Bounded Approximate Decentralised Coordination via the Max-Sum Algorithm

A. Rogers*, A. Farinelli*†, R. Stranders* and N. R. Jennings*

*School of Electronics and Computer Science, University of Southampton, Southampton, UK.

†Department of Computer Science, University of Verona, Verona, Italy.

{acr,af2,rs06r,nrj}@ecs.soton.ac.uk

October 28, 2010

Abstract

In this paper we propose a novel approach to decentralised coordination, that is able to efficiently compute solutions with a guaranteed approximation ratio. Our approach is based on a factor graph representation of the constraint network. It builds a tree structure by eliminating dependencies between the functions and variables within the factor graph that have the least impact on solution quality. It then uses the max-sum algorithm to optimally solve the resulting tree structured constraint network, and provides a bounded approximation specific to the particular problem instance. In addition, we present two generic pruning techniques to reduce the amount of computation that agents must perform when using the max-sum algorithm. When this is combined with the above mentioned approximation algorithm, the agents are able to solve decentralised coordination problems that have very large action spaces with a low computation and communication overhead. We empirically evaluate our approach in a mobile sensor domain, where mobile agents are used to monitor and predict the state of spatial phenomena (e.g., temperature or gas concentration). Such sensors need to coordinate their movements with their direct neighbours to maximise the collective information gain, while predicting measurements at unobserved locations. When applied in this domain, our approach is able to provide solutions which are guaranteed to be within 2% of the optimal solution. Moreover, the two pruning techniques are extremely effective in decreasing the computational effort of each agent by reducing the size of the search space by up to 92%.

1 Introduction

Recently, significant research effort has sought to apply coordination techniques to control physical devices that are able to acquire information from the environment. In these settings, *decentralised* coordination (i.e. no central system exists that controls the coordination process, but the devices coordinate amongst themselves) has proved to be a natural, robust and effective approach to organise the activities of the *embedded agents* that control the devices. For example, decentralised coordination 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, 3]. In both of these domains, and many others besides, decentralised coordination is particularly challenging 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).

Given this background, the problem of decentralised coordination in these domains is often cast as a multi-agent distributed constraint optimisation problem (DCOP). In the constraint optimisation framework the aim is to find the assignment of a set of variables that optimises the aggregation of payoffs (or conversely costs) of a set of soft constraints defined over the values of the variables [4]. In a distributed constraint optimisation problem a set of agents control the value of the variables in the system; jointly aiming to optimise the global reward. DCOP techniques can be directly used to address the decentralised coordination problem described above by representing the possible actions that an embedded agent can take with variables and by encoding payoffs (or costs) for taking joint actions with constraints. These DCOP techniques can be broadly divided into two classes: complete algorithms (i.e., algorithms that always find a solution that optimises the global objective function), such as ADOPT [5], OptAPO [6], DPOP [7], NCBB [8] and AFB [9]; and approximate algorithms such as the Distributed Stochastic Algorithm (DSA) [1], Maximum Gain Message (MGM) [10], and ALS_DisCOP [11] that do not.

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 [12]) 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 above mentioned approaches. For example, MB-DPOP provides a memory bounded algorithm that trades-off the linear message number of DPOP with polynomial message size [13]. In addition, BnB-ADOPT is an extension of ADOPT, using a different search strategy (depth first with branch and bound instead of best first) that consistently reduces computation time [14]. However, while these approaches provide important improvements, to guarantee optimality of the solution, the overall time and/or message complexity is still necessarily 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 (see [4] for a review of such algorithms). Furthermore, such approximate techniques, have been shown to provide solutions which are very close to optimality in several problem instances [1, 10]. However, such approaches fail to provide guarantees on the solution quality in general settings. This is particularly troublesome because the quality of solution to which most approximate algorithms converge is highly dependent on many factors which cannot always be properly assessed before deploying the system. Therefore there is no guarantee against particularly negative behaviours of such techniques on specific pathological instances.

To rectify these shortcomings, we believe the answer is to develop approximate algorithms with quality guarantees. Such approaches can address the trade off between solution quality and computation effort while providing a guaranteed lower bound on the quality of the solution obtained with respect to the optimum. Addressing such trade-offs is particularly important in dynamic settings and when the agents have low computational power, which is usually the case for applications involving embedded devices (such as mobile robots or sensor networks). Moreover, having a bound on the quality of the provided solutions is particularly important for safety critical applications (such as disaster response, surveillance, etc.) because a pathological behaviour of the system is, in this case, simply unacceptable.

Now, there has been some work on providing guarantees on the performance of approximate algorithms in the DCOP framework. In particular, Pearce and Tambe use the concept of k-optimal solutions, where a solution is k-optimal if the corresponding value of the objective function cannot be improved by changing the assignment of any k or less variables [15, 16]. Specifically, Pearce and Tambe provide an approximation ratio (i.e., the ratio between the unknown optimal solution and the approximate solution [17]) for k-optimal algorithms which is valid for any DCOP with non-negative reward structure [15]. However, the accuracy of the approximation ratio, in any particular setting, depends on the number of agents, on the arity of the constraint functions and on the value of k. Specifically, the approximation bound is more accurate when k is higher but less accurate when the number of agents in the system grows. Thus, their approach provides a poor approximation bound when the number of agents grows. Moreover, finding a k-optimal solution of higher k requires, in general, exponentially more computation and communication. Better approximation bounds can be provided assuming some *a priori* knowledge on the reward structure. For example, Bowring et al. show that the approximation bounds can be improved by assuming the knowledge of the ratio between the least minimum reward to the maximum reward [16]. In this approach, the bound is significantly improved, and the bound decreases consistently when the number of agents grows. However, we will show in Section 4.3 that the resulting bound is still significantly larger than that produced by our approach.

Data dependent approximation approaches with guarantees have also been investigated. For example, Petcu and Faltings propose an approximate version of DPOP [18], and Yeoh et al. provide a mechanism to trade-off solution quality for computation time for the ADOPT and BnB-ADOPT algorithms [19]. Such mechanisms work by fixing an approximation ratio and reducing computation or communication overhead as much as possible to meet that ratio. While empirical results show that such approaches significantly improve the efficiency of their complete counterparts (i.e., DPOP and BnB-ADOPT), there is no guarantee or bound on the computation time or communication overhead required to achieve the predetermined bound.

Against this background, here we propose a novel decentralised coordination approach that is able to make efficient use of constrained computational and communication resources, while providing accurate *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 [12, 20]. The max-sum algorithm belongs to the Generalised Distributive Law (GDL) framework [21], a family of techniques frequently used in information theory for decoding error correcting codes[1] [22] and to solve graphical models (e.g., to find the maximum a posteriori assignment in Markov random fields [23] or compute the posterior probabilities [24]). 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

[1]The turbo codes are probably the most important representative application for which GDL techniques are used. See [22], chapter 48.4.

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, in this work, we build on the existing max-sum algorithm and propose a new algorithm that provides bounded approximate solutions on general constraint networks with bounded reward functions. We do so by removing cycles in the original constraint network, specifically 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. We note that the same guarantees can be obtained by using any distributed optimization algorithm that runs in linear time on tree-structured network. Thus, the results in this paper pertaining to bounded approximate solutions are not limited to the max-sum algorithm. However, our specific choice of the max-sum algorithm here is driven by its efficiency in terms of low communication overhead (specifically in the number of messages), low computational requirement and ease of decentralisation. Other possible choice yielding the same results in term of efficiency would similar message passing algorithms such as DPOP or the cluster tree elimination algorithm [25]. However, as shown in [26], the GDL framework (of which max-sum is an instance) generalises many optimisation algorithms based on dynamic programming, including both DPOP and cluster tree elimination.

Building on this result, we then go on to show that we can further improve the computational efficiency of our algorithm by reducing the search space that each agent needs to consider. This is important, since many practical problems exhibit search spaces which quickly become intractable even for approximated techniques. In order to achieve this, we develop two generic action pruning algorithms. The first attempts to discard dominated actions of individual agents (i.e. those that can never be part of an optimal solution) before the max-sum algorithm is run (and thus, this approach also generalises to other distributed optimization algorithms). The second uses branch and bound to reduce the space of joint actions that needs to be considered whilst running the max-sum algorithm.

To evaluate the effectiveness of the two algorithms in a realistic application, we consider a disaster response scenario where a set of mobile sensors are tasked to gather information on spatial phenomena, such as temperature or the concentration of potentially toxic chemicals. To predict environmental conditions in parts of the environment that can not be sensed directly, these sensors need to identify and model the spatial and temporal dynamics of the monitored phenomena. Moreover, the sensors need to coordinate their movements to collect the most informative measurements needed to predict these environmental conditions as accurately as possible [27]. This problem is particularly challenging from a coordination standpoint because the sensors need a sophisticated model to represent the complex spatial and temporal correlations of the monitored phenomena (and here we use the Gaussian processes to perform this role), which results in a high computational overhead when evaluating the possible joint actions of the sensors. Moreover, to achieve effective solutions, mobile sensors have to coordinate on paths, rather than single actions, thus dealing with a large search space.[2] Thus, to effectively apply max-sum in a computationally challenging domain, such as the mobile sensors one, we can use these two new pruning algorithms to drastically

[2]A path is a sequence of single actions, thus the number of possible paths grows exponentially with the length of the sequence. However, by coordinating on sequences of actions, robots are able to better predict which are the most informative measurements, and thus coordinate more effectively.

reduce the required number of function evaluations, thus alleviating a major bottleneck of this algorithm for practical applications.

In more detail, this work makes the following contributions to the state of the art:

1. We propose a novel approach for decentralised coordination that provides bounded approximate solutions. This is the first approach to provide guarantees on convergence and solution quality for the max-sum algorithm in a decentralised coordination setting (and as noted, earlier, it is also applicable to other distributed optimization algorithms that run in linear time on tree-structured network). In particular, our approach exploits the fact that we can calculate a weight for each edge of the original loopy constraint graph that characterises the maximum effect that the removal of that 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. Moreover, we present a fully decentralised algorithm (building on Gallager, Humblet and Spira's algorithm for finding minimum spanning trees [28]) 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.

2. We empirically evaluate our bounded approximate approach in a synthetic scenario analysing the solution and approximation ratio obtained in a generalisation of the distributed graph colouring algorithm, which is a canonical problem frequently used to evaluate DCOP techniques (e.g., [5] and [6]). 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, and we show that this is much more accurate than the previous theoretical bound for k-optimal algorithms.

3. We develop two novel, generic pruning techniques to reduce the computational overhead of max-sum when applied to problems with a large action space. The first method attempts to reduce the number of actions that each agent needs to consider *before* running the max-sum algorithm. This algorithm prunes the dominated actions of each agent, which will never be selected by the decentralised coordination procedure, regardless of the actions of other agents. The second technique is based on a branch and bound search, which is performed when computing the joint actions that maximise the utility of the whole system.

4. Finally, we apply the developed decentralised coordination techniques to the mobile sensor domain. We show that our approach is able to provide an effective on-line coordination approach for the mobile sensors. In particular, we empirically show that a coordination algorithm based on max-sum outperforms a greedy non-coordinated algorithm by up to 50% in this domain. Moreover, the use of the bounded approximate algorithm results in solutions that are within 2% of the optimal. At the same time, by applying the two pruning techniques the action space is reduced by 92%, thus significantly reducing the computational overhead.

The rest of this paper is structured as follows: Section 2 formally defines the decentralised coordination problem we address and Section 3 provides a brief outline of the max-sum algorithm. Section 4 presents our approach to provide bounded approximate solutions and Section 5 then details our techniques to speed-up the computation performed by the max-sum approach. Section 6 empirically evaluates our approach in the mobile sensor domain. Section 7 puts our work in perspective with previous approaches and, finally, Section 8 concludes and discusses future work.

2 The Decentralised Coordination Problem

We formulate the decentralised coordination problem we address as a DCOP. Following the standard DCOP formulation, we have 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 represents the possible actions that the controlling agents can execute and can take values over a finite domain $\mathbf{d}_i$. 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, function $F_i(\mathbf{x}_i)$ denotes the value for each possible assignment of the variables in $\mathbf{x}_i$ and represents the joint payoff that the corresponding agents achieve. 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 value of each variable, $\mathbf{x}^*$, such that the sum of all functions in the system is maximised (i.e., social welfare maximisation):

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

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 and functions that the agents control.

3 Basics of the Max-Sum Algorithm

Notation used in this section

- $q_{i\to j}(x_i)$ is the message sent from variable x_i to function F_j.
- α_{ij} is the normalising constant for the message $q_{i\to j}(x_i)$.
- $\mathcal{M}_i$ is set of function indexes, indicating which function nodes are connected to variable node x_i.
- $r_{j\to i}(x_i)$ is the message sent from function F_j to variable x_i.
- $\mathcal{N}_j$ is the set of variable indexes, indicating which variable nodes are connected to function node F_j $\mathbf{x}_j \backslash x_i \equiv \{x_k : k \in \mathcal{N}_j \setminus i\}$.

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

[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.

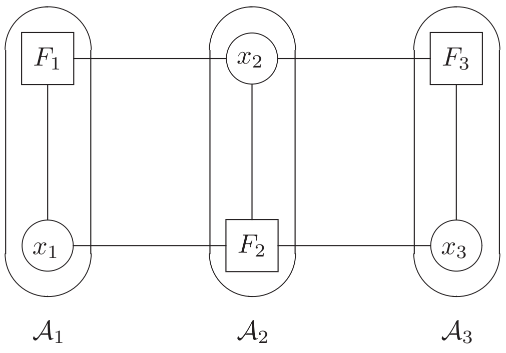


Figure 1: An example factor graph for agents $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$.

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 for performing computations associated to a set of functions. In the figure, for ease of presentation only, we show a situation where each agent is responsible for assigning a single variable and for performing the computation for a single function. However, this is not a requirement for the application of the max-sum algorithm and in general agents can be responsible for a set of variables for the computation of an arbitrary number of functions. In the example, $\mathbf{x}_1 = \{x_1, x_2\}$, $\mathbf{x}_2 = \{x_1, x_2, x_3\}$ and $\mathbf{x}_3 = \{x_2, x_3\}$. Notice that $F_2(\mathbf{x}_2)$ is not a pairwise interaction and in general, there is no requirement that the utility functions should decompose into additive constraints between variables. The max-sum algorithm then operates directly on the factor graph representation described above, 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 set of function indexes, indicating which function nodes are connected to variable node i, and α_{ij} is a normalisation factor (the details of which will be dicussed shortly).

- **From function to variable:**

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

  where $\mathcal{N}_j$ is a set 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, thereby 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. In this case, the optimal variable assignment is found by locally calculating the function, $z_i(x_i)$, once the variable node has received a message from each of its connected function nodes.

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

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. In cyclic graphs, messages are usually normalised to prevent them from increasing endlessly. This is achieved by setting the normalising constant α_{ij} in Equation 2 such that $\sum_{x_i} q_{i \to j}(x_i) = 0$.[4] Extensive empirical evidence demonstrates that, despite the lack of convergence guarantees, the GDL algorithms (e.g., sum-product, max-product, max-sum, etc.) do in fact generate good approximate solutions when applied to cyclic graphs in this way [29]. Interesting results have been obtained for characterising the quality of solutions at convergence. Specifically, for the max-product algorithm[5] it can be shown that when the algorithm converges, 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 [23]. Characterising the properties of these algorithms in terms of convergence and solution quality guarantees is still an ongoing area of research, and to date significant results have been obtained only for graphs with specific topologies (e.g., several researchers have focused on the analysis of the convergence and solution quality in graphs containing just a single loop [30, 31]).

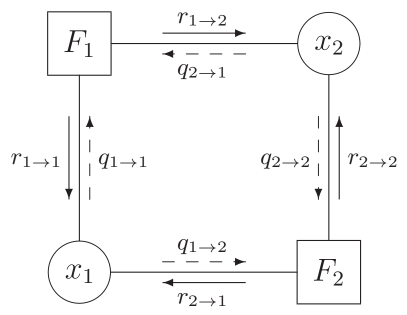
To better explain the operations performed by the max-sum algorithm we now detail an execution example. To make the example easier we consider a simple factor graph composed of two variables and two functions, each variable has a domain composed of three values indicated as r, b, g. Figure 2(a) shows the factor graph in this case, and the max-sum messages for a single iteration. Figure 2(b) shows the table form of the functions and the operations required to compute the exemplar message $r^3_{2 \to 2}(x_2)$, where the superscript indicates the iteration for the message computation. At the first iterations all the q messages are initialised to zero, and therefore the r messages are a maximisation of the sending function over the variable which is not receiving the message (e.g., $r^0_{2 \to 2}(x_2) = max_{x_1}[F_2(x_1, x_2)]$). At each iteration each variable computes its individual z function and chooses the value that maximise it. For this particular example the messages reach a fixed point after just six iterations and the z functions converge to $z_1(x_1) = \{< x_1 = r, 9 >, < x_1 = b, 14 >, < x_1 = g, 4 >\}$ $z_2(x_2) = \{< x_2 = r, 18 >, < x_2 = b, 21 >, < x_2 = g, 18 >\}$[6]. The algorithm would then find the optimal assignment $x_1 = b$ and $x_2 = b$ obtaining a total utility of 12.

[4]Note that this normalisation will fail in the case of a negative infinity utility that represents a hard constraint on the solution. However, it is still possible to use the max-sum algorithm in this context by simply replacing the negative infinity reward with one whose absolute value is greater than the sum of the maximum values of each function. This ensures both that the normalisation works correctly, and that the reward is still sufficiently negative to effectively act as a hard constraint (i.e. there can be no solution that violates this constraint that has a higher utility than one that does not).

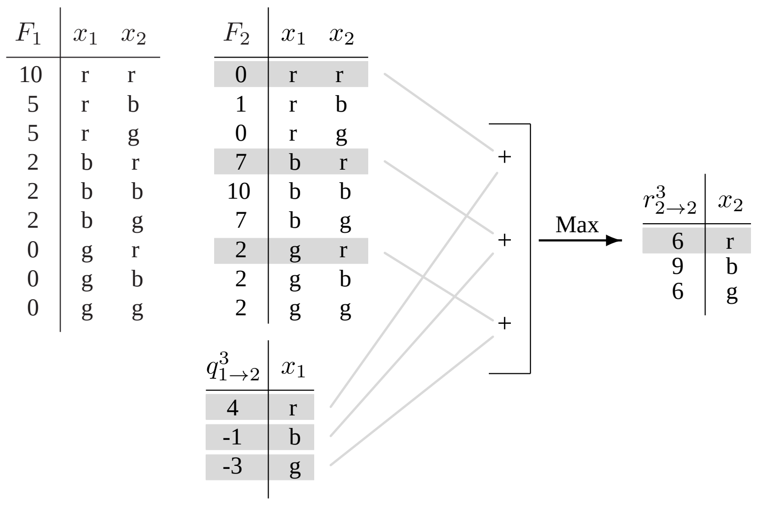
[5]The same results hold for the max-sum algorithm as it can be considered as a derivative of the max-product algorithm when we consider the log domain [12].

[6]For a complete trace of the max-sum algorithm on an exemplar problem, see [12].

(a)



(b)



F_1	x_1	x_2
10	r	r
5	r	b
5	r	g
2	b	r
2	b	b
2	b	g
0	g	r
0	g	b
0	g	g

F_2	x_1	x_2
0	r	r
1	r	b
0	r	g
7	b	r
10	b	b
7	b	g
2	g	r
2	g	b
2	g	g

$q^3_{1\to 2}$	x_1
4	r
-1	b
-3	g

$r^3_{2\to 2}$	x_2
6	r
9	b
6	g

Figure 2: Execution example for the max-sum algorithm showing (a) the factor graph and messages exchanged in each iteration of the algorithm, and (b) the table form of the functions and computation of an exemplar function to variable message.

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 in the system) [12]. However, as with the approximate algorithms mentioned in the introduction, the lack of guaranteed convergence and guaranteed solution quality, limits the use of the standard max-sum algorithm in many application 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 [32] or pseudo-trees [7]. However, such arrangements result in an exponential element in the computation of the solution or in the communication overhead. For example, DPOP is functionally equivalent to performing max-sum over a pseudo-tree formed by depth-first search of the constraint graph, and the resulting maximum message size is exponential with respect to the width of the pseudo tree. This exponential element is unavoidable in order to guarantee optimality of the solution and is tied to the combinatorial nature of the optimisation problem. However, as discussed in the introduction, such exponential behaviour is undesirable in systems composed of devices with constrained computational resources. Thus, 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

Notation used in this section

- $FG(\mathbf{x}, \mathbf{F}; E)$ is a factor graph.
- E is the set of links connecting function and variable nodes in the factor graph.
- $\mathbf{x}^*$ is the optimal variable assignment for the constraint network.
- $\tilde{\mathbf{x}}$ is the optimal variable assignment for the tree structured constraint network.
- $\tilde{V} = \sum_i F_i(\tilde{\mathbf{x}}_i)$ is the approximate solution obtained with the assignment $\tilde{\mathbf{x}}$.
- $V^* = \sum_i F_i(\mathbf{x}_i^*)$ is the optimal solution.
- ρ_{FG} is the approximation ratio.
- $e_{ij} \in E$ are the dependencies links between variables and functions.
- w_{ij} is the weight associated with dependency link e_{ij}.
- $\mathbf{x}_i^t$ is the set of dependent variables for function F_i which will be part of the tree-structured constraint network.
- $\mathbf{x}_i^c$ is the set of dependent variables for function F_i which will not be part of the tree-structured constraint network.
- $B_i(\mathbf{x}_i^c)$ is the maximum impact on the solution for a set of removed dependencies of function F_i.
- $B = \sum_i B_i(\mathbf{x}_i^c)$ is the maximum impact on the solution for a set of removed dependencies.
- $\tilde{V}^m = \sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i)$ is the optimal solution to the tree structured constraint network.
- C is the set of couples of indices $< i, j >$ that identify the edges removed from the factor graph.
- $W = \sum_{<i,j>\in C} w_{ij}$ is the sum of the weights of removed edges.

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 so-

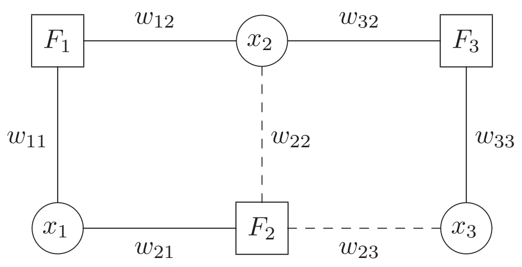


Figure 3: Example of a factor graph containing cycles and a spanning tree formed by removing the edges between the variables nodes x_2 and x_3 and the function node F_2.

lution of this new problem. Moreover, by removing cycles in this way, we do not incur the exponential communication cost that is typical of complete approaches (as discussed above and in the introduction). With our approach, the size of exchanged messages will be proportional only to the size of the domain of the variables involved, as opposed to the exponentially sized messages that are typical of complete algorithms. Also, the amount of computation required to perform the maximisation step when calculating function to variable messages, is exponential only in the number of variables directly involved in the function.[7] If the arity of the functions is bounded (e.g. we have only pairwise interactions) this computation is polynomial.

However, 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 solution to our original problem. Nonetheless, 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 quantify the maximum impact that each dependency 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 $V^* \leq \rho_{FG}\tilde{V}$, where our approximate solution $\tilde{V} = \sum_i F_i(\tilde{\mathbf{x}}_i)$ and the optimal solution $V^* = \sum_i F_i(\mathbf{x}_i^*)$. Note that the approximation ratio ρ_{FG} is dependent on the particular instance of the problem. Thus, instead of bounding the performance of our algorithm on a large class of problems, we compute a data-dependent bound for any specific problem instance. As a result, this bound is tighter than a theoretical bound for a wider class of problems.

The key property of our algorithm is that it puts weights on the dependency links between variables and functions. These weights quantify the maximum impact that removing a dependency may have. In more detail, 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 3 shows the same factor graph in Figure 1 with the weights, and a possible spanning tree, (solid lines represent links present in the spanning tree, and dashed lines represent links that were present in the original cyclic factor graph, but have been removed to

[7]More specifically, when a function F_i which depends on a set of variables $|\mathbf{x}_i| = n$ sends a message to one of its variables x_j the amount of computation required will be d^n, where d is the size of the variables' domain.

form the spanning tree).[8]

Given these concepts, our approach proceeds as follows:

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

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

   For example, w_{23} reported in Figure 3 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 then 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 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 dependent variables which have not been removed and $\mathbf{x}_i^c$ represents those that have. For example, in Figure 3 we have $\mathbf{x}_2^t = \{x_1\}$ 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 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 $\cup_i \mathbf{x}_i^t = \mathbf{x}$.

   Now, given a function F_i we define the maximum impact of a set of removed dependencies as:

$$B_i(\mathbf{x}_i^c) = \begin{cases} \max_{\mathbf{x}_i \setminus \mathbf{x}_i^c} \left[\max_{\mathbf{x}_i^c} F_i(\mathbf{x}_i) - \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i) \right] & \text{if } \mathbf{x}_i^c \neq \emptyset \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

   where $\mathbf{x}_i^c$ is the set of variables removed from the function dependency. By computing $B_i(\mathbf{x}_i^c)$, we are evaluating the maximum impact of all the removed dependencies from a function to form a spanning tree. For example, considering our running example reported in Figure 3 we have:

$$B_2(x_2, x_3) = \max_{x_1} \left[\max_{x_2, x_3} F_i(x_1, x_2, x_3) - \min_{x_2, x_3} F_i(x_1, x_2, x_3) \right]$$

   This represents the maximum impact on the solution quality when both variables x_2 and x_3 are removed. Finally, we define the sum of the maximum impact of removed dependencies from the factor graph as:

$$B = \sum_i B_i(\mathbf{x}_i^c)$$

[8] This figure will be used as a running example to clarify the key steps of the approach.

3. We now 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:

$$\tilde{\mathbf{x}} = \arg\max_{\mathbf{x}} \sum_i F_i'(\mathbf{x}_i^t) = \arg\max_{\mathbf{x}} \sum_i \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i) \tag{7}$$

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, and we shall shortly prove that this approximate solution is within a calculated bound from the optimum solution. More precisely:

$$V^* \leq \rho_{FG} \tilde{V} \tag{8}$$

where the approximation ratio $\rho_{FG} = 1 + (\tilde{V}^m + B - \tilde{V})/\tilde{V}$, 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. Recall that V^* is the unknown optimal solution to the original cyclic constraint network and $\tilde{V}$ is our approximate solution evaluated on the cyclic constraint network.

This result follows directly from the following theorem which bounds the difference between the computed solution $\tilde{\mathbf{x}}$ and the optimal solution $\mathbf{x}^*$:

4.1 Analysis of Bound

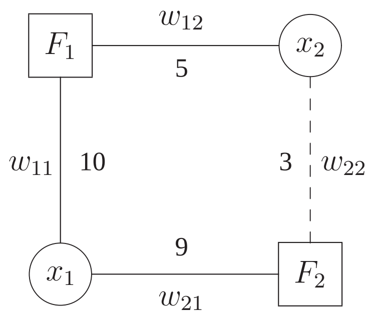
A bounded approximate solution described above is dependent on the properties of the following theorem:

Theorem 1. *Bounded Approximation*

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

This theorem states that the unknown optimal solution V^* is never greater than the sum of the optimal solution computed on the tree structured constraint network and B. This allows us to have an upper bound on the unknown optimal solution and thus to provide a bounded approximation of the original problem. The complete proof of this theorem can be found in Appendix A.

The result stated in Theorem 1 is valid for any spanning tree of the original problem. However, the approximation ratio ρ_{FG} is influenced by which dependencies are removed and is thus dependent on the specific problem instance. Nonetheless, we can provide a general approximation ratio ρ, which is independent of the specific problem instance by performing a worst case analysis of ρ_{FG}. Specifically, assuming that we know the maximum fraction of rewards $\frac{M}{m}$ across all functions, then a worst case approximation ratio for the optimal solution is $\rho = \frac{M}{m}$. We note that this result is in accordance with the analysis performed in [16]. See Appendix B for the full derivation



F'_2	x_1
0	r
7	g
2	b

Figure 4: Tree structured factor graph obtained with the bounded max-sum algorithm for the original factor graph of Figure 2(a).

of this result. Note that the smaller the ratio between the maximum and minimum payoff, the better will be the bound. However, if we have functions that are not bounded (e.g., they can have arbitrarily high payoff) the approximation ratio we provide will not be significant in the worst case.

In practice we use the fact that ρ_{FG} is dependent on the specific problem instance, and therefore we can exploit the structure of the problem to provide a better approximation ratio. Specifically, ρ_{FG} depends on the number of dependencies that we need to cut to build the spanning tree. Clearly this value is higher (and thus the approximation ratio will be worse) for graphs with many cycles. Moreover, while ρ depends on the ratio between the maximum and the minimum payoff across all functions, ρ_{FG} depends on the sum of the impacts of the removed dependencies only. Therefore, if we carefully remove dependencies which have a low impact on the solution quality we can provide very good approximation ratios.

To better explain the operations performed by the bounded max-sum algorithm consider again the factor graph reported in Figure 2(a). The first step of our algorithm is to compute the weights for each link in the factor graph, and we show these in Figure 4 using the functions reported in Figure 2(b). We then form a new factor graph which is a spanning tree of the original factor graph. More specifically, we remove link e_{22}, which is the one with the smallest weight (this is shown as a dashed line in the figure), and thus, $B = w_{22} = 3$. Moreover, we replace the function $F_2(x_1, x_2)$ with $F'_2(x_1) = min_{x_2}[F_2(x_1, x_2)]$. Now, we run the max-sum algorithm on the new factor graph. Since this new factor is a tree, the max-sum algorithm is guaranteed to converge to the optimal solution, which in this case is $x_1 = r$ and $x_2 = r$. This achieves a utility on the new factor graph, $\tilde{V}^m$, of 10 and a utility on the original factor graph, $\tilde{V}$, of 11. Furthermore, in this case we have that $\rho_{FG} = 1+(10+3-11)/11 = 13/11$, and thus, we know that the unknown optimal solution must be greater than $\tilde{V}$, but no more than $\rho_{FG}\tilde{V}$, which in this case is 13. Now, recall that the optimal solution for the original factor graph was shown in Section 3 to be $x_1 = b$ and $x_2 = b$ yielding a total utility of 12 (V^*). Thus, as required we have that $V^* \leq \rho_{FG}\tilde{V}$.

4.2 Decentralised Bounded Max-Sum

Having described our approach, and discussed the approximation ratio that we can provide, we now detail 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 approximation ratio, and (ii) initiating the max-sum algorithm and propagating the information required to compute the approximation ratio to the agents. In this section, we describe the approach for factor graphs containing n-ary constraint functions; we specify the computation of the approximation ratio when only pairwise constraint functions are present in Section 4.2.2

4.2.1 Spanning Tree Formation

As described earlier, we aim to remove cycles from the factor graph to guarantee convergence of the max-sum algorithm. Moreover, we want to remove dependencies which have minimal impact on the solution quality. We can do this by finding a spanning tree that minimises the sum of the weights of the removed edges. To this end, we use the weights of each edge to compute a maximum weight spanning tree, T. Notice that, by finding a maximum weight spanning tree we effectively minimise the sum of the weights of the removed edges. Moreover, if we indicate with $W = \sum_{<i,j>\in C} w_{ij}$, where C is the set of couples of indices $< i, j >$ that identify the edges removed from the factor graph, we can then show that $W \geq B$, i.e., the sum of the weight of removed edges is an upper bound of B (the proof of this is provided in Appendix A, Lemma 2). Therefore, by minimising the sum of the weights of removed edges we are minimising the approximation ratio ρ_{FG}.

The computation of the maximum spanning tree can be performed 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 [28]. This is a distributed, asynchronous algorithm, for general, undirected graphs.[9] GHS is optimal in terms of communication cost $O(n \log n + E)$ and has a running time of $O(n \log n)$, where n is the number of nodes in the factor graph.

We briefly describe the GHS algorithm here and refer to [28] 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.

4.2.2 Max-Sum Initiation and 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

[9] 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. However, other distributed algorithms do exist which have a lower bound for running time e.g. [33].

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.[10] 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 the COMPLETE message 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.[11] 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 ρ_{FG}.

When the leaf nodes receive this final MAXSUM message, the B and solution propagation phase starts. During this phase, nodes propagate tuples composed of BSUM (which will accumulate the value of B specified in Equation 6) and SOLUTION messages.[12] If the leaf is a variable node it creates an empty BSUM and an empty SOLUTION message. If it is a function node F_i, it creates a BSUM message of value equal to $B_i(\mathbf{x}_i^c)$ where $\mathbf{x}_i^c$ is the set of local deleted variables, 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 BSUM 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 local B_i component, and the value of $F_i(\tilde{\mathbf{x}}_i)$, respectively. When the root has received all the BSUM and SOLUTION messages, both are propagated back down the tree, informing each node of the total B, 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 $\rho_{FG} = 1 + (\tilde{V}^m + B - \tilde{V})/\tilde{V}$. The number of messages for each information propagation phase is equal to the number of edges in the spanning tree (i.e., $|\mathbf{F}| + |\mathbf{x}| - 1$), and thus, while the size of each message depends on the message type, it is always constant with respect to the number of nodes in the factor graph (e.g., a MAXSUM message involving variable x_i contains $|\mathbf{d}_i|$ values while BSUM and SOLUTION messages contain one value each).

4.3 Approximation Ratio for Pairwise Interactions

Note that when the interactions are pairwise[13] and thus at most one dependency is removed from each function node, there is a direct link between B_i and the removed

[10]Note that, in our case, one of the two core nodes will always be a function node because the factor graph is a bipartite graph, and the core nodes are connected.

[11]In settings where the choice of variable assignment may not be unique (most commonly, when the functions return integer payoffs) an additional value propagation phase may be used at this point. See [34] for details.

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

[13]The focus on pairwise interactions is a very common approach in the DCOP literature, which is why we pay specific attention to this type of interactions in this section.

weight. Specifically, since each function F_i has exactly two edges, B_i will be either zero (when no dependencies are removed for that function) or the weight of the removed dependency. Consequently, by minimising the sum of the removed weights, we directly minimise the approximation ratio. Therefore, by using the approach presented in the previous section we find the optimal set of dependencies to be removed, i.e. the set of dependencies that provide the minimum approximation ratio.

However, in general, when multiple dependencies may be removed from any function node, this is no longer the case. For example, consider Figure 3, and suppose the spanning tree is a maximum spanning tree. This implies that e_{23} and e_{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 $B_i(x_2, x_3) = \max_{x_1}[\max_{x_2,x_3} F_2(x_1, x_2, x_3) - \min_{x_2,x_3} F_2(x_1, x_2, x_3)]$ 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 B 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, such that $B \leq W$, and thus, our bounded approximate solution is still valid in these cases. The proof that this inequality holds is presented in Appendix A.

4.4 Empirical Evaluation

We now present an empirical evaluation of our bounded approximate algorithm, in particular we wish to evaluate the significance of the approximation ratio that our approach can provide. Recall that the lower the approximation ratio the better. This empirical evaluation is required because our approximation ratio depends on the specific problem instance, and in particular on the topology of the constraint network (i.e., mainly on the number of loops) and on the ratio between the maximum and minimum payoff of the constraint functions (as discussed above). Thus, here we consider a set of decentralised coordination problems where a set of agents is arranged in a 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. Since there are pairwise interactions we have $B = W$ and we are able to compute the minimum approximation ratio ρ_{FG}.

We consider two different graph topologies: random graphs and graphs from the ADOPT repository which represents a large class of graph colouring problems that have previously been used to benchmark DCOP approaches (available from `http://teamcore.usc.edu/dcop/`). In both cases, graphs were selected with different link densities (i.e. the average connection per agents) and different numbers of nodes.

A random payoff matrix is associated with each edge of the graph, 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 distribution, and we consider two different distributions: a gamma distribution with $\alpha = 9$ and $\beta = 2$, and a uniform distribution with range $(0, 1)$. Both produce strictly positive payoffs, but only the uniform distribution has finite support.

This setting generalises the distributed graph colouring problem, which is a canonical problem frequently used to evaluate DCOP techniques (e.g., [5] and [6]). In the standard graph colouring domain the value of W that 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 by differentiating the values of

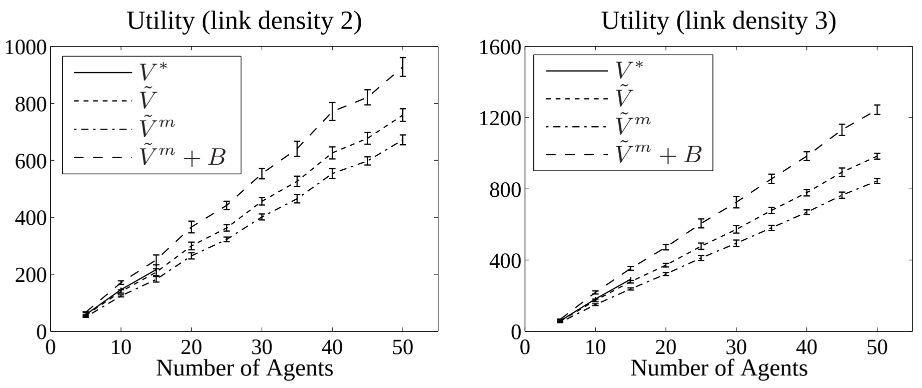


Figure 5: Empirical results for the utility when varying the number of agents and the link density (using random graphs with payoffs drawn from a gamma distribution).

constraint functions; moreover, the use of a gamma distribution introduces significant variance such that some dependencies have a higher impact than others. By having different values for different constraint functions and dependencies, we consider situations where constraints among the actions of some agents are more important than others for the global solution. This makes the evaluation analysis more significant, and it better represents realistic applications, such as cooperative exploration with mobile sensors, which are the main application focus of this work. An empirical evaluation of our approach in the mobile sensor domain will be presented in Section 6.4.

For each configuration, we consider the following four 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.
- $\tilde{V}^m + B$: The upper bound on the value of the unknown optimal solution computed by our approach.
- V^*: The optimal solution computed using a previously published modified version of ADOPT [35].[14]

We first consider the case of random graphs with payoffs drawn from a gamma distribution, and in Figure 5 we show the results obtained for link densities of 2 and 3.[15] For each configuration, we report the average value and the 95% confidence in-

[14]For the results reported here we used the code available at `http://teamcore.usc.edu/dcop` published by the authors of the paper. Specifically, we used the preprocessing policy named DP2 in their paper, which out-performs alternatives in their empirical evaluation. Furthermore, note that ADOPT normally minimizes the constraint costs in a DCOP while here we wish to maximise the sum of the rewards. However, since there are no infinitely high rewards in any problem instance considered here, we are able to determine an arbitrary fixed maximum threshold, M, for each specific instance, and then translate the reward function, r(x), to a cost function, c(x), such that c(x) = M - r(x). ADOPT can then be used to solve the resulting minimisation problem.

[15]These values are in the range often used for benchmarking DCOP techniques on random graph colouring instances [5].

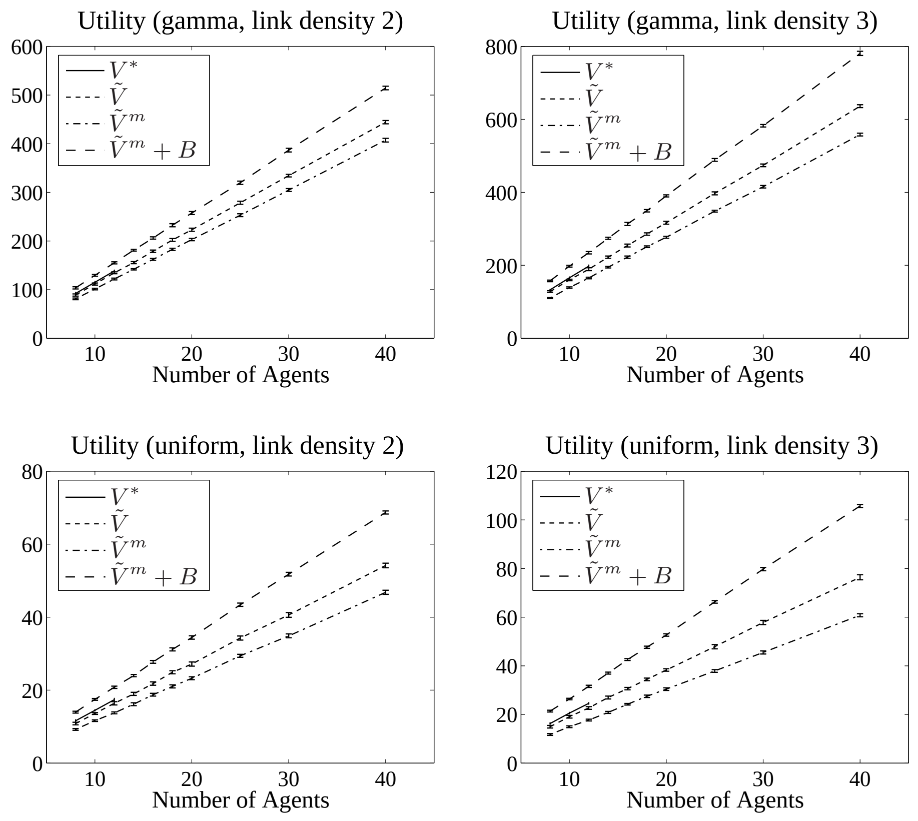


Figure 6: Empirical results for the utility when varying the number of agents, the link density and the distribution for payoffs (using graphs from the ADOPT repository).

terval computed over twenty repetitions.[16] Since the optimal utility is computed by a complete algorithm, we were able to compute this metric only for smaller numbers of agents (e.g., up to 15). Our results show that the actual utility that our approach computes is extremely close to the optimal solution (in the experiments the minimum ratio was 95%). Thus showing that, from an empirical point of view, our approach provides very good approximations. More importantly, however, the approximation ratio we guarantee is significant. In the experiments ρ_{FG} was never above 1.27, and was typically 1.23.

To illustrate the insensitivity of these results to the particular graph topology and payoff distribution, in Figure 6 we show the results for graphs from the ADOPT repository with payoffs drawn from both gamma and uniform distributions. The same measures described above ($\tilde{V}^m$, $\tilde{V}$, $\tilde{V}^m + B$, V^*) averaged over all the different graph instances available in the ADOPT repository (25 instances) and the 95% confidence interval. Results show that the behaviour of our approach is similar across the different payoff distributions we considered. In more detail, the approximation ratio is slightly better (i.e., lower) for the gamma distribution than uniform but it is very significant for

[16]The small confidence interval shows that twenty repetitions provide, for our experimental setting, a good sample size to assess the statistical significance of the results.

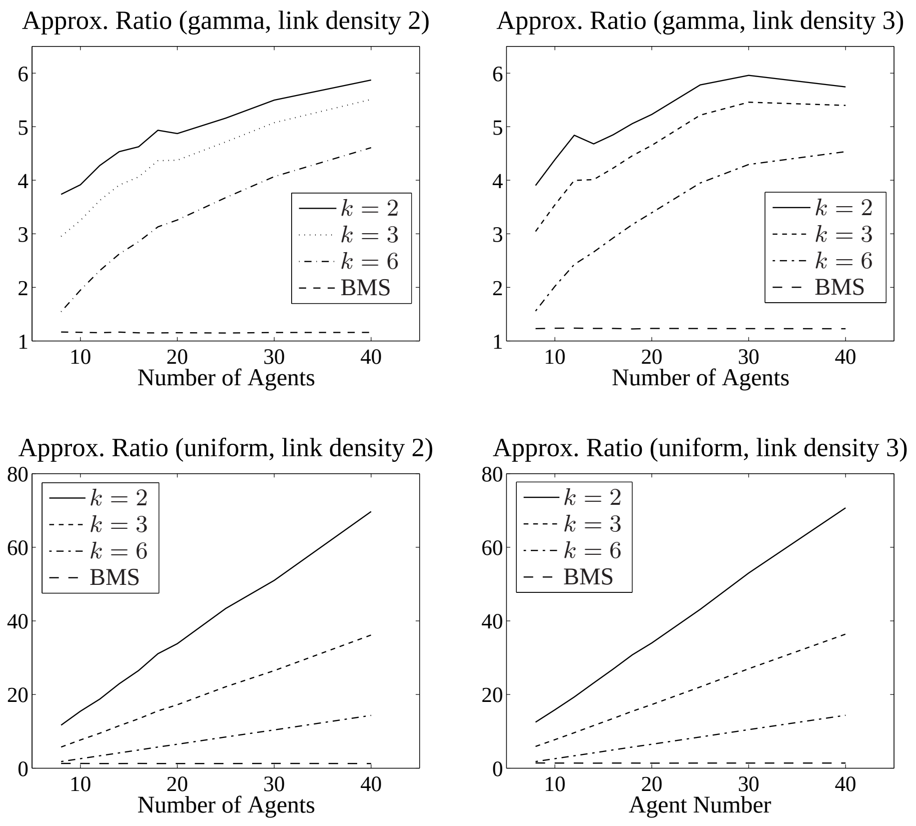


Figure 7: Empirical results for the approximation ratio obtained with our bounded max-sum (BMS) and the approximation ratio computed using the k-optimal analysis (using graphs from the ADOPT repository).

both the payoff distributions. In particular, the worst approximation ratio was approximately 1.24 and 1.43 with an average of 1.2 and 1.33 respectively. As before, the actual utility computed by our approach is extremely close to the optimal solution.[17]

To analyse the significance of the approximation ratio that our approach provides, we compute for the same data set the approximation ratio obtained with the k-optimality framework, using the formulas provides in [16] for general constraint networks. For our data set the constraint arity is 2 ($m = 2$) and we compute the average ratio of the least minimum reward to the maximum reward (indicated with β in [16]) for the different distributions and agent numbers. Our results, in Figure 7, show that the approximation ratio obtained using the bounded max-sum approach (labelled as BMS) is much more significant than that obtained using the k-optimality framework (labelled by their k value). Clearly, by increasing k it is possible to achieve better approximation ratios, however this would result in an exponential increase in the computation required to obtain a k-optimal solution and, in fact, the most widely used approximate algorithms in the field uses $k = 1$ or $k = 2$ [1, 10]. Recall however that the approximation ratio

[17]As in the previous results, the value for the optimal utility is computed by a complete algorithm, and thus, we were able to report values only up to 12 agents.

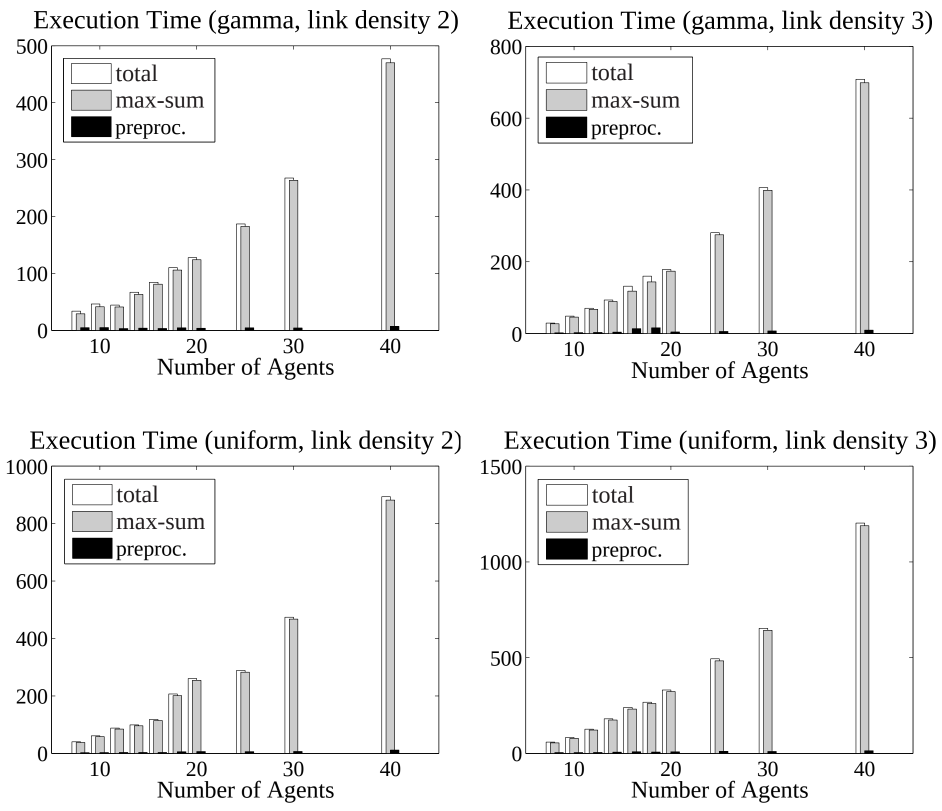


Figure 8: Empirical results for the execution time in milliseconds for pre-processing the factor graph and running the max-sum algorithm, varying the number of agents, the link density and the distribution for payoffs (using graphs from the ADOPT repository).

computed with the bounded max-sum is specific to the problem instance while the one provided by the k-optimality framework is not, and as such can be computed without running any solution algorithm.

We now consider the execution time of our approach on the same data set described above. Specifically, we measure the computational time (in milliseconds) required to form the spanning tree and compute the weights (this operation is labeled *prepoc.* in the figures) and the time required to run the max-sum algorithm over the spanning tree. Out results, in Figure 8, show that our approach scales very well with the number of agents, having a total running time of approximately 1 second on the most complex problem instance in the ADOPT graph repository (specifically, a graph with 40 agents and a link density of 3). Moreover, the running time of the approach is dominated by the execution of max-sum on the spanning tree and the preprocessing time is negligible.

Finally, we present a comparison of the utility obtained using the bounded max-sum approach proposed here, the loopy version of max-sum (i.e., max-sum running directly on the input constraint network) and the optimal utility computed using a previously published modified version of ADOPT [35]. Our aim here is to compare the utility obtained using the max-sum and its bounded version with respect to the optimal utility for larger problem instances. To this end we focus on the graphs of the ADOPT

Agents	Loopy max-Sum	Bounded max-Sum ($\tilde{V}$)	ADOPT (V^*)
8	142.80 ± 0.00	142.80	142.80
10	162.36 ± 2.25	167.20	169.03
12	203.67 ± 0.00	197.83	203.67
14	221.86 ± 2.84	211.65	221.86
16	261.42 ± 3.53	247.47	264.72
18	293.82 ± 0.00	287.56	-
20	329.10 ± 0.00	310.79	-
25	399.14 ± 1.66	390.45	-
30	500.63 ± 0.00	486.04	-
40	614.12 ± 9.37	615.50	-

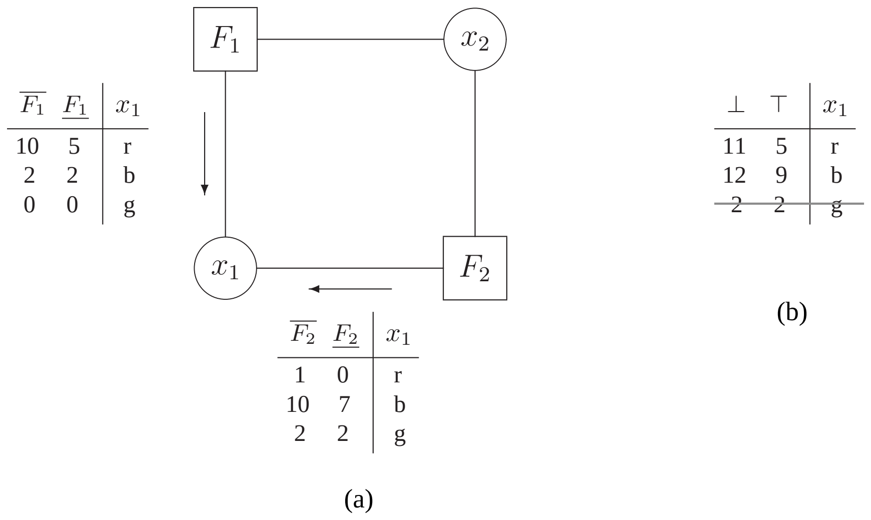
Table 1: Utility comparison for max-sum, bounded max-sum and ADOPT [35] on one problem instance from ADOPT graph repository whilst varying the number of agents (using payoffs drawn from a gamma distribution).

repository with link density 3, we use the gamma payoff distribution mentioned above, and we run the various algorithm on a single problem instance for each agent number. Using this approach we were able to compute the optimal utility up to 16 agents, but for higher number of agents the version of ADOPT we used could not terminate the computation within the imposed time limits of five hours. Table 1 reports the results obtained. Since both the bounded max-sum and ADOPT approaches are deterministic, the reported values are the utilities obtained on a single execution of each algorithm. However, for the loopy max-sum we report the average utility obtained over twenty repetitions together with the 95% confidence interval.

Our results show that the utility values obtained by the loopy max-sum are very close to the ones provided by the bounded max-sum approach, with the loopy max-sum being marginally superior for most of the problem instances. Moreover, both loopy and bounded max-sum achieve results which are very close to the optimal. This results confirm that loopy max-sum is able to provide very good empirical results, and show that the bounded version proposed here achieves similar performance providing guarantees on the solution quality.

5 Speeding Up Message Computation

As mentioned in the introduction, many practical applications inherently have large action spaces for individual agents. For example, in the mobile sensor domain that will be discussed in Section 6, each sensor (agent) can reposition itself to many different locations, and follow different paths on its way to those locations. Moreover, to evaluate the utility gained by the entire team of sensors, a computationally expensive function needs to be evaluated. Therefore, in general, the straightforward application of max-sum to compute the optimal joint action (see Equation 1) is not practical, because the computation of the messages that are sent from function F_j to variable x_i (Equation 3) is a major bottleneck. The naïve way of computing these messages for a given variable x_i is to enumerate the entire domain of $\mathbf{x}_j$ (i.e. the domain of F_j), and evaluate F_j for each element. Since the size of this joint action space grows exponentially with both the number of agents, and the number of possible actions for each agent, the amount of



$\overline{F_1}$	$\underline{F_1}$	x_1
10	5	r
2	2	b
0	0	g

$\overline{F_2}$	$\underline{F_2}$	x_1
1	0	r
10	7	b
2	2	g

$\perp$	$\top$	x_1
11	5	r
12	9	b
~~2~~	~~2~~	~~g~~

Figure 9: First iteration of the action pruning algorithm for variable x_1 showing (a) the messages sent by function nodes F_1 and F_2, and (b) the resulting removal of 'g' from the domain of variable x_1.

computation can quickly become prohibitive in many settings. This is especially true when evaluating F_j is costly. Therefore, in this section, we present two novel pruning algorithms that drastically reduce the size of the joint action space that needs to be searched without sacrificing solution quality. In the remainder of this section, we will describe each algorithm in turn.

5.1 The Action Pruning Algorithm

The first algorithm attempts to reduce the number of actions each agent needs to consider *before* running the max-sum algorithm. This algorithm prunes the dominated actions that can never maximise the social welfare (Equation 1), regardless of the actions of other agents. More formally, a value $d \in \mathbf{d}_i$ of variable x_i is dominated if there exists a value $d' \in \mathbf{d}_i$ such that:

$$\forall d_{-i} \in \bigtimes_{k=1, k\neq i}^{n} \mathbf{d}_k : \sum_{j \in \mathcal{M}_i} F_j(d, d_{-i}) \leq \sum_{j \in \mathcal{M}_i} F_j(d', d_{-i}) \tag{10}$$

where, as in Section 3, $\mathcal{M}_i$ is a set of function indexes, indicating which function nodes are connected to variable node i. Now, by removing these dominated actions, the optimal solution remains unchanged. However, since a number of dominated actions are pruned, the size of the joint action space is reduced.

Just as with the max-sum algorithm itself, this algorithm is implemented by message passing, and operates directly on the variable and function nodes of the factor graph, making it fully decentralised:

- **From function to variable**: The message from function F_j to x_i contains the minimum $\underline{F_j}(x_i)$ and the maximum $\overline{F_j}(x_i)$ value of F_j with respect to $x_i = d$, for all actions $d \in \mathbf{d}_i$, the domain of x_i (see Algorithm 1).

- **From variable to function**: Variable x_i sums the minimum and maximum values from each of its adjacent functions, and prunes dominated actions. It then informs neighbouring functions of its updated domain (see Algorithm 2).

Figure 9 reports the messages that variable x_1 would received at the first iteration of the action pruning algorithm using the example factor graph and functions presented in Figure 2. In this case, given the received messages, variable x_1 will be able to prune g from its domain.

Using this distributed algorithm, functions continually refine the bounds on the utility for a given value of a variable, which potentially causes more actions to be pruned. Therefore, it is possible that action pruning starts by pruning a single action, which results in further actions being pruned throughout the entire factor graph.

Algorithm 1 Algorithm for computing pruning message from function F_j to variable $x_i : i \in \mathcal{N}_j$

```
1: compute $\underline{F_j}(x_i) \leq \min_{\mathbf{x}_j \setminus x_i} F_j(x_i, \mathbf{x}_j \setminus x_i)$
2: compute $\overline{F_j}(x_j) \geq \max_{\mathbf{x}_j \setminus x_i} F_j(x_i, \mathbf{x}_j \setminus x_i)$
3: send $\langle \overline{F_j}(x_i), \underline{F_j}(x_i) \rangle$ to $x_i$
```

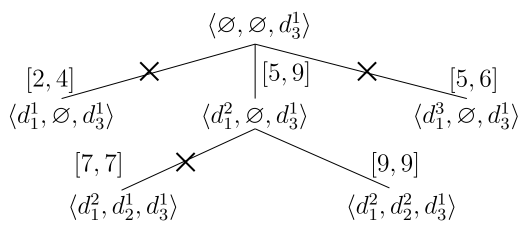
Algorithm 2 Algorithm for computing pruning messages from variable x_i to all functions $F_j : j \in \mathcal{M}_i$

```
1: **if** a new message has been received from all $F_j : j \in \mathcal{M}_i$ **then**
2:     compute $\bot(x_i) = \sum_{j \in \mathcal{M}_i} \underline{F_j}(x_i)$
3:     compute $\top(x_i) = \sum_{j \in \mathcal{M}_i} \overline{F_j}(x_i)$
4:     **while** $\exists d \in \mathbf{d}_i : \top(d) < \max \bot(x_i)$ **do**
5:         $\mathbf{d}_i \leftarrow \mathbf{d}_i \setminus \{d\}$    *Remove dominated value $d$*
6:     **end while**
7:     send updated domain $\mathbf{d}_i$ to all $F_j : j \in \mathcal{M}_i$
8: **end if**
```

This algorithm terminates once the messages exchanged between the functions and variables converge. That is, when all messages along all edges in the factor graph are equal to the previously received messages. Thus, a node in the factor graph can initiate the max-sum algorithm once it has received the same message twice from each neighbour. Also note that termination is guaranteed because of the fact that every variable has a finite number of actions; during each iteration either at least one variable value is pruned or the algorithm has converged. To see why this is true, note that for the bounds on F_i for a certain value d to change, at least one variable value needs to get pruned. Otherwise, the messages sent from variables to functions will be identical, and all variables receive the same message twice, which results in the termination of the algorithm.

5.2 The Joint Action Pruning Algorithm

Now, whereas the first algorithm runs as a preprocessing phase to max-sum, the second algorithm is geared towards speeding up the computation of the messages from function to variable (see Equation 3), *during* the execution of the max-sum algorithm. In contrast to reducing the action space of individual agents, which was the goal of the



$\langle \varnothing, \varnothing, d_3^1 \rangle$
$[2,4]$
$[5,9]$
$[5,6]$
$\langle d_1^1, \varnothing, d_3^1 \rangle$
$\langle d_1^2, \varnothing, d_3^1 \rangle$
$\langle d_1^3, \varnothing, d_3^1 \rangle$
$[7,7]$
$[9,9]$
$\langle d_1^2, d_2^1, d_3^1 \rangle$
$\langle d_1^2, d_2^2, d_3^1 \rangle$

Figure 10: Search tree for computing $r_{j\to 3}(d_3^1)$ (a single element of the message from F_j to x_3). The numbers between the brackets indicate lower and upper bounds on the maximum value in the subtree.

first algorithm, this algorithm attempts to reduce the size of the joint action space that has to be searched by applying branch and bound.

A naïve way of computing this message to a single variable x_i is to determine the maximum utility for each of agent i's actions by exhaustively enumerating the joint domain of the variables in $\mathbf{x}_j \setminus \{x_i\}$, and evaluating the expression between brackets in Equation 3, which we denote by:

$$\tilde{r}_{j\to i}(\mathbf{x}_j) = F_j(\mathbf{x}_j) + \sum_{k\in\mathcal{N}_j\setminus x_i} q_{k\to j} \tag{11}$$

Instead of just considering joint actions, we now allow some actions to be undetermined, and thus, consider *partial* joint actions. By doing so, we can create a search tree on which we can employ branch and bound to significantly reduce the size of the domain that needs to be searched. In more detail, to compute $\tilde{r}_{j\to i}(d_i^k)$ for $d_i^k \in \mathbf{d}_i$ (a single element of the message from F_j to variable x_i), we create a search tree $\mathcal{T}(d_i^k)$ as follows:

- The root r of $\mathcal{T}(d_i^k)$ is a partial joint action $\hat{\mathbf{d}}_r = \langle \varnothing, \ldots, \varnothing, d_i^k, \varnothing, \ldots, \varnothing \rangle$, which indicates that x_i has been assigned the value d_i^k, and the remaining variables are unassigned (denoted by $\varnothing$).
- The set of children of a node p represented by partial action $\hat{\mathbf{d}}_p = \langle d_1^{(1)}, \ldots, d_l^{(l)}, \varnothing, \ldots \varnothing, d_i^k, \varnothing, \ldots, \varnothing \rangle$ is obtained by assigning to the first unassigned variable (x_{l+1}) each of its possible actions: $Children(\hat{\mathbf{d}}_p) = \{\langle d_1^{(1)}, \ldots, d_l^{(l)}, d_{l+1}, \varnothing, \ldots, \varnothing, d_i^i, \varnothing, \ldots, \varnothing \rangle | d_{l+1} \in \mathbf{d}_{l+1}\}$. Thus, the node $\hat{\mathbf{d}}_p$ has $|\mathbf{d}_{k+1}|$ children.
- The leafs of the tree represent a (fully determined) joint action: $\forall i \in \mathcal{N}_j : x_i \neq \varnothing$. In the search tree, leafs are assigned a value that results from evaluating Equation 11 for the corresponding action.

The leaf with the maximum value found in $\mathcal{T}(d_i^k)$ represents the action that maximises Equation 3 for value d_i^k, and thus has the desired value for $r_{j\to i}(x_i)$. Now, to find this value efficiently using branch and bound, we need to be able to find bounds on the maximum value found in a subtree of $\mathcal{T}(d_i^k)$. These bounds depend on F_j and the received messages $q_{k\to j}$. Now, in many cases we can put bounds on the maximum of the former, that is obtained by further completing a partial joint action in a subtree of $\mathcal{T}(d_k^i)$. We will show an example case in Section 6, where we apply these techniques on the mobile sensor domain.

To illustrate this method with a simple example, however, Figure 10 shows a partially expanded search tree for computing a single element $r_{j\to 3}(d_3^1)$ of a message from function F_j to variable x_3. Given the lower and upper bounds on the maximum (denoted between brackets), subtree $\left\langle d_1^1, \varnothing, d_3^1 \right\rangle$ can be pruned immediately after expanding the root. Similarly, subtree $\left\langle d_1^3, \varnothing, d_3^1 \right\rangle$ is pruned after expanding leaf $\left\langle d_1^2, d_2^2, d_3^1 \right\rangle$, which has the desired maximum value.

Now, since the utility functions F_j are domain dependent, there is no general way of computing the aforementioned bounds. However, in most domains, such as the mobile sensor domain which will be introduced in Section 6, a partial joint action has a meaningful interpretation that can lead to an intuitive way of computing the bounds on the maximum of F_j in any subtree of $\mathcal{T}$. We will come back to this in Section 6.

6 The Mobile Sensor Domain

Notation used in this section

- $\mathcal{S} = \{S_i | i = 1 \ldots M\}$ is the set of M mobile sensors.
- $\mathcal{G} = (V, E)$ is the layout of the physical environment.
- E is the possible movements between locations V, with each $v \in V$ embedded in a 2D plane.
- $\mathcal{P}$ is the spatial phenomena that is monitored by the sensors.
- $T = \{t_1, t_2, \ldots\}$ is a sequence of discrete timesteps of unknown length.
- $\mathcal{L}_t = \left(l_t^1, \ldots, l_t^M\right)$ are the sensors' locations at time $t \in T$ where $l_t^i \in V$.
- $O_t = \left(o_t^1, \ldots, o_t^M\right)$ are the measurements taken by the sensors at timestep $t \in T$.
- $o = \langle \mathbf{x}, y \rangle$ is a single measurement of the scalar field.
- $\mathcal{P}$
  $\mathbf{x} = (v, t)$ a location v and timestep t tuple.
- y a measured value.
- $K(\mathbf{X}, \mathbf{X}')$ is the covariance matrix.
- σ_f is the hyperparameter that models the signal variance of the phenomenon.
- l is the hyperparameter that models the length-scale of the phenomenon.
- $H(A|B)$ is the conditional entropy of sample set A given the sample set B.
- $\rho_A(B)$ is the incremental value of adding sample set A to sample set B.

In this section we present the mobile sensor coordination problem that illustrates our approach, in which mobile sensors collect measurements of a spatial phenomenon (such as temperature, radiation, pressure and gas concentration) at discrete points in time and space. Using a statistical model, the sensors model and predict values of this phenomenon at locations and times for which samples are not available. Applications for this approach include environmental monitoring, military surveillance, and disaster response, in which mobile sensors can play a crucial role in improving situational awareness. This is a particularly challenging problem because of the sophisticated statistical models needed to represent the environmental phenomena, and the fact that sensors have to coordinate to collect informative measurements as a team. These properties make it an interesting benchmark problem for the techniques developed in previous sections.

This section is organised as follows. First, we formalise the mobile sensor coordination problem in Section 6.1. In Section 6.2, we show how spatial phenomena can be modelled using a Gaussian process. Next, we show how to apply the max-sum algorithm by defining the coordination problem in terms of decision variables and utility functions in Sections 6.3.1 and 6.3.2. Finally, we empirically evaluate the max-sum algorithm in this domain.

6.1 Problem Formulation

In this section we present a formalisation of the environmental monitoring problem for multiple mobile sensors. This formalisation is inspired by [36], and has been extended for multiple sensors with limited local knowledge.

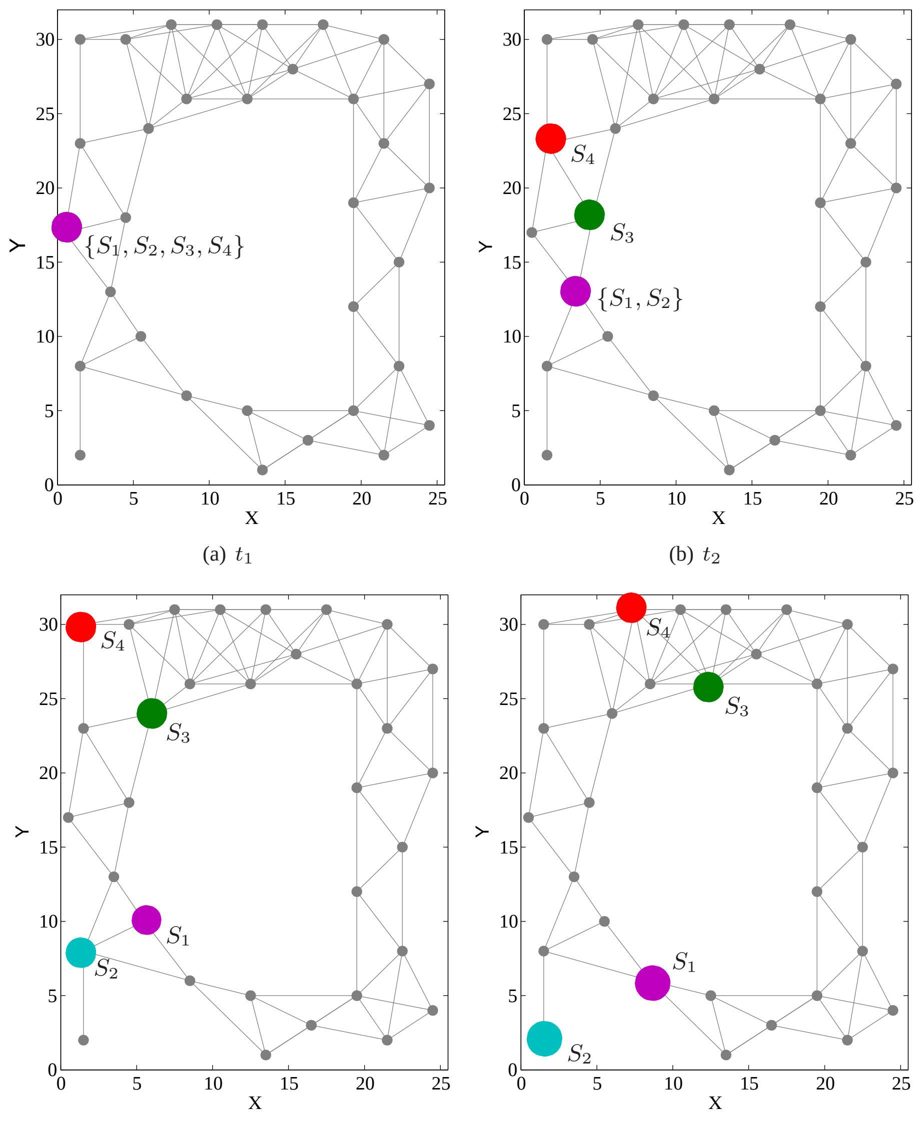
Intuitively, an environment is defined by its physical layout, and by the phenomenon that exist within it. More formally, we can denote an environment and the mobile sensors by a tuple $\mathcal{E} = (\mathcal{S}, \mathcal{G}, \mathcal{P}, T)$, where:

- $\mathcal{S} = \{S_i | i = 1 \dots M\}$ is the set of M mobile sensors;
- $\mathcal{G} = (V, E)$ encodes the layout of the physical environment, where E denotes the possible movements between locations V, with each $v \in V$ embedded in a 2D plane;
- $\mathcal{P}$ is a spatial phenomena that is monitored by the sensors in $\mathcal{S}$. Here, we explicitly model phenomenon $\mathcal{P}$ as a scalar field defined on one temporal and two spatial dimensions: $\mathcal{P} : V \times T \to \mathbb{R}$.
- $T = \{t_1, t_2, \dots\}$ models time as a sequence of discrete timesteps of unknown length.[18]

Furthermore, we denote the sensors' locations at time $t \in T$ by the M-tuple $\mathcal{L}_t = \left(l_t^1, \dots, l_t^M\right)$, where $l_t^i \in V$. At every timestep $t \in T$, the sensors take measurements $O_t = \left(o_t^1, \dots, o_t^M\right)$ at locations L_t by sampling from $\mathcal{P}$: $o_t^i = \mathcal{P}(l_t^i, t)$, and move to a new location adjacent to the current location in V: $l_{t+1}^i \in adj_{\mathcal{G}}(l_t^i)$. To illustrate this formal model with an example, Figure 11 shows the position of a team of four sensors in an example environment during the first four timesteps.

Given this model, the sensors' challenge is to monitor $\mathcal{P}$ at all locations V at time t. Since the number of sensors M is generally much smaller than $|V|$, the sensors need to not only take measurements at locations $\mathcal{L}_t$, but also *predict* the value of $\mathcal{P}$ at time t for every location V, based on observations made earlier. In order to do this, we associate to the measurement at location $v \in V$ at time t a continuous random variable $\mathcal{X}_{v,t}$, and use a statistical model to predict values at locations V. As we shall discuss in the next section, we will model the phenomenon $\mathcal{P}$ with a Gaussian process, that encodes both its spatial and temporal correlations.

[18]In uncertain and dynamic scenarios, the mission time is often not known beforehand.



(a) t_1 (b) t_2 (c) t_3 (d) t_4

Figure 11: Four timesteps of a team of sensors $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$ moving in an environment whose layout is defined by a graph $\mathcal{G} = (V, E)$, pictured in grey. E contains a pair of locations (v_i, v_j) when they are less than 7.5 meters apart. The initial deployment of the sensors is $\mathcal{L}_1 = (v_1, v_1, v_1, v_1)$, where $v_1 = (0.5, 17) \in V$ (if sensors occupy the same location, only one of them is shown). Phenomenon $\mathcal{P}$ is not shown.

Now, in order to move in such a way to collect those samples that improve the accuracy with which measurements at unobserved locations can be predicted, the sensors need to be able to determine the informativeness of samples that may be collected along their path. Here, the informativeness of a set of samples is quantified by a function $f(\mathcal{X})$ of a set of random variables $\mathcal{X} = \{\mathcal{X}^{(1)}, \mathcal{X}^{(2)}, \dots\}$ that correspond to these samples. Depending on the context, $f(\mathcal{X})$ can take on different forms [36]. In this paper, $f(\mathcal{X})$ equals the entropy $H(\mathcal{X})$ of $\mathcal{X}$.[19]

To measure the performance of the sensors, we use the root mean squared error (RMSE) of the sensors' predictions.[20] In order to so, we denote the predictions that the sensors make at time t by $P_t = \{p_t^v | v \in V\}$ and the actual values of the environmental parameter at those locations by $A_t = \{a_t^v | v \in V\}$. The RMSE for timestep t is then defined as:

$$RMSE_t = \sqrt{\frac{\sum_{v \in V}(a_t^v - p_t^v)^2}{|V|}} \tag{12}$$

In the upcoming sections we will show how the max-sum algorithm can be employed to minimise the RMSE. First, however, we will explain how the spatial phenomena are modelled, and how we obtain a measure of uncertainty about the state of the spatial phenomenon that is strongly correlated with the RMSE.

6.2 Modelling the Spatial Phenomena

In order to predict measurements at unobserved locations, we model the spatial phenomenon $\mathcal{P}$ with a Gaussian process (GP) [39]. A GP is a principled Bayesian method of performing inference over functions, and have been shown to be very suitable for modelling spatial phenomena [40, 37, 36, 41]. By using a GP, $\mathcal{P}$ can be estimated at any location and at any point in time using the set of samples collected by the sensors so far.[21]

In more detail, a single sample o of the scalar field $\mathcal{P}$ is a tuple $\langle \mathbf{x}, y \rangle$, where $\mathbf{x} = (v, t)$ denotes the location and time at which the sample was taken, and y the measured value. Now, if we collect the location vectors $\mathbf{x}$ in a matrix $\mathbf{X}$, and the measurements y in a vector $\mathbf{y}$, the predictive distribution of the measurement at spatio-temporal coordinates $\mathbf{x}_*$, conditioned on previously collected samples $O_t = \langle \mathbf{X}, \mathbf{y} \rangle$ is Gaussian with mean μ and variance σ^2 given by:

$$\mu = K(\mathbf{x}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{y} \tag{13}$$

$$\sigma^2 = K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{x}_*) \tag{14}$$

where $K(\mathbf{X}, \mathbf{X}')$ denotes the matrix of covariances for all pairs of rows in $\mathbf{X}$ and $\mathbf{X}'$. Each element of this covariance matrix is obtained by evaluating a function $k(\mathbf{x}, \mathbf{x}')$, called a covariance function, which encodes the spatial and temporal correlations of the pair $(\mathbf{x}, \mathbf{x}')$. Generally, covariance is a non-increasing function of the distance in space

[19]Here we exploit one of the attractive properties of the Gaussian process, whereby the uncertainty of a sample at any point in time or space can be predicted without having to explicitly reason about the actual value of $\mathcal{P}$. For more details about the reasons for choosing this metric, see [27].

[20]This measure was chosen because it has been often used in related work to ascertain the accuracy of sensor predictions [37, 38].

[21]We chose to use a GP because of its versatile and flexible nature. However, the techniques discussed in the remainder of this section are not specific to the use of a GP to model the environment; any other model can be used, as long as it provides some measure of uncertainty in the environment.

and time, and a prototypical choice of a covariance function is the squared exponential function where the covariance decreases exponentially with this distance:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\tfrac{1}{2}|\mathbf{x} - \mathbf{x}'|^2/l^2\right) \tag{15}$$

where σ_f and l are called *hyperparameters* that model the signal variance and the length-scale of the phenomenon respectively. The former models the amplitude of the signal, while the latter determines how quickly the phenomenon varies over time and space.[22]

One of the key features of the GP is that the posterior variance in Equation 14 is independent of actual measurements $\mathbf{y}$. This allows the sensors to determine the entropy reduction that results from collecting samples along a certain path without the need of actually collecting them. Moreover, since the predictive distribution is Gaussian, the entropy $H(\mathcal{X})$ of random variables $\mathcal{X}$ corresponding to a set of potential samples is $\ln\sqrt{2\pi\sigma^2 e}$, where σ^2 is directly obtained from Equation 14.

6.3 Applying the Max-Sum Algorithm

In order to apply the max-sum algorithm to the coordination problem defined in Section 6.1, we need to define a mapping between the concepts of the max-sum algorithm and the concepts in the mobile sensor domain. The three key concepts in max-sum are agent, variable and utility function, which we map to sensor, decision variable and information value respectively. In this mapping, each sensor S_i is modelled as an autonomous agent $\mathcal{A}_i$ that has a single decision variable p_i (p for 'path'). This variable represents the path that it will travel along in the next l timesteps. This variable and its domain will be defined in Section 6.3.1. The information value function U_i, or utility function, encodes the value of the samples that are collected along a sensor's path, given the paths along which the other sensors decide to move. Thus, U_i depends on p_i and (a subset of) the other sensor's variables. These functions will be defined in Section 6.3.2. By applying max-sum in this fashion, we aim to find a collection of paths (a joint path) of finite length along which sensors collectively gather the samples of maximum value. Clearly, if these paths are of length l, the sensors will need to use max-sum to coordinate their moves every $m \leq l$ timesteps. After following (a portion of) their paths, the sensors' action space will have changed, and so will their utility functions. Therefore, the factor graph encoding the current coordination problem changes over time, and consequently, the sensors interleave coordinating using the max-sum algorithm with movement through their environment.

In the remainder of this section, we show how the variables of the factor graph are defined, as well as the functions. Furthermore, we show that the sum of individual sensors' utilities equals the utility of the team, and, as a result of which, we can use max-sum to find the paths that maximise team utility (as described in Equation 1). Finally, we show how to apply the pruning algorithms from Section 5 by computing the various required bounds.

6.3.1 Decision Variables

In Section 6.1 we defined the graph $\mathcal{G}$ that defines the layout of the sensors' environment. Observations can only be collected at the vertices of $\mathcal{G}$, and moves between two

[22] A slightly modified version of Equation 15 allows for different length-scales for the spatial and temporal dimensions of the process.

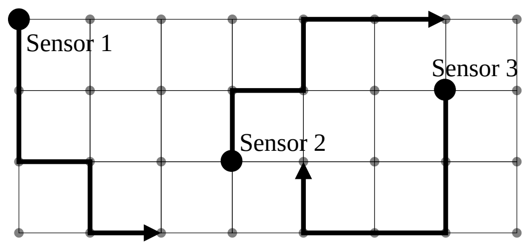


Figure 12: A joint move of length 5 for sensors on a lattice graph.

vertices are only allowed if this graph contains an edge between them. As a result, the set of observations A_i that sensor i can collect is restricted by its current location and the layout of the environment.

Now, given a sensor's current location v_i, and path length l, the set of all possible paths that the sensor can currently consider is denoted by d_i. The joint action space $\mathbf{d}$ of the team of sensors is then the Cartesian product of all individual action spaces: $\mathbf{d} = \times_{i=1}^{M} \mathbf{d}_i$. Each element $d \in \mathbf{d}$ is thus a collection of M paths of length l; one for each sensor. An example of such a joint move for three sensors, consisting of a path of length 5 for each of them, is shown in Figure 12. Specifically, a joint move is an ordered list of vertices of $\mathcal{G}$, at each of which the sensor makes an observation of the spatial phenomenon. Thus, there exists a correspondence between a path and a collection of random variables that are observed along that path. Therefore, with some slight abuse of notation, we can treat every element in $\mathbf{d}$ as both a path and a set of (potential) samples. Thus, we can now assign to each sensor i a decision variable p_i, which takes values in the set $\mathbf{d}_i$, representing all moves that sensor i is currently considering.

6.3.2 Utility Functions

Given the definition of the function f that assigns a value to a set of samples, the team utility of collecting a set of samples A, given that samples B were collected previously is equal to the conditional entropy of A given B and is denoted by $H(A|B)$. By exploiting the chain rule of entropy, which states that $H(X,Y) = H(X|Y) + H(Y)$, we can decompose the team utility of collecting A into a sum of the utility obtained by single sensors that each collect a subset A_i of A, such that $\cup_{i=1}^{M} A_i = A$, as follows:

$$\begin{aligned}
H(A|B) &= H(A_1|B) + H(A_2|A_1,B) + \ldots + H(A_n|A_1,\ldots,A_{n-1},B) \\
&= [H(A_1 \cup B) - H(B)] + [H(A_1 \cup A_2 \cup B) - H(A_1 \cup B)] + \ldots + \\
&\quad [H(A_1 \cup \ldots \cup A_n \cup B) - H(A_1 \cup \ldots \cup A_{n-1} \cup B)] \\
&= \rho_{A_1}(B) + \rho_{A_2}(A_1 \cup B) + \ldots + \rho_{A_n}(A_1 \cup \ldots \cup A_{n-1} \cup B) \\
&= \sum_{i=1}^{n} \rho_{A_i}\left(\bigcup_{j=1}^{i-1} A_j \cup B\right) \qquad (16)
\end{aligned}$$

Where $\rho_A(B)$ is defined as the *incremental value* of adding A to B: $\rho_A(B) = H(A \cup B) - H(B)$.

Informally, Equation 16 ensures that the team utility is a sum of the incremental values by adding samples A_i to the samples collected by sensors $j < i$. We will call the individual factors of this sum the *sensor utility*.

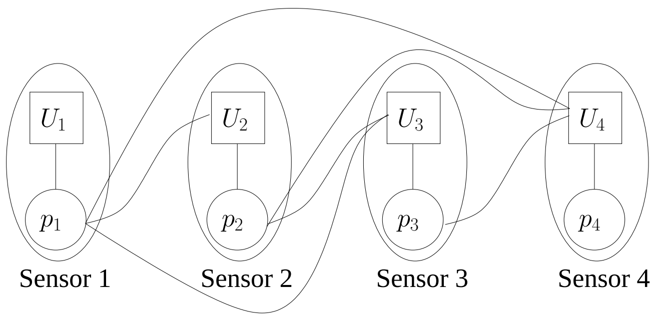


Figure 13: A factor graph encoding the mobile sensor coordination problem for four sensors.

Definition 1 (Sensor Utility)**.** *Sensor i's contribution to the team utility is:*

$$U_i(A_1, \ldots, A_i) = \rho_{A_i}\left(\bigcup_{j=1}^{i-1} A_j \cup B\right)$$

So, in order to calculate its utility, a sensor need only be aware of the samples collected by sensors with a lower ID. Moreover, by summing the contributions by individual sensors, we obtain $H(A|B) = H(A_1 \cup \ldots \cup A_M|B)$, which is the team utility. Note, that it is possible to further factorise the utility functions if the samples collected by one agent are independent of those of another agent. In this case, the edge connecting the function node of each of these agents to the variable node of the other agent can be removed. The correlation length of the Gaussian process provides a clear metric to perform this edge removal. However, in this evaluation, we do not do so, since this edge removal is effectively performed by the construction of the maximum spanning tree within the bounded max-sum algorithm.

Combining this observation with the definition of the sensors' decision variables and the correspondence between observation sets and paths, the goal of the team is now to find joint move $\mathbf{p}^*$ such that:

$$\mathbf{p}^* = [p_1^*, \ldots, p_M^*] = \underset{p_1 \in \mathcal{A}_i, \ldots, p_M \in \mathcal{A}_M}{\arg\max} \sum_{i=1}^{M} U_i(p_1, \ldots, p_i) \tag{17}$$

In other words, the sensors attempt to find joint move $\mathbf{p}^*$ that maximises the team utility by maximising the sum of their contributions as defined in Definition 1. Since the team utility is the sum of the sensors' utility functions, max-sum can be readily applied to solve this coordination problem.

Finally, to give an example of a factor graph resulting from combining the variables from the previous section with the utility functions defined in this section, Figure 13 shows the factor graph for solving the coordination problem with four sensors.

6.3.3 Applying the Pruning Algorithms

As mentioned in Section 5, the straightforward application of max-sum in domains where the utility functions are expensive to evaluate leads to a prohibitive computational cost. Clearly, this the case in the mobile sensors domain, where determining the value of a sample involves the inversion of a potentially very large matrix $K(\mathbf{X}, \mathbf{X})$ (see Equation 14). Thus, the use of the pruning algorithms presented in Sections 5.1 and 5.2 could be particularly advantageous in this setting.

Now, in order to compute the necessary bounds for these two pruning algorithms, we need to use domain specific knowledge, since these bounds are context dependent. In particular, we will exploit various properties of the GP in order to efficiently compute (approximations to) these bounds.

Firstly, we derive the bounds on the utility functions for the pruning algorithm from Section 5.1. Note that, given the highly non-linear relations expressed in Equation 14 on which the agents' utility functions U_i are based, it is very difficult to compute tight bounds on $\overline{U}_i$ and $\underline{U}_i$ in Algorithm 1 without exhaustively searching the domain of $\mathbf{p}_i$ for utility function U_i. Needless to say, this would defeat the purpose of this pruning technique. Nonetheless, experimentation shows that by computing these bounds in a greedy fashion, a very good approximation is obtained. In order to do this, the lower bound $\underline{U_i}(p_n)$ on a move p_n is obtained by selecting the neighbouring agents one at a time, and finding the move that reduces the utility of agent i's move the *most*. In a similar vein, the upper bound $\overline{U_i}(p_n)$ is obtained by selecting those moves of other sensors that reduce the utility the *least*.

Next, we derive bounds on the maximum utility found in subtrees $\mathcal{T}$—the search tree for the joint action pruning algorithm defined in Section 5.2. To compute these bounds on $U_i(\hat{\mathbf{d}})$, for a partial joint action $\hat{\mathbf{d}}$, first note that this partial joint action represents a situation in which only a subset of the sensors have determined their move. Using this interpretation, we can obtain bounds as follows. The upper bound on this value is obtained by disregarding the sensors that have not yet determined their move (i.e. sensors i for which $p_i = \varnothing$). To see why this results in an upper bound, note that the act of collecting a sample always reduces the value of other samples (an example of the information never hurts principle), disregarding the samples of these 'undecided' sensors will give an upper bound on the maximum. Thus, this value is computed by evaluating a modified version of the utility function as follows: $U_i(\{p_j \in \mathbf{p}_i : p_j \neq \varnothing\})$.

To obtain a lower bound on the maximum, we exploit a property of the covariance function in Equation 15. This property causes the interdependency between the value of samples to weaken as the distance between them increases. So, in order to calculate a lower bound on the maximum, we compute the value of the sample in the event that the undecided sensors move away from sensors i's destination. This results in minimum correlation between the sample and the samples collected by the undecided sensors, thus increasing the value of this sample. In many cases, this results in a very tight lower bound on the maximum of the sample.

6.4 Empirical Evaluation

In this section, we evaluate the algorithms developed in this paper on the mobile sensor domain to ascertain their effectiveness in a setting inspired by a real-life application. Specifically, we first evaluate the speed up resulting from applying the pruning algorithms described in Section 5. The mobile sensor setting is a very suitable candidate to do this, since the computational overhead incurred by evaluating the utility functions is significant in this setting. Second, we evaluate the bounded max-sum algorithm from Section 4 in combination with these pruning algorithms, and determine the difference between the difference between the optimal and obtained solution. This evaluation is similar to Section 4.4, however, instead of considering randomly generated coordination problems, we use the more realistic mobile sensor domain.

6.4.1 Effectiveness of the Pruning Algorithms

To empirically evaluate the two pruning algorithms in the mobile sensor domain, we simulated five sensors on a lattice graph measuring 26 by 26 vertices. The data was generated by a GP with a squared exponential covariance function (see Equation 15) with a spatial length-scale of 10 and a temporal length-scale of 150. This means that the spatial phenomenon has a strong correlation along the temporal dimension, and therefore changes slowly over time. [23]

Now, at every m time steps, the sensors plan their motion for the next l time steps ($l \geq m$). In what follows, this strategy is referred to as MSm-l. Now, instead of considering all possible paths of length l from an agent's current position, which would result in a very high computational overhead, the action space is limited to the locations in G that can be reached in l time steps in 8 different directions, corresponding to the major directions on the compass rose. In the first experiment, we benchmarked MS1-1 and MS1-5 against four strategies often found in the literature:

- **Random:** Randomly moving sensors.
- **Greedy:** Sensors that greedily maximise the value of the sample collected in the next move without coordination. This strategy was included to determine the effect of coordination between sensors.
- **J(umping) Greedy:** The same as Greedy, except that these sensors can instantaneously jump to any location. This strategy will act as an upper bound on the achievable performance of a greedy strategy, since it is not constrained by the movement graph $\mathcal{G}$.
- **Fixed:** Fixed sensors that are placed using the algorithm proposed in [37]. This is an algorithm that positions fixed (i.e. non-mobile) sensors as to minimise the entropy at all monitored locations.

The average root mean squared error (RMSE) over 100 time steps is plotted in Figure 14(a). From this figure, it is clear that both MS strategies outperform the Greedy and Random strategies, since both have no more than one step look ahead, and the MS strategies compute coordinated paths of length 1 and 5. Furthermore, the prediction accuracy of MS1-5 is comparable to that of JGreedy, whose movement is not restricted by graph G. Moreover, it shows that increasing the look ahead improves the solution quality: the length of the considered paths from 1 to 5 reduces the RMSE by approximately 30%.

In the second set of experiments, we analysed the speed-up achieved by applying the two pruning techniques described in Section 5. Figure 14(b) shows the percentage of joint actions pruned plotted against the number of neighbouring agents. With 5 neighbours, the two pruning techniques combined prune around 92% of the joint moves. With such a number of neighbouring agents, the agents are strongly clustered, which occurs rarely in a large environment. However, should this happen, the utility

[23] These parameters were chosen to generate challenging coordination instances. For example, by using a high value for the spatial length-scale, sensors are able to cover the entire area without needing to move. Similarly, with a very high value for the temporal length-scale, the sensors need to traverse the environment only once, since the phenomenon changes very little over time. Thus, while not necessarily being the worst-case scenario, the problems generated by setting the parameters to these values represent the most challenging instances we managed to create.

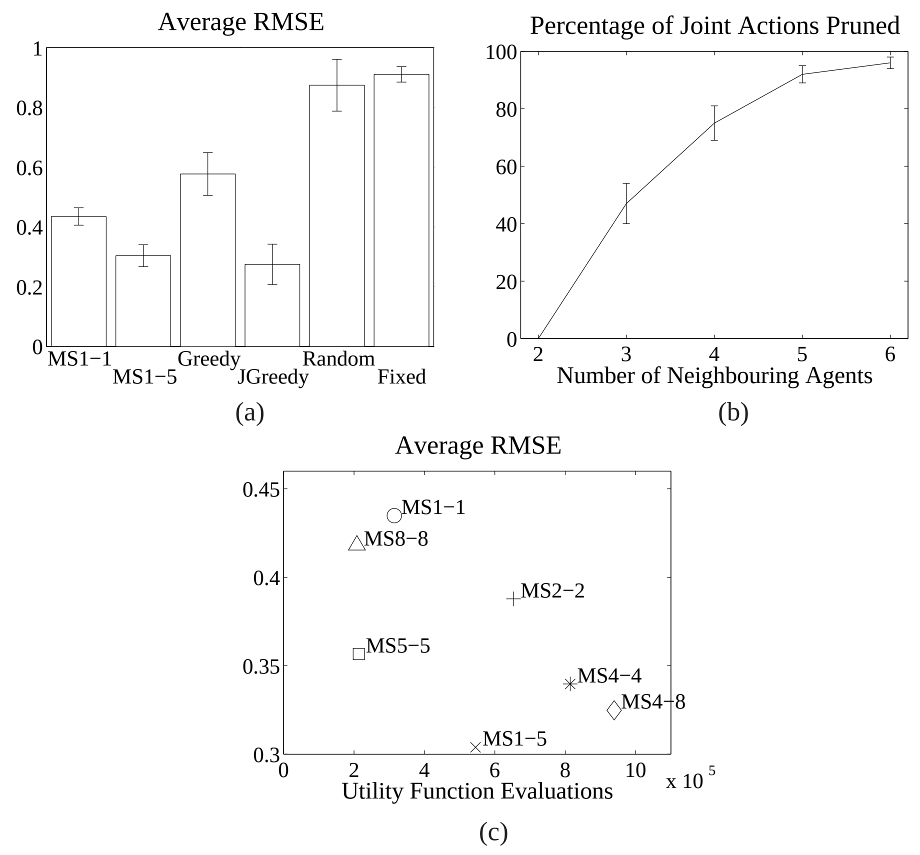


Figure 14: Empirical results for the pruning algorithms showing (a) the average root mean squared error (RMSE), (b) the percentage of joint actions pruned, and (c) the number of utility function evaluations plotted against the average root mean squared error achieved. Error bars indicate the standard error of the mean.

function needs to be evaluated for only 8% of roughly 8^5 joint actions, thus greatly improving the algorithm's efficiency.

In the third experiment, we performed a cost/benefit analysis of various MSm-l strategies. More specifically, we examined the effect of varying m and l on both the number of utility function evaluations, and the resulting RMSE. Figure 14(c) shows the results. The results of MS1-1, MS2-2, MS4-4, MS5-5, and MS8-8 show an interesting pattern. Up to and including $m = l = 4$, both the number of function evaluations and the average RMSE decrease. This is due to the fact that planning longer paths is more expensive, but results in lower RMSE. However, for $m, l > 4$, the action space becomes too coarse (since only 8 directions are considered) to maintain a low RMSE. At the same time, the number of times the agents coordinate reduces significantly, resulting in a lower number of function evaluations. Finally, MS1-5 and MS4-8 provide a compromise; they compute longer paths, but coordinate more frequently. This leads to more computation compared to MS5-5 and MS8-8, but results in significantly lower RMSE, because agents are able to 'reconsider' their paths midway.

6.4.2 Empirical Evaluation of the Bounded Max-Sum Algorithm

In the previous set of experiments, we focused exclusively on the effectiveness of the pruning algorithms in a setting where the evaluation of utility function is computationally demanding. In the second set, we combined the pruning algorithms with the bounded max-sum algorithm presented in Section 4 to determine whether (i) the optimal solution is preserved by using the pruning algorithms, and (ii) to determine the solution quality provided by the bounded max-sum algorithm compared to the optimal solution computed by enumerating the entire joint action-space. The latter presents an empirical estimation of the approximation computed by this algorithm in a realistic and demanding setting.

In more detail, these experiments used the same environment layout and GP settings as before. During each simulation, which lasted for 200 timesteps, the sensors computed paths of length 8 at 4 timestep intervals (i.e. MS4-8). We performed simulations with 3, 4, 5, and 6 sensors, both *with* and *without* the action pruning algorithm from Section 5.1, starting the sensors from random locations in each run. In what follows, the simulation with M sensors, and pruning turned on is denoted by PM, and with pruning turned off, by NPM. In both case, the joint action pruning algorithm from Section 5.2 was always used.

Figure 15(a) shows the average utility obtained during 200 timesteps of the simulation over twenty repetitions. In particular, it reports the four metrics described in Section 4.4. The results show that the solution computed using the bounded max-sum algorithm is very close to the optimal solution. More specifically, the minimum ratio between the computed solution and the optimal solution over 160 runs was 98%, thus showing that the use of this algorithm leads to very good approximations to the optimal solution. Moreover, the graph shows that the use of the pruning techniques results in a slight tightening of the bound on the optimal solution. More importantly, it corroborates the theoretical claim that the optimal solution is preserved when applying the pruning algorithms (see Section 5.1).

Figure 15(b) shows the benefits of applying the two pruning algorithms more clearly. This figure includes four key metrics:

1. Cache misses: the number of times the utility functions actually needed to be evaluated for different joint actions.
2. The total number of function calls.
3. The number of nodes that needed to be expanded in the search tree (described in Section 5.2) to find the optimal value.
4. The total number of nodes that the full search tree contains.

From this figure, we note that the action pruning approach described in Section 5.1 results in reductions in all four of these metrics of approximately one order of magnitude. By removing dominated action choices, the coordination problem is simplified and agents need perform less evaluations of the costly utility function. Furthermore, we note that the joint action pruning algorithm described in Section 5.2 results in a reduction in the number of nodes of the search tree that must be expanded, compared to the total number of nodes in the tree, of up to two orders of magnitude.[24]

[24]Note that in settings where the computational cost of performing the utility function evaluation dominants other processing, this will translate into a significant runtime improvement. However, in general, the runtime of the algorithm will also depend on many other domain specific factors (such as the computational resources of the agents, and even the communication delays as they exchange messages).

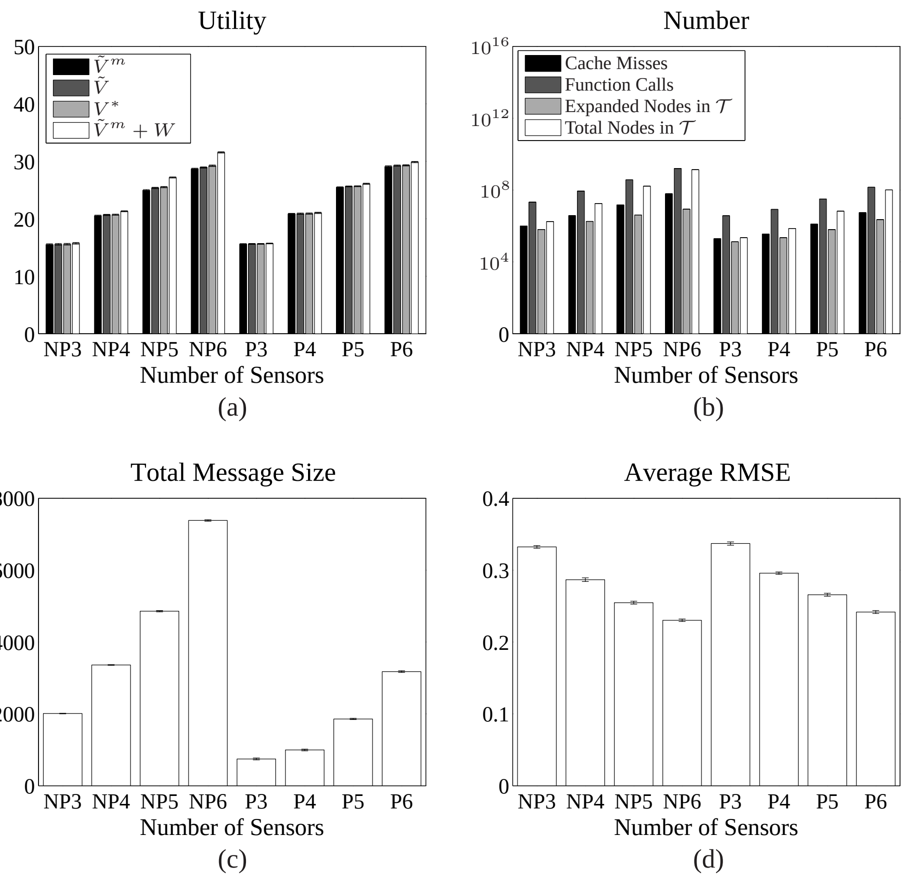


Figure 15: Empirical results for the bounded max-sum algorithm showing (a) the average utility, (b) the number of cache misses, the total number of utility function calls, the number of expanded partial joint actions (expanded nodes in search tree $\mathcal{T}$), and the maximum number of partial joint actions that could have been evaluated (total number of nodes in search tree $\mathcal{T}$), (c) the average root mean squared error (RMSE), (d) the total message size in terms of the number of values exchanged. PM means that M sensors are deployed, and that the action pruning algorithm from Section 5.1 is used, and NPM means M sensors without action pruning. Error bars indicate the standard error of the mean.

Figure 15(c) shows the required amount of communication needed for coordination. The most notable conclusion that can be drawn from this figure is the strong reduction in message size when the action pruning algorithm is used. Since the action space of individual sensors is reduced by pruning dominated actions, the number of values contained in the messages exchanged between functions and variables (Equations 2 and 3) is significantly reduced, resulting in a lower communication overhead.

Finally, Figure 15(d) shows the obtained solution quality in terms of the average RMSE. Unlike the utility reported in Figure 15(a), this figure shows a slight decrease in solution quality when using the pruning algorithms. This is caused by the fact that the sensors minimise entropy in their environment, which, despite being strongly linked, does not directly translate into a decrease of RMSE. Put differently, directed by the utility function that incentivises entropy reduction, the pruning technique in Section 5.1 that operates on the action space of individual agents, prunes actions that would have led to lower RMSE. Fortunately, this effect is limited, as the maximum increase of RMSE found over 160 runs was only 3.5%.

In summary, in this section, we demonstrated the effectiveness of the bounded max-sum algorithm and the two pruning algorithms in a setting where the utility functions are computationally expensive to evaluate. We showed that, by using the two pruning algorithms, the number of function evaluations is reduced by roughly two orders of magnitude for a joint action space of size 8^6. Moreover, the results showed that by using the bounded max-sum results, we obtain solutions that are guaranteed to be no further away than 2% from the optimal solution. Generalising from these specific results, these experiments clearly show the effectiveness of the developed techniques for real-life applications with complex interactions between agents.

7 Related Work

As described in the introduction, approximation ratios have previously been provided for k-optimal algorithms in the area of DCOPs [15, 16]. In this case, the k-optimal solution for a DCOP is a solution that cannot be improved by changing the assignment of any k or less variables. Many well known local algorithms for DCOPs are guaranteed to provide k-optimal solutions. In particular, any locally hill climbing algorithm is k-optimal for $k = 1$, such as for example DSA [1] and MGM [10]. While, a $k = 2$ variant of MGM, termed MGM-2, has been presented [10]. Specifically, Pearce and Tambe provide approximation ratios which are valid for any DCOP with non-negative reward structure, and which are dependent on the arity of the constraint functions, the number of agents participating in the DCOP and the value of k. Moreover, they provide both general bounds which are not dependent on the constraint graph structure, and tighter bounds for specific structures (e.g. ring and star graph structures). More recently, Bowring et al. have improved on this bound by assuming *a priori* information concerning the DCOP reward structure [16]. With respect to this work, our approach is somewhat complementary, as here we provide an approximation ratio which is more accurate but is dependent on the specific problem instance, while their approach provides a less accurate bound which, in turn, is more general. In more detail, the approximation ratio provided within the k-optimality framework is dependent on the number of agents and thus scales poorly when the number of agents in the system grows.[25] Conversely, our approximation ratio is dependent on the reward structures (because it requires functions to be bounded), and on the constraint graph structure (because it is more accurate when less cycles are present in the constraint network). However, it is not dependent on the number of agents present in the system and, since it exploits the specific constraint graph structure, it is able to provide very accurate approximation ratios.

An alternative approximation approach, proposed by Yeoh et al., is based on the ADOPT algorithm, and its extension BnB-ADOPT [19]. This algorithm fixes a predetermined error bound for the optimal solution, and stops when a solution that meets this error bound is found. Their approach is similar to our work in that it is dependent on the problem instance. Specifically, in their case, the error bound is fixed and the algorithm will stop only when such a bound is obtained. The number of cycles required by

[25]See the discussion in Section 4.4 for a more detailed comparison of our approach with [16].

the algorithm to converge is dependent on the particular problem instance, and, in the worst case, remains exponential. Our approach in contrast, is guaranteed to converge after a polynomial number of cycles (i.e., twice the depth of the tree structured factor graph), but the approximation ratio is dependent on the particular problem instance. Therefore, our approach tries to minimise computation and communication, by trading off solution quality. This requirement is driven by our focus on decentralised coordination for embedded agents, where constraints on communication and computation are crucial for the practical applicability of the coordination approach.

Similar considerations hold with respect to A-DPOP [18], an extension of the DPOP algorithm that computes approximate solutions. A-DPOP attempts to reduce message size (which is exponential in the original DPOP algorithm in the width of the pseudo tree) by optimally computing only a part of the messages and approximating the rest (with upper and lower bounds). Given a fixed approximation ratio, A-DPOP can then reduce message size to meet this ratio, or alternatively, given a fixed maximum message size, it propagates only those messages that do not exceed that size. As a result of this, the computed solution is not optimal, but approximate. Moreover, as discussed above, since the algorithm fixes a desired approximation ratio, the message size remains exponential. In contrast, if we would fix the maximum message size in our approach, the approximation ratio is dependent on the specific problem instance. Furthermore, note that in the A-DPOP case, there is no mechanism to minimise the approximation ratio, which in our approach is provided by considering the maximum spanning tree of the constraint network.[26]

Our use of tree structures to obtain an approximation of the original problem shares similarities with previous work in information theory where a dependence tree is used to approximate a generic joint probability distribution of random discrete variables. In particular, it has been shown that a maximum weight dependence tree provides the best tree approximation of the joint probability distribution [42]. In contrast, our contribution addresses a decentralised decision problem as opposed to a centralised tree parametrisation of an unknown joint probability. Consequently, 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 [42]. Techniques based on tree-decomposition have also been previously used in the area of constraint optimisation. In particular, in [43] 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.[27] In contrast, here we focus on removing cycles from the original problem instance to optimise the approximation ratio, maintaining a low communication and computation overhead, because we focus on decentralised coordination for resource constrained embedded agents.

Finally, our action pruning approach shares some similarities with the directed soft arc consistency approach proposed by Matsui et al. [44]. This work proposes a distributed algorithm to perform directed soft arc consistency on pseudo-trees, and shows that this approach can be efficiently combined with common search algorithms (e.g., ADOPT). Our action pruning approach has a similar spirit as it is a distributed prepro-

[26]If we force A-DPOP to have polynomial message size, as it is the case with our approach, the algorithm would compute a DFS tree and remove all other edges, without considering the impact of removed dependencies on the approximation ratio.

[27]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 [43] for further details.

cessing scheme that results in a faster algorithm. Moreover it is somewhat similar to standard arc consistency for constraint satisfaction problems as it tries to delete useless values from variable domains. However, our action pruning approach, in contrast to the method presented in [44], does not try to reduce the range of the values of the functions. This is motivated by the fact that max-sum is not a search algorithm and so it would not benefit from such reduction as it is the case with search algorithms such as ADOPT. Here, we are more concerned to reduce the number of actions that each functions must consider when computing the messages of the max-sum algorithm as this is the main source of algorithm's computational complexity. Nonetheless, applying soft arc consistency could potentially result in a better approximation ratio for the bounded max-sum, as reducing the range of the cost functions could result in smaller weights. However further investigations would be required to see whether the pre-processing overhead to apply directed soft arc consistency would be worth the possible reduction in the approximation ratio, and this falls outside the scope of the current paper.

8 Conclusions and Future Work

In this paper we proposed a novel approach to decentralised coordination which is particularly suited for embedded computationally constrained agents. Our approach is based on a factor graph representation of the constraint network (i.e. the interactions between agents) and builds on the max-sum algorithm. Our approach guarantees accurate bounded approximate solutions, while maintaining a very low computation and communication overhead. Given any particular instance of a general constraint network, our approach is able to compute a solution and to provide an approximation ratio for the unknown optimal solution. This is achieved without incurring the typical exponential cost of optimal approaches, thus resulting in a very effective and efficient technique. Moreover, by applying two novel generic pruning techniques, we are able to reduce the computation that each agent must perform when computing the approximate solution, thus further improving the computational efficiency of our approach.

We apply our approach in a mobile sensor domain to assess its practical benefits. In this domain mobile sensors must coordinate their actions to gather the most informative measurements from the environment. In this setting, we develop a factor graph representation for this specific coordination problem, and show how our approach can be used as a solution technique. Moreover, we show how the two developed pruning techniques can be used in this specific domain to further speed up the max-sum message computation. Empirical results showed that our novel technique is extremely effective, providing accurate solutions which are guaranteed to be no further away than 2% from the optimal. Moreover, the use of the pruning techniques proved to be very successful in speeding-up the computation of the max-sum message: for 5 sensors, these techniques prune 92% of joint moves, thus significantly reducing the number of utility function evaluations, which are particularly expensive in our domain.

Many possible future directions stem from this work. A first interesting research direction is to investigate techniques to further reduce the approximation ratio. A possible approach is to iteratively apply our algorithm while clustering variable and function nodes (as proposed in [32]) 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. In this respect, it would also be interesting to consider the use of state of the art techniques for constraint satisfaction. As mentioned in the previous section, soft arc consistency could be applied, as proposed in [44], to preprocess the constraint network before applying our approach. Another interesting possibility would be to investigate the use of cutset schemes to obtain tree-structured network [25]. For example, the cycle cutset decomposition could be used to completely remove cycles, or alternatively, more general cutset schemes (b-cutset) could be exploited to obtain constraint networks with a bounded induced width. A similar approach has already been succesfully used in [13], and in this context, our idea of finding a maximum-weight spanning tree could be used as an heuristic to choose the cutset variables.

A second interesting direction is to investigate the use of region based message passing techniques, such as the family of generalised belief propagation approaches [45], as solution techniques for our constraint network. GBP is a generalisation of the standard GDL techniques (such as max-sum) and operates on a *region graph*, which is obtained by dividing the factor graph into specific regions based on the factor graph topology.[28] Messages are then computed for regions and sent from one region to another. Recent empirical results show that region based techniques such as GBP are able to outperform standard GDL techniques, with a minimal extra cost in terms of computation. Moreover, GBP has similar guarantees on solution optimality as the standard GDL techniques, namely it is optimal when the region based graph does not contain cycles [45]. Therefore, investigating possible extensions of our bounded max-sum algorithm to GBP techniques appears to be a promising direction. More generally speaking, the application of GDL-based techniques to decentralised coordination appears to be a very promising direction, resulting in effective and efficient solutions. Furthermore, many important aspects which are specific to the coordination of embedded agents still need to be investigated. For example, agents usually have heterogeneous computation and communication capabilities, and this could potentially be taken into account when assigning the responsibility for variables and factor node computation to the different agents in order to better exploit the limited resources of the system.

Finally, an important research direction is to go beyond the limited look-ahead used here for coordinating the paths of the mobile sensors, and investigate the use of sequential decision making approaches. The sequential aspect is inherent to a wide variety of applications involving embedded agents, however a key issue is to keep the computational costs under control. To this end, the factorisation of the objective function seems to be a promising idea, and again, GDL-based approaches appear to be very well suited solution techniques.

Acknowledgements

This paper is a significantly extended version of paper [27]. The work reported in this paper was jointly funded by the Systems Engineering for Autonomous Systems (SEAS) Defence Technology Centre established by the UK Ministry of Defence and the ALADDIN (Autonomous Learning Agents for Decentralised Data and Information Systems) project; a BAE Systems and EPSRC (Engineering and Physical Research Council) strategic partnership (EP/C548051/1).

[28]A region is formed by a sub-set of factor nodes and all variable nodes that are connected to them. A region usually includes short loops in the factor graph to have good approximations.

A Proof of Bounded Approximation

To prove theorem 1 we first introduce and prove the following lemma:

Lemma 1.
$$\forall i, \mathbf{x} \ \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) + B_i(\mathbf{x}_i^c) \geq \max_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) \tag{18}$$

Proof of Lemma 1. Expanding $B_i(\mathbf{x}_i^c)$ as defined in Equation 6 we have:

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

where $\mathbf{x}_i^t = \mathbf{x}_i \setminus \mathbf{x}_i^c$ as in Section 4. By contradiction, let us consider an assignment ${\mathbf{x}''}_i^t$ such that:

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

We can rewrite the previous expression as:

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

where

$${\mathbf{x}'}_i^t = \arg\max_{\mathbf{x}_i^t} \left[\max_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) - \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) \right]$$

However, this is a contradiction with respect to the definition of ${\mathbf{x}'}_i^t$. Therefore Lemma 1 must hold. □

Lemma 2.
$$B \leq W \tag{19}$$

Proof of Lemma 2. We can rewrite the above inequality as:

$$\sum_i B_i(\mathbf{x}_i^c) \leq \sum_{<i,j> \in C} w_{ij}$$

where C is the set of couples of indices $< i, j >$ that identify the edges removed from the factor graph. We can prove this inequality by showing that $\forall i \ B_i(\mathbf{x}_i^c) \leq \sum_{j \in I(\mathbf{x}_i^c)} w_{ij}$ where $I(\mathbf{x}_i^c)$ is the set of variable indexes which have been removed. We proceed by expanding $B_i(\mathbf{x}_i^c)$ as defined in Equation 6 and w_{ij} as defined in Equation 5 to give:

$$\max_{\mathbf{x}_i^t} \left[\max_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) - \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^t; \mathbf{x}_i^c) \right] \leq \sum_j \max_{\mathbf{x}_i \setminus j} \left[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i) \right]$$

For functions that have no dependencies removed we have $\mathbf{x}_i^c = \emptyset$, therefore $B_i(\emptyset) = 0$. Consequently, $I(\mathbf{x}_i^c) = \emptyset$, and thus, since the sum of the weight will be zero, the above inequality holds. For functions that have at least one dependency removed, we can substitute the left term of the above inequality with:

$$\max_{\mathbf{x}_i^t} \left[\max_j \left[\max_{\mathbf{x}_i^c \setminus j} \left[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i) \right] \right] \right]$$

However, this term is less than or equal to:

$$\max_{\mathbf{x}_i^t}\left[\sum_j \max_{\mathbf{x}_i^c \setminus j}\left[\max_{x_j} F_i(\mathbf{x}_i) - \min_{x_j} F_i(\mathbf{x}_i)\right]\right]$$

which, in turn, is less than or equal to the right hand side of our original expression. □

Theorem. *Bounded Approximation*

$$\sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i) + B \geq \sum_i F_i(x_i^*) \tag{20}$$

Proof of Theorem Bounded Approximation. By considering the definition of $\tilde{\mathbf{x}}$ given in Equation 7, we can write the following inequality:

$$\sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i) \geq \sum_i \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^*)$$

This inequality holds because $\tilde{\mathbf{x}}$ is defined as the assignment that maximises the problem on the tree structured network, and thus, the value given by that assignment on the tree structured problem will be higher or equal than any other possible assignment. Specifically, it will be greater or equal than the optimal assignment of the original problem $\mathbf{x}^*$, and thus, we can write:

$$\sum_i \min_{\mathbf{x}_i^c} F_i(\tilde{\mathbf{x}}_i) + B \geq \sum_i \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^*) + B$$

adding the same quantity B to both terms of the equation. Then using Lemma 1 we know that:

$$\sum_i \min_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^*) + B \geq \sum_i \max_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^*)$$

Now, since:

$$\sum_i \max_{\mathbf{x}_i^c} F_i(\mathbf{x}_i^*) \geq \sum_i F_i(x_i^*)$$

our bounded approximation holds. □

B Derivation of worst case approximation ratio

We derive here the worst case approximation ratio $\rho = M/m$ starting from the ρ_{FG}.

Derivation of worst case approximation ratio. The worst case, for ρ_{FG}, happens when the optimal solution on the spanning tree equals the optimal solution evaluated on the original graph. This is the worst case because the approximation ratio is directly dependent on $\tilde{V}^m$ and inversely dependent on $\tilde{V}$, but we know that $\tilde{V}^m \leq \tilde{V}$. Intuitively, this is the worst case since in this case we overestimate the impact of the removed dependency the most (i.e., the actual impact is zero for the computed solution). When $\tilde{V}^m = \tilde{V}$ we have $\rho = 1 + B/\tilde{V}^m$. Moreover, let us denote the maximum ratio between the minimum reward and maximum reward across all functions with M/m where m and M are the minimum and maximum reward for the function that maximises the reward ratio and m_i and M_i are the minimum and maximum reward for function i. We can then write:

$$\rho = 1 + \frac{B}{\sum_i^{|\mathbf{F}|} m_i} = 1 + \frac{\sum_i^{|\mathbf{F}|}(M_i - m_i)}{\sum_i^{|\mathbf{F}|} m_i}$$

This is because we assume, being a worst case analysis, that all functions need to have at least one dependency cut, therefore $\tilde{V}^m = \sum_i^{|\mathbf{F}|} m_i$ and that for all functions the dependencies we cut have the highest possible impact on the solution quality (i.e. $\forall i\ B_i(\mathbf{x}_i^c) = M_i - m_i$). Then we have $\rho = \frac{\sum_i^{|\mathbf{F}|} M_i}{\sum_i^{|\mathbf{F}|} m_i} \leq M/m$.

$\square$

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., Martinez, K., Jennings, N.R.: A utility-based adaptive sensing and multi-hop communication protocol for wireless sensor networks. ACM Transactions on Sensor Networks **6**(3) (2010) Article 27

[3] Rogers, A., Corkill, D.D., Jennings, N.R.: Agent technologies for sensor networks. IEEE Intelligent Systems **24**(2) (2009) 13–17

[4] Chapman, A., Rogers, A., Jennings, N.R., Leslie, D.: A unifying framework for iterative approximate best response algorithms for distributed constraint optimisation problems. The Knowledge Engineering Review (2011) In Press

[5] Modi, P.J., Shen, W., Tambe, M., Yokoo, M.: ADOPT: Asynchronous distributed constraint optimization with quality guarantees. Artificial Intelligence (161) (2005) 149–180

[6] 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. (2004) 438–445

[7] Petcu, A., Faltings, B.: DPOP: A scalable method for multiagent constraint optimization. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence. (2005) 266–271

[8] Chechetka, A., Sycara, K.: No-commitment branch and bound search for distributed constraint optimization. In: Proceedings of Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems. (2006) 1427 – 1429

[9] Gershman, A., Meisels, A., Zivan, R.: Asynchronous forward bounding for distributed cops. Journal of Artificial Intelligence Research **34** (2009) 61–88

[10] Maheswaran, R.T., Pearce, J.P., Tambe, M.: Distributed algorithms for DCOP: A graphical game-based approach. In: Proceedings of the Seventeenth International Conference on Parallel and Distributed Computing Systems. (2004) 432–439

[11] Zivan, R.: Anytime local search for distributed constraint optimization. In: Proceedings of the Twenty-Third Conference on Artificial Intelligence. (2008) 393–398

[12] Farinelli, A., Rogers, A., Petcu, A., Jennings, N.R.: Decentralised coordination of low-power embedded devices using the max-sum algorithm. In: Proceedings of the Seventh International Conference on Autonomous Agents and Multiagent Systems. (2008) 639–646

[13] Petcu, A., Faltings, B.: MB-DPOP: A new memory-bounded algorithm for distributed optimization. In: Proceedings of the Twentieth International Joint Conference on Artificial Intelligence. (2007) 1452–1457

[14] Yeoh, W., Felner, A., Koenig, S.: BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems. (2008) 591–598

[15] Pearce, J.P., Tambe, M.: Quality guarantees on k-optimal solutions for distributed constraint optimization problems. In: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence. (2007) 1446–1451

[16] Bowring, E., Pearce, J., Portway, C., Jain, M., Tambe, M.: On k-optimal distributed constraint optimization algorithms: New bounds and algorithms. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems. (2008) 607–614

[17] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. The MIT press (2001)

[18] Petcu, A., Faltings, B.: A-DPOP: Approximations in distributed optimization. In: Principles and Practice of Constraint Programming (2005) 802–806

[19] Yeoh, W., Sun, X., Koenig, S.: Trading off solution quality for faster computation in DCOP search algorithms. In: Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence. (2009) 354–360

[20] Kok, J.R., Vlassis, N.: Collaborative multiagent reinforcement learning by payoff propagation. Journal Machine Learning Research **7** (2006) 1789–1828

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

[22] MacKay, D.J.C.: Information Theory, Inference, and Learning Algorithms. Cambridge University Press (2003)

[23] 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

[24] Weiss, Y., Freeman, W.: Correctness of belief propagation in gaussian graphical models of arbitrary topology. Neural Computation **13**(10) (2001) 2173–2200

[25] Dechter, R.: Constraint Processing. Morgan Kaufmann (2003)

[26] Vinyals, M., Rodríguez-Aguilar, J.A., Cerquides, J.: Constructing a unifying theory of dynamic programming DCOP algorithms via the generalized distributive law. Journal of Autonomous Agents and Multi Agent Systems (2010) 1–26

[27] Stranders, R., Farinelli, A., Rogers, A., Jennings, N.R.: Decentralised coordination of mobile sensors using the max-sum algorithm. In: Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence. (2009) 299–304

[28] Gallager, R.G., Humblet, P.A., Spira, P.M.: A distributed algorithm for minimum-weight spanning trees. ACM Transactions on Programming Languages and Systems **5**(1) (1983) 66–77

[29] Frey, B.J., Dueck, D.: Clustering by passing messages between data points. Science **315**(5814) (2007) 972–976

[30] Aji, S.M., Horn, G.B., Mceliece, R.J.: On the convergence of iterative decoding on graphs with a single cycle. In: Proceedings of the International Symposium on Information Theory. (1998) 276

[31] Weiss, Y.: Correctness of local probability propagation in graphical models with loops. Neural Computation **12**(1) (2000) 1–41

[32] 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

[33] Awerbuch, B.: Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (1987) 230–240

[34] 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

[35] Ali, S.M., Koenig, S., Tambe, M.: Preprocessing techniques for accelerating the DCOP algorithm adopt. In: Fourth International Joint Conference on Autonomous Agents and Multiagent Systems (2005) 1041-1048

[36] Meliou, A., Krause, A., Guestrin, C., Hellerstein, J.M.: Nonmyopic informative path planning in spatio-temporal models. In: Proceedings of the Twenty-Second Conference on Artificial Intelligence. (2007) 602–607

[37] Guestrin, C., Krause, A., Singh, A.P.: Near-optimal sensor placements in gaussian processes. In: Proceedings of the Twenty-Second International Conference on Machine Learning. (2005) 265–272

[38] Krause, A., Guestrin, C.: Near-optimal observation selection using submodular functions. In: Proceedings of the Twenty-Second Conference on Artificial Intelligence. (2007) 1650–1655

[39] Rasmussen, C.E., Williams, C.K.I.: Gaussian processes for machine learning. The MIT Press (2006)

[40] Osborne, M.A., Rogers, A., Ramchurn, S.D., Roberts, S.J., Jennings, N.R.: Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In: Proceedings of the Seventh International Conference on Information Processing in Sensor Networks. (2008) 109–120

[41] Low, K.H., Dolan, J.M., Khosla, P.: Adaptive multi-robot wide-area exploration and mapping. In: Proceedings of the Seventh International Conference on Autonomous Agents and MultiAgent Systems. (2008) 23–30

[42] Chow, C.K., Liu, C.N.: Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory **14**(3) (1968) 462–467

[43] Dechter, R., Kask, K., Larrosa, J.: A general scheme for multiple lower bound computation in constraint optimization. Principles and Practice of Constraint Programming **2239** (2001) 346–360

[44] Matsui, T., Silaghi, M.C., Hirayama, K., Yokoo, M., Matsuo, H.: Directed soft arc consistency in pseudo trees. In: Proceedings of The Eighth International Conference on Autonomous Agents and Multiagent Systems. (2009) 1065–1072

[45] Yedidia, J., Freeman, W., Weiss, Y.: Constructing free-energy approximations and generalized belief propagation algorithms. IEEE Transactions on Information Theory **51**(7) (2004) 2282–2312