Sensor Placement and Coordination via Distributed Multi-Agent Cooperative Control

Alexandros Papangelis[1], Vangelis Metsis[1], John Shawe-Taylor[2] and Fillia Makedon[1]
[1]Heracleia Human-Centered Computing Laboratory
Dept. of Computer Science and Engineering
University of Texas at Arlington
416 Yates St., Arlington, TX 76019-0015
[2]Centre for Computational Statistics and Machine Learning
University College London
London, WC1E 6BT, UK
alexandros.papangelis@mavs.uta.edu, vmetsis@uta.edu, jst@cs.ucl.ac.uk, makedon@uta.edu

ABSTRACT

This paper examines the problem of sensor placement and coordination to maximize the sensor utilization when monitoring different types of environments. Our assumption is that the sensors are mobile and each sensor can have more than one type of sensing capabilities which can be active or not at each specific moment. The goal is to maximize the amount of information collected from the environment, given the limited amount of resources that the total of the available sensors can provide, and at the same time to be fault tolerant in failures of individual sensors by using a decentralized approach that re-organizes their placement in case of failures.

We tackle this problem by employing a decentralized multi-agent coordination framework using message passing and the Max-Sum algorithm for building and maintaining a common picture of the area to be monitored. We show that by representing each sensor as an independent agent which can take decisions individually and at the same time can affect the decisions of its neighbouring sensor-agents we can provide a robust and efficient system for the monitoring of life-critical environments such as assistive environments or governmental infrastructures.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; I.2.9 [**Robotics**]: Sensors; G.2.2 [**Graph Theory**]: Graph algorithms

General Terms

Algorithms, Reliability, Measurement, Experimentation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PETRA '10 Samos, Greece
Copyright 2010 ACM 978-1-4503-0071-1/10/06 ...$10.00.

Keywords

Mobile sensors, sensor placement and coordination, multi-agent systems, Max-Sum algorithm

1. INTRODUCTION

In recent years a lot of research effort has been put in creating environments where computers seamlessly interact with humans with the intention to enhance their productivity or make their lives easier. Such environments can be Assistive Living Homes, where automation aims at reducing the workload of conventional nursing services and facilitate independent living [12], critical infrastructures such as energy production power plants where continuous monitoring is necessary to prevent and deal with potential accidents [8] and even military applications. To achieve so the computers have to detect the human activity and the changes that are taking place in areas of interest. This is usually done by employing a set of sensing devices (or sensors) which can monitor the target humans or environments [2]. The issue that rises from such an employment is how to optimally place the sensing devices in a given space and how to coordinate their functionality and movement in order to maximize the information being captured and at the same time minimize the cost and/or the intrusiveness.

Different researchers follow different approaches for solving this problem which can be categorized according to if the environment to be monitored is static or dynamic, if the placement and configuration of the sensors is a one shot attempt or it adapts to the varying conditions, if the provided solution is optimal or approximate and if the organization architecture is based on a centralized or decentralized paradigm [1].

Since the problem of optimal sensor placement in the general case is of exponential complexity, Lin et. al. [11] propose a near optimal solution based on the simulated annealing approach under the constraints of the cost limitation and the complete coverage. Following a different approach, Guestrin et. al [6], propose an approximation method based on mutual information criteria for monitoring special phenomena which are modeled as Gaussian Processes (GPs). The limitation of the above two techniques is that both

the sensor placement and their functionality is static which means that once the deployment has be completed the state and position of the sensors cannot be altered to adapt to environmental changes.

To tackle the problem of dynamically changing environments, researchers have turned their attention to more adaptive methods, which can deal with emerging situations and tune the behavior or the positioning of the sensing devices accordingly. However, the increased complexity of the problem introduces an even greater complexity in the calculations required to solve it. For this reason there has been a significant amount of work in decentralized approaches where the decisions for the position and function of a sensor or a group of sensors is taken locally relying on the information collected from the near neighborhood. A popular decentralized approach is to represent each sensor as an autonomous agent which can interact only with nearby sensor-agents [7]. Such a scenario is realistic and desirable in most cases since most sensing devices have limited communication range and processing capabilities. The coordination of the sensors then is based on multi-agent coordination methodologies. Patricio et. al. [13] have proposed a multi-agent framework for surveillance purposes where each camera is represented by an agent and the surveillance process is distributed over several agent-cameras, according to their individual ability to contribute their local information to the global target solution. In their joint work, Farinelli at. al. [4, 5], and Stranders et. al. [16, 15, 17] have based the decentralized coordination of multiple agents on local message passing coordinated by the graph theoretic method of Max-Sum algorithm [18]. By maximizing the social welfare of the group of agents they can optimize the sense/sleep cycles of a set of wireless sensors and path planning for monitoring and predicting the state of spatial phenomena. In addition, Rogers et. al. [14] experiment with a hardware implementation of the same method on Texas Instrument CC2430 System-on-Chips to show the feasibility of the algorithm in real life applications.

This paper extends the above work by testing it's applicability to optimal sensor placement and functionality coordination of a set of mobile wireless sensors. We explain the theoretical framework behind multi-agent coordination based on message passing and the Max-Sum algorithm, and we introduce an extended version of the Max-Sum algortihm that can be used to achieve the required quality of coverage of an area. We call this extended version Extended Max-Sum Decentralised Coordination (EMSDC) algorithm. We assume that each region of the area to be covered is not always equally important but can be described by a Gaussian map. Furthermore, the importance of each region can vary over time which requires that the placement of the sensors has to be dynamically reformed to cover possibly new regions of interest. Finally we show that our method is robust to individual device failures.

The rest of this paper is organized as follows. In section 2 we introduce the theoretical framework which the multi-agent coordination is based on. This includes the factor graphs, the message passing and the Max-Sum decentralized coordination algorithm. Following, in section 3 we present our experimental evaluation of the proposed methodology and analyse the outcomes. Last, in section 4 we conclude by discussing the findings of this work and possible future improvements.

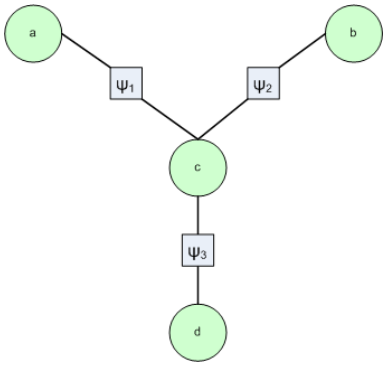

Figure 1: Example of a factor graph.

2. THEORY AND FORMALIZATION

In this section we present the theory behind our novel Extended Max-Sum Decentralised Coordination (EMSDC) algorithm which is used for the optimal sensor placement and functionality coordination by our multi-agent system. It is a message passing algorithm applied on a factor graph, so here we will see what is a factor graph, what is message passing and how the Sum-Product [10], Max-Product [19] and Max-Sum Decentralised Coordination (MSDC) algorithms [4, 5, 9, 16, 15, 18] are extended to EMSDC. At the end, to help the reader understand how the system works in a real application, we give a toy execution example.

2.1 Factor Graphs

Factor graphs are graphical models that are used to represent functions of the form:

$$f(x_1, x_2, \ldots, x_n) = \prod_i \phi_i(X^i) \tag{1}$$

where X^i are subsets of $x_1, x_2, \ldots, x_n$ and $\phi_i(X^i) = p(x_i|parents(x_i))$.

A factor graph has two types of nodes. Variable nodes, that represent variables of the environment and factor nodes, that represent the factors $\phi_i(X^i)$. Edges are only allowed between variable nodes and factor nodes. For example, the distribution $p(a, b, c, d) = \psi_1(a, c)\psi_2(b, c)\psi_3(c, d)$ can be represented by the factor graph in Figure 1.

Note that we can have directed edges on factor graphs. Factor graphs with directed edges have the advantage that we can easily infer the assumed dependencies between variables.

2.2 Message Passing

It is possible to do inference simply by making calculations. All we need to do is create a model - a probability distribution function that describes the environment - and then use the basic rules of probability to mechanically compute the quantity of interest, i.e. make an inference about the environment. Let us see an example. Suppose we have the distribution $p(a, b, c, d) = p(a|b)p(b|c)p(c|d)p(d)$, where the variables can be either in state 0 or in state 1, and that

we want to compute $p(a=0)$:

$$p(a=0) = \sum_{b,c,d} p(a=0,b,c,d) = \sum_{b,c,d} p(a=0|b)p(b|c)p(c|d)b(d) \tag{2}$$

Since we have three "free" variables and each variable can be in one of two available states, in order to calculate $p(a=0)$ we need to sum 8 products. For distributions involving more variables and more states that the variables can be in, this computation can be difficult or even impossible to do. An easy and better way to compute $p(a=0)$ is to represent the distribution as a factor graph and apply a message passing algorithm on that factor graph. But let us first see how message passing works. The main idea is to push as far to the right as possible the summation over variable d:

$$p(a=0) = \sum_{b,c} p(a=0|b)p(b|c) \underbrace{\sum_{d} p(c|d)p(d)}_{\gamma_d(c)} \tag{3}$$

We name the summation over d $\gamma_d(c)$, which is a two state potential. We can think of $\gamma_d(c)$ as a message from variable d to variable c. Similarly now, we push as far to the right as possible the summation over c:

$$p(a=0) = \sum_{b} p(a=0|b) \underbrace{\sum_{c} p(b|c)\gamma_d(c)}_{\gamma_c(b)} \tag{4}$$

and we name the summation over c $\gamma_c(b)$, which is again a two state potential. The summation over b cannot be pushed. Using this technique we only need to sum 6 products, since we have three summations over two states each time.

Having understood the concept behind message passing, we can now see some specific algorithms.

Sum Product Algorithm: The $\gamma_c(b)$ message can be considered as a message from variable c to variable b, i.e. a variable to variable message. In order to do efficient inference though, we need to represent the distribution as a factor graph, and exchange messages between its nodes. Since a factor graph has edges only between variable nodes and factor nodes, we need to define variable to factor and factor to variable messages. Sum - Product algorithm defines these messages:

Initialisation: Messages from external factor nodes are initialized to the factor. Messages from external variable nodes are initialized to unity.

Variable to Factor Message: To compute a message from a variable x to a factor f, we have to multiply all incoming messages to variable x, from all its neighbours except f:

$$\mu_{x\to f}(x) = \prod_{g\in\{ne(x)\setminus f\}} \mu_{g\to x}(x) \tag{5}$$

Factor to Variable Message: To compute a message from a factor f to a variable x, we have to multiply the sum of the factor's value for all its neighbouring variables except from variable x and the product of all incoming messages from all neighbouring variables except x:

$$\mu_{f\to x}(x) = \sum_{y\in X^f\setminus x} f(X^f) \prod_{y\in\{ne(f)\setminus x\}} \mu_{y\to f}(y) \tag{6}$$

Marginal: To compute $p(x)$, we have to multiply all incoming messages to variable x:

$$p(x) = \prod_{f\in ne(x)} \mu_{f\to x}(x) \tag{7}$$

A very interesting problem is to find the most probable "state" of the distribution, that is find the combination of states for the variables that maximize the distribution:

$$\arg\max_{x_1,\ldots,x_n} p(x_1,\ldots,x_n) \tag{8}$$

We can use a message passing algorithm on this problem as well, and this idea leads us to the Max-Product algorithm [5].

2.2.1 Max Product Algorithm

Initialisation and Variable to Factor Message: Messages are exactly the same as in Sum - Product algorithm.

Factor to Variable Message: To compute a message from a factor to a variable we only need to replace the summation with maximization:

$$\mu_{f\to x}(x) = \max_{y\in X^f\setminus x} f(X^f) \prod_{y\in\{ne(f)\setminus x\}} \mu_{y\to f}(y) \tag{9}$$

Marginal: To compute the optimal state for variable x, we have to find which state maximizes the product of all incoming messages to variable x:

$$x^* = \arg\max_{x} \prod_{f\in ne(x)} \mu_{f\to x}(x) \tag{10}$$

These two algorithms have a serious disadvantage. They involve the calculation of many (and sometimes large) products of small numbers. This leads to system errors when the calculation is done by computer. To avoid this we can calculate the logarithm of the messages instead:

$$\mu'_{x\to f}(x) = log\mu_{x\to f}(x) \tag{11}$$

This is the idea behind the Max-Sum Decentralised Coordination algorithm.

2.3 Max-Sum Decentralised Coordination Algorithm

In this algorithm, factors represent utility functions, and are connected to variables that are necessary for their calculation.

Initialisation: We randomly initialize each variable node, and randomly initialize all outgoing messages. The algorithm is guaranteed to converge to a solution [5].

Variable to Factor Message: To compute a message from a variable n to a factor m, we have to calculate the sum of incoming messages from all neighbouring factors except factor m, plus a normalisation constant α:

$$Q_{n\to m}(x_n) = \alpha_{nm} + \sum_{m'\in ne(n)\setminus m} R_{m'\to n}(x_n) \tag{12}$$

α_{nm} is a scalar such that:

$$\sum_{x_n} Q_{n\to m}(x_n) = 0 \tag{13}$$

Factor to Variable Message: To compute a message from a factor m to a variable n, we essentially do local blind search combined with information coming from the graph.

We have to find the combination of states for m's neighbouring variables, except for n, that maximize the utility (i.e. the factor's value) plus the sum of all incoming messages except the one that came from n:

$$R_{m\to n}(x_n) = \max_{X_{m\setminus n}} (U(X_m) + \sum_{n'\in ne(m)\setminus n} Q_{n'\to m}(x_{n'})) \quad (14)$$

where X_m is a set of variables that are the neighbours of m.

Marginal: To compute the probability that a variable x is in a specific state, we have to sum all incoming messages to x:

$$Z_n(x_n) = \sum_{m\in ne(n)} R_{m\to n}(x_n) \quad (15)$$

In acyclic factor graphs these Z_n represent a solution to the utility maximization problem (or social welfare maximization problem). If however the factor graph contains cycles, Z_n represent an approximate solution:

$$Z_n \approx \max_{X_{m\setminus n}} \sum_{m=1}^{M} U_m(X_m) \quad (16)$$

To compute the optimal state for a variable x, we simply have to calculate:

$$\arg\max_{x_n} Z_n(x_n) \quad (17)$$

A pseudo-code description of the Max-Sum algorithm can be found in [18].

To better understand how this algorithm works, let us see a non-mathematical explanation of the messages exchanged:

- When a factor M sends an R message to a variable N, it is as if the factor says: "N, I prefer that you are in state S".

- When a variable N sends a Q message to a factor M, it is as if the variable says: "M, my neighbours prefer my state to be T".

- When a variable n calculates Z, it is as if the variable says: "I am choosing a state based on which state my neighbours mostly prefer for me".

2.4 The Extended Max-Sum Decentralised Coordination Algorithm (EMSDC)

To deal with the problem of optimal placement, we created an extended version of the Max-Sum Decentralised Coordination (MSDC) algorithm. This version, unlike MSDC, takes into account not only the state of the agents but also the location of the agents (represented by utility - factor pairs). The main idea is that each agent has two types of states, task and location. This means that instead of having one factor graph, we have two, where at the second one variables represent agents' locations and utilities measure how good these locations are for each agent (typically a measure of the overlap with its neighbours multiplied by a gaussian function). We then run the MSDC two times, once for each factor graph, i.e. once for task selection and once for placement. As is the case with task selection, the agents exchange preferences on each other's location instead of their own actual location. Each agent then tries to push its neighbours away, to the direction that maximises each neighbour's utility. MSDC's performance has already been proven in [5]. Our algorithm's running time is twice the running time of MSDC, but asymptotically the complexities are the same.

The agents exchange messages in order to maximise that utility. The extra messages exchanged are:

From Factor to Variable:

$$T_{m\to n}(x_n) = \max_{d\in D}(U_m(p_n, d)) \quad (18)$$

where D are all possible directions (e.g. Up, Down, Left, Right) and p_n is the location of the n_{th} agent.

Marginal: The marginal is similar to $Z_n(x_n)$:

$$W_n(p_n) = \sum_{m\in M(n)} T_{m\to n}(p_n) \quad (19)$$

To compute the optimal position for a variable p, we calculate:

$$\arg\max_{p_n} W_n(p_n) \quad (20)$$

In practice, however, agents exchange Q messages as defined in [5] and a unified $RT_{m\to n}(x_n, p_n)$ message:

$$RT_{m\to n}(x_n, p_n) = [R_{m\to n}(x_n), T_{m\to n}(p_n)] \quad (21)$$

The marginal becomes:

$$ZW_n(x_n, p_n) = [Z_n(z_n), W_n(p_n)] \quad (22)$$

Each agent then computes its optimal state and position by maximising:

$$\arg\max_{x_n, p_n} ZW_n(x_n, p_n) \quad (23)$$

which means that finally each agent selects the pair (state, location) with the greatest gain. The distance that the agents will move towards their selected direction is a tuneable parameter of the algorithm.

Here is a toy example of EMSDC, using 3 single tasking agents that begin close to each other and can choose from 3 different states. We use 4 directions and the graph colouring problem for task selection, thus the RT message will be [Red, Green, Blue, Up, Down, Left, Right]. All agents begin from state Red. Note that we do not show all decimal digits due to space constraints:

$Q_{1\to 2}(x_2)$ = [0.0 0.0 0.0]
$Q_{1\to 2}(x_3)$ = [0.0 0.0 0.0]
$Q_{1\to 2}(x_1)$ = [0.0 0.0 0.0]
$RT_{1\to 2}(x_2)$ = [$\underbrace{-0.05\ 0.1\ 0.1}_{Tasks\,(R_{1\to 2}(x_2))}$ $\underbrace{272\ 192\ 231\ 237}_{Directions\,(T_{1\to 2}(x_2))}$]
$RT_{1\to 3}(x_3)$= [-0.05 0.1 0.1 393 326 328 391]
$RT_{1\to 1}(x_1)$ = [0.1 -0.05 -0.05 0.09 0.092 0.094 0.097]
$ZW_1(x_1)$ = [0.0 0.0 0.0 0.0 0.0 0.09 **0.097**]
$Agent_1$ remains **Red** and is moving **Right**.

$Q_{2\to 1}(x_1)$ = [0.0 0.0 0.0]
$Q_{2\to 3}(x_3)$ =[-0.1 0.05 0.05]
$Q_{2\to 2}(x_2)$ =[-0.1 0.05 0.05]
$RT_{2\to 1}(x_1)$ = [0.1 -0.05 0.15 174 219 192 207]
$RT_{2\to 3}(x_3)$ = [0.15 0.0 0.15 261 251 212 295]
$RT_{2\to 2}(x_2)$ = [-0.05 0.1 -0.05 0.118 0.113 0.114 0.117]
$ZW_2(x_2)$ = [-0.1 **0.2** 0.05 **272** 192 231 237]
$Agent_2$ is changing to **Green** and moving **Up**.

$Q_{3\to 1}(x_1)$ = [0.050 -0.1 0.050]
$Q_{3\to 2}(x_2)$ = [-0.1 0.05 0.05]
$Q_{3\to 3}(x_3)$ = [-0.049 -0.050 0.099]
$RT_{3\to 1}(x_1)$ = [0.0 0.15 0.15 242 270 271 242]

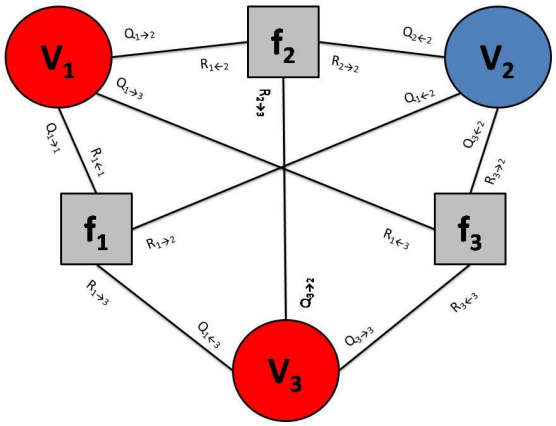

Figure 2: Example of messages being exchanged in the factor graph.

$RT_{3\to 2}(x_2)$ = [0.049 0.050 0.050 251 231 270 206]
$RT_{3\to 3}(x_3)$ = [0.15 0.0 0.0 0.129 0.124 0.125 0.128]
$ZW_3(x_3)$ = [0.1 0.1 **0.25** 654 577 541 **687**]
$Agent_3$ is changing to **Blue** and moving **Right**.

In this toy example the system converges after just one cycle, since all agents now have different colours (Red, Green and Blue). But we can see that during this process the agents exchange preferences about each other's position as well. In Figure 2 you can see the factor graph representation and the messages being exchanged between the different nodes of the graph.

3. EXPERIMENTAL SET-UP AND AND SYSTEM EVALUATION

For our experiments we assume that we have a large area to cover and that each sensor's range is circular. Some sections of this area are more interesting than others (high traffic areas, accident prone areas, etc.) and we want them covered even if some sensors overlap. We represent this using a 3-dimensional Gaussian map.

We currently have not implemented other constraints, such as walls, but EMSDC can easily be applied to a constrained environment.

For the task selection part we model each task with a colour and try to solve the graph colouring problem. We use the graph colouring utility from [5]:

$$U_m(X_m) = \gamma_m(x_m) - \sum_{i\in ne(m)} \sum_{j\in C(i,m)} x_i \oplus x_j \qquad (24)$$

where $x_i \oplus x_j$ is 1 if $x_i = x_j$ and 0 otherwise, and C is defined as:

$$C(i,m) = \{k \in ne(m) | k > i \wedge (i \in ne(k) \vee k \in ne(i))\} \qquad (25)$$

For the placement part we used the following utility:

$$U'_m(X_m) = \sum_{i\in ne(m)} \sqrt{(p_m - p_i)^2} \qquad (26)$$

where p_i is the position of the i_{th} agent.

When we have a Gaussian Map we simply multiply U'_m with the Gaussian:

$$U'_m(X_m) = \sum_{i\in ne(m)} (\sqrt{(p_m - p_i)^2} G(p_i)) \qquad (27)$$

In order to calculate the total coverage, we need to first calculate the total overlap between the agents. The calculation of the overlap however can be very hard in the general case. We tackle this problem using an approximate Monte Carlo technique. We randomly choose 50,000 points, which is like throwing 50,000 darts at the area to cover, and we count the number of hits for each dart. If there are no hits that point is not covered, whereas, if there are more than one, there is overlap at that point. We can then measure the hits per dart and name that ratio *reward*. We increment the *reward* by one for each dart with exactly one hit, and by $\frac{1}{hits}$ if there are more that one hits. The optimal would be 1 hit per dart, thus $optimal = 50,000$.

Another way would be to increment *reward* by one for each dart with more than one hit, and then use the following formula to calculate the area covered by the agents (which is $\sum_{i=1}^{N} \pi\rho_i^2 - overlap$):

$$area = \frac{\frac{reward}{optimal} areaToCover}{min\{areaToCover, \sum_{i=1}^{N} \pi\rho_i^2\}} \qquad (28)$$

We consider that a cycle has passed when all agents have had the opportunity to change their state. We run each experiment for 250 cycles. Each agent's range was set to 30 units. The dimensions of the environment are 460 x 980 = 450800 units, so each agent alone can cover 0.6% of the area. Each experiment was run 10 times for each dimension, and we calculated the average.

To perform the experimental evaluation we created a simulation environment based on JADE [3]. We assume that the information we gain through sensor measurements can be incorporated to each sensor's utility function. Each sensor is represented by a JADE agent, listens to the network for any new messages and responds accordingly. At any time we can ask a sensor agent for its state, and that agent will then calculate and return $\arg\max_{x_n,p_n} ZW_n(x_n, p_n)$. The sensor agents keep exchanging messages until the system converges.

It is proven in [5] that MSDC will converge even if the initial states of the variables in the factor graph are random. This means that the order by which messages are exchanged does not matter. So, in our simulation we introduce an agent that simulates the network infrastructure and forces the sensor agents to broadcast Q and RT messages to their neighbours in an orderly fashion. When all agents have finished broadcasting Q and RT messages, a cycle has passed and the network agent then forces the sensor agents to compute their state (task and location). We used this approach because it is easier to monitor when a cycle has passed and take more accurate measurements.

3.1 Optimal Placement (of Single-Tasking Agents) with Gaussian Map

In most real world applications, when monitoring an area, there are important and not that important sections of that area. For example in an assistive living apartment, we probably do not want to monitor the inside of a closet or a storage area rarely used, and instead we want to focus on high traffic areas, such as the bathroom or the kitchen. In our model we represent this using a 3-dimensional Gaussian map.

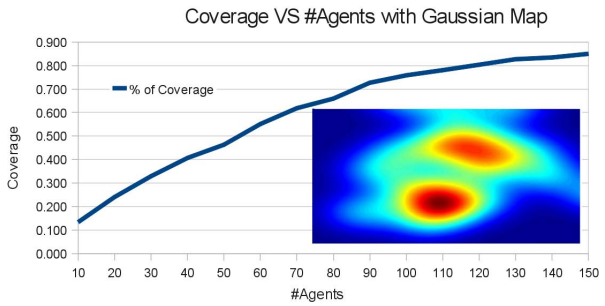

Figure 3: Percentage of coverage versus number of agents when using a Gaussian map.

For our experiments we used the Gaussian map (as viewed from above) depicted in Figure 3, and run the algorithm for 10 to 150 agents. Figure 3 shows the percentage of the covered area versus the number of agents. We can see that the total area covered rises rapidly in the lower dimensions and slower in the higher dimensions. This is because the agents are trying to cover the high interest areas (red) first, leaving others (blue) less covered. This might be a desirable feature, since we may have some overlap in the red areas, but this also means redundancy and increased fault tolerance. It is possible to tune the algorithm and put more weight on the overlap between the agents and less on the effect of the Gaussian map. This way we will have less overlap in the red areas and the agents will spread more.

3.2 System response to environmental changes

An interesting problem is how the system will respond to a change in the environment. To model this, we use two different Gaussian maps, depicted in figure 4, where the second map (bottom image) has one more "important" region. This could be an event like a fire in the kitchen or a person falling in the bathroom. The first image shows the initial random placement of the sensors, before the EMSDC algorithm has been run. The percentage of coverage in this case is 39.1% The second (middle) image shows the coverage of the area after the execution of the EMSDC algorithm but before the occurrence of the critical event. The coverage in that case is 68.5%. We then changed the map, at which point the coverage suddenly became 60.7% since an important area was not covered. After the algorithm ran for 100 cycles the new resulting coverage increased to 64.2%. Figure 4-bottom, shows the final position of the agents. We can clearly see that the agents adapt very well to the change in their environment. Note that it is not possible to achieve the initial percentage of coverage with the same number of sensors, since after the map change there is a bigger amount of "important" regions to be covered.

3.3 Fault Tolerance

The two main benefits of using multi agent systems are decentralised control, meaning that each agent performs small tasks that can be performed by low cost devices, and fault tolerance. Here we prove that EMSDC performs very well in the presence of failures.

To test the fault tolerance of the system, we compared it to a static system, i.e. a system where the sensors cannot move to compensate for failures. We run EMSDC using 100

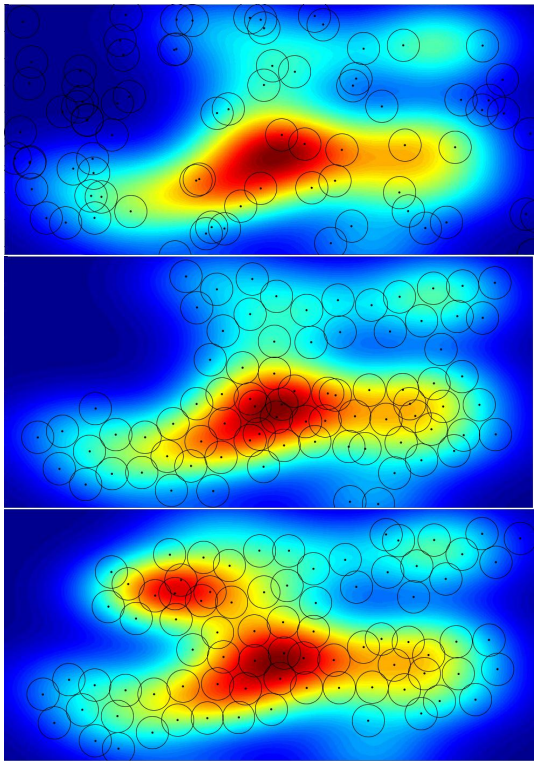

Figure 4: The top image shows a random starting state of the system when using a Gaussian map. The middle image shows the state of the system after the execution of the EMSDC algorithm. The bottom image shows the final state of the system after a change in the Gaussian map has taken place and the system has converged to a new solution.

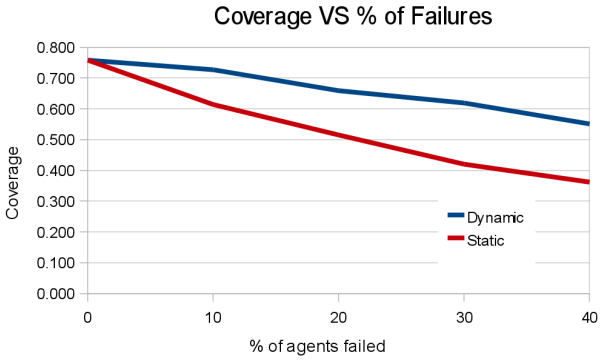

Figure 5: Percentage of coverage versus percentage of agents failed, for a static and dynamic system. We can see that the dynamic system adapts well to failures.

agents, calculated the coverage after 0% to 40% agents have failed randomly and compared the results with the coverage of EMSDC in the presence of failures. To simulate failures, we use a model where each sensor has a probability p to fail at each cycle. After that point the sensor becomes useless either because it cannot take measurements or because it cannot transmit/receive data. We can see the results in figure 5. The very small decrease in coverage for EMSDC means that as the agents in important sections fail, others come and take their place. Contrary if the placement of the sensors could not be re-organized after some sensors have failed, important regions could remain uncovered and that would result in a rapid decrease in percentage of coverage. As we can see from the graph, with 10% of sensors failed we already have a 13% difference in the coverage between the static approach and our dynamic system.

Although not shown here, the system is also very tolerant to lossy communication, since messages sent at time t are not very different from messages sent at time $t-1$. Each agent stores in memory the received Q and R messages, so at each cycle, if an incoming message is lost the agent will use the previously stored one. Also, messages are exchanged constantly and rapidly, so the agent will most likely receive within reasonable time an updated message coming through a lossy link. Lossy communication can slow down the convergence of the system into an optimal state but does not significantly affect the final outcome.

3.3.1 Optimal Placement of Multi-Tasking Agents

A multi tasking agent is an agent that can perform more than one tasks simultaneously, e.g. capturing of video, sound and temperature. It is not necessary that all agents are capable of performing the exact same tasks.

It is very easy to generalise this model for multi tasking agents. Instead of using a set of tasks for each agent that represents the tasks each agent is currently performing, we can do the following:

If we have K different tasks that the agents can perform (in the general case each agent is capable of simultaneously performing a subset of the K tasks), we can use the MSDC algorithm where each agent will have $2^{|K|}$ possible states. We can then convert the current state (a number from 0 to $2^{|K|}$) to a binary number (after adding any necessary 0's), where each digit represents a task, and if the i_{th} digit is "1" the agent is performing the i_{th} task whereas if it is "0" the agent is not performing that task.

In the current version, our system can coordinate the agents taking into consideration the tasks that each sensor-agent can perform. Agents that can perform the same set of tasks should not overlap or if they overlap they should not be in the same state. However, in order for this feature of the system to have an acceptable performance, the utility of each agent should keep a balance among the degree of overlap, the importance of the covered space and the different conflicting tasks. In the ideal case each task should have a different importance map, since monitoring e.g. the temperature in the kitchen may be more important than capturing video. Therefore sensors with capabilities of temperature measurement should be favoured more for being placed there.

Such feature in not implemented in our system and thus the agents can take arbitrary placements and states considering only the general importance of the space to be covered and the overlaps. We plan to include this feature in future versions of our system.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have addressed the problem of placement and coordination of wireless mobile sensors. The proposed algorithm, the Extended Max-Sum Decentralised Coordination algorithm which is an extension of Max-Sum Decentralised Coordination algorithm [5] is already proven to perform very well on the coordination part. Here we have proved its value on the sensor placement problem and the resulting algorithm manages to place and coordinate sensors near optimally. The algorithm is robust, fault tolerant, it can adapt to major changes in the environment and is suitable for low power sensors [5, 18] with low processing capabilities.

In the future we will work more on multi tasking mobile sensors, and introduce two classes of mobile sensors: active and passive. An active sensor is a sensor over which we have control (i.e. we can place) and a passive is one over which we have no control, for example an embedded/wearable sensor on a person's jacket that can move around arbitrarily. We will also introduce several types of constraints in the environment, such as walls and forbidden sensors in some areas, as well as sensor battery life. Battery life can have a huge impact on the decisions made by the algorithm and is a crucial part of a real world application. Last, we plan to apply the algorithm in a 3 dimensional environment where each sensor can cover a volume instead of a surface. This introduces additional placement constraints and makes the calculation of overlaps more challenging.

5. ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work is supported in part by the National Science Foundation under award numbers CT-ISG 0716261 and MRI 0923494. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] B. R. Abidi, N. R. Aragam, Y. Yao, and M. A. Abidi. Survey and analysis of multimodal sensor planning and integration for wide area surveillance. *ACM Computing Surveys (CSUR)*, 41(1):7, 2008.

[2] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. A survey on wireless multimedia sensor networks. *Computer Networks*, 51(4):921–960, 2007.

[3] F. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. Springer, 2008.

[4] A. Farinelli, A. Rogers, and N. Jennings. Maximising sensor network efficiency through Agent-Based coordination of Sense/Sleep schedules. In *Workshop on Energy in Wireless Sensor Networks in conjuction with DCOSS*, 2008.

[5] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised coordination of low-power embedded

devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 639–646. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[6] C. Guestrin, A. Krause, and A. P. Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 265–272. ACM New York, NY, USA, 2005.

[7] B. Horling and V. Lesser. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, 19(04):281–316, 2005.

[8] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, page 263. ACM, 2007.

[9] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th international conference on Information processing in sensor networks*, page 10. ACM, 2006.

[10] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[11] F. Y. S. Lin and P. L. Chiu. A near-optimal sensor placement algorithm to achieve complete coverage/discrimination in sensor networks. *IEEE Communications Letters*, 9(1):43–45, 2005.

[12] V. Metsis, Z. Le, Y. Lei, and F. Makedon. Towards an evaluation framework for assistive environments. In *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments*, page 12. ACM, 2008.

[13] M. A. Patricio, J. Carbo, O. Perez, J. Garcia, and J. M. Molina. Multi-agent framework in visual sensor networks. *EURASIP Journal on Advances in Signal Processing*, 2007, 2007.

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

[15] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of continuously valued control parameters using the max-sum algorithm. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 601–608. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[16] R. Stranders, A. Farinelli, A. Rogers, and N. R. Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proc. 21st Int. Joint Conf. on AI (IJCAI), Pasadena, USA*, 2009.

[17] R. Stranders, A. Rogers, and N. Jennings. A Decentralised On-Line Coordination Mechanism for Monitoring Spatial Phenomena with Mobile Sensors. In *Proceedings of the Second International Workshop on Agent Technology for Sensor Networks (ATSN)*, page 9.

[18] A. Waldock, D. Nicholson, and A. Rogers. Cooperative control using the max-sum algorithm. 2008.

[19] 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):736–744, 2001.