Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm

A. Farinelli*, A. Rogers*, A. Petcu†, N. R. Jennings*

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

† AI Laboratory (LIA), Swiss Federal Institute of Technology in Lausanne, CH-1015 Lausanne, Switzerland.
adrian.petcu@epfl.ch

ABSTRACT

This paper considers the problem of performing decentralised coordination of low-power embedded devices (as is required within many environmental sensing and surveillance applications). Specifically, we address the generic problem of maximising social welfare within a group of interacting agents. We propose a novel representation of the problem, as a cyclic bipartite factor graph, composed of variable and function nodes (representing the agents' states and utilities respectively). We show that such representation allows us to use an extension of the max-sum algorithm to generate approximate solutions to this global optimisation problem through local decentralised message passing. We empirically evaluate this approach on a canonical coordination problem (graph colouring), and benchmark it against state of the art approximate and complete algorithms (DSA and DPOP). We show that our approach is robust to lossy communication, that it generates solutions closer to those of DPOP than DSA is able to, and that it does so with a communication cost (in terms of total messages size) that scales very well with the number of agents in the system (compared to the exponential increase of DPOP). Finally, we describe a hardware implementation of our algorithm operating on low-power Chipcon CC2431 System-on-Chip sensor nodes.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence*

General Terms

Algorithms, Experimentation

Keywords

Sum-product, DCOP, sensor networks, coordination

1. INTRODUCTION

Increasing attention is currently being devoted to applications involving low-power, wireless devices that are deployed within the environment, and seek to acquire and integrate information. Relevant examples include rescue robotics [12] and the use of sensor networks for performing wide-area surveillance in security [5] and for monitoring environmental phenomena [10]. Within all these domains, a fundamental challenge is to coordinate the activities of the physically distributed devices in order to achieve good system-wide performance, given the specific constraints of each device (such as limited power, communication and computational resources), and the fact that each device can typically only communicate with the few other devices in its local neighbourhood (due to the use of low-power wireless transceivers, the small form factor of the device and antenna, and the hostile environments in which they are deployed). Additional challenges arise through the need to perform such coordination in a decentralised manner such that there is no central point of failure and no communication bottleneck, that the computation required for the coordination is shared over the distributed resources available, and that the solution scales well as the number of devices within the network increases.

Given this background, a multi-agent systems (MAS) approach is a natural one for modelling such autonomous decentralised systems, and within the academic literature of MAS, problems similar to those described above are often represented as *Distributed Constraint Optimisation Problems* (DCOPs). Furthermore, a number of complete algorithms that generate optimal solutions have been proposed for solving them: including OptAPO [4], ADOPT [8] and DPOP [11]. More specifically, OptAPO uses a partially centralised approach in which mediator agents compute solutions for portions of the overall problem. In contrast, both ADOPT and DPOP preprocess the constraint graph, arranging it into a *Depth First Search* (DFS) tree, and then exchange messages over this tree.

Now, while these algorithms represent significant contributions in their own domain, they do not address many of the additional challenges that are present when the agents correspond to embedded devices. In particular, optimality demands that some aspect of these algorithms is exponential. For example, within OptAPO mediator agents may be required to perform calculations that grow exponentially with the size of the portion of the overall problem that they are responsible for. Similarly, the number of messages that agents exchange when using ADOPT is exponential in the height of the DFS tree, and for PDOP, it is exponential in the width of the tree. Such exponential relationships are simply unacceptable for embedded devices that exhibit constrained computation, bandwidth and memory resources. For example, our work is motivated by the problem of coordinating the sense/sleep cycles of energy-harvesting sensor nodes within a network deployed for wide-area surveillance. Within this application, the requirement to operate for an indefinite lifetime imposes the use of extremely low-power devices, and thus we are using the Chipcon CC2431 System-on-Chip (see section 7 for further details). This device incorporates a RF transceiver whose message buffer is limited to just 8 kByte RAM, and thus, such optimal algorithms are unusable for all but

Cite as: Decentralised Coordination of Low-Power Embedded Devices Using the Max-Sum Algorithm, A. Farinelli, A. Rogers, A. Petcu and N. R. Jennings, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 639-646.

the very smallest of networks. Furthermore, this RF transceiver exhibits lossy communication such that messages may be lost, and nodes may be temporarily disconnected from the network. Thus, a practical decentralised coordination algorithm must also be robust to this lossy communication, and only limited results exist for these optimal algorithms [7].

In contrast, a large number of approximate stochastic algorithms have also been proposed for solving DCOPs. These algorithms are typically based upon entirely local computation, whereby each agent updates its state based only on the communicated (or observed) states of those local neighbours that influence its utility. As such, these approaches are well suited for large scale distributed applications, and in this context, the Distributed Stochastic Algorithm (DSA) is one of the most promising; having been proposed for decentralised coordination within sensor networks [1] and benchmarked on DCOP problems [15]. However, algorithms of this type often converge to poor quality solutions since agents do not explicitly communicate their utility for being in any particular state, but only communicate their preferred state (i.e. the one that will maximise their own utility) based on the current preferred state of their neighbours.

Thus, against this background, there is a clear need for coordination algorithms that make efficient use of constrained computational and communication resources, and yet are able to effectively represent and communicate complex utility relationships through the network. It is this requirement that we address in this paper, and to this end, we present an approximate decentralised solution that can be applied to the general problem of maximising the social welfare of a group of agents (i.e. maximising the sum of the utilities of each individual agent) when the utility of any individual agent is dependent on its own state, and the state of a number of interacting neighbours. Our solution is based upon message passing techniques that are frequently used in the context of information theory to decompose complex computations on single processors. However, they have never previously been exploited as a basis for multi-agent coordination. In particular, we exploit the extensive evidence that demonstrates that the sum-product algorithm (and its derivative, the max-sum algorithm) generate good approximate solutions when applied to cyclic graphs (in the context of approximate inference through 'loopy' belief propagation on Bayesian networks [9], iterative decoding of practical error correcting codes [2], and solving large scale K-SAT problems involving thousands of variables [6]), due to their ability to propagate information around the network such that they converge to a *neighborhood maximum*, rather than a simple local maximum [14].

Thus, in more detail, this paper makes the following contributions:

1. We show that a novel representation of the social welfare maximisation problem, as a cyclic bipartite factor graph composed of variable and function nodes (representing the agents' states and utilities), allows us to use an extension of the max-sum algorithm to generate approximate solutions to this global optimisation problem through decentralised local message passing between interacting agents.
2. We empirically evaluate this approach on a canonical coordination problem (graph colouring). We benchmark our approach against local best response, a state of the art approximate algorithm (DSA), and a complete algorithm (DPOP). We show that our algorithm converges to better solutions than the approximate algorithms (showing up to 1/5 of the total number of conflicts over time), exhibits a near linear increase in total message size as the number of agents increases (compared to the exponential increase of DPOP), and is robust to lossy communication (performing well when even 90% of messages are dropped).
3. We prove the practical application of our approach by describing how we have implemented the graph colouring problem benchmarked above in hardware on the Chipcon CC2431 System-on-Chip (a wireless embedded device developed to form the core of future low-power sensor nodes).

Our results indicate that our formalism provides a resource efficient means to perform decentralised coordination within networks of low-power embedded devices.

The remainder of this paper is structured as follows: in section 2 we formally describe the social welfare maximisation problem that we tackle. In section 3 we describe message passing algorithms in general, before describing our decentralised coordination algorithm in section 4. In section 5 we describe our graph colouring benchmark, and then present our empirical evaluation in section 6. In section 7 we describe a hardware implementation of our algorithm, before concluding in section 8.

2. PROBLEM DESCRIPTION

As described above, our goal is to solve general coordination problems, and thus, we consider the general case in which there are M agents, and the state of each agent may be described by a discrete variable x_m. Each agent interacts locally with a number of other agents such that the utility of an individual agent, $U_m(\mathbf{x}_m)$, is dependent on its own state and the states of these other agents (defined by the set $\mathbf{x}_m$). We make no assumptions regarding the structure of the individual utility functions and there is no requirement that they are known to other agents.

Within this setting, we wish to find the state of each agent, $\mathbf{x}^*$, such that social welfare of the whole system (i.e. the sum of the individual agents' utilities) is maximised:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \sum_{m=1}^{M} U_m(\mathbf{x}_m) \tag{1}$$

Furthermore, in order to enforce a truly decentralised solution, we assume that each agent only has knowledge of, and can directly communicate with, the few neighbouring agents on whose state its own utility depends. In this way, the complexity of the calculation that the agent performs depends only on the number of neighbours that it has (and not the total size of the network), and thus, we can achieve solutions that scale well.

Given this problem description, we now describe a general class of algorithms which have previously been used to solve similar problems, before presenting the details of our formalism for addressing the specific problem at hand.

3. THE MESSAGE PASSING APPROACH

The fundamental algorithm on which we base our formalism is commonly used in the context of information theory in order to decompose complex calculations by exploiting the fact that the functions being handled can be factorised (i.e. expressed as the product of simpler expressions) [3]. In particular, we focus on the sum-product algorithm, and then describe how derivatives of this algorithm (specifically the max-sum algorithm) can be used to maximise social welfare within a decentralised agent system.

3.1 The Sum-Product Algorithm

Let us initially consider a function, F, that is dependent on N variables, $\mathbf{x} = \{x_1 \dots x_N\}$, and is defined as a product of M factors[1],

[1]Note that this is the same as the number of agents above, and the reason for this will become clear in section 4.

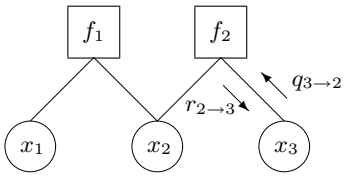

Figure 1: Example acyclic factor graph for the function $F = f_1(x_1, x_2)f_2(x_2, x_3)$, squares represent function nodes and circles variable nodes.

such that:

$$F(\mathbf{x}) = \prod_{m=1}^{M} f_m(\mathbf{x}_m) \tag{2}$$

where each of the factors $f_m(\mathbf{x}_m)$ is a function of a subset $\mathbf{x}_m$ of the variables that make up $\mathbf{x}$. A function of this form can be represented using a *factor graph*, which is a bipartite graph composed of two kinds of element: variable nodes and function nodes. Variable nodes are connected only to function nodes and function nodes are only connected to variable nodes. For example, figure 1 shows a bipartite graph representation for the function $F = f_1(x_1, x_2)f_2(x_2, x_3)$.

This factor graph representation can be applied in many settings. For example, to represent a graphical model in probability theory where each factor represents a probability distribution over the variables to which it is connected, or an error correcting code where variables and functions represent source bits and parity-check bits [3]. A common requirement in these settings (in order to calculate the marginal probability of a variable within a graphical model or to calculate the probability that a particular code word was transmitted) is to calculate the marginal function, $z_n(x_n)$, that describes the total dependency of the global function $F(\mathbf{x})$ on variable x_n. It is given by:

$$z_n(x_n) = \sum_{\{x_{n'}\}, n' \neq n} F(\mathbf{x}) \tag{3}$$

The sum-product algorithm provides an efficient local message passing procedure to compute the marginal functions of all variables simultaneously. It does so by iteratively propagating messages along the edges of the corresponding factor graph. These messages take two forms: messages from functions to variables, $r_{n\to m}$, and messages from variables to functions, $q_{m\to n}$. The messages are specified as follows:

From variable to function:

$$q_{n\to m}(x_n) = \prod_{m' \in M(n)\setminus m} r_{m'\to n}(x_n) \tag{4}$$

From function to variable:

$$r_{m\to n}(x_n) = \sum_{\mathbf{x}_m \setminus n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)\setminus n} q_{n'\to m}(x_{n'}) \right) \tag{5}$$

where $\mathcal{N}(m)$ is the set of indexes of variables connected to the function f_m, $\mathcal{M}(n)$ is the set of indexes of functions connected to the variable x_n, and finally $\mathbf{x}_m \setminus n \equiv \{x_{n'} : n' \in \mathcal{N}(m) \setminus n\}$.

For cycle-free graphs (e.g. trees) the sum-product algorithm calculates the exact marginal function for each variable, and has an efficient update rule in which the messages converge after a number of steps equal to the diameter of the graph [3]. Following this update rule, leaf nodes (i.e. nodes which have only one neighbour) initiate the process by sending to all their parents an identity message. Specifically, each leaf variable node, x_n, sends a message $q_{n\to m}(x_n) = 1$, and each function leaf node, f_m, sends a message $r_{m\to n}(x_n) = f_m(x_n)$. Each time a node receives a message from an edge e, it computes outgoing messages based on equations 4 and 5, then sends the messages to all remaining edges. When a variable node has received a message from all its neighbours it can compute the exact marginal value according to the following equation:

$$z_n(x_n) = \prod_{m \in M(n)} r_{m\to n}(x_n) \tag{6}$$

In addition, to the procedural update rule described above, nodes may also be initialised randomly, and then update their outgoing messages at any time and in any sequence (i.e. asynchronously). In this case, the messages are still guaranteed to converge, and hence, the marginal values calculated using equation 6 also converge to the exact solution. The convergence time in this case is again proportional to the diameter of the graph [3].

3.2 The Max-Product Algorithm

The max-product algorithm is a derivative of the sum-product that computes the values of $\mathbf{x}$ that maximise the function $F(\mathbf{x})$. Since our final goal is to maximise social welfare, the max-product algorithm is more useful within our formalism. It is obtained by simply replacing the summation function in the expression for the 'function to variables' messages by a maximum function:

From variable to function:

$$q_{n\to m}(x_n) = \prod_{m' \in M(n)\setminus m} r_{m'\to n}(x_n) \tag{7}$$

From function to variable:

$$r_{m\to n}(x_n) = \max_{\mathbf{x}_m \setminus n} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)\setminus n} q_{n'\to m}(x_{n'}) \right) \tag{8}$$

Crucially, now rather than calculating the marginal functions of any variable, the algorithm calculates $\arg\max_{\mathbf{x}} F(\mathbf{x})$, and the product of the messages flowing into any variable now represents:

$$z_n(x_n) = \max_{\mathbf{x}_m \setminus n} \prod_{m=1}^{M} f_m(\mathbf{x}_m) \tag{9}$$

such that if $\mathbf{x}^* = \arg\max_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x}_m)$ then each individual component is given by $x_n^* = \arg\max_{x_n} z_n(x_n)$. Thus, a global maximisation task is solved via local message passing, and again, this result is optimal for acyclic graphs.

3.3 Cyclic Graphs

While the algorithms described above are only *provable* optimal and guaranteed to converge when applied to acyclic graphs, it is common to apply them to cyclic graphs anyway. This is often called 'loopy belief propagation', and means that the aggregation of messages flowing into each variable now only represents an approximate solution to the maximisation problem:

$$z_n(x_n) \approx \max_{\mathbf{x}_m \setminus n} \prod_{m=1}^{M} f_m(\mathbf{x}_m) \tag{10}$$

The performance of the sum-product and max-product algorithms when applied to cyclic graphs has been the subject of much recent research within the areas of statistical physics and machine learning. While only limited theoretical results exist for the validity of

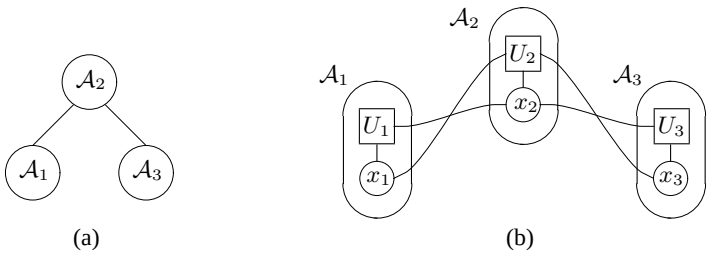

Figure 2: Diagram showing (a) the interactions of agents, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, and (b) the resulting cyclic factor graph.

this approach on graphs of arbitrary topology[2] [13, 14], there exists extensive empirical evidence of their effectiveness in practical problems. Examples include their use for the iterative decoding of error correcting codes [2], and their use within the Survey Propagation algorithm where they represent the state of the art approach to solving constraint satisfaction problems, and have been applied to K-SAT problems involving up to 10,000 variables [6].

Since messages now propagate in loops their value might grow indefinitely, and thus, we must normalise the variable to function message as they are updated (i.e. "on-the-fly" normalisation):

From variable to function:

$$q_{n\to m}(x_n) = \alpha_{nm} \prod_{m' \in M(n)\setminus m} r_{m'\to n}(x_n) \tag{11}$$

where α_{nm} is a scaler chosen such that

$$\sum_{x_n} q_{n\to m}(x_n) = 1 \tag{12}$$

This normalisation prevents us from explicitly calculating the value of $\max_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x}_m)$, however, we are still able to compute $\arg\max_{\mathbf{x}} \prod_{m=1}^{M} f_m(\mathbf{x}_m)$, which is all that we require to address the social welfare maximisation problem described in section 2.

4. THE MAX-SUM DECENTRALISED COORDINATION ALGORITHM

As described in section 2, we are seeking to solve a global social welfare maximisation problem in a decentralised manner through local message passing. In order to use the results of the previous section, we must first express this problem in terms of a factor graph, and to do so, we note that we can represent an agent as a function and a variable representing its state and utility. The utility of any agent is a function of its own state, and the state of a small number of neighbouring agents, and thus, the function node of a single agent is connected to its own variable node, and the variable nodes of a number of neighbouring agents.

For example, consider the case shown in figure 2a where three agents interact, and agent 1's utility is dependent on its own state and that of agent 2, agent 3's utility is dependent on its own state and the state of agent 2, and finally, agent 2's utility is dependent on its own state and that of agents 2 and 3. The equivalent factor graph is show in figure 2b, and in this case the social welfare is defined as $\sum_{m=1}^{3} U_m(\mathbf{x}_m) = U_1(x_1, x_2) + U_2(x_1, x_2, x_3) + U_3(x_2, x_3)$.

Having represented the social welfare maximisation problem as a factor graph, we can simply apply the max-product algorithm described in the previous section, in order to calculate the messages that should be exchanged between agents. More precisely, since in this case we are maximising a summation, rather than a product, we operate in the logarithm space, and thus, we use the a derivative of the max-product algorithm; the max-sum algorithm. Thus, we define $R_{m\to n} = \log r_{m\to n}$, $Q_{n\to m} = \log q_{n\to m}$, $Z_n(x_n) = \log z_n(x_n)$ and $U_m(\mathbf{x}_m) = \log f_m(\mathbf{x}_m)$. Using these identities and taking logarithms of both sides of the update rules of the max-product algorithm (with on-the-fly normalisation) shown in equations 11 and 12 allows us to write:

From variable to function:

$$Q_{n\to m}(x_n) = \alpha_{nm} + \sum_{m' \in M(n)\setminus m} R_{m'\to n}(x_n) \tag{13}$$

where α_{nm} is a scaler chosen such that:

$$\sum_{x_n} Q_{n\to m}(x_n) = 0 \tag{14}$$

From function to variable:

$$R_{m\to n}(x_n) = \max_{\mathbf{x}_m \setminus n} \left(U_m(\mathbf{x}_m) + \sum_{n' \in N(m)\setminus n} Q_{n'\to m}(x_{n'}) \right) \tag{15}$$

Again, we may calculate the sum of messages flowing into each variable in order to calculate the marginal function of each variable:

$$Z_n(x_n) = \sum_{m \in M(n)} R_{m\to n}(x_n) \tag{16}$$

Since $\log F(\mathbf{x}) = \sum_{m=1}^{M} \log f_m(\mathbf{x}_m) = \sum_{m=1}^{M} U_m(\mathbf{x}_m)$, these marginal functions represent solutions to the social welfare maximisation problem. However, under our representation the factor graphs that we deal with will almost always contain cycles, since the dependency between the agents is usually mutual (i.e. agent 1's state affects the utility of agent 2, and likewise, agent 2's state affects the utility of agent 1), and thus, they will be approximate solutions:

$$Z_n(x_n) \approx \max_{\mathbf{x}_m \setminus n} \sum_{m=1}^{M} U_m(\mathbf{x}_m) \tag{17}$$

By simply finding $\arg\max_{x_n} Z_n(x_n)$, each individual agent is thus able to determine which state it should adopt such that social welfare is maximised. Furthermore, note that the largest calculation that any agent performs (the update of the function to variable messages shown in equation 15) is exponential only in the number of neighbours that it has. This is typically much less than the total number of agents within the system, and thus the algorithm scales well as more agents are added to the system.

The cyclic nature of the factor graph that we face means that there is no need for a formal update schedule, and the agents may

[2]These results indicate that when the algorithm converges, it does not converge to a simple local maximum, but to a neighborhood maximum that is guaranteed to be greater than all other maxima within a particular large region of the search space. Depending on the structure of the factor graph, this neighborhood can be exponentially large [14].

simply randomly initialise their outgoing messages, and then update them whenever they receive an updated message from a neighbouring agent. Furthermore, since the calculation described in equation 17 can be performed at any time (using the most recent messages received), agents have a continuously updated estimate of their optimum state[3]. The final state of the algorithm depends on the structure of the agents' utility functions (as we shall show in section 6), and, in general, we observe three behaviours: i) The preferred states of all agents converge to fixed states that represent either the optimal solution, or a solution close to the optimal, and the messages also converge (i.e. the updated message is equal to the previous message sent on that edge), and thus, the propagation of messages ceases. ii) The agents' preferred states converge as above, but the messages continue to change slightly at each update, and thus continue to be propagated around the network. iii) Neither the agents' preferred states, nor the messages converge and both display cyclic behaviour.

Thus, depending on problem being addressed, and the convergence properties observed, the algorithm may be used with two different termination rules: i) Continue to propagate messages until they converge, either changing the state of the agent continuously to match the optimum indicated, or only after convergence has occurred. ii) Propagate message for a fixed number of iterations per agent (again either changing the state of the agent continuously or only at termination).

Finally, we note that if messages are continuously propagated, and the states of the agents are continuously updated, then the algorithm may be applied to dynamic problems where the interactions between agents, or the utilities resulting from these interactions, may change at any time. For example, within tracking problems where the decentralised coordination algorithm is being used to focus different sensors onto different targets (as described in [15]), then the utilities of each sensor are continually changing due to the changing position of targets, and the actions of other sensors. Thus, by continually propagating messages each agent is able to maintain a continuously updated estimate of the state that it should adopt in order to maximise social welfare in this dynamic problem[4].

5. GRAPH COLOURING BENCHMARK

In order to empirically evaluate the performance of the decentralised coordination algorithm described above, we perform a comparison on a set of instances of distributed graph colouring problems. The graph colouring problem is a canonical DCOP problem [4, 8], and is one that has previously been used to benchmark coordination algorithms for sensor networks [1, 15].

More formally, in distributed graph colouring problems, agents are located at the nodes of a graph, and must select their colour (i.e. the state) from a set of possible colours (i.e. $x_m \in 1, \ldots, c$) in order that they avoid conflicts (i.e. choosing the same colour) with other agents connected to themselves via an edge. Thus, the utility of each agent is expressed as:

$$U_m(\mathbf{x_m}) = \gamma_m(x_m) - \sum_{i \in \mathcal{N}(m) \setminus m} x_m \otimes x_i \qquad (18)$$

where:

$$x_i \otimes x_j = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{otherwise} \end{cases} \qquad (19)$$

[3]This is in contrast to optimal methods such as DPOP that require that the root node of the pseudo-tree informs all the other nodes of their optimal state once it has received all the necessary messages.

[4]Again, this is in contrast to optimal solutions that would repeatedly "freeze" the problem into a sequence of "frames", and then calculate the optimal solution to these increasingly out-of-date problems.

0:
$$\gamma_1(x_1) = [\mathbf{0.1}, -0.1]$$
$$\gamma_2(x_2) = [-0.1, \mathbf{0.1}]$$
$$\gamma_3(x_3) = [-0.1, \mathbf{0.1}]$$

1:
$$R_{1\to2}(x_2) = max_{x_1}(U_1(x_1, x_2) + Q_{1\to1}(x_1)) = [-0.1, 0.1]$$
$$R_{1\to1}(x_1) = max_{x_2}(U_1(x_1, x_2) + Q_{2\to1}(x_1)) = [0.1, -0.1]$$

2:
$$Z_1(x_1) = R_{1\to1}(x_1) + R_{2\to1}(x_1) = [\mathbf{0.1}, -0.1]$$
Variable x_1 state is 0

3:
$$Q_{2\to3}(x_2) = R_{1\to2}(x_2) + R_{2\to2}(x_2) = [-0.1, 0.1]$$
$$Q_{2\to1}(x_2) = R_{2\to2}(x_2) + R_{3\to2}(x_2) = [0, 0]$$
$$Q_{2\to2}(x_2) = R_{1\to2}(x_2) + R_{3\to2}(x_2) = [-0.1, 0.1]$$

4:
$$R_{2\to3}(x_3) = [0.2, -0.2]$$
$$R_{2\to1}(x_1) = [0.2, -0.2]$$
$$R_{2\to2}(x_2) = [-0.1, 0.1]$$

5:
$$Z_2(x_2) = [-0.2, \mathbf{0.2}]$$
Variable x_2 state is 1

6:
$$Q_{3\to2}(x_3) = R_{3\to3}(x_3) = [0, 0]$$
$$Q_{3\to3}(x_3) = R_{2\to3}(x_3) = [0.2, -0.2]$$

7:
$$R_{3\to2}(x_2) = [-0.1, 0.1]$$
$$R_{3\to3}(x_3) = [0.0, 0.0]$$

8:
$$Z_3(x_3) = [\mathbf{0.2}, -0.2]$$
Variable x_3 state is 0

Figure 3: Example execution of our decentralised coordination algorithm on a 3-agent 2-colour graph colouring problem.

and, $\gamma_m(\mathbf{x_m}) << 1$, reflects the agents' preference for any particular colour in the absence of conflicts[5]. As before, the goal is to find the state of each agent such that social welfare is maximised, and hence, the total number of conflicts is minimised.

Before presenting the full evaluation, we present a simple 2-colour example (involving three agents connected as shown in figure 2) to illustrate the operation of the algorithm described in the previous section (see figure 3). Step 0 shows the preferences for each agent, and the following steps show the messages from variable to function (as specified in equation 13), from functions to variables (as specified in equation 15) and the marginal function computation (as specified in equation 16) for each agent. Notice that messages are functions of length equal to the number of possible states (colours) of each agent, and that the messages flowing into any variable represent the utility that the rest of the system can obtain if that variable adopts any particular state (colour). The initial messages from variables to functions are all set to zeros.

6. EMPIRICAL EVALUATION

Having presented a simple example, we now present a full empirical evaluation of our formalism on three sets of graphs that represent a wide range of possible structures. These three sets are:

1. 50 instances of 3-colour random graphs with 10, 20, 30, 40 and 50 nodes, that have an average of 3 links per node, and are known to be colourable.
2. 40 instances of 3-colour random graphs with 10, 14, 18 and 25 nodes taken from the ADOPT graph repository (available from `http://teamcore.usc.edu/dcop/`). As above, we use graphs with a link density of 3 which results in over-constrained graphs that are not generally colourable with 3 colours [8].
3. 5 instances of 4-colour colourable regular lattice graphs with 36, 49, 64, 81 and 100 nodes that are placed on a regular grid and are connected with their eight immediate neighbours (including diagonal neighbours).

[5]Note that this preference breaks the symmetry of the optimisation problem, and has no affect on the optimal solution.

We benchmark our max-sum decentralised coordination algorithm against two alternative approximate algorithms and a complete algorithm:

1. BR - *Best Response*: At each time step, each agent chooses the best state for its variable (i.e. the one that minimizes the conflicts), according to the current states of its neighbours. When a state change occurs, the agent sends the updated state information to all of its neighbours. BR represents a lower bound on the performance of any algorithm since it uses the minimum computation and communication possible.

2. DSA - *Distributed Stochastic Algorithm*: As with best response, except that an agent only actually performs the state change according to a predefined probability (called the *activation probability*). Whenever an agent's preferred state actually changes, it sends a message to its neighbours informing them of this fact. We set the activation probability to 0.6 (additional experiments indicate little sensitivity to the exact value of this parameter in our setting).

3. DPOP - *Dynamic Programming Optimisation Protocol*: A complete algorithm that maintains optimality by preprocessing the constraint graph to produce a pseudo-tree, and then local performs message passing on this tree [11].

Using these three sets of graphs, we evaluate each of these algorithms by performing repeated simulations (10 times per graph instance) and measuring a number of different metrics.

6.1 Conflicts Over Time

We first evaluate the performance of our method against the alternative approximate solutions by considering both the quality of the graph colouring solutions that it produces and the speed with which they are derived. More specifically, we run all the approximate algorithms for a fixed number of execution cycles (50 in this case), and then sum the number of conflicts over this execution time. Following [4] we define a cycle to mean the period in which all agents have had the opportunity to update their states and have delivered their outgoing messages[6].

Figure 4 shows the results of these experiments on the three different sets of graphs. We first consider the colourable random and lattice graphs and note that our max-sum decentralised coordination algorithm (denoted by *MS* in the figure) outperforms the other algorithms, showing up to $1/5$ of the number of conflicts over time displayed by both alternative approximate approaches. Note that the improvement is greatest for larger numbers of agents, since while the alternative approaches are using only local information regarding the preferred states of their neighbours, the max-sum algorithm is able to effectively exploit information regarding the utilities of these neighbouring agents to obtain very good solutions.

In the case of graphs from the ADOPT repository, our method performs less well for small numbers of agents. While in the previous two cases, the preferred state of the agents was seen to converge to the optimum, or a state close to the optimum, this is not the case here, and we observe cyclic behaviour in both the messages being propagated and the preferred state of the agents. We address this behaviour, and present a solution in the next section.

6.2 Convergence

In the preceding experiments, we did not use any termination criteria for our algorithm, but simply let the algorithm run for a given

[6]We implement this within each cycle by selecting agents in a random order, updating their messages and state based on the previous messages that they had received, and then propagating their outgoing messages to their neighbours.

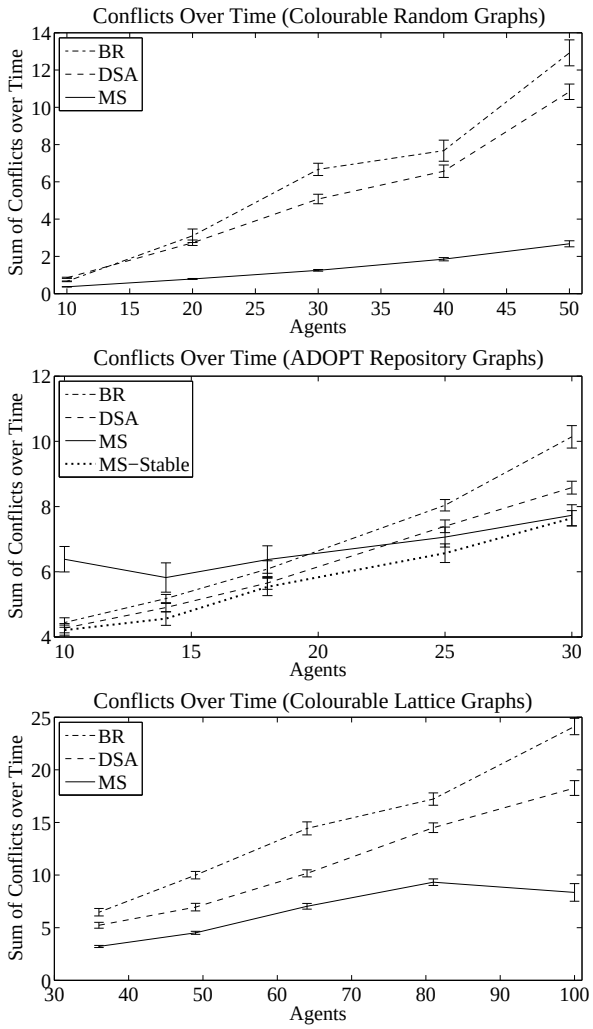

Figure 4: Results for the three graph colouring datasets showing the number of conflicts summed over execution time. Simulation results are averaged over 10 runs on each graph instance, and error bars represent the standard error of the mean.

number of cycles and then stopped it (as per the discussion in section 4). While this is suitable in cases where typical problem instances can be empirically evaluated prior to deployment of the real system, this is clearly not always the case. Thus, we may also require that both the messages and the state of the agents converge, such that the agents may stop exchanging messages when an updated message matches that which was last sent.

The poor results for the ADOPT repository graphs shown in figure 4 are due to the fact that these over-constrained graphs exhibit more complex structures than the other test sets. The colourable random graphs generally contain large loops that span the entire graph, while the lattice graphs contain many small regular loops. In contrast, the graphs of the ADOPT repository represent a combination with both large and small irregular loops, and hence, they represent a more challenging and interesting test set. For this reason, the rest of the analysis is performed on this set only.

We can address the poor convergence observed by modifying the utility function of an agent to consider not only the constraints with its direct neighbours, but also the constraints among these neigh-

Num. Agents	%Not Conv.	Sol. Quality	Conv. Cycles
Best Response (BR)			
10	0	0.39±[0.15]	2.31±[0.09]
14	0	0.87±[0.13]	2.56±[0.13]
18	0	1.23±[0.27]	2.67±[0.07]
25	0	2.36±[0.18]	3.04±[0.1]
Distributed Stochastic Algorithm (DSA)			
10	0.37	0±[0.16]	42.01±[3.58]
14	0.3	0.26±[0.14]	37.21±[2.48]
18	0.36	0.28±[0.22]	39.43±[2.47]
25	0.43	0.98±[0.16]	43.22±[1.09]
Max-sum Algorithm (MS)			
10	0.93	2.22±[0.53]	47.44±[0.99]
14	0.55	0.81±[0.44]	37.03±[3.89]
18	0.64	0.72±[0.42]	40.68±[2.79]
25	0.73	0.62±[0.38]	40.96±[2.51]
Max-sum Algorithm - Stabilised (MS-Stable)			
10	0	0.02±[0.15]	9.76±[0.32]
14	0	0.1±[0.21]	10.82±[0.53]
18	0	0.42±[0.25]	11.66±[0.38]
25	0	0.44±[0.3]	12.98±[0.52]

Table 1: Comparison of convergence properties of our decentralised coordination algorithms using graphs from the ADOPT repository graphs.

bours. This amounts to a change in the utility function to:

$$U_m(\mathbf{x_m}) = \gamma_m(x_m) - \sum_{i \in \mathcal{N}(m)} \sum_{j \in C(i,m)} x_i \otimes x_j \qquad (20)$$

where:

$$C(i,m) = \{k \in \mathcal{N}(m) | k > i \wedge (i \in \mathcal{N}(k) \vee k \in \mathcal{N}(i))\} \quad (21)$$

and $\gamma_m(x_m)$ and $x_i \otimes x_j$ are as before[7]. The results of this improvement are shown in figure 4 for the ADOPT repository graphs (shown as MS-Stable). As can be seen, the modified utility function effectively prevents cycling, and causes both the messages and the preferred states of the agents to converge, and our formalism now significantly outperforms the earlier utility function.

Table 1 reports these results in more detail. In particular, it shows the percentage of runs that converged, the mean and standard error of the mean for the solution quality at convergence (i.e. the number of conflicts from optimal, where the optimal is calculated using the DPOP algorithm), and the number of cycles that were required to reach convergence (i.e. when no further messages were exchanged). Note that our max-sum coordination algorithm with a modified utility function converges in all cases. By propagating utility information among the agents, it is able to converges more rapidly than DSA, and yet it also generates better quality solutions that are extremely close to optimal.

6.3 Total Message Size Exchanged

While we have used DPOP to calculate the optimal colouring of the ADOPT repository graphs we have not explicitly discussed its performance yet. We now do so in order to compare the total size of messages exchanged among the agents for each algorithm. Figure 5 shows a comparison the total size of messages (on a logarithmic scale) that were exchanged[8] in order to reach convergence to the

[7]Note that in order to implement this, agents must inform their neighbours of their own neighbour list, but this represents a comparatively small communication overhead (e.g. in a static setting it need be communicated only once).

[8]Here we count the number of real numbers exchanged. For example, BR and DSA communicate their preferred state only and thus use a single real number per message, our max-sum decentralised coordination message requires six real numbers (three states for both variable and function out-going messages), and DPOP requires a real number to represent each agent-state combination.

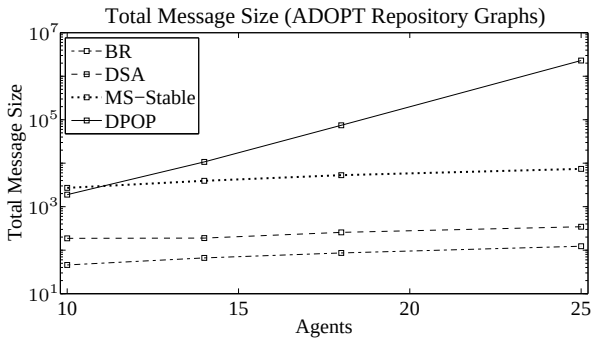

Figure 5: Total size of messages exchanged by algorithms on the ADOPT repository graphs.

solutions shown in table 1. Note that while the approximate algorithms show only very gradual dependence between the number of agents and the total size of messages exchanged, the rising line for DPOP clearly indicates an exponential relationship, and for just 25 agents, DPOP already requires over 100 times the total message size. Furthermore, while the approximate algorithms iteratively exchange messages of small size, DPOP propagates ever larger messages up the pseudo-tree. For the graphs containing 10, 14, 18 and 25 agents, the largest single message observed in our experiments consisted of 2187, 6561, 59049 and 1594323 different utility values. Using standard 32 bit floating point numbers to represent each utility value would means that the memory of the Chipcon CC2431 SoC is exceeded when there are just 10 agents[9].

6.4 Message Loss

Finally, we note that figure 5 shows that our algorithm exchanges a total messages size that is at least one order of magnitude greater than the two alternative approximate algorithms (BR and DSA). The reasons for this are that our algorithm communicates utility information over each possible state every time it receives an updated message itself, whereas the two alternative algorithms only communicate their preferred state, and only communicate this information when it changes. However, if the communication channels between agents are lossy such that messages are lost (as is likely with low-power wireless devices), the minimal communication strategy of the alternative algorithms is disadvantageous. Specifically, figure 6 shows the total conflicts over time (as was shown in figure 4) as the probability of successful transmission of agent-to-agent messages decreases (using the graphs from the ADOPT repository). Note that the performance of our algorithm degrades extremely slowly, since it effectively retransmits the same information at each update. This result shows that our algorithm is extremely robust to lossy communication, and also indicates that further improvements in the cost of communication can be accrued.

7. HARDWARE IMPLEMENTATION

In order to prove the practical applicability of our max-sum decentralised coordination algorithm, we have implemented it, and the graph colouring benchmark problem, in hardware using the Chipcon CC2431 System-on-Chip (SoC). This is a low-power device incorporating an IEEE 802.15.4 compliant RF transceiver, 128 kByte flash memory, 8 kByte RAM, and a 32 MHz 8 bit 8051 microcontroller in a single 7x7mm package, and is intended to form the core of future low-power sensor nodes (see figure 7). The con-

[9]Even if memory constraints are eliminated, the time to communicate these large messages over the 250 kbps channels used by these low power devices is still prohibitive in large networks.

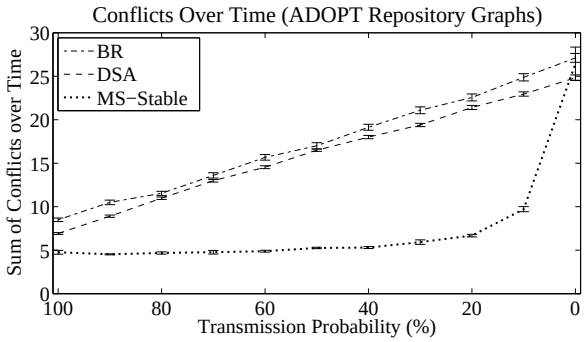

Figure 6: Results showing the number of conflicts over time, as the transmission probability of messages is decreased (using the ADOPT repository graphs).

strained computational power and memory resources mean that optimal algorithms, such as DPOP, cannot be used.

Our max-sum algorithm runs continuously on each sensor node (with no termination rule), and whenever two nodes are within a pre-specified distance (measured by received signal strength) they consider themselves to be connected within the constraint graph. Each node transmits out-going messages and updates their preferred state exactly as described in section 4, and as the agents are moved relative to one another, they continuously select a preferred colours (shown using LEDs) to avoid clashes with connected agents (whose number is shown by the 7-segment display). Videos are available at `www.youtube.com/v/T6H1AwQ2gXw` and `www.youtube.com/v/D6vWvs3Lsj0`.

8. CONCLUSIONS

In this paper we addressed the need for resource efficient decentralised coordination algorithms for low-power embedded devices. We showed that a novel representation of this problem, as a cyclic bipartite factor graph, allowed us to use an extension of the max-sum algorithm to generate approximate solutions to a global optimisation problem through local decentralised message passing between agents. In an empirical evaluation, this algorithm was shown to be robust to lossy communication, and to generate solutions close to the optimum with a communication cost (in terms of total messages size) that scales very well with the number of agents in the system (compared to the exponential increase of DPOP). Finally, we proved the practical application of our approach by implementing it on a low-power embedded device.

Our future work in this area considers two main directions. First, we would like to address more specifically the coordination of the sense/sleep cycle of sensor nodes within the wide area surveillance sensor network that motivates our work. Here, we intend to benchmark our approach as the action space of the sensor nodes increases, as the utility calculation becomes more complex, and as the interactions of agents change over time.

Second, we would like to more formally investigate how the convergence properties of our max-sum algorithm depend on the structure of the problem to which it is applied. Our goal is to provide conditions for convergence, and to bound the approximate solutions that are obtained. In this respect, the extremely good convergence properties of the modified utility function that we introduced in section 6.2 suggest this may be possible within the graph colouring domain that we have considered here.

9. ACKNOWLEDGMENTS

The work reported on here was undertaken as part of the ARGUS II DARP (Defence and Aerospace Research Partnership), the Data Information Fusion Defence Technology Centre (DIF DTC) Phase II 'Adaptive Energy-Aware Sensor Networks' project, and the EPSRC funded Market-Based Control project (GR/T10664/01). The ARGUS II DARP is a collaborative project involving BAE SYSTEMS, QinetiQ, Rolls-Royce, the University of Oxford and the University of Southampton, and is funded by the industrial partners together with the EPSRC, MoD and DIUS. The DIF DTC project is joint funded by MoD and General Dynamics UK. We would like to thank Luke Teacy and Dimosthenis Karatzas for development work deploying the algorithms on the Chipcon CC2431 nodes.

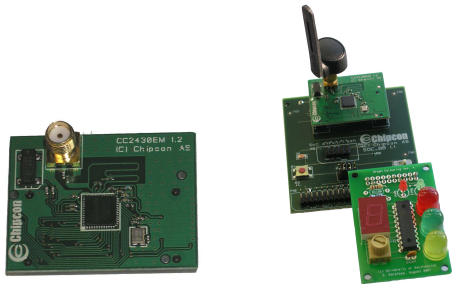

Figure 7: Hardware implementation of our max-sum decentralised coordination algorithm on the Chipcon CC2431 SoC.

10. REFERENCES

[1] S. Fitzpatrick and L. Meetrens. *Distributed Sensor Networks A multiagent perspective*, chapter Distributed Coordination through Anarchic Optimization, pages 257–293. Kluwer Academic, 2003.

[2] D. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45(2):399–431, 1999.

[3] D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[4] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proceedings of 3rd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'04)*, pages 438–445, 2004.

[5] A. Makarenko and H. Durrant-Whyte. Decentralized data fusion and control algorithms in active sensor networks. In *Proceedings of 7th International Conference on Information Fusion (Fusion'04)*, pages 479–486, 2004.

[6] M. Mezard, G. Parisi, and R. Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297(5582):812–815, 2002.

[7] P. Modi, S. Ali, W. Shen, and M. Tambe. Distributed constraint reasoning under unreliable communication. *Proceedings of Distributed Constraint Reasoning Workshop at 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems*, 2003.

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

[9] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence (UAI'99)*, pages 467–475, 1999.

[10] P. Padhy, R. K. Dash, K. Martinez, and N. R. Jennings. A utility-based sensing and communication model for a glacial sensor network. In *Proceeding of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'06)*, pages 1353–1360, 2006.

[11] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, (IJCAI'05)*, pages 266–271, 2005.

[12] P. Rybski, S. Stoeter, M. Gini, D. Hougen, and N. Papanikolopoulos. Effects of limited bandwidth communications channels on the control of multiple robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 369–374, 2001.

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

[14] W. Y. and F. W.T. On the optimality of solutions of the max-product belief propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):723–735, 2001.

[15] W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *Proceedings of the 2nd Int. Joint Conference on Autonomous Agent and Multiagent Systems (AAMAS'03)*, pages 185–192, 2003.