Max/Min-sum Distributed Constraint Optimization through Value Propagation on an Alternating DAG

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

ABSTRACT

Distributed Constraint Optimization Problems (DCOPs) are NP-hard and therefore the number of studies that consider incomplete algorithms for solving them is growing. 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 produce high quality solutions. More specifically, 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 algorithms 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 in linear time. (2) Identifying major weaknesses of Max-sum and Max-sum_AD that cause inconsistent costs/utilities to be propagated and affect the assignment selection. (3) Solving the identified problems by introducing value propagation to Max-sum_AD. Our empirical study reveals a large improvement in the quality of the solutions produced by Max-sum_AD with value propagation (VP), when solving problems which include cycles, compared with the solutions produced by the standard Max-sum algorithm, Bounded Max-sum and Max-sum_AD with no value propagation.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: [Multiagent systems]

General Terms

Algorithms, Experimentation

Keywords

DCOP, Incomplete Algorithms, GDL

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 [10, 12, 21, 15, 16].

A number of studies on DCOPs presented complete algorithms [10, 12, 5]. However, since DCOPs are NP-hard, there is a growing interest in the last few years in incomplete DCOP algorithms [9, 21, 17]. Although incomplete algorithms do not guarantee that the obtained solution is optimal, they are applicable for large problems and compatible with real time applications.

Local search algorithms for DCOPs are incomplete algorithms whose general structure 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. 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) [9]; the agent that can improve its state the most in its neighborhood replaces its assignment. A stochastic decision whether to replace an assignment is made by agents in the distributed stochastic algorithm (DSA) [21].

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 [4]. Max-sum is an incomplete *GDL* algorithm [1]. In contrast to standard local search algorithms, agents in Max-sum do not propagate assignments but rather calculate utilities (or costs) for each possible value assignment of 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 [17, 15] and task allocation for rescue teams in disaster areas [13]. 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 [14]. This version required the elimination of some of the problem's constraints in order to reduce the DCOP to a tree structured problem that 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 [4]. In fact, in some of the cases where it does not converge, it traverses states with low quality and thus, at the end of the run a poor quality solution is reported. This pathology apparently occurs when the constraint graph of the problem includes cycles of various sizes [4]. Unfortunately, many DCOPs that were investigated in previous studies are dense and indeed include such cycles (e.g., [10, 5]). Our experimental study revealed that on random problems of different density parameters and problem sizes, and on problems with structured constraints (graph coloring), Max-sum does not converge.

In this paper we contribute to the development of incomplete

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

algorithms for solving DCOPs by:

1. Proposing a new version of the Max-sum algorithm that 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. The order divides the set of neighbors for each agent to two disjoint subsets, one of agents that come before it in the order in which it receives messages, and the other of neighbors that come after it in the order in which it sends messages (notice that the order is on the direction of messages and not on the agents' actions).

   In order not to ignore constraints of the DCOP, after a number of iterations in which all agents perform concurrently, 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. We identify major weaknesses of Max-sum and Max-sum_AD, which are their performance in the presence of ties and the possibility that best costs/utilities computed for the same variable consider different value assignments and thus are not valid. We demonstrate that the propagation of such inconsistent information results in a distorted selection of assignments by the agents.

3. We solve the problems of inconsistent cost propagation and tie breaking by using value propagation. After performing the algorithm in both directions and allowing it to converge for the second time after considering all the problem's constraints, we require that agents add to the messages they send the value assignment they have selected and that only constraints with these values are considered. Thus, we prevent the possibility that different agents consider costs/utilities that are based on conflicting value assignments. We note that value propagation has been used in previous studies for complete GDL algorithms [12, 18]) for breaking ties, including for the production of the optimal solution for the tree structure factor graph in Bounded Max-sum [14]. However, to best of our knowledge, ours is the first use of value propagation in GDL algorithms for solving DCOPs with cyclic factor graphs.

Our empirical study demonstrates the success of Max-sum_AD combined with value propagation in comparison with the standard Max-sum algorithm and with Bounded Max-sum when solving random DCOPs and graph coloring problems (problems that include cycles on which Max-sum fails to converge).

The rest of this paper is organized as follows: We present related work in Section 2. DCOPs are presented in Section 3. Section 4 presents the standard Max-sum algorithm. The Max-sum_AD algorithm is presented in Section 5. Section 6 identifies the need for value propagation (VP) and further describes how VP is combined with Max-sum_AD. Section 7 includes an evaluation of the proposed algorithm in comparison with Max-sum and Bounded Max-sum. In Section 8 we discuss how dynamic spanning trees can be generated as part of the Max-sum_AD algorithm which can be used to produce bounded approximation of the optimal solution. Our conclusions are presented in Section 9.

2. RELATED WORK

Many algorithms for solving DCOPs were proposed in the last decade. These can be divided into complete and incomplete algorithms. While some complete algorithms such as ADOPT, BnB-ADOPT [10, 20], and AFB [5] perform distributed search, GDL-based complete algorithms implement a dynamic programming approach [1, 12, 18]. The first to apply this approach to DCOP were Petcu and Faltings by proposing the DPOP algorithm [12]. DPOP performs dynamic programming on a pseudo-tree structure. Since pseudo trees may have limited branching in dense problems, recent studies investigated alternative structures (e.g., junction trees) in order to increase the parallelism in GDL based algorithms [3, 18].

The other set of incomplete algorithms for solving DCOPs includes search algorithms as well. As mentioned in the Introduction, in standard local search algorithms for DCOPs (e.g., DSA) agents exchange messages in synchronous iterations and the algorithms differ in the method agents use to decide when and how to replace their assignment.

In [9, 11], a different approach towards local search for solving DCOPs was proposed. In these studies, completely exploitive algorithms are used to converge to local optima solutions, which are guaranteed to be within a predefined distance from the global optimal solution. The approximation level is dependent on a parameter k, which defines the size of coalitions that agents can form. These k size coalitions transfer the problem data to a single agent, which performs a complete search procedure in order to find the best assignment for all agents within the k size coalition. As a result, the algorithm converges to a state that is *k optimal* (k-opt) [11], i.e., no better state exists if k agents or fewer change their assignments.

The production of k-opt solutions may require solving an exponential number of problems of size k. To overcome this shortcoming, recent studies have proposed alternatives for the selection of small local environments that would be solved optimally in order to produce quality guarantees on the overall solution. One alternative, t-distance, created environments dependent on the distance of nodes in the constraint network [6]. While this alternative reduced the number of problems that need to be solved it did not bound the size of the problems that are solved. The most recent approach included the generation of environments that were bounded both by distance and size [19]. Thus, the number of problems to solve is bounded by the number of agents and the problems to solve by the predefined size.

Aggregation of agents' constraints was also used in an attempt to cope with the in-convergence of Max-sum [4]. 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 that are not included in the constraints (the constraint between agents A_1 and A_2 is revealed to agent A_3). Furthermore, the amount of aggregated information was not limited and in dense problems could result in a single agent holding a large part of the problem's constraints (partial centralization). Another approach is to aggregate constraints and unite nodes in the constraint graph so that the resulting graph would be a tree [18]. However, the result of this rearrangement of the constraint graph is the need to perform exponential computation and transfer exponential communication that will result in a complete solution.

In this paper we focus on incomplete GDL algorithms that avoid partial centralization and clustering of agents, and attempt to solve the original DCOPs as do standard complete algorithms (e.g., ADOPT and DPOP), *one-opt* distributed local search algorithms (e.g., DSA

and MGM) and as the standard Max-sum algorithm does [14].

The alternating direction approach we implement in this paper is inspired by algorithms for solving asymmetric distributed constraints problems [2]. However, unlike in the case of asymmetric problems where the motivation for this approach was preserving privacy, in this paper the motivation is strictly algorithmic.

3. DISTRIBUTED CONSTRAINT OPTIMIZATION

To avoid confusion, and without loss of generality, in the rest of this paper we will assume all problems are minimization problems as presented in the early DCOP papers (e.g., [10]). Thus, we assume that all constraints define costs and not utilities. The GDL algorithm for minimization problems is actually a *Min-sum* GDL algorithm. However, we will continue to refer to it as Max-sum since this name is widely accepted. Our description of a DCOP is also consistent with the definitions in many DCOP studies, e.g., [10, 12, 5].

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 that 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 *complete assignment* is a partial assignment that includes all the variables ($vars(PA) = \mathcal{X}$). An *optimal solution* is a complete assignment with minimal cost.

For simplicity, all DCOPs considered in this paper are binary DCOPs in which each agent holds exactly one variable.

4. STANDARD MAX-SUM

The Max-Sum algorithm [4] is a GDL algorithm [1] that operates on a *factor graph* [7] that 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 that it is involved in. Similarly, a function-node is connected to all variable-nodes that represent variables in the original DCOP that 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 that represent its own variables and for each function-node, one of the agents whose variable is involved in the constraint it represents, performs its role. Variable-nodes and function-nodes are considered "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 its own variable and the role of one of the function-nodes representing a constraint it is involved in, e.g., in this factor graph agent A_1 takes the role of function-node f_1 which represents the constraint between its own variable x_1 and variable x_3 held by agent A_3.

[1]We preserve the terminology of [4] and call constraint representing nodes in the factor graph "function-nodes".

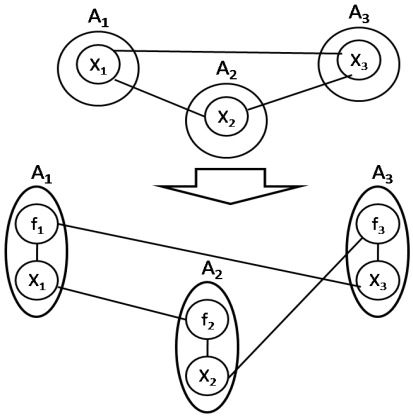

Figure 1: Transformation of a DCOP to a factor graph

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'

Figure 2: Standard Max-sum.

Figure 2 presents a sketch of the Max-sum algorithm.[2] 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 content of messages sent by the factor graph nodes. A message sent from a variable-node x to a function-node f at iteration i, includes for each of the values $d \in D_x$ the sum of costs for this value it received from all function neighbors apart from f in iteration $i-1$. Formally, for value $d \in D_x$ the message will include:

$$\sum_{f' \in F_x, f' \neq f} cost(f'.d) - \alpha$$

where F_x is the set of function-node neighbors of variable x and $cost(f'.d)$ is the cost for value d included in the message received from f' in iteration $i-1$. α is a constant that is reduced from all costs included in the message (i.e., for each $d \in D_x$) in order to prevent the costs carried by messages throughout the algorithm

[2]In contrast to previous papers on Max-sum, we present it using a pseudo-code. This is following standard DCOP literature, e.g., [10, 12, 21]. Nevertheless, only the presentation is different. The algorithm itself is identical to the algorithm presented in [4, 14].

from growing arbitrarily. Selecting α to be the average on all costs included in the message is a reasonable choice for this purpose [4, 14]. Notice that as long as the amount reduced from all costs is identical, the algorithm is not affected by this reduction since only the differences between the costs for the different values matter.

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 minimal cost of any combination of assignments to the variables involved in f apart from x and the assignment of value d to variable x. Formally, 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, X_f is the set of variable-node neighbors of function-node f, and $cost(x'.d')$ is the cost that was received in the message sent from variable-node x' in iteration $i-1$, for the value d' that is assigned to x' in a.

While the selection of value assignments to variables is not used to generate the messages in the Max-sum algorithm, in every iteration an agent can select its value assignment and the assignment selected by agents at the end of the run is the reported solution. Each variable-node selects the value assignment that received the lowest sum of costs included in the messages that were received most recently from its neighboring function-nodes. Formally, 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 value assignment.

5. MAX-SUM ON AN ALTERNATING DAG (MAX-SUM_AD)

In order to guarantee the convergence of Max-sum we need to avoid the pathology described in [4], caused by cycles 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 indices of agents performing their role in the algorithm. A node whose role is performed by agent A_i is ordered before a node whose 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 the function-nodes and break ties among function-nodes using their indices.

Once we define an order on all nodes in the factor graph, each agent (node) can divide its set of neighbors to two disjoint subsets, the subset of neighbors in the factor graph that come before it in the order, from whom it receives messages, and the subset of neighbors that are ordered after it, to whom it sends messages.

Next, 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 indices, send messages only to agents with larger indices than their own). Notice that all agents perform concurrently in each iteration of the algorithm, thus the order does not affect the agents' actions, only the direction of messages.

Max-sum_AD (node n)

```
1.  currebt_order ← select an order on all nodes in the factor graph
2.  N_n ← all of n's neighboring nodes
3.  while (no termination condition is met)
4.      N_prev_n ← {n̂ ∈ N_n :
            n̂ is before n in current_order}
5.      N_follow_n ← N_n \ N_prev_n
6.      for(k iterations)
7.          collect messages from N_prev_n
8.          for each n' ∈ N_follow_n
9.              if (n is a variable-node)
10.                 produce message m_n' using
                        messages from N_n \ {n'}
11.             if (n is a function-node)
12.                 produce message m_n' using constraint
                        and messages received from N_n \ {n'}
13.             send m_n' to n'
14.     current_order ← reverse(current_order)
```

Figure 3: Max-sum_AD.

After l iterations in the selected direction, the order is reversed and messages are sent for the next l iterations only in the opposite direction (i.e., to agents with lower indices). In each direction the Max-sum algorithm is performed as described in Section 4 with the exception of the restriction on the messages that are sent. Agents always consider the last message received from all their neighbors (regardless of the direction) when producing a new message. For example, when variable-node x produces a message it would send 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. Notice that the most recent message from a neighbor that is before x according to the current order was received following the previous iteration, while from a neighboring function-node that is after x according to the current order, the last message was 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.

Figure 3 presents a sketch of the Max-sum_AD algorithm. It differs 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 1, 4, 5 and 14).
Next we prove the convergence of Max-sum_AD when performed in a single direction.

LEMMA 1. *Given o, an order on the nodes of the factor graph FG, for any node $n \in FG$, if l is the length of the longest path in FG according to o that reaches n, then after l iterations, the content of the messages n receives does not change as long as messages are sent according to o.*

Proof: We prove by a complete induction on the length of the longest path to node n according to o. A node n', which is first according to o, i.e., the length of the longest path that reaches it according to o is equal to zero, does not receive messages from any other node. We assume the correctness of the Lemma for any node n' that the longest path that reaches it is $l' < l$. We now check the correctness of the Lemma for node n, where the longest path reaching it is equal to l. The longest path to all of the neighbors of node n that are ordered before it according to o must be shorter than l. Thus, according to the assumption, after $l-1$ iterations of

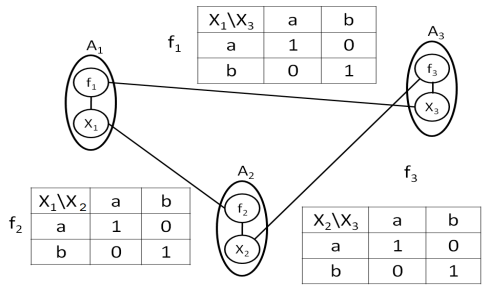

Figure 4: Example of the need to break ties

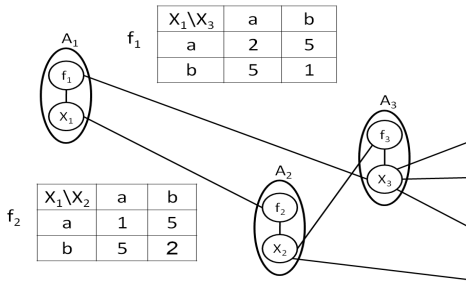

Figure 5: Example of the need for value propagation beyond ties

the algorithm the messages they receive will not change. Since the content of messages produced by agents in the Max-sum algorithm is dependent on the content of messages they received last, then from the $l'th$ iteration and on these neighbors of node n will be sending identical messages to n. Thus, the Lemma holds for node n as well. □.

The first immediate corollary from Lemma 1 is that after a number of iterations equal to the diameter of the factor graph FG, all the nodes in FG will continue to receive the same messages until o is reversed. Moreover, since agents use the last messages they have received in order to select their value assignments, the value assignments will not change either. Thus, the algorithm converges to a single complete assignment.

The decision to escape this fixed assignment by changing direction is an algorithmic decision. If this decision is made, the algorithm will converge again after a linear number of iterations in the reversed order, but not necessarily to a better solution. We demonstrate in our empirical study that one version of Max-sum_AD monotonically improves the states it converges to after each direction change and that another does not.

Notice, that after the first alternation of direction, although we send messages only in a single direction, the data passed in the last messages that were received before the change in direction is used for the calculation of the content of the following messages to be sent and for value assignment selections. Thus, after the first change of direction, all the constraints of the problem are considered.

6. MAX-SUM_AD WITH VALUE PROPAGATION

In this section we introduce value propagation into the Max-sum_AD algorithm. We start by presenting the motivation for this addition and then go into the algorithmic details.

6.1 Motivation for Value Propagation

In order to understand the need for value propagation in Max-sum in general and specifically in Max-sum_AD we identify two phenomena that deteriorate the ability of agents to identify the value assignments that will minimize the cost of the solution.

We illustrate the first phenomenon in the following example: The factor graph depicted in Figure 4 presents the standard need for value propagation as identified in previous studies on complete GDL algorithms [12, 18]. Each of the function-nodes computes for each of the values of its neighboring variable-nodes the minimal cost that it can offer by assigning a value to its other variable-node neighbor. In this specific symmetric example, each function identifies that for each value a, when assigning b to the other variable the cost is 0. Similarly for each value b, when assigning a to the other variable the cost is 0 as well. Thus, all messages sent by function-nodes to variable-nodes contain zeros for all values. As a result, messages from variable-nodes to function-nodes contain zero costs as well. In other words, in this specific problem, no information is propagated to the agents throughout the algorithm run. At the end of the run the variable-nodes are indifferent regarding their possible value assignments and if they all select value a the solution cost is 3. It is easy to see that there exists a solution in which one variable is assigned a and the other two assign b with a cost of 1.

While for complete algorithms ties are the only motivation for value propagation, the following example demonstrates that this is not the case in an incomplete Max-sum algorithm such as Max-sum_AD. Consider the factor graph depicted in Figure 5. Assume the order is according to the agents' indices. Function f_1 will send to x_3 costs 2 and 1 for its values a and b, respectively. Similarly, f_2 will send to x_2 costs 1 and 2 for its values a and b, respectively. Thus, the algorithm will converge to a solution that includes assignments $\langle x_2, a\rangle$ and $\langle x_3, b\rangle$. However, the selection of a for x_2 was under the assumption that value a is selected for x_1 while the selection of b for x_3 is under the assumption that x_1 is assigned b. Since only one of them can be assigned to x_1 the contribution of functions f_1 and f_2 to the solution cost is 6. If we would have selected first the assignment of x_1 and then the assignments for x_2 and x_3 accordingly, we could have reached a solution in which the contribution of those two functions was 3 (e.g., $\langle x_1, a\rangle$,$\langle x_2, a\rangle$ and $\langle x_3, a\rangle$). This inconsistency in the use of the information propagated by Max-sum_AD does not affect only the assignment selection. The information passed by messages is used to generate additional messages and therefore inconsistent information is propagated further to other agents in the distributed system. [3]

6.2 Introducing value propagation into Max-sum_AD

We overcome the pathologies we identified above by using a value propagation procedure similar to the method used in complete GDL algorithms for avoiding ties [12, 14, 18]. On iterations in which we perform value propagation we require that variable-nodes include in their messages to function-nodes their selected value assignments. Function-nodes select the best cost considering only the value assignments they received from their variable-node neighbors, which are ordered before them. Formally, in iterations in which we perform value propagation the message sent by function-node f to variable x includes as before for each $d \in D_x$:

$$min_{ass_{-x}} cost(\langle x, d\rangle, ass_{-x})$$

However, for a variable-node from which a value assignment was received in its latest message, the term $min_{ass_{-x}}$ considers only

[3]We demonstrate the phenomenon for Max-sum_AD since it is easier to follow. In standard Max-sum, such inconsistent information concerning the conflicting assignment of some node is propagated in all directions and through cycles, fed back to the node itself.

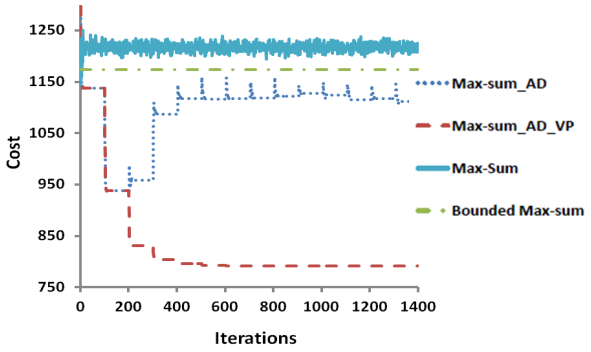

Figure 6: Solution cost of the Max-sum versions when solving problems with low density ($p_1 = 0.2$)

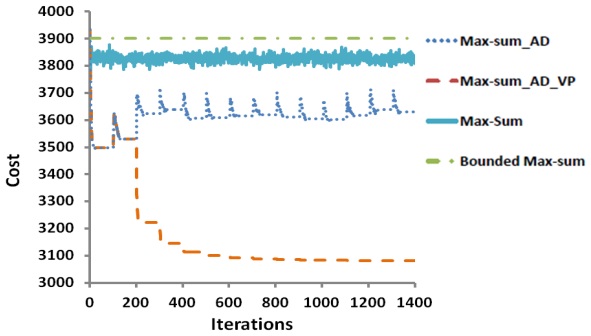

Figure 7: Solution cost when solving problems with high density ($p_1 = 0.6$)

this value assignment. Specifically, if a value assignment was received from each of the neighboring variable-nodes then $min_{ass_{-x}}$ is a single partial assignment.

In order to demonstrate the value propagation procedure, consider once again the factor graph depicted in Figure 5. Variable-node x_1 selects value a and includes this selection in the messages to function-nodes f_1 and f_2. Then f_1 calculates for the values of x_3 the costs 2 and 5 for values a and b, respectively. Similarly, function f_2 calculates costs 1 and 5 to the values a and b of variable-node x_2.

The timing for starting value propagation has a major effect on its success. If we would start value propagation from the first iteration, the sum of costs that indicate to agents which value assignments are better will not be propagated through the system. Thus, the selection of the first value assignment will be done in complete entropy and the following assignments can lead to an assignment with low quality. Instead, we start the value propagation procedure only after the algorithm converged in both directions (after the second order alternation). At this time agents have considered all the problem's constraints and have enough knowledge to make a quality selection of value assignments. Our experimental study also indicates that the best assignment found by Max-sum_AD without value propagation is after its second convergence (just before the second change of direction).

7. EXPERIMENTAL EVALUATION

In this section we present experiments that demonstrate the advantage of the proposed Max-sum_AD algorithm when combined with value propagation, over existing versions of the Max-sum algorithm.

The first set of experiments was performed on minimization random DCOPs in which each agent holds a single variable. Each variable had ten values in its domain. The network of constraints in each of the experiments was generated randomly by selecting the probability p_1 for 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 [8, 5].

The experimental setup included problems generated with 50 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 function-node.

We compared the Max-sum_AD algorithm with and without value propagation with the Max-sum and Bounded Max-sum algorithms [14]. We generated 50 random problems and ran the algorithms for 1400 iterations on each of them. The results we present are an average of those 50 runs. To make sure that the Max-sum_AD algorithms converge, we changed directions every 100 iterations, which is the longest possible path in the DAG (in case the graph has a chain structure). For each of the algorithms we present the sum of the costs of constraints included in the assignment it would have selected in each iteration.

Figure 6 presents the costs of the solutions found by the four algorithms when solving problems of relatively low density ($p_1 = 0.2$). It is most apparent that the Max-sum algorithm does not converge and it traverses complete assignments with high costs. The results of Bounded Max-sum are slightly better than the results of standard Max-sum. Max-sum_AD converges to a solution of lower cost even before the first direction change (in the first 100 iterations) and to a much better solution after the direction change. However, after the following direction changes it converges to solutions with higher costs. The solutions with the lowest costs are found by Max-sum_AD with value propagation. Since we begin the value propagation only after the second direction change, in the first 200 iterations it is identical to Max-sum_AD. However, after the second direction change it converges to a solution with a much smaller cost. It is interesting to mention that after the following direction changes it continues to improve until it finally converges to the solution with the lowest cost after the fifth direction change.

Figure 7 presents similar results for problems with higher constraint density – $p_1 = 0.6$. Here, it is notable that Bounded Max-sum finds solutions with higher costs than the standard Max-sum algorithm. This is reasonable since the more dense the problem is, the more edges are removed from the graph by Bounded Max-sum in order to produce the tree structured factor graph or, in other words, more constraints are ignored when producing the solution. The final difference in cost between the two versions of the Max-sum_AD algorithm seems similar to the results obtained for low density problems. However, here on dense problems, most of the reduction in cost by Max-sum_AD_VP was made after the second change of direction when the value propagation phase began. In addition, it is notable that the value propagation version of Max-sum_AD keeps improving even after 800 iterations (7 direction changes).

In our second set of experiments we present the relation between the costs of the solutions found by the different versions of the Max-sum algorithm and the optimal solution. Figures 8 and 9 present results of the four Max-sum versions solving smaller random problems on which we could run an exhaustive algorithm and

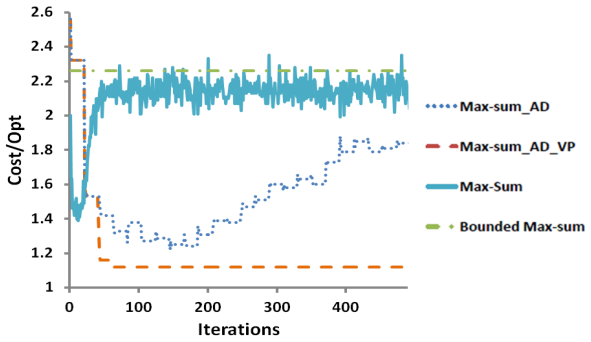

Figure 8: Solution cost when solving small problems with low density ($p_1 = 0.3$)

find the optimal solution. The problems included 10 agents, each holding a single variable with 5 values in its domain. Once again, we generated sparse and dense problems. However, the density parameters used were $p_1 = 0.3$ and $p_1 = 0.7$. This is because a lower density parameter for such small problems may generate problems with multiple components. On these small problems we could guarantee convergence by changing directions in Max-sum_AD every 20 iterations. Therefore, we ran the algorithms for a smaller number of iterations (500). The results presented include the factor from the optimal solution obtained by dividing the average cost of the assignments found by the algorithms in each iteration by the cost of the optimal solution.

It is interesting to observe that for both density parameters Max-sum_AD with value propagation found a solution with an average cost very close to the average optimal cost (1.12 factor for the sparse problems and 1.07 for the dense problems). It is also notable that Bounded Max-sum finds solutions with low quality for these problems (e.g., a factor larger than 2.5 of the average optimal cost for the dense problems). We assume this is caused by the smaller domains in these problems, which result in problems in which the constraints have larger differences among them. Thus, ignoring some of the constraints as done by Bounded Max-sum can have a larger effect on the result. It is interesting to notice the phenomenon that applies both to standard Max-sum and Max-sum_AD without value propagation. They both produce better results in the beginning of the run (i.e., in the early iterations of the algorithm) and then their performance deteriorates. This phenomenon is more apparent when solving low density problems. It is clear that the deterioration that these algorithms exhibit is prevented by value propagation. Thus, we assume that this deterioration is related to the propagation of inconsistent costs through the distributed system as we described in Section 6.1.

In the last set of experiments we generated graph coloring problems that include many ties. The problems we used included 50 agents, each holding a single variable and 3 colors in the domain of each variable. The density parameter was $p_1 = 0.05$, i.e., each agent had 2.5 neighbors on average. The problems are standard graph coloring problems, i.e., neighbors with identical colors induce a cost of 1 while for neighbors with different colors the cost is zero. The results presented in Figure 10 validate the observation we presented in Section 6.1 of Max-sum's weakness in the presence of ties. Both Max-sum and Max-sum_AD without value propagation are stuck with their initial assignment since no information is propagated through the system. On the other hand, Bounded Max-sum and Max-sum_AD_VP both include value propagation; therefore they are able to break the ties and produce high quality solutions. Max-sum_AD with VP still finds a solution with a smaller cost than Bounded Max-sum.

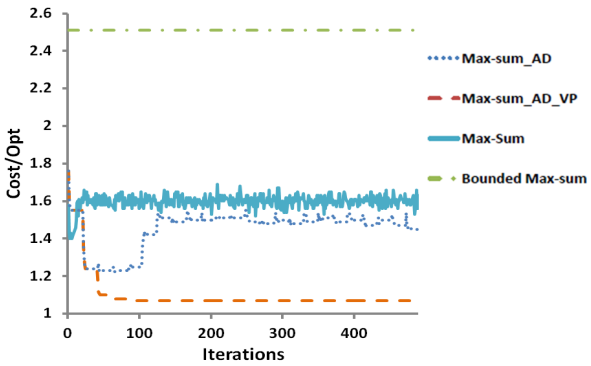

Figure 9: Solution cost when solving small problems with high density ($p_1 = 0.7$)

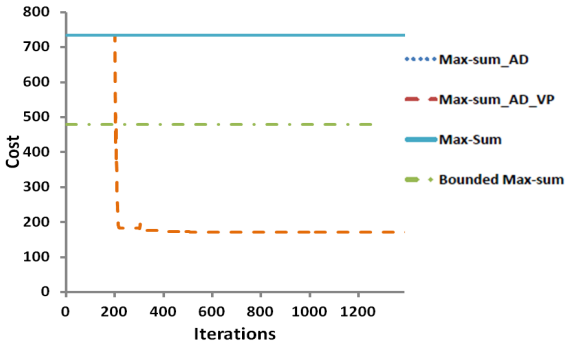

Figure 10: Solution cost when solving graph coloring problems

8. BOUNDED APPROXIMATION IN MAX-SUM_AD

While incomplete algorithms are not guaranteed to find the optimal solution, some of the recent studies on incomplete methods for DCOPs have offered quality guarantees and bounds on the distance of the produced solution from the optimal solution (e.g., [11, 14]). Specifically, *Bounded_Max-sum* [14] exploits the ability of Max-sum to find an optimal solution on a tree structured graph, and produces a bound from the optimal solution by reducing the problem to such a graph and accounting for the costs of the removed edges. This approach can be applied in Max-sum_AD as well. Using the messages of the Max-sum_AD algorithm we can dynamically generate a spanning tree of the factor graph and then run the algorithm on this tree in both directions to obtain the optimal solution.

In order to generate a spanning tree of the factor graph we need to identify cycles in the factor graph. To this end, we can carry the path (list of nodes) in a single direction on the algorithm's messages. When node n receives two messages in a single direction from two different neighbors that indicate that a node n' contributed to the costs calculation of both messages, it detects a cycle. By removing one of the edges through which it received these messages the agent can eliminate the cycle. If all cycles are eliminated, the result will be a spanning tree that was found in linear time. Running Max-sum_AD in both directions on the generated tree and then using value propagation in an additional iteration would give us the optimal assignment for this tree structured graph. Accounting for removed edges as in [14] would result in a bounded approximation.

We note that the method described above is naive in its selection of removed edges and therefore is expected to give worse bounds than the bounds found by [14]. We leave for future work the in-

vestigation of the information that can be accumulated in the alternating process of Max-sum_AD that can result in producing tighter bounds.

9. 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 when the algorithm is performed in a single direction, it converges after a linear number of iterations. After performing a linear number of iterations in each direction the algorithm converges to a high quality solution after considering all of the problem's constraints. If we keep alternating directions the algorithm converges to states that are not necessarily monotonically improving. In fact, our empirical results reveal that after the second direction change, the algorithm explores states of lower quality.

We further identify a possible reason for this behavior of the algorithm. We demonstrate that costs calculated by the algorithm propagated and used for selecting value assignments by agents are often considering different value assignment for the same variable and thus are inconsistent. In order to overcome this shortcoming of the algorithm we propose the use value propagation. To validate that we propagate values with high quality we begin the value propagation phase after the algorithm has converged for the second time and considered all the problem's constraints.

Our empirical study shows the advantage of the combination of Max-sum_AD with value propagation over previous versions of the Max-sum algorithm. Value propagation allows the algorithm to monotonically improve the solutions it finds in each direction until it converges to a solution with much higher quality than the other version of Max-sum find. On small problems for which we were able to find the optimal solution using a complete algorithm, Max-sum_AD_VP found solutions that approximate the optimal solution by a factor of roughly 1.1 on average.

Acknowledgment: We thank Arnon Netzer and Or Peri for producing the results of the complete algorithm and for their advice on how to improve the paper.

10. REFERENCES

[1] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] I. Brito, A. Meisels, P. Meseguer, and R. Zivan. Distributed constraint satisfaction with partially known constraints. *Constraints*, 14(2):199–234, 2009.

[3] I. Brito and P. Meseguer. Improving dpop with function filtering. In *AAMAS*, pages 141–148, 2010.

[4] 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.

[5] A. Gershman, A. Meisels, and R. Zivan. Asynchronous forward bounding. *J. of Artificial Intelligence Research*, 34:25–46, 2009.

[6] C. Kiekintveld, Z. Yin, A. Kumar, and M. Tambe. Asynchronous algorithms for approximate distributed constraint optimization with quality bounds. In *AAMAS*, pages 133–140, 2010.

[7] F. R. Kschischang and B. J. F. andH. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 47:2:181–208, Febuary 2001.

[8] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Intelligence*, 159:1–26, 2004.

[9] 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.

[10] 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.

[11] J. P. Pearce and M. Tambe. Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In *IJCAI*, Hyderabad, India, January 2007.

[12] A. Petcu and B. Faltings. A scalable method for multiagent constraint optimization. In *IJCAI*, pages 266–271, 2005.

[13] S. D. Ramchurn, A. Farinelli, K. S. Macarthur, and N. R. Jennings. Decentralized coordination in robocup rescue. *Comput. J.*, 53(9):1447–1461, 2010.

[14] A. Rogers, A. Farinelli, R. Stranders, and N. R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artif. Intell.*, 175(2):730–759, 2011.

[15] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of continuously valued control parameters using the max-sum algorithm. In *AAMAS (1)*, pages 601–608, 2009.

[16] 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.

[17] 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.

[18] M. Vinyals, J. A. Rodríguez-Aguilar, and J. Cerquides. Constructing a unifying theory of dynamic programming dcop algorithms via the generalized distributive law. *Autonomous Agents and Multi-Agent Systems*, 22(3):439–464, 2011.

[19] M. Vinyals, E. Shieh, J. Cerquides, J. A. Rodriguez-Aguilar, Z. Yin, M. Tambe, and E. Bowring. Quality guarantees for region optimal dcop algorithms. In *Proc. of 10th Int. Conf. on Autonomous Agentsand Multiagent Systems (AAMAS 2011)*, pages 133–140, Tapei, 2011.

[20] W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. *J. Artif. Intell. Res. (JAIR)*, 38:85–133, 2010.

[21] 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.