Decentralised Coordination of Continuously Valued Control Parameters using the Max-Sum Algorithm

R. Stranders, A. Farinelli, A. Rogers and N. R. Jennings
School of Electronics and Computer Science
Southampton, SO17 1BJ, UK.
{rs06r,af2,acr,nrj}@ecs.soton.ac.uk

ABSTRACT

In this paper we address the problem of decentralised coordination for agents that must make coordinated decisions over continuously valued control parameters (as is required in many real world applications). In particular, we tackle the social welfare maximisation problem, and derive a novel continuous version of the max-sum algorithm. In order to do so, we represent the utility function of the agents by multivariate piecewise linear functions, which in turn are encoded as simplexes. We then derive analytical solutions for the fundamental operations required to implement the max-sum algorithm (specifically, addition and marginal maximisation of general n-ary piecewise linear functions). We empirically evaluate our approach on a simulated network of wireless, energy constrained sensors that must coordinate their sense/sleep cycles in order to maximise the system-wide probability of event detection. We compare the conventional discrete max-sum algorithm with our novel continuous version, and show that the continuous approach obtains more accurate solutions (up to a 10% increase) with a lower communication overhead (up to half of the total message size).

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Theory

Keywords

DCOP, Coordination, Computational Geometry

1. INTRODUCTION

Networks of sensing devices that acquire, integrate and wirelessly communicate information are gaining increasing attention within the research community and have found applications within areas such as multi-sensor target tracking, unmanned autonomous vehicles for rescue robotics and wide-area surveillance. In many of these applications, a key challenge to successful deployment is to enable the physically distributed devices to coordinate their individual sensing actions in order to act together toward system-wide goals. Examples of such coordination include controlling the orientation of multiple fixed sensors to better localise, identify and track a target [3], continuously controlling the trajectory (and hence the position) of multiple mobile sensors to explore an uncertain and dynamic environment [5], and in the wide-area surveillance scenario that we consider in detail here, coordinating the sense/sleep cycle of energy constrained sensors in order to maximise the system-wide probability of detecting an event [7]. Such coordination is particularly challenging in these domains since the devices often have constrained computational resources (often due to the requirement of minimising the power used by the device) and they can typically only communicate with a small number of nearby devices (due to the use of lower power wireless communication).

To address these challenges, multiagent system approaches have been widely used, and in doing so, the coordination problem faced by the devices is often cast as a distributed constraint optimisation problem (DCOP), since this allows a wide range of existing DCOP algorithms to be immediately applied. Such algorithms can be broadly divided in two main classes: complete algorithms that generate optimal solutions such as ADOPT [13], OptAPO [12], and DPOP [15]; and approximate algorithms such as the Distributed Stochastic Algorithm (DSA) [3] or Maximum Gain Message [11]. Now, while complete algorithms provide guarantees on the solution quality, they also exhibit an exponentially increasing coordination overhead (either through the size and/or number of messages exchanged, or in the computation required by each device) as the number of devices in the network increases. Conversely, approximate algorithms require very little local computation or communication, but often converge to poor quality solutions because the devices do not propagate information across the whole network. Rather, local information is only used by neighbouring devices. For example, in DSA each agent communicates its preferred action (e.g., the one that will maximise its own utility) based on the current preferred actions of its neighbours only.

While such approaches show promise, they all assume that each device has a small number of discrete actions to choose between. However, in many applications, and in all of the examples described earlier, each device must actually make a choice regarding one or more continuous valued control parameters (i.e. its orientation, its heading and velocity, or the time at which it activates its sensors). While these continuous control parameters may be discretised in order to apply existing DCOP algorithms, such discretisation results in sub-optimal solutions even when complete DCOP algorithms are used. Furthermore, increasing the discretisation in order to improve the solution quality rapidly increases the computation and communication overhead of the coordination.

Thus, against this background, there is a clear need for a distributed constraint optimisation algorithm that can be applied within these applications to make coordinated decisions regarding contin-

Cite as: Decentralised Coordination of Continuously Valued Control Parameters using the Max-Sum Algorithm, R. Stranders, A. Farinelli, A. Rogers and N. R. Jennings, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

uous valued control parameters. It is this requirement that we address in this paper, and to this end, we present a novel distributed constraint optimisation algorithm for continuous control parameters that can be applied to any setting where the interaction between the devices can be described by piecewise linear utility functions. We choose this particular class of utility function because it can be manipulated using standard techniques from computational geometry, resulting in a computationally efficient algorithm, but at the same time it is a general class of function that can be used to approximate any arbitrary continuous function. We then demonstrate and empirically evaluate this algorithm by applying it to the problem of coordinating the sense/sleep cycle of energy constrained sensors deployed within a network for wide-area surveillance.

In more detail, the starting point for our work is a broad class of algorithms usually referred to under the framework of the generalised distributive law (GDL) [1]. Such algorithms have been widely used in the field of information theory and probabilistic inference to decompose complex computations on single processors [10], and more recently both complete and approximate algorithms from this framework have been applied to the coordination of networked sensing devices within the domain of discrete control parameters [14, 2]. In particular, one of the approximate algorithms, based upon the max-sum algorithm, has been shown to generate solutions closer to the optimum than previous approximate stochastic DCOP algorithms [2]. It does so with an acceptable computation and communication overhead when benchmarked against representative complete algorithms (specifically DPOP), and it has been shown to be robust to message loss. Due to the fact that this algorithm exhibits these properties, it has been deployed and validated on low-power embedded devices. Nevertheless, despite the attractive properties of this algorithm, and the fact that the GDL framework is very general, it has never before been used in the context of distributed optimisation, to solve problems involving variables with continuous domains. It is this specific challenge that we address in this paper. To do so, however, requires that the fundamental operations of the max-sum algorithm be redefined for the continuous space in which they now operate.

Thus, in more in detail, we make the following contributions:

1. We derive an efficient representation that uses simplexes to describe the utility interactions of continuous valued control parameters (or variables). We then derive exact algorithmic solutions for the fundamental operations required to apply the max-sum algorithm in the domain of continuous control parameters and piecewise linear utility functions; specifically, addition and marginal maximisation of general n-ary piecewise linear functions.
2. We demonstrate the applicability of this algorithm by applying it to the problem of coordinating the sense/sleep cycles of energy constrained sensors such that the system-wide probability of detecting an event is maximised.
3. We empirically evaluate the performance of our continuous max-sum algorithm in this setting by comparing it against the conventional discrete version in which the continuous valued control parameter is artificially discretised. We show that the continuous max-sum algorithm is able to provide a more accurate solution (up to 10% increase in solution quality), while allowing a more compact representation of the utility functions. This, in turn, results in a lower coordination overhead in terms of message size (the continuous max-sum algorithm shows a reduction of up to half the total message size).

The remainder of this paper is structured as follows: in Section 2 we formally describe the social welfare maximisation problem that we face. In Section 3 we detail the max-sum algorithm, while in Section 4 we present our representation of the utility as piecewise linear functions, and the way we perform the operations needed by the max-sum algorithm. In Section 5 we describe our test-bed and present our experimental results, before concluding in Section 6.

2. PROBLEM DESCRIPTION

We now formally describe the decentralised coordination problem that we address in this paper. In Section 5 we instantiate this general problem to the specific case of a sensor network composed of energy constrained sensors performing a wide-area surveillance task. We consider a set of M agents, each of which has a single[1] continuously valued control parameter x_m whose domain is a closed and bounded interval in $\mathbb{R}$. Each agent interacts directly with a set of other agents, such that the utility of an agent, $U_m(\mathbf{x}_m)$, is dependent on the value of its own control parameter and that of those other agents with which it interacts (defined by the vector $\mathbf{x}_m$). For example, in the specific wide area surveillance problem that we will detail in Section 5, the control parameter of each agent represents the time at which it decides to activate its sensor, the interactions arise through its sensor's sensing field overlapping with those of other nearby agents, and the utility describes the probability that any agent will detect an event. As stated previously, we focus on utility functions that can be represented as piecewise linear functions. Therefore, $U_m(\mathbf{x}_m)$ is a multivariate piecewise linear function for every m.

Within this setting, we wish to find the value of each agent's control parameter, $\mathbf{x}^*$, such that the sum of the individual agents' utilities (commonly referred to as social welfare within the multi-agent systems literature[2]) is maximised:

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

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 that influence its own utility directly. In this way, the complexity of the calculation that the agents perform depends on the number of neighbours that it has (and not the total size of the network), and thus, we can achieve solutions that scale well.

3. THE MAX-SUM ALGORITHM FOR DECENTRALISED COORDINATION

In order to apply the max-sum algorithm, we represent the optimisation problem described in Equation 1 as a bipartite factor graph. For example, Figure 1(a) shows three interacting agents, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, and the resulting factor graph consisting of variable and function nodes representing each agent's control parameter and utility is shown in Figure 1(b). In order to avoid unnecessary repeated computation, we further decompose the factor graph to separate functions that represent the interactions between agents, and also any preferences over control parameter values that individual agents may have (shown in Figure 1(c)). The decomposition ensures that the sum of the functions is equal to the sum of the original agent utilities, such that $U_1(x_1, x_2) + U_2(x_1, x_2) + U_3(x_2, x_3) = F_1(x_1) + F_2(x_2) + F_3(x_3) + F_4(x_1, x_2) + F_5(x_2, x_3)$.[3]

[1]Note that our representation allows multiple control parameters per agent, with possibly different domains, but for ease of exposition we present just the single case here.

[2]The same problem is also referred to as the optimal control problem in control theory [14].

[3]The functions F_i are also piecewise linear.

In order to have a truly decentralised computation, it is necessary to assign the function that represents interactions between agents to one of the agents that is involved in the interaction. For example, Figure 1(c) shows a possible assignment of functions to agents where the agent that has the highest identifier is responsible for the shared function. The assignment can be done arbitrarily, since any assignment ensures that the sum of the agent's utilities is equal to the sum of the functions, although allocations that balance the computational load between the agents would clearly be desirable.

The max-sum algorithm then operates directly on the factor graph representation described above. When this graph is cycle free, the algorithm is guaranteed to converge to the global optimal solution such that it finds the combination of states that maximises the sum of the agents' utilities [10]. When applied to cyclic graphs, there is no guarantee of convergence, but extensive empirical evidence demonstrates that this family of algorithms generate good approximate solutions [9, 4]. Within this setting, the max-sum algorithm specifies the messages that should be passed from variable to function nodes, and from function nodes to variable nodes. These messages are defined as:

- **From variable to function:**

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

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

- **From function to variable:**

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

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

The messages flowing into and out of the variable nodes within the factor graph are functions of a single variable that represent the total utility of the network for each possible value of that variable. At any time during the propagation of these messages, agent i is able to determine which state it should adopt such that the sum over all the agents' utilities is maximised. This is done by locally calculating the function, $z_i(x_i)$, from the messages flowing into agent i's variable node:

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

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

The messages described above may be randomly initialised, and then updated whenever an agent receives an updated message from a neighbouring sensor; there is no need for a strict ordering or synchronisation of the messages. In addition, the calculation of the marginal function shown in Equation 4 can be performed at any time (using the most recent messages received), and thus, sensors have a continuously updated estimate of their optimum state.

The final state of the algorithm depends on the structure of the agents' utility functions, and, in general, three behaviours can be observed:

1. 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.
2. 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.
3. 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 different termination rules:

1. Continue to propagate messages until they converge, either changing the state of the agents continuously to match the optimum indicated, or only after convergence has occurred.
2. Propagate messages for a fixed number of iterations per agent (again either changing the state of the agent continuously or only at termination).

The first termination rule favours the quality of the solution. When the algorithm converges, it does not converge to a simple local maximum, but to a neighbourhood maximum that is guaranteed to be greater than all other maxima within a particular large region of the search space [16]. Depending on the structure of the factor graph, this neighbourhood can be exponentially large. However, only limited guarantees for convergence of the max-sum algorithm exist, and for general factor graphs the algorithm might not converge. For practical applications, therefore, the second termination rule is often preferred. In fact, empirical evidence shows that the max-sum algorithm reaches good approximate solutions in few iterations. In addition, in dynamic scenarios where the utilities of the agents or the interactions between them change over time, the max-sum algorithm can run indefinitely without any termination rule; each agent can decide at every cycle which state to choose based on Equation 4, and operates on a continuously changing coordination problem.

Note that at this stage, the description of the max-sum algorithm applies equally well to both continuously valued and discrete variables. However, the summation and marginal maximisation operations required in Equations 2 and 3 are much more readily implemented in the case of discrete variables. In the next section, we describe an efficient representation that uses simplexes to describe the utility interactions over continuous valued variables. This then allows us to derive exact algorithmic solutions for these operations.

4. MAX-SUM IN CONTINUOUS SPACE

As described above, in order to apply the max-sum algorithm to cases in which the variables can take on continuous values, we need to be able to express each individual agent's utility $U_i(\mathbf{x}_i)$ as a function of these continuous variables, and perform the fundamental operations of the max-sum algorithm on these functions. As mentioned earlier, in this paper, we restrict our attention to cases in which this function $U_i(\mathbf{x}_i)$ is a multivariate continuous piecewise linear function (CPLF). We choose this particular class of utility function because they represent a general class that can be used to approximate any arbitrary continuous function. Furthermore, under this restriction, the two aforementioned operations have an attractive geometric interpretation that makes it possible to define and manipulate them using standard techniques from computational geometry, hence allowing the continuous versions of the operations required by max-sum to be performed. Thus, more specifically, in this section we show how to:

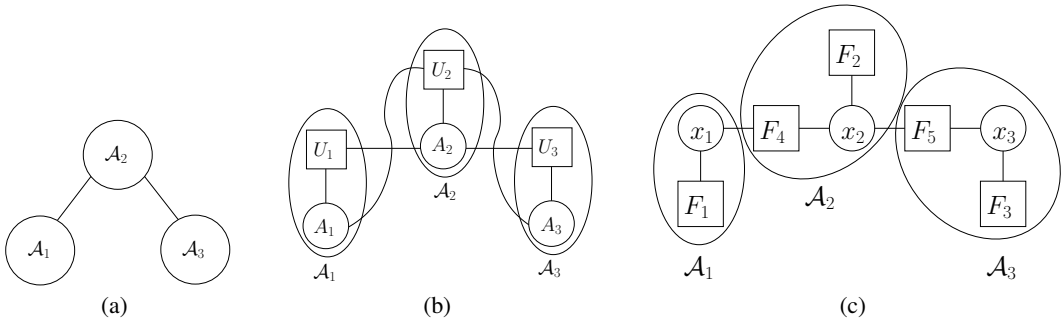


Figure 1: Diagram showing (a) the interactions of agents, $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$, (b) factor graph representing the agents' utility (c) factor graph representing the agents' interactions.

1. Represent each agents' utility function as a continuous piecewise linear function (CPLF).
2. Perform the summation of two continuous piecewise linear functions (in order to perform the necessary summation of the utility function, $F_j(\mathbf{x}_j)$, and the summation of incoming messages, $\sum_{k \in \mathcal{N}_\mathbf{j} \setminus i} q_{k \to j}(x_k)$ shown in Equation 3).
3. Calculate the marginal maximisation of a continuous piecewise linear function with respect to a single variable (in order to perform the necessary $\max$ operation in Equation 3).

We next present a formal definition of a CPLF, before going on to derive exact algorithmic solutions for computing the sum of two CPLFs, and the marginal maximisation of a CPLF with respect to a single variable. Finally, we connect the formalism developed here to the max-sum algorithm as described in the previous section.

4.1 Representing CPLFs with Simplexes

A continuous piecewise linear function is a function whose domain can be partitioned into a set of convex polytopes[4], such that it is linear on each of these polytopes. For one variable, a CPLF is a function that can be represented with a finite number of line segments, while a CPLF of two variables can be represented by a finite number of two-dimensional polygons. An example of the latter is given in Figure 3. The domain of the function in this figure is partitioned into triangles (shown on the $(x1, x2)$ plane) on which the function is linear.

In our formalism, we use n-dimensional simplexes, or n-simplexes, to partition the domain of an n-ary CPLF. The reason for this is that an n-simplex is the simplest n-dimensional polytype and are therefore easy to manipulate. More specifically, an n-simplex is constructed by taking the convex hull of a set of $n+1$ affinely independent vertexes in $\{\mathbf{p}_1, \ldots, \mathbf{p}_{n+1}\} \in \mathbb{R}^m$ $(m \geq n)$, and is denoted by Δ^n. Note that we will omit the superscript n when its value is clear from the context. The set of points enclosed by a simplex is given by the following equation:

$$\Delta^n = \{\mathbf{x} \in \mathbb{R}^m | \sum_{i=1}^{n} a_i \mathbf{p}_i = \mathbf{x}, \sum_i a_i = 1, \forall i : a_i \geq 0\} \quad (5)$$

Now, an n-ary CPLF $f : \mathcal{D} \to \mathbb{R}$ is defined by a set of n-simplexes in $\mathbb{R}^{n+1}$: $\{\Delta_1, \ldots, \Delta_m\} \subset \mathbb{R}^{n+1}$. Here, domain $\mathcal{D}$ is the Cartesian product of the domains of its variables $(x_1, \ldots, x_n)$:[5]

[4]A convex polytope is a multi-dimensional generalisation of the two-dimensional convex polygon. In n dimensions, it is a convex hull of at least $n+1$ points.

[5]In what follows, $(x_1, \ldots, x_n)$ and $\mathbf{x}$ are used interchangeably.

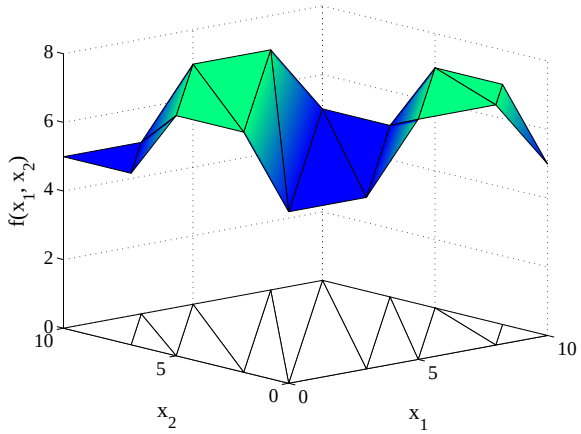


Figure 3: An example of a CPLF in two dimensions. This CPLF encodes the utility of two sensors S_1 and S_2 in the wide area surveillance scenario described in Section 5.1. Sensor S_1 can be active for $l_1 = 2$ out of every $L = 10$ time units, while S_2 can be active for $l_2 = 5$ out of every 10 time units.

$\mathcal{D} = \mathcal{D}_{x_1} \times \cdots \times \mathcal{D}_{x_n}$. Since each x_i is a scalar, $\mathcal{D}_{x_i}$ is a closed interval in $\mathbb{R}$. Consequently, $\mathcal{D}$ is an interval in $\mathbb{R}^n$, or an n-cube. From the definition of a CPLF, we require that the projection of the simplexes that make up f is a partition $P_\mathcal{D}$ of $\mathcal{D}$. More formally, $P_\mathcal{D} = \{\widetilde{\Delta}_i | 1 \leq i \leq m, \bigcup_i \widetilde{\Delta}_i = \mathcal{D}, \widetilde{\Delta}_i \bigcap \widetilde{\Delta}_j = \emptyset, 1 \leq i < j \leq m\}$, where $\widetilde{\Delta}_i$ is the projection of Δ_i onto the $\mathbf{x}$ hyperplane.

4.2 Summation of Two CPLFs

In order to perform the summation of two CPLFs g and h, we need to compute the simplexes that make up function f such that $\forall \mathbf{x} \in \mathcal{D} : f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x})$ holds. We denote the operator that adds two CPLFs as $\oplus$, which works in two steps. First, we need to compute a domain partition P_f of f, such that P_f contains a corner[6] at every corner in g and h. Second, we need to compute the values of f at each vertex $\mathbf{p}$ of the simplexes that partition f. The latter step is trivial; for each vertex $\mathbf{p}$, evaluate $g(\mathbf{p})$ and $h(\mathbf{p})$ and add them together. However, the former step is a little more involved, since computing the domain partition of f involves overlaying or *merging* the partitions P_g and P_h in order to determine where the sum of g and h might have a corner. Algorithm 1 details how to perform this partition and it proceeds in two main steps:

1. Copy partition P_g to the variable P_f that contains the result

[6]A corner is the location at which two simplexes meet at an angle.

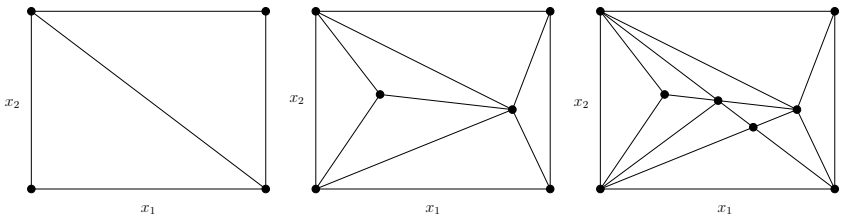


Figure 2: (a) Domain partition P_g of function g. (b) Domain partition P_h of function h. (c) Merged partition P_f of function $f = g \oplus h$.

Algorithm 1 An algorithm for merging two partitions

Input: *Partitions P_g and P_h*
Output: *Partition $P_f = P_g \oplus P_h$*
1: $P_f \leftarrow P_g$
2: ***for all*** $\Delta \in P_h$ ***do***
3: ***for all*** $\mathbf{p} \in \{\mathbf{p}_1^\Delta, \ldots, \mathbf{p}_{n+1}^\Delta\}$ ***do***
4: $P_f \leftarrow \mathcal{S}(P_f, \mathbf{p})$
5: ***end for***
6: ***end for***
7: ***for all*** $\Delta_h \in P_h$ *and* $\Delta_f \in P_f$ ***do***
8: ***for all*** *intersections* $\mathbf{p}$ *of the edges of* Δ_h *with the* $(n-1)$*-faces of* Δ_f ***do***
9: $P_f \leftarrow \mathcal{S}(P_f, \mathbf{p})$
10: ***end for***
11: ***end for***
12: ***return*** P_f

while it is constructed (line 1).

2. Merge every simplex in P_h with P_f by overlaying it on P_f:
   (a) Add the vertexes of all simplexes in P_h to P_f (lines 2 to 6).
   (b) Add the edges of the simplexes in P_h to P_f (lines 7 to 13). These edges are the corners of h, and therefore need to be present in P_f.

As an example, Figures 2(c) shows the output of Algorithm 1 on the partitions shown in Figures 2(a) and 2(b). Note that the partition of f shown in Figure 2(c) indeed has corners at every location where function g and h have corners.

To complete the specification of Algorithm 1, we need to define the split operator $\mathcal{S}$ that is essential in the merge process (lines 4 and 10). Specifically, the split operator $\mathcal{S}$ partitions a simplex Δ^n around a point $\mathbf{x}$: $\mathcal{S}(\Delta^n, \mathbf{x}) = \{\Delta_1^n, \ldots \Delta_m^n\}$. Thus, each $\Delta_i^n \in \mathcal{S}(\Delta^n, \mathbf{x})$ is obtained by creating a simplex with vertexes $\{\mathbf{x}, \mathbf{p}_1, \ldots, \mathbf{p}_{n+1}\} \setminus \{\mathbf{p}_i\}$. Depending on the location of $\mathbf{x}$ in Δ^n, the split operator creates a different number of simplexes. In more detail, depending on the complexity of the face[7] of Δ^n on which $\mathbf{x}$ lies, $\mathcal{S}$ splits Δ^n into at least 1, and at most n simplexes. Figures 4(b) and 4(c) show how the 2-simplex in 4(a) is split on points on a 2-face (body) and a 1-face (edge) respectively. Note that, in the latter case, the simplex is split in two, since vertexes $\{\mathbf{p}_1, \mathbf{p}_2, \mathbf{x}\}$ are not affinely independent, and therefore do not form a simplex. Splitting on a 0-face (or vertex) does not split the simplex, neither does splitting on a point outside the simplex. To avoid cluttering the notation in Algorithm 1, we denote the operation of splitting all simplexes in a partition P on a point as $\mathcal{S}(P, \mathbf{x})$, which is shorthand for $\cup_{\Delta \in P} \mathcal{S}(\Delta, \mathbf{x})$.

[7]The faces of a simplex are simplexes that make up its boundaries. The *complexity* of a face of an n-simplex is its dimensionality, which ranges from 1 to n. A face of complexity i is called an i-face. A 0-face is a vertex of the simplex, a 1-face is an edge, a 2-face is a triangle, etc. The n-face of the simplex is the simplex itself, which is also referred to as the *body*.

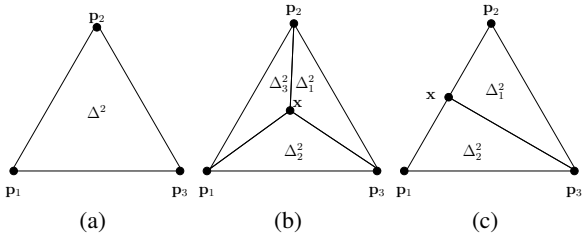


Figure 4: (a) A 2-simplex. (b) Splitting a 2-simplex on point $\mathbf{x}$ on a 2-face. (c) Splitting a 2-simplex on point $\mathbf{x}$ on a 1-face (edge)

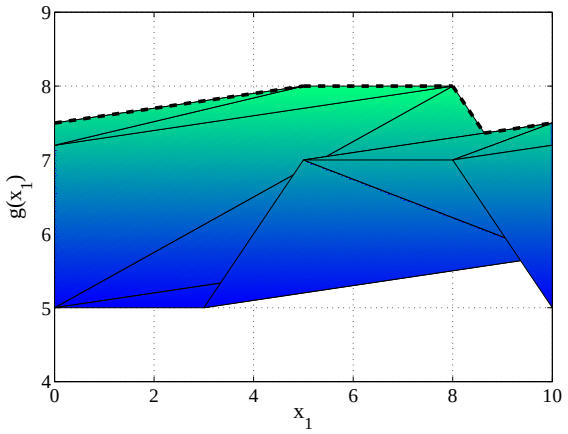


Figure 5: A CPLF $y = g(x_1, x_2)$ projected onto the (x_1, y) plane. The dotted line indicates the upper envelope of these simplexes, and equals $g(x_1) = \max_{x_2} g(x_1, x_2)$.

4.3 Marginal Maximisation of a CPLF

Marginal maximisation is the second operator that is needed in max-sum. It takes as input a function $y = f(x_1, \ldots, x_n)$, and computes a single-dimensional CPLF $f(x_i) = \max_{\mathbf{x} \setminus i} f(x_1, \ldots, x_n)$. This is achieved by first projecting all simplexes of f onto the (x_i, y) plane and then extracting the *upper envelope*. In more detail, in order to project a n-simplex Δ, we project each of its $m = \binom{n}{2}$ edges to obtain a set of line segments $S_\Delta = \{s_1, \ldots, s_m\}$. The *upper envelope* $\hat{U}_S$ of the set S of all projected line segments

of all simplexes that make up f is then a function:

$$\hat{U}_S(x) = \max\{s(x)|s \in S \wedge x \in [s_s, s_e]\}$$

where $[s_s, s_e]$ is the closed interval on which line segment s is defined. The upper envelope of a set of n line segments can be computed in $O(n \log n)$ operations [6].

Figure 5 shows an example of this operation. In this figure the function $y = g(x_1, x_2)$ is projected onto the (x_1, y) plane. This function $g(x_1, x_2)$ is the sum of the function from Figure 3 and the function $h(x_1, x_2) = 0.1x_2$ that slightly shears it along the x_2 axis[8]. The figure also shows the upper envelope of this projection that is the result of applying the marginal maximisation operator on this function g with respect to variable x_1.

4.4 Instantiating the Continuous Max-Sum Algorithm

Now that we have defined the CPLF and the two required operations, we can instantiate the max-sum algorithm for continuous variables, by defining the processes through which messages between the variables and functions are computed:

- From variable to function (Equation 2). Since the messages $r_{k\to i}(x_i)$ are real-valued functions of a single continuous variable x_i, computing $q_{i\to j}$ involves summing over single-dimensional CPLFs using the $\oplus$ operator of Section 4.2. The addition of scalar a_{ij} is then a trivial operation.
- From function to variable (Equation 3). The computation of the message $r_{k\to i}(x_i)$ proceeds in two steps. First, the expression between the brackets is evaluated. The first term in this expression is the utility of agent j, which is a CPLF. The second term is the sum of multiple single-dimensional CPLFs of *different* variables, which is a multi-dimensional CPLF. So, to evaluate this expression, we add the first and second term using the $\oplus$ operator, which, again, results in a CPLF. Second, we use the marginal maximisation operator on this CPLF to obtain the message as required.

Now that we have performed the necessary steps to instantiate the continuous max-sum algorithm, we will compare its performance with that of the discrete max-sum algorithm next.

5. EMPIRICAL EVALUATION

To empirically validate the performance of our continuous max-sum algorithm, we focus on its application for coordinating a network of energy constrained wireless sensors deployed for a wide area surveillance task. We choose this application domain since it represents a challenging real world coordination problem which involves continuously valued control parameters, and also allows us to empirically compare the performance of our novel continuous max-sum algorithm against the conventional discrete one. We note that the discrete max-sum algorithm has previously been benchmarked against approximate (DSA) and complete (DPOP) techniques for DCOPs in a graph k-colouring test-bed [2], and was shown to obtain better solutions than DSA while scaling very well with the number of agents, in terms of total message size (as opposed to the exponential increment exhibited by DPOP).

In the following, we first describe the problem scenario in detail and then instantiate the agents' utilities in this setting. We then describe the application of both the conventional discrete max-sum algorithm and our novel continuous max-sum algorithm to this problem, and empirically compare their properties.

[8]This is equivalent to receiving a message $q_{2\to f}(x_2) = 0.1x_2$ from variable x_2.

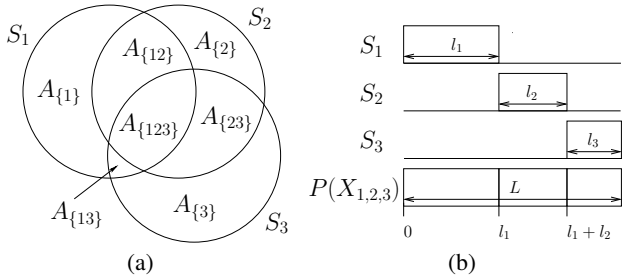


Figure 6: (a) Example coordination problem in which three sensors, $\{S_1, S_2, S_3\}$, have sensing fields that overlap (b) An optimal solution

5.1 The Wide Area Surveillance Scenario

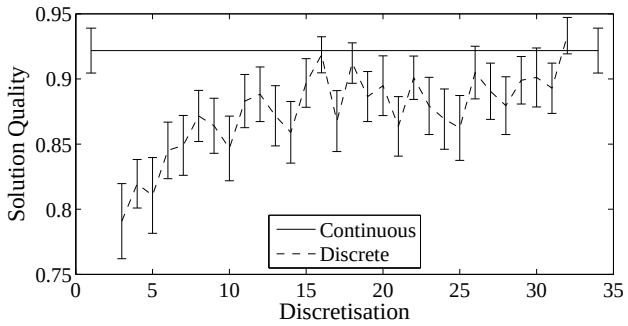
We consider a network of wireless sensors that are randomly deployed within some area to detect events (e.g. vehicle and pedestrian activity in an urban setting). We assume that these sensors are able to harvest energy from the environment (e.g. using a photovoltaic cell or vibration-harvesting microgenerators), but at a rate that is insufficient to allow them to be powered continually. Thus at any time a sensor can be in one of two states: either sensing or sleeping. In the former state the sensor consumes energy at a constant rate, and is able to interact with the surrounding environment (e.g. it can detect events within its sensing field and communicate with other sensors). In the latter state the sensor can not interact with the environment but it consumes negligible energy. To maintain energy-neutral operation [8], and thus exhibit an indefinite lifetime, sensors adopt a repeated schedule of length L, during which each sensor can be active for only a given time l_i. This amount of time depends on specific characteristics of the environment surrounding the sensor, and the way energy is harvested. For example, if a sensor is using solar panels to harvest energy, sensors which are in shaded regions will have shorter duty cycles compared to those located in spots with a greater exposure to sunlight.

The sensing ranges of multiple sensors will typically overlap. However, just a single sensor is required to be active in order to detect an event. Thus, there is no gain for the system in having more than one sensor actively sensing the same region (i.e. we have a sub-additive utility function), and hence, to maximise the probability of detecting events while maintaining energy neutral operations, sensors whose sensing fields overlap should coordinate the start times of their duty cycles. Therefore, in this setting, the continuously valued control parameter, x_i, represents the time at which sensor i will start sensing, while the domain over which this variable can take values is the interval $[0, L]$. Once the sensors have decided on the value of this parameter, they will repeat this schedule indefinitely.

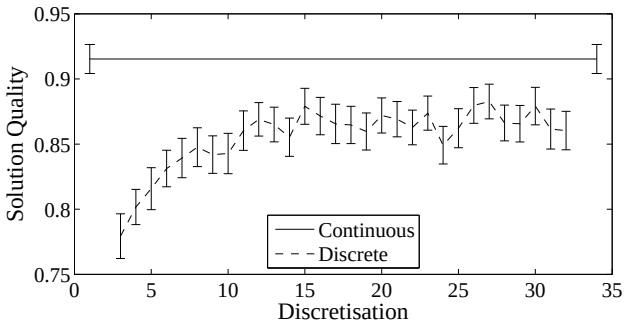
5.2 Applying the Max-Sum Algorithm

In order to apply the max-sum algorithm, in either its continuous or discrete forms, it is first necessary to instantiate the agents' utility functions for this problem. Thus, we define $\mathbf{N}_i$ to be a set of indexes indicating which other sensors' sensing fields overlap with that of sensor i and $\mathbf{k}$ is any subset of $\mathbf{N}_i$ (including the empty set). $A_{\{i\}\cup\mathbf{k}}$ is the area that is overlapped only by sensor i and those sensors in $\mathbf{k}$. For example, with respect to Figure 6(a), which shows the three overlapping sensors, the area $A_{\{1,2\}}$ is the area that is sensed only by sensors 1 and 2.[9] In a slight abuse of notation,

[9]Here, we assume that sensors have knowledge of the overlaps they have with their neighbours. While this may not hold in all applica-



(a) Solution quality after 20 iterations



(b) Averaged solution quality over 20 iterations

Figure 7: Solution quality as a fraction of the optimal solution. Error bars are the standard error in the mean.

we represent the entire sensing area of sensor S_1 as S_1, and thus, note that the area $A_{\{1,2\}}$ is different from $S_1 \cap S_2$ because the area $S_1 \cap S_2$ would include also the sub area $S_1 \cap S_2 \cap S_3$. In general, we have:

$$A_{\{i\}\cup\mathbf{k}} = \bigcap_{j\in(\{i\}\cup\mathbf{k})} S_j \setminus \bigcup_{l\notin(\{i\}\cup\mathbf{k})} S_l \tag{6}$$

The utility of sensor i is then simply given by the weighted sum of the probability of detecting an event in any particular sub area:

$$U_i(\mathbf{x}_i) = \sum_{\mathbf{k}\subseteq\mathbf{N}_i} \frac{A_{\{i\}\cup\mathbf{k}}}{|\{i\}\cup\mathbf{k}|} \times P(\mathbf{x}_{\{i\}\cup\mathbf{k}}) \tag{7}$$

where $P(\mathbf{x}_{\{i\}\cup\mathbf{k}})$ is the probability of detecting an event per unit area given the combined sensing schedules of sensors $\{i\}\cup\mathbf{k}$. In our experiments, we assume that this is simply given by the fraction of the time during which at least one sensor is actively sensing during the interval of length L. Note, that we divide each sub area by the number of sensors who can sense it to avoid double-counting areas that are represented by multiple sensors. In addition, when the set $\mathbf{k}$ is empty we consider the area covered only by the single sensor. For example, the utility of sensor S_2 shown in Figure 6(a), is calculated by considering the areas $A_{\{2\}}$, $A_{\{1,2\}}$, $A_{\{2,3\}}$ and $A_{\{1,2,3\}}$.

Given this utility function we can now decompose it into a factor graph on which we run the max-sum algorithm. As discussed in Section 3, there are several ways of doing this. Here, we use a separate function to represent the utility of each sub area and connect this function to all variable nodes that represent sensors whose sensing field overlaps with this sub area. We then remove the functions that model areas where more than two sensors overlap, which is equivalent to only considering pairwise interactions between agents. This is a very common approach in the DCOP literature, and one that reduces the computational complexity of the coordination, while still providing good solutions in this particular scenario. However, we note that our formalism supports modeling higher order interactions. We can now apply both versions of the max-sum algorithm directly to this factor graph.

5.2.1 The Discrete Version

To apply the discrete max-sum algorithm, we artificially discretise the continuously valued control parameter and only allow the sensors to select x_i from a set of d discrete values in the range $[0, L]$. The summation and marginal maximisation operators of the max-sum algorithm are performed over this discrete sample space, and the messages exchanged by the sensors are represented by single-dimensional functions evaluated at the d possible sample points.

5.2.2 The Continuous Version

To apply the continuous max-sum algorithm, we use the results derived in Section 4 to represent the utility functions of the factor graph as continuous piecewise linear functions. For example, the function in Figure 3 encodes the utility for two sensors overlapping in a sub area, with $l_1 = 2$, and $l_2 = 5$. The function has a minimum plateau when the active sensing periods of both sensors completely overlap in time (e.g. when $x_1 = x_2 = 0$), and a maximum plateau when sensors do not (e.g. when $x_1 = 0$ and $x_2 \in [2, 5]$).

Consider the following example that illustrates the difference between the discrete and continuous max-sum algorithms. Suppose that sensors are deployed as in Figure 6(a), and suppose that to maintain energy neutral operations, the three sensors can be active for l_1, l_2, and l_3 time units out of L (with $l_1 + l_2 + l_3 = L$). In this case, an optimal solution[10] is the one reported in Figure 6(b) where $x_1 = 0$, $x_2 = l_1$ and $x_3 = l_1 + l_2$. Notice that, while continuous max-sum is able to assign any value in the interval $[0, L]$ to the sensors' variables, discrete max-sum will be able to find the optimal solution only if the chosen discretisation includes the points l_1 and $l_1 + l_2$. However, as previously mentioned, the sensors' duty cycles depend on the sensor deployment and on the environment configuration, and thus they are not known before-hand. Hence, it is not possible to always choose a discretisation that includes the optimal solution and thus discrete max-sum will result in suboptimal solutions.

5.3 Experimental Results

As described above, we performed experiments comparing the performance of discrete and continuous max-sum algorithms to assess the benefits and limitations of our approach. In particular, we compare the two algorithms by considering the quality of the solution and the communication overhead in terms of the size of messages that the agents must exchange. We performed experiments on deployments of 10 sensors, that are randomly scattered across a unit square. These sensors have a circular shaped sensing area with a radius of 0.2. The sensors' duty cycles l_i are drawn from a uniform distribution over $[0.3, 0.6]$.

During the experiments we ran both versions of the max-sum algorithm for 20 iterations. Each experiment consisted of a single run of our continuous max-sum algorithm, and multiple runs of the discrete max-sum algorithm with increasing levels of discretisation.

tions, we use this simplified model here to focus on the coordination aspects of this application scenario.

[10]Note that, in this case, there is an infinite number of optimal solutions, which can be generated by shifting the starting times of all sensors by an equal amount.

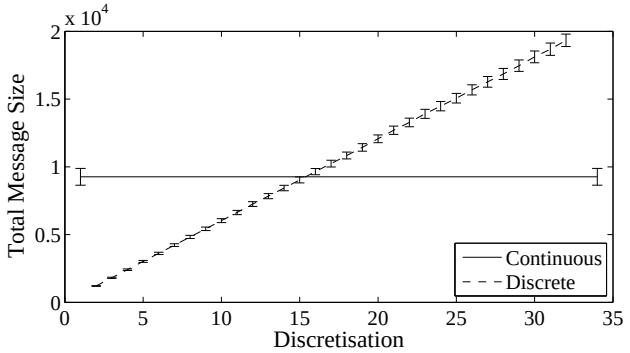


Figure 8: Total number of values exchanged between the agents. Error bars are the standard error in the mean.

Figure 7 shows the aggregated results of 100 runs, where the solution quality is expressed in terms of the optimal solution computed with a centralised simulated annealing algorithm.

Specifically, Figure 7(a) reports the quality of the final solution (e.g., after the 20 iterations), while Figure 7(b) reports the average quality of the solutions obtained after each iteration. The latter metric incorporates information on how the algorithms behave over time; the quicker the algorithms converge towards better solutions, the higher the average.

Figure 7(a) shows that the final solutions produced by continuous max-sum are better than those produced by discrete the discrete version. In particular, continuous max-sum exhibit up to a 10% increase in the solution quality for low discretisation levels. Moreover, Figure 7(b) shows that, when considering the average quality of solution, the difference is more pronounced also for higher discretisation levels, thus showing that continuous max-sum is able to reach good, stable solutions quicker than the discrete version.

In terms of total message size, we can conclude from Figure 8 that, as expected, the communication overhead of discrete max-sum increases proportionally to the level of discretisation. Significantly, the continuous max-sum algorithm achieves better solution quality over the entire range of discretisations, even when the message size of the discrete max-sum algorithm is greater than the continuous version. Thus the continuous version generates better solutions, and also requires less communication overhead.

6. CONCLUSIONS

In this paper, we presented a novel decentralised algorithm for social welfare maximisation in multiagent systems. In particular, we focussed on scenarios where the agents' control parameters are continuous, and the interactions between the agents is expressable as a piecewise linear functions. We extended the existing max-sum algorithm to operate in the domain of continuous variables by using techniques from computational geometry. We empirically evaluated our approach by applying it to a wide area surveillance scenario where energy constrained agents need to coordinate their sense/sleep schedules to maximise the probability of event detection in a decentralised fashion. We compared the continuous max-sum algorithm with its conventional discrete counterpart, and showed that our continuous algorithm is able to achieve solutions of better quality while exhibiting a lower communication overhead.

Our future work in this area is to extend the proposed approach by considering arbitrary functions that describe the agents' interactions. A promising direction to do this is to investigate the use of techniques such as Gaussian processes (GPs). GPs provide a powerful mathematical tool to approximate arbitrary continuous functions, and thus, could be extremely useful to describe the agents' interactions that are non-linear. An application scenario where this is the case is, for example, the coordinated exploration of an uncertain environments using mobile sensors. In this setting, the sensors' goal is to reduce the uncertainty they have about the environment, therefore the agent interactions are related to the amount of information that sensors obtain and such measures usually require the use of strongly non-linear functions.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

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

[2] A. Farinelli, A. Rogers, A. Petcu, and N.R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proc. of the 7th Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 639–646, 2008.

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

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

[5] B. Grocholsky, J. Keller, V. Kumar, and G. Pappas. Cooperative air and ground surveillance. *IEEE Robotics & Automation Magazine*, 13(3):16–25, 2006.

[6] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inf. Process. Lett.*, 33(4):169–174, 1989.

[7] C. Hsin and M. Liu. Network coverage using low duty-cycled sensors: Random & coordinated sleep algorithm. In *Proc. of the 3rd Int. Symposium on Information Processing in Sensor Networks*, pages 433–442, 2004.

[8] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *ACM Transactions on Embedded Computing Systems*, 6(4), 2007.

[9] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 42(2):498–519, 2001.

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

[11] R. J. Maheswaran, J. Pearce, and M. Tambe. A family of graphical-game-based algorithms for distributed constraint optimization problems. In *Coordination of Large-Scale Multiagent Systems*, pages 127–146. Springer-Verlag, 2005.

[12] R. Mailler and V. Lesser. Solving distributed constraint optimization problems using cooperative mediation. In *Proc. of the 3rd Int. Joint Conf. on Autonomous Agents and MultiAgent Systems*, pages 438–445, 2004.

[13] 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.

[14] M. Paskin, C. Guestrin, and J. McFadden. A robust architecture for distributed inference in sensor networks. In *Proc. of the 4th Int. Symposium on Information Processing in Sensor Networks*, page 55–62, 2005.

[15] A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence*, pages 266–271, 2005.

[16] Y. Weiss and W. T. Freeman. 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.