated DCOPs.

Our experimental setup includes problems generated with 35 agents each. The factor graph generated for all versions of the Max-sum algorithm had agents performing the role of the variable-nodes representing their own variables, and for each constraint, we had the agent with the smaller index involved in it perform the role of the corresponding

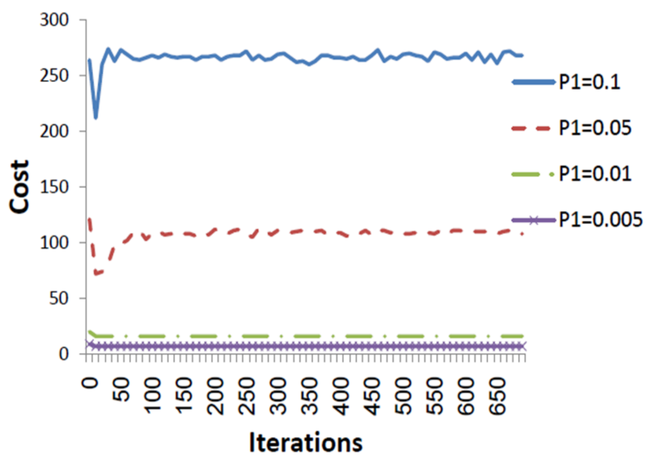


Fig. 4. Solution cost of Max-sum for every 10th iteration when solving problems with very low density

function-node. Figure 4 presents results for the standard Max-sum algorithm solving problems with very low density. Only for problems with extremely low density the algorithm converges. Thus, for the comparison of Max-sum with Max-sum_AD, we generated problems with two density parameters $p_1 = 0.1$ and $p_1 = 0.5$. For both of these density parameters Max-sum did not converge.

The Max-sum algorithm was compared with four versions of Max-sum_AD: $plain$, $standard$, $RNIS$ and ROS (see Section 5 for their description). We generated 50 random problems and ran the algorithms for 700 iterations on each of them. The results we present are an average on those 50 runs. To make sure that the Max-sum_AD algorithms converge we changed directions every 70 iterations (except in the RNIS version) which is the longest possible path in the DAG (in case the graph has a chain structure).

For each of the algorithms we present both the sum of the costs of constraints in the assignment it would have selected in each iteration and the *anytime value* (the best sum of costs found for some state visited up to this iteration). The framework proposed in [14] enhances DCOP local search algorithms with the *anytime* property. It uses a *Breadth First Search* (BFS) tree on the constraints graph in order to accumulate the costs of agents' states in the different steps during the execution of the algorithm. The anytime property in this framework is achieved with a very low overhead in time, memory and communication. In addition, it preserves a higher level of privacy than other DCOP algorithms which use tree structures [14].

Figure 5 presents for problems with constraint density $p_1 = 0.1$, for every ten'th iteration, the cost of the solution that would have been selected by the algorithm if the run would terminate at this iteration (we do not present the cost at each iteration to prevent the figure from being too dense). It is apparent that while Max-sum traverses states of low quality (with high costs) and the plain version of Max-sum_AD converges to the same solution over and over again, the versions of Max-sum_AD which are combined with exploration methods traverse lower cost states. The performance of $RNIS$ dete-

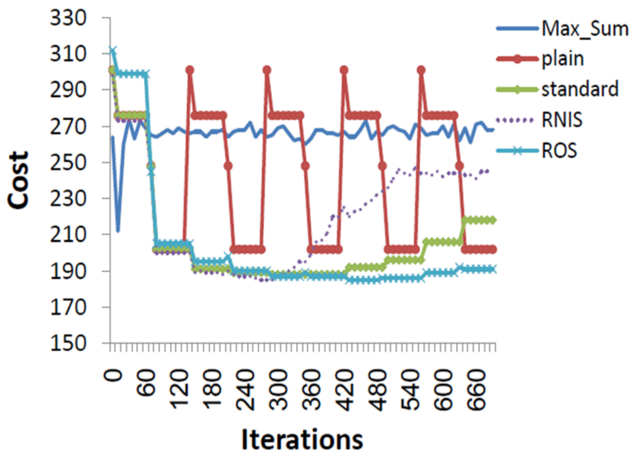


Fig. 5. Solution cost for every 10th iteration when solving problems with low density ($p_1 = 0.1$)

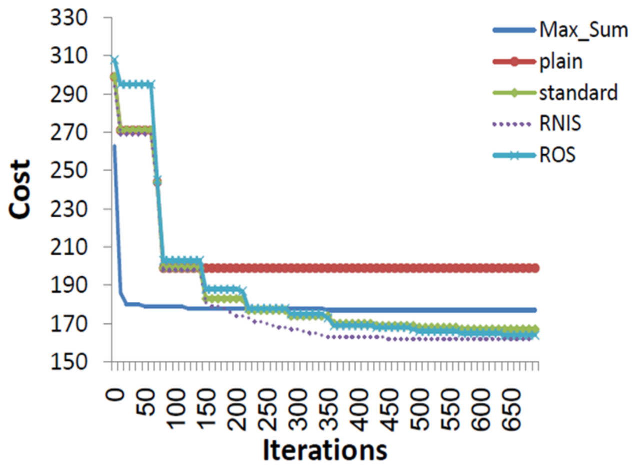


Fig. 6. Anytime cost for every 10th iteration when solving problems with low density ($p_1 = 0.1$)

riorates after the fourth change in direction and later the performance of the $standard$ heuristic deteriorates as well. On the other hand, the ROS heuristic continues to converge to high quality states with low costs. An interesting phenomenon to point out is that Max-sum visits relatively high quality states in the early iterations of the algorithm before its deterioration to an unsteady traverse of states with low quality. It seems that

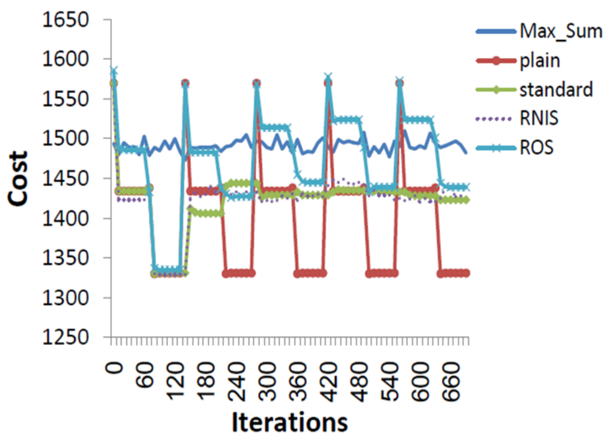


Fig. 7. Solution cost for every 10th iteration when solving problems with high density ($p_1 = 0.5$)

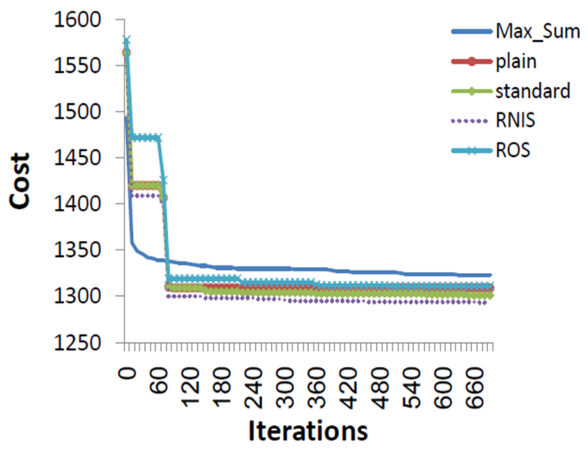


Fig. 8. Anytime cost for every 10th iteration when solving problems with high density ($p_1 = 0.5$)

it takes a number of iterations before the effect of the cycles on its performance begin. The anytime results for this experiment are presented in Figure 6. Notice that the anytime result selects the best among states in all the iterations and not only the ones which their cost was presented in Figure 5.

Similar results are presented in Figures 7 and 8 for problems with constraint density $p_1 = 0.5$. On dense problem the advantage of the different versions of Max-sum_AD over the standard Max-sum is more apparent. In addition the $plain$ version of the algorithm converges to a solution of high quality and the advantage achieved by the exploration methods is less apparent. Here, the exploitive performance which we observed

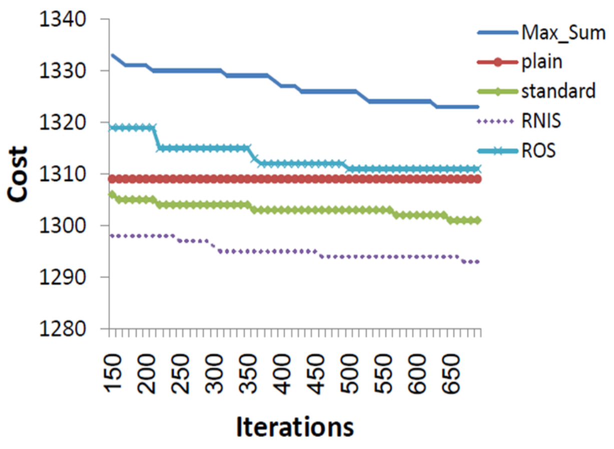


Fig. 9. A closer look at the anytime cost for every 10th iteration, starting after 140 iterations ($p_1 = 0.5$)

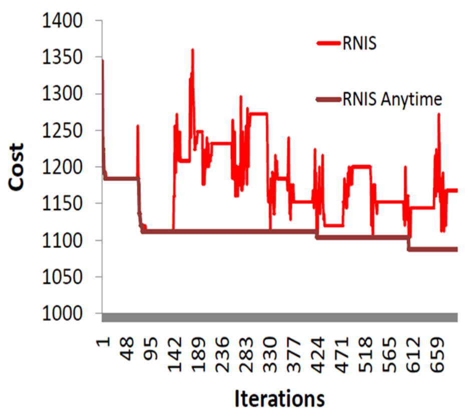
for Max-sum in the first iterations of Figure 5 does not appear. This is probably because the size of the cycles is smaller, thus their destructive effect is instantaneous. A closer look at the anytime result of the algorithms after the first 140 iterations is presented in Figure 9. While the ROS method does not improve on the $plain$ method, the standard method and RNIS find solutions with lower costs. All the versions of Max-sum_AD outperform Max-sum. It is important to notice that while the average cost in each iteration as presented in Figure 7 does not reveal an advantage of the exploration methods the anytime result does. This is because different iterations were successful when solving different problems. Thus, the average in each iteration does not reveal this success.

Figure 10 demonstrates the balance between exploitation and exploration of the proposed methods by presenting the states in the run of a single problem for ROS and RNIS. It is apparent that after each change in direction there is an exploration phase and a convergence to a solution. RNIS converges only when the random selected number of iterations in the same direction is large enough while ROS converges after each change in direction.

7 Conclusion

The Max-sum algorithm offers an innovative approach for solving DCOPs. Unfortunately, when problems include cycles of various sizes in the factor graph, the algorithm does not converge and the states it visits are of low quality.

In this paper we proposed a new version of the Max-sum algorithm, Max-sum_AD, which guarantees convergence. Max-sum_AD uses an alternating DAG to avoid cycles. We proved that the algorithm converges if the number of iterations it performs in a single direction is equal to or larger than the longest path in the DAG.



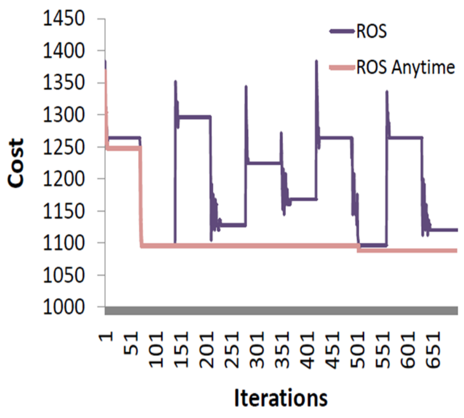


Fig. 10. A single run of RNIS (left) and ROS (right)

The guaranteed convergence of the strictly exploitive ("plain") algorithm serves as a baseline on which we add exploration elements and allow the algorithm to continue the search for a high quality solution. The use of the algorithm within the anytime framework proposed in [14] allows the selection of the best among the complete assignments it converges to as the algorithm's outcome.

Our empirical study reveals the advantage of the Max-sum_AD algorithm when combined with exploration methods over Max-sum. In the future we intend to investigate the compatibility of our exploration methods to realistic applications.

Acknowledgment: We thank Alessandro Farinelli for helping us understand the Max-sum algorithm.

References

1. B. Awerbuch and C. Scheideler. obust random number generation for peer-to-peer systems. *Principles of Distributed Systems, Lecture Notes in Computer Science*, 4305:275–289, 2006.
2. A. Farinelli, A. Rogers, and N. R. Jennings. Bounded approximate decentralised coordination using the max-sum algorithm. In *Proc. 13th Workshop on Distributed Constraint Reasoning (DCR) at IJCAI-09)*, pages 46–59, Pasadena, CA, July 2009.
3. A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. 7th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-08)*, pages 639–646, 2008.
4. A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding. *J. of Artificial Intelligence Research*, 34:25–46, 2009.
5. J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159:1–26, 2004.
6. R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for dcop: A graphical-game-based approach. In *Proc. Parallel and Distributed Computing Systems PDCS)*, pages 432–439, September 2004.
7. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. Adopt: asynchronous distributed constraints optimizationwith quality guarantees. *Artificial Intelligence*, 161:1-2:149–180, January 2005.

8. A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.
9. S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings. Decentralized coordination in robocup rescue. *Comput. J.*, 53(9):1447–1461, 2010.
10. M. E. Taylor, M. Jain, Y. Jin, M. Yokoo, and M. Tambe. When should there be a "me" in "team"?: distributed multi-agent optimization under uncertainty. In *Proc. of the 9th conference on Autonomous Agents and Multi Agent Systems (AAMAS 2010)*, pages 109–116, May 2010.
11. W. T. L. Teacy, A. Farinelli, N. J. Grabham, P. Padhy, A. Rogers, and N. R. Jennings. Max-sum decentralised coordination for sensor systems. In *AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, pages 1697–1698, 2008.
12. X. S. W. Yeoh and S. Koenig. Trading off solution quality for faster computation in dcop search algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 354–360, July 2009.
13. W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. *Artificial Intelligence*, 161:1-2:55–88, January 2005.
14. R. Zivan. Anytime local search for distributed constraint optimization. In *AAAI*, pages 393–398, Chicago, IL, USA, 2008.