Decentralised Coordination of Mobile Sensors Using the Max-Sum Algorithm

Ruben Stranders[*] and **Alessandro Farinelli**[*†] and **Alex Rogers**[*] and **Nicholas R. Jennings**[*]
{rs06r,af2,acr,nrj}@ecs.soton.ac.uk
[*] School of Electronics and Computer Science, University of Southampton, UK
[†] Department of Computer Science, University of Verona, Italy

Abstract

In this paper, we introduce an on-line, decentralised coordination algorithm for monitoring and predicting the state of spatial phenomena by a team of mobile sensors. These sensors have their application domain in disaster response, where strict time constraints prohibit path planning in advance. The algorithm enables sensors to coordinate their movements with their direct neighbours to maximise the collective information gain, while predicting measurements at unobserved locations using a Gaussian process. It builds upon the max-sum message passing algorithm for decentralised coordination, for which we present two new generic pruning techniques that result in speed-up of up to 92% for 5 sensors. We empirically evaluate our algorithm against several on-line adaptive coordination mechanisms, and report a reduction in root mean squared error up to 50% compared to a greedy strategy.

1 Introduction

In disaster response, and many other applications besides, the availability of timely and accurate information is of vital importance. Thus, the use of multiple mobile sensors for information gathering in crisis situations has generated considerable interest.[1] These mobile sensors could be autonomous ground robots or unmanned aerial vehicles. In either case, while patrolling through the disaster area, these sensors need to keep track of the continuously changing state of spatial phenomena, such as temperature or the concentration of potentially toxic chemicals. The key challenges in so doing are twofold. First, the sensors cannot cover the entire environment at all times, so the spatial and temporal dynamics of the monitored phenomena need to be identified in order to predict environmental conditions in parts of the environment that can not be sensed directly. Second, the sensors need to coordinate their movements to collect the most informative measurements needed to predict these environmental conditions as accurately as possible.

[1] For example, one of the missions of both the Aladdin project (`http://www.aladdinproject.org`) and the Centre for Robot Assisted Search & Rescue (`http://crasar.csee.usf.edu`) is to use autonomous robots for information gathering in disaster response scenarios.

Recent work has addressed similar challenges by modelling the spatial and temporal dynamics of the phenomena using Gaussian processes (GPs) [Rasmussen and Williams, 2006]. GPs are a powerful Bayesian approach for inference about functions, and have been shown to be an effective tool for capturing the dynamics of spatial phenomena [Cressie, 1993]. This principled approach to modelling the environment has been used to compute informative deployments of fixed sensors [Guestrin *et al.*, 2005], and informative paths for single [Meliou *et al.*, 2007] and multiple mobile sensors [Singh *et al.*, 2007].

However, the algorithms used to compute these informative deployments and paths are not suitable in our domain, since they are geared towards solving a one-shot optimisation problem in an off-line phase. Moreover, these algorithms are centralised. In hostile environments, this is undesirable, because it creates a single point of failure, thereby increasing the vulnerability of the information stream. Other work has employed on-line decentralised path planning using artificial potential fields to keep sensors in specific favourable formations [Fiorelli *et al.*, 2006], or through multi-agent negotiation techniques to partition the environment and allocate the sensors to these partitions [Ahmadi and Stone, 2006]. However, in general, this work has used representations of the environment, that are less sophisticated than Gaussian processes, and are thus, less applicable for modelling complex spatial and temporal correlations.

To address this shortcoming, Low *et al.* [2008] combine these approaches and use Gaussian processes to represent the environment, and use Markov decision processes to compute non-myopic paths for multiple mobile sensors in an on-line fashion. Whilst such a non-myopic approach avoids the problem of local minima, it incurs significant computational cost (it is only empirically evaluated for systems containing just two sensors), and is again a centralised solution.

Thus, against this background, in this paper, we present a new on-line, decentralised coordination algorithm for teams of mobile sensors. This algorithm computes coordinated paths with an adjustable look-ahead, thus allowing the trade off between computation and solution quality, and uses GPs to represent generic temporal and spatial correlations of the phenomena. To this end, we represent each sensor as an autonomous agent. These agents are capable of taking measurements, coordinating their actions with their immediate neighbours, and predicting the state of the spatial phenomenon at

unobserved locations. We then use the max-sum algorithm for decentralised coordination [Farinelli *et al.*, 2008] to have the agents negotiate a joint plan by exchanging messages with their immediate neighbours. By applying max-sum in this manner, every agent controls its own movements using information it possesses locally, and the coordination mechanism is decentralised. We choose max-sum because it has been shown to generate good solutions to decentralised coordination problems, while limiting computation and communication. However, a standard application of max-sum is still too computationally costly within our particular domain. Thus, we introduce two novel and generic pruning techniques that speed up the max-sum algorithm, and hence, make decentralised coordination tractable in our application.

In more detail, in this paper we contribute to the state of the art in the following ways:

- We cast the multi-sensor monitoring problem as a decentralised constraint optimisation problem (DCOP), and present a new decentralised on-line coordination mechanism based on the max-sum algorithm to solve it.
- We present two novel, generic pruning techniques specifically geared towards reducing the number of function evaluations that is performed by max-sum. Thus alleviating a major bottleneck of this algorithm.
- We empirically show that a specific instantiation of our approach prunes 92% of joint moves for 5 sensors, and outperforms a greedy single step look-ahead algorithm by up to 50% in terms of root mean squared error.

The remainder of this paper is structured as follows. In section 2 we give a formal problem description. Section 3 describes how spatial phenomena are modelled. In section 4, we present our distributed algorithm, which we empirically evaluate in section 5, before concluding in section 6.

2 Problem Description

The problem formulation described in this section was inspired by [Meliou *et al.*, 2007], and has been extended to deal with multiple sensors and limited local knowledge. Consider an environment in which M sensors monitor spatial phenomena that are modeled by a scalar field $\mathcal{F} : \mathbb{R}^3 \rightarrow \mathbb{R}$, defined on one temporal and two spatial dimensions, at a finite set of locations $V = \{v_1, v_2 \dots\} \subset \mathbb{R}^2$, and an indeterminate[2] number of of discrete time steps $T = \{t_1, t_2, ...\}$. To the measurement at location $v \in V$, and time t we associate a continuous random variable, $\mathcal{X}_{v,t}$. The set of all random variables is denoted by $\mathcal{X}$. The layout of the physical environment is given by a graph $G = (V, E)$, where E encodes the possible movements between locations V. The locations accessible from v are denoted by $adj_G(v)$. Since it is generally not possible to visit all locations V during a single time step, each sensor selects an adjacent location at which to take a measurement at time step $t + 1$. Values at locations V are subsequently predicted with a statistical model using all measurements that the sensors have gathered so far. In order to do this, we model the scalar field $\mathcal{F}$ with a GP (see next section) that encodes both its spatial and temporal correlations.

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

Now, in order to select their movements, sensors need to be able to predict the informativeness of the samples that are collected along their paths with respect to the missing ones. Here, the informativeness of a set of samples O is quantified by a function $f(O)$, that, depending on the context, can take on different forms [Meliou *et al.*, 2007]. Our choice for this function f will be derived in the next section.

Given this formalisation, we define the multi-sensor monitoring problem as follows. For every time step t, maximise $f(O_t)$, where $O_t = \cup_{i=1}^{M} O_t^i$ is the set of samples collected by all M sensors up to time step t at which the prediction is made. Moreover, while doing so, sensors can only communicate with and be aware of their immediate neighbours, such that no single point of control exists.

This problem is very challenging even for a single sensor. We therefore propose a distributed algorithm that computes paths with an adjustable look-ahead in Section 4, but first we discuss the way in which the spatial phenomena are modelled and derive function f.

3 Modeling the Spatial Phenomena

In order to predict measurements at unobserved locations, we model the scalar field $\mathcal{F}$ with a GP. Using a GP, $\mathcal{F}$ can be estimated at any location and at any point in time based on a set of samples collected by the sensors [Rasmussen and Williams, 2006]. In more detail, a single sample o of the scalar field $\mathcal{F}$ is a tuple $\langle \mathbf{x}, y \rangle$, where $\mathbf{x} = (v, t)$ denotes the location and time at which the sample was taken, and y the measured value. Now, if we collect the training inputs $\mathbf{x}$ in a matrix $\mathbf{X}$, and the outputs y in a vector $\mathbf{y}$, the predictive distribution of the measurement at spatio-temporal coordinates $\mathbf{x}_*$, conditioned on previously collected samples $O_t = \langle \mathbf{X}, \mathbf{y} \rangle$ is Gaussian with mean μ and variance σ^2 given by:

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

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

where $K(\mathbf{X}, \mathbf{X}')$ denotes the matrix of covariances for all pairs of rows in $\mathbf{X}$ and $\mathbf{X}'$. These covariances are obtained by evaluating a function $k(\mathbf{x}, \mathbf{x}')$, called a covariance function, which encodes the spatial and temporal correlations of the pair $(\mathbf{x}, \mathbf{x}')$. Generally, covariance is a non-increasing function of the distance in space and time. For example, a prototypical choice is the squared exponential function where the covariance decreases exponentially with this distance:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\tfrac{1}{2} |\mathbf{x} - \mathbf{x}'|^2 / l^2\right) \tag{3}$$

where σ_f and l are called *hyperparameters* that model the signal variance and the length-scale of the phenomenon respectively. The latter determines how quickly the phenomenon varies over time and space[3]. If these hyperparameters are unknown before deployment of the sensors, they can be efficiently learnt on-line from collected samples using Bayesian Monte Carlo [Osborne *et al.*, 2008].

One of the features of the GP is that the posterior variance in Equation 2 is independent of actual measurements $\mathbf{y}$. This

[3]A slightly modified version of Equation 3 allows for different length-scales for the spatial and temporal dimensions of the process.

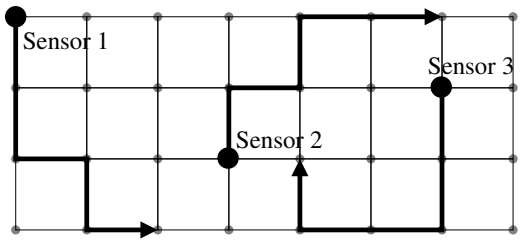


Figure 1: Joint plan of length 5 for sensors on a lattice graph.

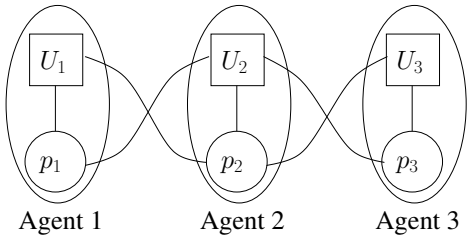


Figure 2: Factor graph with three agents.

allows the sensors to determine the variance reduction that results from collecting samples along a certain path without the need of actually collecting them. Using this feature, we define the value $f(O)$ to be the *reduction in entropy* that results at the coordinates of O after taking these samples[4]. This function exhibits the property of *locality* [Guestrin *et al.*, 2005], that is exploited by our algorithm. This means that the correlation between two samples decreases rapidly (exponentially in the case of Equation 3) with increasing distance, such that samples that are far apart can be considered uncorrelated, and thus, mobile sensors that are far apart need not explicitly coordinate.

4 Decentralised Coordination

Since, in general, it is too expensive to perform on-line non-myopic path planning, especially because the number of time steps in T (the mission time) is unknown beforehand, we present an algorithm that computes joint moves with a finite, adjustable look-ahead. An example of such a joint move for three sensors, consisting of a path of length 5 for each of them, is shown in Figure 1. The sensors run the algorithm every n time steps[5] to plan joint paths of length $l \geq n$. The algorithm is flexible, in that it allows paths of various lengths, and additional constraints, to be considered. For instance, the sensors can coordinate over all possible paths of length l, or only those moves that keep them within their own partition of the environment.

Given any particular potential joint move, our algorithm must calculate the reduction in entropy, $f(\cup_{i=1}^{M} O_t^i)$, that will result. In order to do so, we apply the chain-rule of entropies[6] to convert $f(\cup_{i=1}^{M} O_t^i)$ to $\sum_{i=1}^{M} f(O_t^i)$. This modified problem can then be cast as a decentralised constraint optimisation problem (DCOP), which lends itself to an agent-based solution paradigm. In this paradigm, sensors are modelled as agents that jointly plan their paths in order to maximise the value of the collected samples.

More formally, we denote the decision variable of agent i as p_i, which takes values in the set $A_i = \{a_i^1, \ldots, a_i^{q_i}\}$, representing all q_i moves that agent i is currently considering. The value of the samples that agent i collects along its path, given the movement of other sensors, is denoted by $U_i(\mathbf{p}_i)$, where $\mathbf{p}_i$ is the vector of decision variables on which agent i's utility depends (thus, $p_i \in \mathbf{p}_i$ will always hold). By the property of *locality* of f, this dependency relation is usually limited to a small set of nearby agents. Now, the agents will collectively attempt to find joint move $\mathbf{p}^* = [p_1^*, \ldots, p_M^*]$, such that:

$$\mathbf{p}^* = \arg\max_{\mathbf{p}} \sum_{i=1}^{M} U_i(\mathbf{p}_i) \tag{4}$$

or, in other words, the joint move that maximises the total value obtained by the agents.

4.1 The Max-Sum Message-Passing Algorithm

The coordination problem encoded by Equation 4 is a DCOP, which can be solved by a wide range of algorithms. Unfortunately, many of these algorithms either compute the optimal solution at exponential cost, either in terms of the number or size of messages that are exchanged between agents (e.g. DPOP [Petcu and Faltings, 2005]), or require little local computation and communication, but produce approximate solutions (e.g. the Distributed Stochastic Algorithm [Fitzpatrick and Meertens, 2003]). However, there exists a class of algorithms usually referred to under the framework of the Generalised Distributive Law [Aji and McEliece, 2000], that can be used to obtain good approximate solutions. The max-sum message passing algorithm is one member of this class that is of particular interest here. This algorithm has been shown to compute better quality solutions than the approximate class with acceptable computation compared to representative complete algorithms [Farinelli *et al.*, 2008].

In more detail, the max-sum algorithm operates on a *factor graph*: an undirected bipartite graph in which vertices represent variables p_i and functions U_j. In such factor graphs, an edge exists between a variable p_i and a function U_j iff $p_i \in \mathbf{p}_j$, (i.e., p_i is a parameter of U_j). Using the max-sum algorithm we exploit the fact that an agent's utility depends only on a subset of other agents' decision variables (locality), and that the global utility function is a sum of each agent's utility. Figure 2 shows an example factor graph that encodes Equation 4 for the coordination problem of Figure 1. In this example, the utility of agent 1 depends on its own action, and that of agent 2, so $\mathbf{p}_1 = \{p_1, p_2\}$. Similarly, $\mathbf{p}_2 = \{p_1, p_2, p_3\}$, and $\mathbf{p}_3 = \{p_2, p_3\}$.
In yet more detail, using max-sum, each agent computes:

$$\tilde{U}_i(p_i) = \max_{\mathbf{p}_{-i}} \sum_{i=1}^{M} U_i(\mathbf{p}_i) \tag{5}$$

in a distributed way (i.e. based on local information and communication with direct neighbours). Agent i's optimal move

[4]Note that Guestrin *et al.* [2005] show that mutual information can also be used to define this value. However, the gain in doing so is small, and it is much more computationally expensive to evaluate.

[5]For simplicity, and w.l.o.g. we assume that the speed of the sensors is 1 unit per time step.

[6]This rule states that $H(\mathcal{X}_{o_1}, \ldots, \mathcal{X}_{o_n}) = H(\mathcal{X}_{o_1} | \mathcal{X}_{o_2}, \ldots, \mathcal{X}_{o_n}) + H(\mathcal{X}_{o_2} | \mathcal{X}_{o_3}, \ldots, \mathcal{X}_{o_n}) + \cdots + H(\mathcal{X}_{o_n})$.

p_i^* is then obtained as follows:

$$p_i^* = \arg\max_{p_i} \tilde{U}_i(p_i) \quad (6)$$

In order to do this, messages are passed between the functions U_i, and the variables in $\mathbf{p}_i$ as described below[7]:

- **From variable to function:**

$$Q_{p_n \to U_m}(p_n) = \alpha_{nm} + \sum_{\substack{U_{m'} \in adj(p_n) \\ U_{m'} \neq U_m}} R_{U_{m'} \to p_n}(p_n) \quad (7)$$

  where α_{nm} is a normalising constant that is chosen such that $\sum_{p_n} Q_{n \to m}(p_n) = 0$, to prevent the messages from growing arbitrarily large.

- **From function to variable:**

$$R_{U_m \to p_n}(p_n) = \max_{\mathbf{p}_{-n}} \left[U_m(\mathbf{p}_m) + \sum_{\substack{p_{n'} \in adj(U_m) \\ p_{n'} \neq p_n}} Q_{p_{n'} \to U_m}(p_{n'}) \right] \quad (8)$$

Finally, $\tilde{U}_i(p_i)$ is obtained by summing the most recent messages $R_{U_{m'} \to p_i}(p_i)$ received by p_i. This computation is guaranteed to be exact when the factor graph is acyclic. However, since dependencies are usually mutual (i.e. agent i's action influences agent j's utility and vice versa), the factor graph will normally contain cycles. In this case, max-sum computes approximate solutions (i.e $\tilde{U}_i(p_i) \approx \max_{\mathbf{p}_{-i}} \sum_{i=1}^{M} U_i(\mathbf{p}_i)$). Despite this, however, there exists strong empirical evidence that max-sum produces good results even in cyclic factor graphs [Farinelli *et al.*, 2008].

4.2 Speeding up Message Computation

The straightforward application of max-sum to solve Equation 4 is not practical, because the computation of the messages from function to variable (Equation 8) is a major bottleneck. A naïve way of computing these messages for a given variable p_n is to enumerate all joint moves (i.e. the domain of $\mathbf{p}_m$), and evaluate U_m for each of these moves. Since the size of this joint action space grows exponentially with both the number of agents, and the number of available moves for each agent, the amount of computation quickly becomes prohibitive. This is especially true when evaluating U_m is costly, as is the case in the mobile sensors domain[8]. Therefore, we introduce two pruning algorithms to reduce the size of the joint action space that needs to be considered. These algorithms are then applied to our mobile sensor domain, but they can just as easily be applied within other settings.

The Action Pruning Algorithm

The first algorithm attempts to reduce the number of moves each agent needs to consider *before* running the max-sum algorithm. This algorithm prunes the dominated states that can never maximise Equation 4, regardless of the actions of other agents. More formally, a move $a' \in A_n$ is dominated if there exists a move a^* such that:

$$\forall \mathbf{a}_{-n} \sum_{U_m \in adj(p_n)} U_m(a', \mathbf{a}_{-n}) \leq \sum_{U_m \in adj(p_n)} U_m(a^*, \mathbf{a}_{-n}) \quad (9)$$

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

- **From function to variable**: Function U_m sends a message to p_n, containing the minimum and maximum values of U_m with respect to $p_n = a_n$, for all $a_n \in A_n$. (see Algorithm 1).
- **From variable to function**: Variable p_n sums the minimum and maximum values from each of its adjacent functions, and prunes dominated states. It then informs neighbouring functions of its updated domain (see Algorithm 2).

Using this distributed algorithm, functions continually refine the bounds on the utility for a given state of a variable, which potentially causes more states to be pruned. Therefore, it is possible that action pruning starts with a single move, and subsequently propagates through the entire factor graph.

Now, given the highly non-linear relations expressed in Equation 2, on which the agents' utility functions U_m are based, it is very difficult to calculate these bounds exactly, without exhaustively searching the domain of $\mathbf{x}_m$ for utility function U_m. Needless to say, this would defeat the purpose of this pruning technique. Nonetheless, experimentation showed that by computing these bounds in a greedy fashion, a very good approximation is obtained. Thus, the lower bound $\underline{U_m}(a_n)$ on a move a_n is obtained by selecting the neighbouring agents one at a time, and finding the move that reduces the utility of agent m's move the *most*. In a similar vein, the upper bound $\overline{U_m}(a_n)$ is obtained selecting those moves of other sensors that reduce the utility the *least.*

The Joint Action Pruning Algorithm

Whereas the first algorithm runs as a preprocessing phase to max-sum, the second algorithm is geared towards speeding

Algorithm 1 Algorithm for computing pruning message from function U_m to variable p_n

1: compute $\overline{U_m}(p_n) \leq \min_{\mathbf{p}_{-n}} U_m(p_n, \mathbf{p}_{-n})$
2: compute $\underline{U_m}(p_n) \geq \max_{\mathbf{p}_{-n}} U_m(p_n, \mathbf{p}_{-n})$
3: send $\langle \overline{U_m}(p_n), \underline{U_m}(p_n) \rangle$ to p_n

Algorithm 2 Algorithm for computing pruning messages from variable p_n to all functions $U_m \in adj(p_n)$

1: **if** a new message has been received from all $U_m \in adj(p_n)$ **then**
2: compute $\perp(p_n) = \sum_{U_m \in adj(p_n)} \underline{U_m}(p_n)$
3: compute $\top(p_n) = \sum_{U_m \in adj(p_n)} \overline{U_m}(p_n)$
4: **while** $\exists a \in A_n : \top(a) < \max \perp(p_n)$ **do**
5: $A_n \leftarrow A_n \setminus \{a\}$
6: **end while**
7: send updated domain A_n to each $U_m \in adj(p_n)$
8: **end if**

[7] In what follows, we use $adj(U_m)$ to denote adjacent vertices of function U_m in the factor graph, i.e., the set of variables in the domain $\mathbf{p}_m$ of U_m. Similarly, $adj(p_n)$ denotes the set of functions in which p_n occurs in the domain.

[8] Specifically, determining the value of a sample involves the inversion of a potentially very large matrix $K(\mathbf{X}, \mathbf{X})$ (Equation 2).

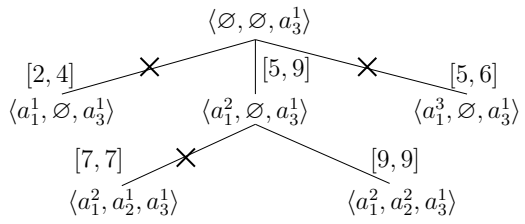


Figure 3: Search-tree for computing $R_{U_m \to x_3}(a_3^1)$ showing lower and upper bounds on the maximum value in the subtree.

up the computation of the messages from function to variable (Equation 8), *while* max-sum is running. A naïve way of computing this message to a single variable p_i is to determine the maximum utility for each of agent i's actions by exhaustively enumerating the joint domain of the variables in $\mathbf{p}_m \backslash \{p_i\}$ (i.e. the Cartesian product of the domains of these variables), and evaluating the expression between brackets in Equation 8, which we denote by $\tilde{R}_{U_m \to p_n}(\mathbf{p}_m)$. This expression is the sum of the utility function U_m and the sum of messages Q.

However, instead of just considering joint moves, we now allow some actions to be undetermined, and thus, consider *partial* joint moves, denoted by $\hat{\mathbf{a}}$. By doing so, we can create a search tree on which we can employ branch and bound to significantly reduce the size of the domain that needs to be searched. In more detail, to compute $R_{U_m \to p_n}(a_n^i)$ (a single element of the message from U_m to variable p_n) for a single state $a_n^i \in A_i$ in the domain of p_n, we create a search tree $\mathcal{T}(a_n^i)$ as follows:

- The root of $\mathcal{T}(a_n^i)$ is a partial joint move $\hat{\mathbf{a}}_r = \langle \varnothing, \ldots, \varnothing, a_n^i, \varnothing, \ldots, \varnothing \rangle$, which indicates that a_n^i is assigned to p_n, and the remaining variables are unassigned (denoted by $\varnothing$).
- The children of a vertex $\langle a_1^{(1)}, \ldots, a_k^{(k)}, \varnothing, \ldots \varnothing, a_n^i, \varnothing, \ldots, \varnothing \rangle$ are obtained by setting the first unassigned variable p_{k+1} to each of its $|A_{k+1}|$ moves.
- The leafs of the tree represent a (fully determined) joint move $\mathbf{a}_m$ (i.e. $\forall i : p_i \neq \varnothing$). In the tree, only leafs are assigned a value, which is equal to $\tilde{R}_{U_m \to p_n}(\mathbf{a}_m)$.

The maximum value found in $\mathcal{T}(a_n^i)$ is the desired value. Now, in order to use branch and bound to find this value, we need to put bounds on the maximum value found in a subtree of $\mathcal{T}(a_n^i)$. These bounds depend on U_m and the received messages Q. Now, in many cases we can put bounds on the maximum of the former, that is obtained by further completing a partial joint move $\hat{\mathbf{a}}'$ in a subtree of $\mathcal{T}(a_n^i)$. The bounds on U_m, combined with the minimum and maximum values of Q for $\hat{\mathbf{a}}'$ (again, by further completing the partial joint move), gives us the desired bounds.

Figure 3 shows an example of a partially expanded search tree for computing a single element $R_{U_m \to x_3}(a_3^1)$ of a message from function U_m to variable p_3. Given the lower and upper bounds on the maximum, subtree $\langle a_1^1, \varnothing, a_3^1 \rangle$ can be pruned immediately after expanding the root. Similarly, subtree $\langle a_1^3, \varnothing, a_3^1 \rangle$ is pruned after expanding leaf $\langle a_1^2, a_2^2, a_3^1 \rangle$, which has the desired maximum value.

To compute these bounds on the maximum of $U_i(\hat{\mathbf{a}})$ in the mobile sensor domain, note that partial joint move $\hat{\mathbf{a}}$ represents a situation in which only a subset of the agents have determined their move. Using this interpretation, we can obtain bounds as follows. The upper bound on this value is obtained by disregarding the agents that have not determined their action (i.e. agents i for which $p_i = \varnothing$). Since the act of collecting a sample always reduces the value of other samples, disregarding the samples of these 'undecided' agents will give an upper bound on the maximum. To obtain a lower bound on the maximum, we use the locality property of f, which tells us that the interdependency between values of samples weakens as their distance increases. So, in order to calculate the lower bound, we move the undecided agents away from agent i's destination.

4.3 Ensuring Network Connectivity

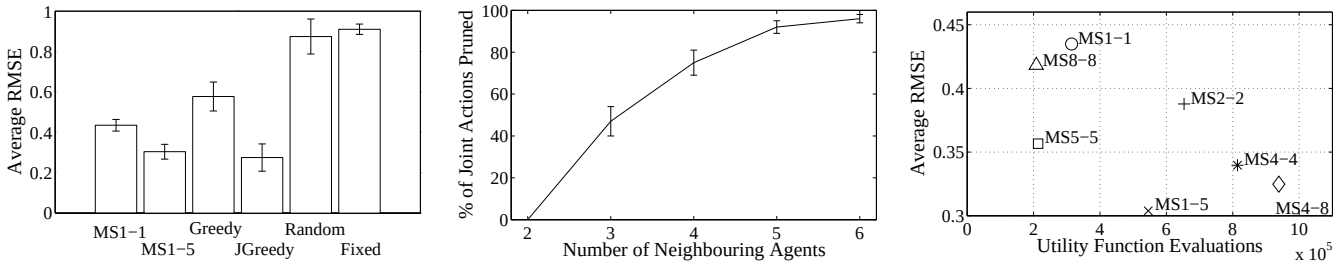
In many situations, it is also important that the sensors maintain network connectivity in order to transmit their measurements to a base station. More importantly in the context of our algorithm, agents need to be able to communicate in order to negotiate over their actions. Not surprisingly, we can use the algorithm to accomplish this, by penalising disconnection from the network in the utility function U_i. To this end, we assume that every agent maintains a routing table that specifies which agents can be reached through each immediate neighbour. Thus, a move is only allowed if all agents will still be reachable through the remaining links. Otherwise, the agent risks disconnection from the network, in which case a large penalty is added to its utility function U_i.

5 Empirical Evaluation

To empirically evaluate our approach, we simulated five sensors on a lattice graph measuring 26 by 26 vertices. The data was generated by a GP with a squared exponential covariance function (see Equation 3) with a spatial length-scale of 10 and a temporal length-scale of 150. This means that the spatial phenomenon has a strong correlation along the temporal dimension, and therefore changes slowly over time. At every m time steps, the sensors plan their motion for the next l time steps ($l \geq m$). In what follows, this strategy is referred to as MSm-l. Now, instead of considering all possible paths of length l from an agent's current position, which would result in a very high computational overhead, the action space is limited to the locations in G that can be reached in l time steps in 8 different directions, corresponding to the major directions on the compass rose. In the first experiment, we benchmarked MS1-1 and MS1-5 against four strategies:

- **Random:** Randomly moving sensors.
- **Greedy:** Sensors that greedily maximise the value of the sample collected in the next move without coordination.
- **J(umping) Greedy:** The same as Greedy, except that these sensors can instantaneously jump to any location.
- **Fixed:** Fixed sensors that are placed using the algorithm proposed in [Guestrin *et al.*, 2005].

The averaged root mean squared error (RMSE) for 100 time steps is plotted in Figure 4(a). From this figure, it is clear that both MS strategies outperform the Greedy, Random, and Fixed strategies. Furthermore, the prediction accuracy of



(a) (b) (c)

Figure 4: The Experimental Results. Errorbars indicate the standard error in the mean.

MS1-5 is comparable to that of JGreedy, whose movement is not restricted by graph G. Moreover, it shows that increasing the length of the considered paths from 1 to 5, reduces the RMSE by approximately 30%.

In the second set of experiments, we analysed the speed-up achieved by applying the two pruning techniques described in Section 4.2. Figure 4(b) shows the percentage of joint actions pruned plotted against the number of neighbouring agents. With 5 neighbours, the two pruning techniques combined prune around 92% of the joint moves. With such a number of neighbouring agents, the agents are strongly clustered, which occurs rarely in a large environment. However, should this happen, the utility function needs to be evaluated for only 8% of roughly 8^5 joint actions, thus greatly improving the algorithm's efficiency.

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

A video demonstrating the mobile sensors with the techniques from Section 4 can be found at `www.youtube.com/ijcai09`.

6 Conclusions

In this paper, we presented an on-line decentralised coordination algorithms for multiple sensors. We showed how the max-sum message passing algorithm can be applied to this domain in order to coordinate the motion paths of the sensors along which the most informative samples are gathered. We also presented two general pruning to speed up the max-sum algorithm. The first attempts to prune actions of the sensors that are not part of the optimal joint move. The second uses branch and bound to reduce the fraction of the joint action space that needs to be searched in order to compute the messages from functions to variables, which is the main bottleneck of the max-sum algorithm. We empirically showed that for 5 sensors, these techniques prune 92% of joint moves, thus significantly reducing the number of utility function evaluations, which are particularly expensive in the mobile sensor domain. Moreover, we showed that root mean squared error with which the spatial phenomenon is predicted by the MS1-5 strategy is approximately 50% lower compared to a greedy single step look-ahead algorithm. Our future work in this area is to extend the proposed approach by adopting techniques from sequential decision making to do non-myopic path planning, while keeping computational costs in check.

Acknowledgments 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/C 548051/1).

References

[Ahmadi and Stone, 2006] M. Ahmadi and P. Stone. A multi-robot system for continuous area sweeping tasks. In *ICRA-06*, pages 1724–1729, 2006.

[Aji and McEliece, 2000] S. M. Aji and R. J. McEliece. Generalized Distributive Law. *IEEE Transactions on Information Theory*, 46(2):325 – 343, 2000.

[Cressie, 1993] N. A. Cressie. *Statistics for Spatial Data*. Wiley-Interscience, 1993.

[Farinelli *et al.*, 2008] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *AAMAS-08*, pages 639–646, 2008.

[Fiorelli *et al.*, 2006] E. Fiorelli, N. E. Leonard, P. Bhatta, D. A. Paley, R. Bachmayer, and D. M. Fratantoni. Multi-AUV control and adaptive sampling in monterey bay. *IEEE Journal of Oceanic Engineering*, 31(4):935 – 48, 2006.

[Fitzpatrick and Meertens, 2003] S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. *Distributed Sensor Networks*, pages 257–295. Kluwer Academic Publishers, 2003.

[Guestrin *et al.*, 2005] C. Guestrin, A. Krause, and A. P. Singh. Near-optimal sensor placements in gaussian processes. In *ICML-05*, pages 265–272, 2005. ACM Press.

[Meliou *et al.*, 2007] A. Meliou, A. Krause, C. Guestrin, and J. M. Hellerstein. Nonmyopic informative path planning in spatio-temporal models. In *AAAI-07*, pages 602–607, 2007.

[Osborne *et al.*, 2008] M. A. Osborne, A. Rogers, S. D. Ramchurn, S. J. Roberts, and N. R. Jennings. Towards real-time information processing of sensor network data using computationally efficient multi-output gaussian processes. In *IPSN-08*, 2008.

[Petcu and Faltings, 2005] Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *IJCAI-05*, pages 266–271, 2005.

[Rasmussen and Williams, 2006] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[Singh *et al.*, 2007] A. Singh, A. Krause, C. Guestrin, W. J. Kaiser, and M. A. Batalin. Efficient planning of informative paths for multiple robots. In *IJCAI-07*, pages 2204–2211, 2007.