Bounded Approximate Decentralised Coordination using the Max-Sum Algorithm

Alessandro Farinelli, Alex Rogers, and Nick R. Jennings

School of Electronics and Computer Science,
University of Southampton, Southampton, SO17 1BJ, UK.
{af2,acr,nrj}@ecs.soton.ac.uk

Abstract. In this paper we propose a novel algorithm that provides bounded approximate solutions for decentralised coordination problems. Our approach removes cycles in any general constraint network by eliminating dependencies between functions and variables which have the least impact on the solution quality. It uses the max-sum algorithm to optimally solve the resulting tree structured constraint network, providing a bounded approximation specific to the particular problem instance. We formally prove that our algorithm provides a bounded approximation of the original problem and we present an empirical evaluation in a synthetic scenario. This shows that the approximate solutions that our algorithm provides are typically within 95% of the optimum and the approximation ratio that our algorithm provides is typically 1.23.

1 Introduction

The development of decentralised coordination techniques is a key issue for many widely studied practical problems, such as resource and task allocation, planning and scheduling. Recently, significant research effort has sought to apply these techniques to control physical devices which are able to acquire information from the environment. For example, agent-based techniques have been used to control the orientation of multiple fixed sensors deployed to localise and track a target [1] and to coordinate sensing and communication in a sensor network deployed to collect environmental data [2]. Decentralised coordination is particularly challenging in these domains because of the constrained computational resources of the devices (due to the requirement of minimising power consumption) and because communication is limited to local neighbours (due to the use of low power wireless communication).

The problem of decentralised coordination in these domains is often cast as a multi-agent Distributed Constraint Optimisation Problem (DCOP), thus allowing the use of a wide range of solution techniques explicitly developed for distributed agent-based systems. Such techniques can be broadly divided into complete algorithms that generate optimal solutions such as ADOPT [3], OptAPO [4], and DPOP [5]; and approximate algorithms such as the Distributed Stochastic Algorithm (DSA) [1] and Maximum Gain Message [6]. Now, while complete algorithms guarantee that they will return the optimum solution, they also exhibit an exponentially increasing coordination overhead (either in the size and/or number of messages exchanged, or in the computation required by each device) as the number of devices in the system increases. Thus, their use in practical applications such as those mentioned above is severely limited. This important issue is partially addressed by extensions of the mentioned approaches. For example, MB-DPOP provides a memory bounded algorithm that trades-off the linear message number of DPOP with polynomial message size [7]. However, while this approach is an improvement, to guarantee optimality of the solution, the overall time and message complexity is still exponential.

In contrast, approximate algorithms require very little local computation and communication, and are, as such, well suited for large scale practical distributed applications in which the optimality of the solution can be sacrificed in favour of computational and communication efficiency. Furthermore, such approximate techniques have been shown to provide solutions which are very close to optimality in several problem instances [1, 6]. However, such approaches fail to provide any guarantees on the solution quality in general settings, and this limits their applicability in many applications domains (particularly safety critical ones).

It is this shortcoming that we address in this paper. Specifically, we propose a novel decentralised coordination algorithm that provides *bounded* approximate solutions. Our point of departure is recent work demonstrating that the max-sum algorithm is a very promising technique for decentralised coordination (and, more generally, constraint reasoning), providing solutions close to optimality while requiring very limited communication overhead and computation [8]. The max-sum algorithm belongs to the Generalised Distributive Law (GDL) framework [9], a family of techniques frequently used in information theory for decoding error correcting codes[1] [10] and to solve graphical models (e.g., to find the maximum a posteriori assignment in Markov random fields [11]). When applied to constraint networks that are trees, the max-sum algorithm is able to provide the optimal solution to the optimisation problem. However, when applied to general constraint networks which typically contain loops, only limited theoretical results hold for the solution quality. While empirical evidence shows that the algorithm is able to find solutions which are very close to the optimal in general problems, there is no guarantee that the algorithms will converge to a solution, and only very limited guarantees on the quality of the solution to which it might converge.

Thus, against this background, in this paper, we build on the existing max-sum algorithm and propose a new algorithm that provides bounded approximate solutions on general constraint networks. We do so by removing cycles in the original constraint network by ignoring dependencies between functions and variables which have the least impact on the solution quality. We then use max-sum to optimally solve the resulting tree structured constraint network, whilst simultaneously computing the approximation ratio for the original problem instance (i.e., the ratio between the unknown optimal solution and the approximate solution provided by our algorithm [12]). Our approach maintains the attractive properties of the max-sum techniques (i.e., low communication overhead and low computational requirement) while providing guarantees on the solution quality provided. The use of tree structures to obtained an approximation of the original problem shares similarities with previous work in information theory [13] where a dependence tree is used to approximate a generic joint probability distribution

[1] The turbo codes are probably the most important representative for which GDL techniques are used.

of random discrete variables. In particular, the authors show that a maximum weight dependence tree provides the best tree approximation of the joint probability distribution. With respect to that work our contribution addresses a decentralised decision problem as opposed to a centralised tree parametrisation of an unknown joint probability. Consequentely, we provide the approximation ratio for our optimisation problem and we consider generic n-ary relationships among variables as opposed to the binary dependence considered in [13]. Also, techniques based on tree-decomposition, have been previously used in the area of constraint optimisation. In particular, in [14] the authors focus on providing bounds on the best-cost extension of a set of variables (i.e., the best value that the target function can achieve for all the possible joint values of the variable set), given a tree-decomposition[2]. In contrast, here we focus on removing cycles from the original problem instance to optimise the approximation ratio. Thus, in more detail, this paper makes the following contributions:

1. We propose a novel weighting for each edge of the original loopy constraint graph. This characterises the maximum effect that the removal of this edge can have on the optimal value of the function to which it was connected. We formally prove that, if we remove edges to create a tree structured constraint network, our algorithm can then compute the approximation ratio for the original problem instance.
2. We present a fully decentralised algorithm (building on Gallager *et al.*'s algorithm for finding minimum spanning trees [15]) that forms a tree structured constraint network by removing those edges with the minimum total weighting (hence minimising the approximation ratio calculated above). The algorithm then initiates max-sum on the resulting tree structured constraint network and distributes the elements required to compute the approximation ratio to all nodes.
3. We empirically evaluate our approach in a synthetic scenario analysing the solution and approximation ratio obtained in various operative conditions. We show that the approximate solutions that our algorithm provides are typically within 95% of the optimum and the approximation ratio that our algorithm provides is typically 1.23.

The rest of this paper is structured as follows: Section 2 formally defines the problem we address. Section 3 provides a brief description of the max-sum algorithm. Section 4 presents our approach and proves that it provides a bounded approximate solution. Section 5 empirically evaluates our approach and Section 6 concludes.

2 Problem Formulation

Following the standard DCOP formulation, our decentralised coordination problem is defined by a set of discrete variables $\mathbf{x} = \{x_1, \ldots, x_m\}$, which are controlled by a set of agents $\mathbf{A} = \{\mathcal{A}_1, \ldots, \mathcal{A}_k\}$, and a set of functions $\mathbf{F} = \{F_1, \ldots, F_n\}$. Each variable x_i can take values over a finite domain $\mathbf{d}_i$ and each function $F_i(\mathbf{x_i})$ is dependent on a subset of variables $\mathbf{x_i} \subseteq \mathbf{x}$ defining the relationship among the variables in $\mathbf{x_i}$. Thus,

[2] Notice that a tree-decomposition for a Constraint Optimisation Problem is not a spanning tree of the original graph, but a tree that has clusters of variables as vertices, and that satisfies the *running intersection* property. See [14] for further details.

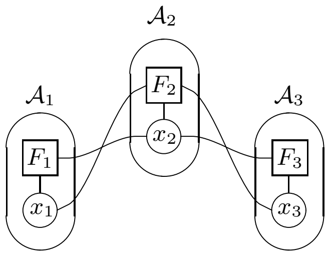

Fig. 1. Diagram showing cyclic factor graph.

function $F_i(\mathbf{x_i})$ denotes the value for each possible assignment of the variables in $\mathbf{x_i}$. Note that this setting is not limited to pairwise (binary) constraints and the functions may depend on any number of variables.

Within this setting, we wish to find the state of each variable, $\mathbf{x}^*$, such that the sum of all functions in the system is maximised:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{i=1}^{n} F_i(\mathbf{x}_i) \tag{1}$$

Furthermore, in order to enforce a truly decentralised solution, we assume that each agent can control only its local variable(s) and has knowledge of, and can directly communicate with, a few neighbouring agents. Two agents are neighbours if there is a relationship connecting variables that the agents control. In this way, the complexity of the calculation that the agent performs depends only on the number of neighbours that it has (and not the total size of the network), and thus, we can achieve solutions that scale well.

3 Basics of the Max-Sum Algorithm

In order to apply max-sum to the optimisation problem described in Equation 1, we represent it as a bipartite factor graph[3]. For example, Figure 1 shows three interacting agents, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$. Variables represent actions that agents can execute, while functions assign utility values for all possible configurations of the variables they depend on, thus describing agent interactions. In general, each agent can be responsible for assigning values to a set of variables, and to perform computations associated to a set of functions. In the Figure, for ease of presentation, we report a situation where each agent is responsible to assign a single variable and to perform computation for a single function. In the example we have that $\mathbf{x}_1 = \{x_1, x_2\}$, $\mathbf{x}_2 = \{x_1, x_2, x_3\}$ and $\mathbf{x}_3 = \{x_2, x_3\}$. Notice that $F_2(\mathbf{x}_2)$ describes a not pairwise interaction. The max-sum algorithm then operates directly on the factor graph representation described above,

[3] From this point onwards, we shall use the terms 'factor graph' and 'constraint network' interchangeably, and note that, agents are responsible for computing and relaying messages of the function and variable nodes that they control.

and does so by specifying the messages that should be passed from variable to function nodes, and from function nodes to variable nodes. These messages are defined as:

– **From variable to function:**

$$q_{i \to j}(x_i) = \alpha_{ij} + \sum_{k \in \mathcal{M}_i \setminus j} r_{k \to i}(x_i) \tag{2}$$

where $\mathcal{M}_i$ is a vector of function indexes, indicating which function nodes are connected to variable node i, and α_{ij} is a normalising constant to prevent the messages from increasing endlessly in cyclic graphs.

– **From function to variable:**

$$r_{j \to i}(x_i) = \max_{\mathbf{x}_j \setminus i} \left[F_j(\mathbf{x}_j) + \sum_{k \in \mathcal{N}_j \setminus i} q_{k \to j}(x_k) \right] \tag{3}$$

where $\mathcal{N}_j$ is a vector of variable indexes, indicating which variable nodes are connected to function node j and $\mathbf{x}_j \backslash i \equiv \{x_k : k \in \mathcal{N}_j \setminus i\}$.

When the factor graph is cycle free, the algorithm is guaranteed to converge to the global optimal solution such that it finds the variable assignment that maximises the sum of the functions. Thus, optimally solving the optimisation problem shown in equation 1. Furthermore, this convergence can be achieved in time equal to twice the depth of the tree by propagating messages from the leaf nodes of the tree to the root and back again. This variable assignment is found by locally calculating the function, $z_i(x_i)$, from the messages flowing into each variable node:

$$z_i(x_i) = \sum_{j \in \mathcal{M}_i} r_{j \to i}(x_i) \tag{4}$$

and hence finding $\arg\max_{x_i} z_i(x_i)$.

When applied to cyclic graphs, the messages within the graph may converge after multiple iterations, but there is no guarantee of this. However, extensive empirical evidence demonstrates that this family of algorithms does in fact generate good approximate solutions when applied to cyclic graphs in this way [16]. When the algorithm does converge, it does not converge to a simple local maximum, but rather, to a neighbourhood maximum that is guaranteed to be greater than all other maxima within a particular large region of the search space [11]. Characterising this region is an ongoing area of research and to date has only considered small graphs with specific topologies (e.g., several researchers have focused on the analysis of the algorithm's convergence in graphs containing just a single loop [11]).

The max-sum algorithm is extremely attractive for the decentralised coordination of computationally and communication constrained devices since the messages are small (they scale with the domain of the variables), the number of messages exchanged typically varies linearly with the number of agents within the system, and the computational complexity of the algorithm scales exponential with just the number of variables on which each function depends (and this is typically much less than the total number of variables) [8]. However, as with the stochastic approaches mentioned earlier, the lack of guaranteed convergence and guaranteed solution quality, limits the use of the standard max-sum algorithm in many applications domains.

A possible solution to address this problem is to remove cycles from the constraint graph by arranging it into tree-like structures such as junction trees [17] or pseudo-trees [5]. However, such arrangements result in an exponential element in the computation of the solution or in the communication overhead (e.g., in DPOP the message size is exponential with respect to the width of the pseudotree). The exponential element is unavoidable to guarantee optimality of the solution and is tied to the combinatorial nature of the optimisation problem.

In the next section we present our alternative approach that ensures the convergence of the algorithm to a bounded approximate solution.

4 The Bounded Max-Sum Algorithm

The basic idea of our approach is to remove cycles from the factor graph, by ignoring some of the dependencies between functions and variables. A dependency directly corresponds to a link between a function node and a variable node in the factor graph, and by removing appropriate dependencies, we can operate max-sum on a cycle free factor graph, hence guaranteeing that the algorithm will converge to the optimal solution of this new problem. Moreover, the size of each message exchanged during this phase will be proportional only to the size of the domain of the variables involved, thus avoiding the exponentially sized messages that are typical of complete algorithms (e.g., DPOP).

Since we ignore some of the dependencies in the factor graph, we cannot guarantee that the solution we obtain in the cycle free factor graph is the optimal one to our original problem. However, as we will show shortly, we can bound the distance of the solution we find on the cycle free factor graph to the optimal solution on the original problem. A key step in this approach is to choose which dependencies to ignore by considering the impact that each has on solution quality.

Specifically, consider a factor graph $FG(\mathbf{x}, \mathbf{F}; E)$ where E is the set of links connecting function and variable nodes. To provide an approximation algorithm, our goal is to compute a variable assignment $\tilde{\mathbf{x}}$ over a spanning tree for the graph FG, such that the $V^* <= \rho(FG)\tilde{V}$, where our approximate solution $\tilde{V} = \sum_i F_i(\tilde{\mathbf{x}}_i)$ and the optimal solution $V^* = \sum_i F_i(\mathbf{x}_i^*)$. Notice that in our approach the approximation ratio is dependent on the particular instance of the problem. We indicate a dependency link with $e_{ij} \in E$ where i is an index over functions and j is an index over variables. Figure 2 reports the factor graph shown in Figure 1 with the weights and a possible spanning tree, solid lines represent links present in the spanning tree while dashed lines represent links present in the original cyclic factor graph, but removed to form the spanning tree. This Figure will be used as a running example to clarify the key steps of the approach. Specifically, our approach proceeds as follows:

1. We define the weight of each dependency link e_{ij} as:

$$w_{ij} = \max_{\mathbf{x}_i \setminus j} \left[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i) \right] \tag{5}$$

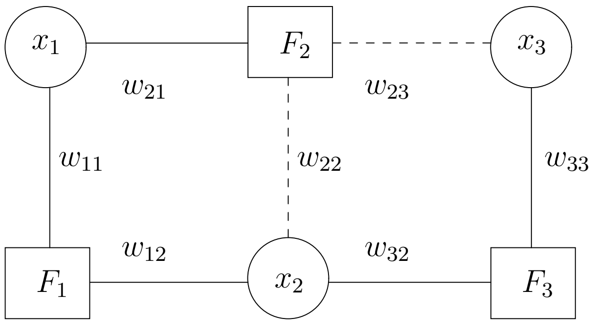

Fig. 2. Example of a factor graph containing cycles and a spanning tree for the factor graph.

For example, w_{23} reported in Figure 2 is computed as

$$w_{23} = \max_{x_1,x_2}\left[\max_{x_3} F_2(x_1,x_2,x_3) - \min_{x_3} F_2(x_1,x_2,x_3)\right]$$

Notice that the weight w_{ij} represents the maximum impact that variable x_j can have over the values of function F_i. In particular, if we ignore variable x_j when maximising F_i the distance between our solution and the optimal will be at most w_{ij}. Thus, the smaller the weight the less important is the dependency in the optimisation process.

2. We remove dependency links from the original cyclic factor graph to form a tree structured graph. For example, in Figure 2 dashed lines represent dependencies that have been removed. For each function within the factor graph, we now have $\mathbf{x}_i = \mathbf{x}_i^t \cup \mathbf{x}_i^c$ where $\mathbf{x}_i^t$ represents the set of dependencies which have not been removed and $\mathbf{x}_i^c$ represents those that have. For example, in Figure 2 $\mathbf{x}_2^t = \{x_1\}$ and $\mathbf{x}_2^c = \{x_2, x_3\}$. Notice that $\mathbf{x}_i^c$ might be empty because no dependency was removed for function i, as it is the case in our running example for $\mathbf{x}_1^c$ and $\mathbf{x}_3^c$ because no dependency was removed for functions F_1 and F_3. However, $\mathbf{x}_i^t$ will always contain at least one element, this follows from the fact that we build a spanning tree of the original factor graph and thus we do not disconnect any element. Consequently, we have that $\cup_i \mathbf{x}_i^t = \mathbf{x}$. We define the sum of the weights of the removed links as:

$$W = \sum_{e_{ij} \in C} w_{ij} \tag{6}$$

where C is the set of links removed from the factor graph.

3. We run the max-sum algorithm on the remaining tree structured factor graph. For functions which have had dependency links removed, we evaluate them by minimising over all values of $\mathbf{x}_i^c$, and thus, the max-sum algorithm optimally solves:

$$\tilde{\mathbf{x}} = \arg\max_{\mathbf{x}} \sum_i \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i) \tag{7}$$

For example, in our case the assignment we obtain after running the max-sum on the spanning tree maximises the function $F_1(x_1, x_2) + F'_2(x_1) + F_3(x_2, x_3)$ where $F'_2(x_1) = \min_{x_2,x_3} F_2(x_1, x_2, x_3)$.

4. The resulting variable assignment, $\tilde{\mathbf{x}}$, represents our approximate solution to the original optimisation problem. As required:

$$V^* <= \rho(FG)\tilde{V} \tag{8}$$

Where the approximation ratio $\rho(FG) = 1 + (\tilde{V}^m + W - \tilde{V})/\tilde{V}$, and $\tilde{V}^m = \sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i)$ represents the optimal solution to the tree structured constraint network. Note that, by removing those dependencies that minimise W, we minimise the approximation ratio[4].

In the next section we formally prove that given the way we compute the weights on the factor graph, equation 8 holds.

4.1 Proof of Bounded Approximation

Since $\rho(FG)\tilde{V} = \sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i) + W$, to show that equation 8 holds it is sufficient to prove the following theorem:

Theorem 1. *Bounded Approximation*

$$\sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i) + W >= \sum_i F_i(x_i^*) \tag{9}$$

To prove this theorem we consider the following property:

Property 1.

$$\forall i, \mathbf{x} \ \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) + \sum_j w_{i,j} >= \max_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) \tag{10}$$

Proof (Proof of property 1). To show that property 1 holds let us first consider the case where $\mathbf{x}_i^c = \{x_j\}$. In this case we have

$$\min_{x_j} F_i(\mathbf{x}_i^t; x_j) + \max_{\mathbf{x}_i^t}[\max_{x_j} F_i(\mathbf{x}_i^t; x_j) - \min_{x_j} F_i(\mathbf{x}_i^t; x_j)] \geq \max_{x_j} F_i(\mathbf{x}_i^t; x_j)$$

By contradiction, let us consider an assignment ${\mathbf{x}''}_i^t$ such that

$$\min_{x_j} F_i({\mathbf{x}''}_i^t; x_j) + \max_{\mathbf{x}_i^t}[\max_{x_j} F_i(\mathbf{x}_i^t; x_j) - \min_{x_j} F_i(\mathbf{x}_i^t; x_j)] < \max_{x_j} F_i({\mathbf{x}''}_i^t; x_j)$$

We can rewrite the previous expression as

$$\max_{x_j} F_i({\mathbf{x}'}_i^t; x_j) - \min_{x_j} F_i({\mathbf{x}'}_i^t; x_j) < \max_{x_j} F_i({\mathbf{x}''}_i^t; x_j) - \min_{x_j} F_i({\mathbf{x}''}_i^t; x_j)$$

[4] In section 4.2 we describe how we use a decentralised maximum spanning tree algorithm to do so.

where

$$\mathbf{x'}^t_i = \arg\max_{\mathbf{x}^t_i}[\max_{x_j} F_i(\mathbf{x}^t_i; x_j) - \min_{x_j} F_i(\mathbf{x}^t_i; x_j)]$$

However, this is a contradiction with respect to the definition of $\mathbf{x'}^t_i$. Therefore property 1 must hold when $\mathbf{x}^c_i = \{x_j\}$.

To prove that property 1 holds also when $|\mathbf{x}^c_i| > 1$ it is sufficient to show that

$$\max_{\mathbf{x}^t_i}[\max_{\mathbf{x}^c_i} F_i(\mathbf{x}^t_i; \mathbf{x}^c_i) - \min_{\mathbf{x}^c_i} F_i(\mathbf{x}^t_i; \mathbf{x}^c_i)] \leq \sum_j \max_{\mathbf{x}_i \backslash j}[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i)]$$

Notice that we can substitute the left term of this expression with

$$\max_{\mathbf{x}^t_i}[\max_j\{\max_{\mathbf{x}^c_i \backslash j}[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i)]\}]$$

However, this term is less than or equal to

$$\max_{\mathbf{x}^t_i}[\sum_j \max_{\mathbf{x}^c_i \backslash j}[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i)]]$$

Which, in turn, is less than or equal to the righthand side of our original expression. Hence property 1 holds for any vector of variables $\mathbf{x}^c_i$.

Proof (Proof of Theorem 1). We can write

$$\sum_i \min_{\mathbf{x}^c_i} F_i(\tilde{\mathbf{x}}_i) + W >= \sum_i \min_{\mathbf{x}^c_i} F_i(\mathbf{x}^*_i) + W$$

This equation holds because we know that

$$\sum_i \min_{\mathbf{x}^c_i} F_i(\tilde{\mathbf{x}}_i) >= \sum_i \min_{\mathbf{x}^c_i} F_i(\mathbf{x}^*_i)$$

holds from the definition of $\tilde{\mathbf{x}}$ and we add the same quantity W to both terms of the equation. Then using property 1 we know that

$$\sum_i \min_{\mathbf{x}^c_i} F_i(\mathbf{x}^*_i) + W >= \sum_i \max_{\mathbf{x}^c_i} F_i(\mathbf{x}^*_i)$$

Now, since

$$\sum_i \max_{\mathbf{x}^c_i} F_i(\mathbf{x}^*_i) >= \sum_i F_i(x^*_i)$$

equation 9 holds.

Note that when the interactions are pairwise and thus at most one dependency is removed from each function node, then, by minimising the sum of the weights, we minimise the impact that this removal has. In general, when multiple dependencies may be removed from any function node, this is no longer the case. For example, consider Figure 2, and suppose the spanning tree is a maximum spanning tree. This implies that w_{23}

and w_{22} are the dependencies, with the minimum total weights, that need to be removed in order to form a spanning tree. However, in this case the possible impact of the removed dependencies on the solution quality will be $\max_{x_1}[\max_{x_2,x_3} F_2(x_1, x_2, x_3) - \min_{x_2,x_3} F_2(x_1, x_2, x_3)]$ which in general is different from $W = w_{22} + w_{23}$. Therefore, when interactions are not pairwise, there might be a combination of dependencies to remove, that has a smaller impact than the W we compute. While it is possible to calculate the impact that removing multiple dependencies has, finding the set that must be removed in order to minimise this impact is a combinatorial problem. Nonetheless, our approach of summing the individual weights overestimates this impact, and thus, our bounded approximate solution is still valid in these cases.

4.2 Decentralised Bounded Max-Sum

Having described our approach, and proved that, given a problem instance, we can provide a bounded approximated solution, we now describe a decentralised implementation of our bounded max-sum algorithm. This implementation has two key steps: (i) forming the spanning tree factor graph which minimises the sum of the weights of the removed edges (hence minimising W), and (ii) initiating the max-sum algorithm and propagating the information required to compute the approximation ration to the agents.

Spanning Tree Formation In order to remove cycles from the given factor graph, FG, we must find a spanning tree that minimises the sum of the weights of the removed edges. To do this, we use the weights of each edge to compute a maximum spanning tree, T. The computation of the maximum spanning tree can be done in a distributed fashion using various message passing algorithms. In particular, here we use the minimum spanning tree algorithm by Gallager, Humblet and Spira (GHS), modified to find the maximum spanning tree [15]. This is a distributed, asynchronous algorithm, for general, undirected graphs[5]. GHS is optimal in terms of communication cost $O(nlogn+E)$ and has a running time of $O(nlogn)$, where n is the number of nodes in the factor graph.

We briefly describe the GHS algorithm here and refer to [15] for a more complete description. Initially, each node (which may be either a variable or a function node) is a *fragment* with level $L = 0$, then each node chooses its maximum weight outgoing edge and attempts to join with the node at the other end. This forms a fragment of level $L = 1$. Nodes in fragments where $L > 0$ co-operate to determine the fragment's maximum weight outgoing edge that will not form a cycle and attempt to join with the fragment on the other end. This occurs by each node finding its maximum weight outgoing edge, and passing this information to a core node, which can then determine the best edge for the whole fragment. Fragments continue to join together in this manner. The two *core nodes* (those at either end of the edge on which the final joining of fragments occurs) are aware when the algorithm terminates, as they will receive reports from each node that they cannot locate any further outgoing edges that will not lead to a cycle.

[5] Notice that our approach is completely generic with respect to the algorithm used to compute the maximum spanning tree. Here the choice of the GHS algorithm is dictated by the low communication overhead and by the ease of implementation.

Max-Sum Initiation & Information Propagation On termination of the GHS algorithm described above, only the two core nodes are aware that the algorithm has completed. Therefore we add a message-passing phase to propagate this information throughout the tree. This procedure also establishes a parent-child hierarchy in the tree, and serves to initiate the max-sum algorithm and information propagation stages. This message-passing phase is initiated by the root node; a role adopted by whichever of the two core nodes is a function node[6]. This root node sends out a `COMPLETE` message to each of its children. When a node receives a `COMPLETE` message, it marks the sender as its parent, and then propagates it down the tree.

When a leaf node receives the `COMPLETE` message the max-sum phase starts. Each node propagates `MAXSUM` messages up the tree, waiting for messages from each child node before sending an updated message to the parent node. The content of the messages are calculated as described in equations 2 and 3, and convergence of the messages to the optimum is guaranteed when the messages have propagated to the root node, and back to the leaf nodes[7]. At this stage, each variable node is aware of both the variable assignment, $\tilde{\mathbf{x}}_i$, that represents the approximate solution to the original optimisation problem, and the value of $\tilde{V}^m = \sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i)$; this is provided directly from the max-sum algorithm and used to calculate $\rho(FG)$.

When the leaf nodes receive this final `MAXSUM` message the weight and solution propagation phase starts. During this phase, nodes propagate tuples composed of `WSUM` (which will accumulate the value of W specified in equation 6) and `SOLUTION` messages[8]. If the leaf is a variable node it creates an empty `WSUM` and an empty `SOLUTION` message. If it is a function node, it creates a `WSUM` message of value equal to the sum of the weights of local deleted edges, and a `SOLUTION` message equal to $F_i(\tilde{\mathbf{x}}_i)$. Both messages are then propagated up the tree, with each internal node waiting to receive messages from all its children before propagating a single new `WSUM` and `SOLUTION` message to its parent. If the internal node is a variable node, then these new messages are simply the sum of the messages from its children. If it is a function node, then they are given by the sum of the messages from its children plus the sum of the weights of its own locally deleted edges, and the value of $F_i(\tilde{\mathbf{x}}_i)$, respectively. When the root has received all the `WSUM` and `SOLUTION` messages, both are propagated back down the tree, informing each node of the total W, and the final solution, $\tilde{V} = \sum_i F_i(\tilde{\mathbf{x}}_i)$.

At this final stage, each agent knows the assignment of the variables that it controls, it knows that this assignment leads to a total solution quality of $\tilde{V}$, and that this solution has an approximation ratio $\rho(FG) = 1 + (\tilde{V}^m + W - \tilde{V})/\tilde{V}$. The number of messages, for each information propagation phase, equals the number of edges in the spanning tree (i.e., $|\mathbf{F}| + |\mathbf{x}| - 1$). Thus, the size of each message depends on the message type, but this is always constant with respect to the number of nodes in the factor graph

[6] Notice that, in our case, one of the two core nodes will always be a function node because the factor graph is a bipartite graph.

[7] In settings where the choice of variable assignment may not be unique (most commonly, when the functions return integer values) an addition value propagation phase may be used at this point. See [18] for details.

[8] Note that these could be propagated in two separate phases, but here we combine them together for efficiency.

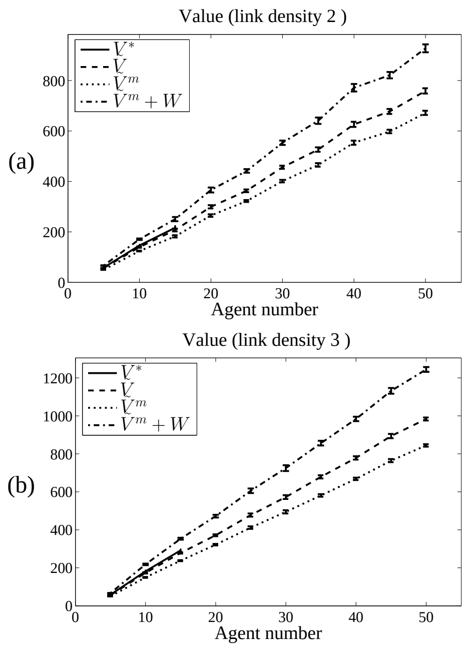

Fig. 3. Graphs showing the utility when varying the number of agents and the link density.

(e.g., a `MAXSUM` message involving variable x_i contains $|\mathbf{d}_i|$ values while `WSUM` and `SOLUTION` messages contain one value each).

5 Empirical Evaluation

We now present an empirical evaluation of our bounded approximate algorithm. This is required because our approximation ratio depends on the specific problem instance. Specifically, we evaluated our approach in a decentralised coordination problem where a set of agents is arranged in a random graph. Each agent controls one variable, with domain $|\mathbf{d}_i| = 3$, and each edge of the graph represents a pairwise constraint between two agents (and thus our W is minimum in this case). A random payoff matrix is associated with each edge, specifying the payoff that both agents will obtain for every possible combination of their variables' assignments. Each entry of the payoff matrix is a real number sampled from a gamma distribution (with $\alpha = 9$ and $\beta = 2$).

This setting generalises the distributed graph colouring problem, which is a canonical problem frequently used to evaluate DCOP techniques (e.g., [3] and [4]). In the graph colouring domain the W our approach provides would simply be the number of edges removed to remove cycles from the graph. The random payoff matrix that we use here enriches the domain making the evaluation analysis more significant, and the gamma distribution, introduces significant variance such that some dependencies have a higher impact than others.

We performed our experiments by generating random graphs with different link density (i.e., average connection per agents) and various number of agents. For each configuration, we extract several performance metrics:

- $\tilde{V}^m$: The solution obtained by the max-sum algorithm on the tree structured constraint network.
- $\tilde{V}$: Our bounded approximate solution, obtained by evaluating the assignment computed by max-sum on the spanning tree, on the original loopy constraint network.
- $\tilde{V}^m + W$: The upper bound on the value of the unknown optimal solution computed by our approach.
- V^*: The optimal solution.

Figures 3(a) and 3(b) show the results obtained for different values of the link density (specifically, 2 and 3)[9] while varying the number of agents. For each configuration, we report the average value and the standard error in the mean over twenty repetitions. Since the optimal utility is computed by complete enumeration of all the possible configurations, we were able to compute this metric only for smaller numbers of agents (e.g., up to 15).

Results show that the actual utility we compute is extremely close to the optimal solution (in the experiments the minimum ratio was 95%). Thus showing that, from an empirical point of view, the solutions we can achieve using this approach provide very good approximations. More importantly the approximation ratio we guarantee is significant. In the experiments $\rho(FG)$ was never above 1.27, and was typically 1.23. Finally, our approach scales very well with the number of agents, having a running time of approximately 800 milliseconds on the most complex problem instance in our data set (i.e., 50 agents and link density 3)[10].

6 Conclusions and Future Work

We developed a novel algorithm for decentralised coordination, which is able to guarantee bounded approximate solutions given particular instances of general constraint networks. Our main future direction is to investigate techniques to further reduce the approximation ratio. A promising direction is to iteratively apply our algorithm while clustering variable and function nodes (as proposed in [17]) to remove cycles without removing dependencies. In this way, we can iteratively decrease the approximation ratio (by removing less dependencies) while paying an increase in communication and computation (due to clustering of nodes), thus allowing a flexible trade-off between solution quality and communication and computation overhead.

[9] These values are in the range often used for benchmarking DCOP techniques on random graph colouring instances [3].

[10] Consider that, on a 3-color random graph problem with link density 2 and 18 agents, ADOPT requires a running time of 200 seconds while our approach requires approximately 100 milliseconds [19].

References

1. Fitzpatrick, S., Meetrens, L.: Distributed Coordination through Anarchic Optimization. In: Distributed Sensor Networks A multiagent perspective. Kluwer Academic (2003) 257–293
2. Padhy, P., Dash, R.K., Martinez, K., Jennings, N.R.: A utility-based sensing and communication model for a glacial sensor network. In: Proceeding of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'06). (2006) 1353–1360
3. Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence Journal (161) (2005) 149–180
4. Mailler, R., Lesser, V.: Solving distributed constraint optimization problems using cooperative mediation. In: Proceedings of Third International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS 2004). (2004) 438–445
5. Petcu, A., Faltings, B.: DPOP: A scalable method for multiagent constraint optimization. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, (IJCAI 2005). (2005) 266–271
6. Maheswaran, R.J., Pearce, J., Tambe, M.: A family of graphical-game-based algorithms for distributed constraint optimization problems. In: Coordination of Large-Scale Multiagent Systems. Springer-Verlag, Heidelberg Germany (2005) 127–146
7. Petcu, A., Faltings, B.: Mb-dpop: A new memory-bounded algorithm for distributed optimization. In: Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI). (2007) 1452–1457
8. Farinelli, A., Rogers, A., Petcu, A., Jennings, N.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proc. of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS). (2008) 639–646
9. Aji, S., McEliece, R.: The generalized distributive law. Information Theory, IEEE Transactions on **46**(2) (2000) 325–343
10. MacKay, D.J.C.: Information Theory, Inference, and Learning Algorithms. Cambridge University Press (2003)
11. Weiss, Y., Freeman, W.T.: On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. IEEE Transactions on Information Theory **47**(2) (2001) 723–735
12. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms, second edition. The MIT press (2001)
13. Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory **it-14**(3) (May 1968) 462–467
14. Dechter, R., Kask, K., Larrosa, J.: A general scheme for multiple lower bound computation in constraint optimization. In: Constraint Programming. (2001) 346–360
15. Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Trans. Program. Lang. Syst. **5**(1) (1983) 66–77
16. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. Science **315**(5814) (February 2007) 972–976
17. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Transactions on Information Theory **42**(2) (2001) 498–519
18. Wainwright, M., Jaakkola, T., Willsky, A.: Tree consistency and bounds on the performance of the max-product algorithm and its generalizations. Statistics and Computing **14**(2) (2004) 143–166
19. Modi, P.J.: Distributed Constraint Optimization for Multiagent Systems. PhD thesis, Dpt. of Computer Science, Univ. of Southern California (2003)