ARTICLE IN PRESS

Pervasive and Mobile Computing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy

Ronen Vaisenberg*, Alessio Della Motta, Sharad Mehrotra, Deva Ramanan

The University of California, Irvine, United States

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Video surveillance
Algorithms
Sensor network
Real-time
POMDP

ABSTRACT

We present a framework for sensor actuation and control in sentient spaces, in which sensors are used to observe a physical phenomena. We focus on sentient spaces that enable pervasive computing applications, such as smart video surveillance and situational awareness in instrumented office environments. Our framework utilizes the spatio-temporal statistical properties of an observed phenomena, with the goal of maximizing an application-specified reward. Specifically, we define an observation of a phenomena by assigning it a discrete value (state) and we model its semantics as the transition between these values (states). This semantic model is used to predict the future states in which the phenomena is likely to be at, based on partially-observed past states. To accomplish real-time agility, we designed an approximate, adaptive-grid solution for Partially Observable Markov Decision Processes (POMDPs) that yields practically good results, and in some cases, guarantees on the quality of the approximation. We use our framework to control and actuate a large-scale camera network so as to maximize the number and type of captured events. To enable real-time control, we implement an action schedule using a table lookup and make use of a factored probability model to capture state semantics. To the best of our knowledge, we are the first to address the problem of actuating a large-scale sensor network based on a real-time POMDP formulation.

1. Introduction

Recent advances in sensing technologies have enabled the infusion of information technology into physical processes. This offers unprecedented opportunities, the impact of which will rival (if not exceed) that of the WWW. Such sensor-enabled environments can realize sentient spaces that have the potential to revolutionize almost every aspect of our society. Sentient systems observe the state of the physical world, analyze and act based on it. Sentient systems enable a rich set of pervasive computing applications including smart video surveillance, situational awareness for emergency response and social interactions in instrumented office environments to name a few. Sentient systems present new forms of computational challenges, from modeling, control, and scheduling perspectives. One has to balance the needs of (possibly multiple) pervasive computing applications with physical constraints of sensors/devices being controlled, all while achieving situational awareness of the physical world being monitored. In this paper, we use additional information, namely the semantics and predictability of the monitored world, to improve the agility of sentient systems. We design real-time algorithms that use probabilistic semantic models to schedule sensor actuation in an "optimal" way. We evaluate our

* Corresponding author. Tel.: +1 9493852016.
E-mail addresses: ronenwai@gmail.com, ronen@ics.uci.edu (R. Vaisenberg), alessio.dellamotta@gmail.com (A.D. Motta), sharad@ics.uci.edu (S. Mehrotra), dramanan@ics.uci.edu (D. Ramanan).

1574-1192/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.pmcj.2013.10.014

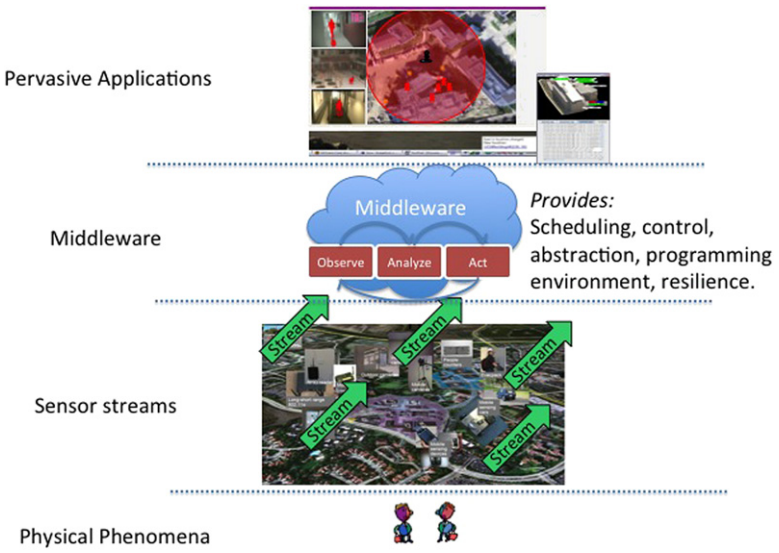


Fig. 1. Multiple layers view of sentient systems.

approach in an instrumented smart building, specifically using it to dynamically control a camera network so as to record the events of interest.

At UCI we are in the process of building a middleware for sentient spaces which we refer to as Satware.[1] Satware is a multimodal sensor data stream querying, analysis, and transformation system. Satware provides applications with a semantically richer level of abstraction of the physical world as compared to raw sensor streams, providing a flexible and powerful application development environment. Decoupling (and abstracting) events from the underlying sensors offers numerous advantages. The middleware is tasked to bridge the semantic gap between the raw sensors observing the environment and the high level pervasive applications, see Fig. 1.

We propose a unique approach to the task of sensor adaptation: We model and learn *phenomena semantics*—the underlaying pattern by which the phenomena evolves over time and space. Phenomena semantics, or *semantics* in the context of this paper, represent the behavioral nature of the phenomena. The word "semantics" is often used to denote the problem of understanding the meaning behind signs, symbols, sounds or expressions. We use semantics to refer to the understanding of the relationship between the current state of the phenomena and the possible future states it can evolve to. In particular, we model the phenomena semantics as state transition probabilities, which in turn allow the sentient system an "understanding" of the way the phenomena evolves over time and space. Such state transition probabilities, in turn, allow the system to compute the expected long term utility of each action and decide on the best course of sensor actuations.

To illustrate how challenging this problem is, consider the following questions:

- How should the phenomena semantics be represented?
- How should we take into account the application requirements, system constraints, and phenomena semantics to address the task of actuation?
- Due to resource constraints, our representation of the current state of the phenomena would, invariably be incomplete. How should we balance the need to reduce the uncertainty with respect to the current state of the phenomena and the need to collect data of interest to the application?

In this paper, we focus on the *Actuation Challenge*: System actuation involves the design of adaptation algorithms that optimize the usage of the infrastructure (via resetting device parameters and system policies) when changes occur. For example, changing the image resolution, or actuating the optical zoom of a camera are examples of such adaptations that are aimed at generating a more detailed observation.

When actuating sensors, the scheduler is constantly facing a tradeoff between actions that have the potential of generating a high resolution image and actions that reduce the uncertainty of the activities that are taking place right now. A greedy approach where the scheduler collects only observations of interest to the pervasive application is, sometimes, not optimal. This is because some actions may be required to reduce uncertainty about the world without immediate benefit to the pervasive application. The more certain the scheduler is about the state of the world, the more likely it is to satisfy the application needs, in the long term.

In such cases, schedulers should balance the demands of an application with the need to better observe the world. We advocate the use of a Partially Observable Markov Decision Process (POMDP) to reason about sensor actuation

[1] Satware is described in more details in [1].

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

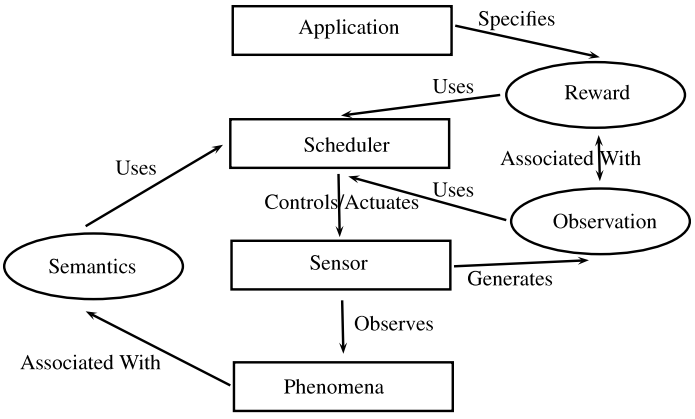


Fig. 2. Sentient spaces high-level concepts.

with physically-realized, non-ideal sensors. We explore our framework in the context of large-scale camera networks. Specifically, we use POMDPs to control pan–zoom–tilt (PZT) parameters of cameras for a variety of pervasive computing applications such as building occupancy estimation, identity recognition, people tracking, etc. Notably, our framework schedules actuations so as to simultaneously maximize the (possibly-conflicting) benefits of differing end applications.

The key technical contributions of this work are as follows: first we propose a general POMDP framework for sensor actuation problems that balances application need and situational awareness. Second, we introduce approximations that are required to scale POMDP solvers for *real-time* actuation of large-scale sensor networks. We further extend the approximation framework to balance the need of multiple monitoring applications. Third, we demonstrate the effectiveness of our system, by comparing it to commonly-used baseline scheduling algorithms, on a recording from a large scale camera network in a campus building. Our results suggest that sensor level challenges can be abstracted and effectively addressed by a middleware layer.

To the best of our knowledge, we are the first to address the problem of actuating a large-scale sensor network based on an approximated, real-time POMDP. We accomplish this by using a semantic model of building activities and approximating the expected utility of scheduler actions. We describe a semantic correlation model that is simple enough to be implemented in real-time, yet powerful enough to capture meaningful semantics of typical behavior. Our action selection process is as fast as a table lookup in real-time. Though we focus on large-scale camera networks, our approach applies to the general problem of sensor selection and actuation in sentient spaces. We briefly discuss such extensions in Section 6. We believe that our approach will serve as a fundamental building block for building next-generation sentient systems.

2. Problem formulation

2.1. Abstract problem formulation

Fig. 2 outlines the high-level concepts of our sensor actuation framework. A pervasive application specifies a reward that is associated with an observation of a phenomena. The scheduler controls and actuates the sensor so as to better observe the phenomena and maximize the application-specific reward. To do so, the scheduler uses semantics that are associated with the phenomena.

Environment model: The environment model is an abstract representation of the monitored environment. For instance, in a monitored building the environment model is used to refer to different regions such as "$r_1 = south$ west hallway, second floor" and "$r_2 = second$ floor kitchen".

Phenomena: The phenomena is any kind of feature property whose value is amenable to observation or estimation, including physical properties, existence and occurrence assessments, etc. The phenomena happens in the instrumented space and changes (states) over time. The phenomena has a measurable value instance (state) at a particular time, written as X_t. For instance, consider $\boldsymbol{N}$ spatial regions of an instrumented building, each of which can be empty or occupied by a person. In this case, there exist 2^N possible phenomena states.

Semantics: We make use of semantic rules that define how the phenomena evolves over time and space. Specifically, in the context of this paper, we model these semantics as $P(\{X_{t+1}, \ldots, X_{t+T}\}|\{X_t, \ldots, X_{t-M}\})$, which is a function that returns predictions of the next $\boldsymbol{T}$ timesteps into the future given the states from the past $\boldsymbol{M}$ timesteps.

Sensor: The sensor is the physical device that is used to observe a phenomena. The system has access to sensors (i.e., cameras) $s_1, \ldots, s_n$ that are controlled by a scheduler.

Scheduler: The scheduler controls the capture and processing parameters at time t by choosing an action $A_t \in A$.

Observation: At any time t, given the phenomena state and scheduler action, the system generates an observation $O(X_t, A_t)$.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

Applications: We assume that multiple pervasive computing applications wish to monitor phenomena of different types. The sentient system is responsible for generating a schedule such that sensor data captured by the system satisfies the application needs.

Rewards: We define $R(O(X_t, A_t)) \rightarrow \mathbb{R}$ as an application-specific reward that depends on the observation, which itself depends on the phenomena state and the selected action. For instance, in our building monitoring setting, an image containing entities will have a higher reward as compared to an image with none. We account for the general setting there may exist multiple applications with different reward functions.

2.2. Modeling the problem as a POMDP

We model the above problem using a POMDP. Other alternatives for a POMDP model include Gaussian Markov random fields [2] or spatial point-process models [3]. One crucial aspect of our approach is that we capture spatial correlations across cameras, which may be difficult for some other models (e.g., spatial Poisson processes typically assume independent events [3]). But more importantly, our POMDP framework provides a formalism for making *decisions* with probability models. As we show, reasoning about decisions is crucial to balance the needs of multiple applications and overall situational awareness. One could substitute other probability models that capture semantic correlation (including random fields or point processes). Indeed, we see this modularity as the key strength of a POMDP formalism.

2.3. A brief overview of MDP and POMDP

We begin our discussion with a brief overview of Markov Decision Processes (MDPs) [4]. A MDP is a four tuple (S, A, P, R) where S is a *finite* set of states, A is a finite set of actions, $P_a(s, s_{next})$ is the probability that action a at state s would lead to s_{next}. $R(s, a)$ is the reward associated with being in state s while taking action a.

State transitions occur (stochastically) between states based on the actions taken. The goal is to find a policy π which specifies an action for each state $\pi(s)$ which has the maximal total expected reward over some horizon: $\sum_t R(s_t, a_t)$. For instance, in our setting, S would be a binary vector indicating if there is a frontal face of a person in a given camera region. A would be a discrete PZT state (region to zoom into or zoom out) of each camera. P is the state transition probability, which represents the "motion semantics" of how people move in the monitored space. For example, people tend to appear in spatially adjacent regions of cameras as they move. $R(s, a)$ would be the reward associated with being at state s while taking action a. For example, a natural reward would be the defined as the number of cameras which are zoomed into a region which contains a person. MDPs can be solved and the optimal policy can be found using dynamic programming [4]. However, in our case, there are several fundamental challenges that prevent us from modeling the space as an MDP: First, the probability transition matrix is too large to even store, as the number of possible world states is $O(2^N)$; each region can contain a face or not and we have N regions.

A more fundamental challenge is that the true state S of the world is not fully observed. For example, a zoomed in camera has uncertainty about the rest of the regions outside of the field of view. This means we *cannot* use the previously presented MDP formulation. Rather, we use the framework of Partially Observable MDPs (POMDPs). Crucially, this requires a scheduler to maintain an internal representation that captures the uncertainty about the outside world. Interestingly, one can formulate a POMDP as a MDP whose states S are continuous "belief states" about the world. Belief states assign every possible world state a probability value. This allows one to represent uncertainty about the state of regions outside the field-of-view.[2] In our POMDP, the number of distinct worlds is 2^N, and so each belief state $s \in S$ is a probability vector of length 2^N whose entries are nonnegative and sum to 1. Hence even representing a single belief state is not tractable. For more details on POMDPs, the interested reader is referred to the surveys in [5,6]. In the rest of this section, we formally specify a POMDP for scheduling sensor actuations in a sensor network.

2.4. Applying the model on a camera network

We focus on an active camera networks, where the set of actions A can be modeled with discrete PZT states of each camera. Each camera can be zoomed out, or zoomed into a particular region within the field of view. We denote the collection of N regions observed throughout the entire network as a vector X_t, where $X_t^r \in \{0, 1\}$ indicates if there exists a person in region r at time t. We assume that the N regions are fully observable when all cameras are actuated to their zoomed out configuration. We use this assumption to allow a learning phase in which the transition semantics of the environment are observed and learned. These environment semantics are later being used to actuate the cameras in real-time. Thus, all zoomable regions are part of the N regions for which we can effectively train a model by a period of observation when zoomed out.

The environment transitions (stochastically) between states as characterized by the state transition function $P(X_{t+1}|X_t)$ (transitions are attributed to activities that take place in the space, e.g., people walking, talking, meeting, etc.).

[2] For example, in a camera network of 6 cameras where each camera has 4 regions, there will constantly be $3 * 6 = 18$ regions out of the 24 for which the scheduler has high uncertainty.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

The physical space is monitored by a network of cameras $Cam = \{1, \ldots, K\}$, each of which can observe certain regions that are specified by the cover function—

$$C(r, j) = \begin{cases} 1 & \text{region } r \text{ is covered by camera j.} \\ 0 & \text{Otherwise.} \end{cases}$$

We write $A_{t-1} \in A$ for the action plan which specifies the PZT actuation state for each camera at the next time step t. We write $O^r(X_t, A_{t-1})$ for the observation at time t at region r given the world is in state X_t and the cameras are actuated according to A_{t-1}:

$$O_t^r = \begin{cases} Sc(X_t^r,\ high) & \exists j \in Cam\ |C(r, j) = 1 \\ & \text{and } A_{t-1} \text{ calls for } j \text{ to zoom in } r. \\ Sc(X_t^r,\ low) & \exists j \in Cam\ |C(r, j) = 1 \\ & \text{and } A_{t-1} \text{ calls for } j \text{ to zoom out.} \\ \text{No Observation} & \text{Otherwise} \end{cases}$$

where Sc, denoting the scheduler observation is defined as:

$$Sc(s, r) = \begin{cases} \text{High Resolution Facial Image} & s = 1 \text{ and } r = \text{ high} \\ \text{Low Resolution Facial Image} & s = 1 \text{ and } r = \text{ low} \\ \text{No Event} & \text{Otherwise.} \end{cases}$$

Thus, there are four possible values available for O_t^r: high resolution facial image, low resolution facial image, no event and no observation. Note the difference between the last two values—no event stands for the scheduler knowing that there is no person in that region while no observation represents the fact that the scheduler has no information about the state of that region. Scheduler actions cannot affect the environment state, but rather its observation of it—$O(X_t, A_{t-1})$. In other words, regardless of the camera configuration (zoomed in or out), the model of state transition remains unchanged. Activity that is taking place in the real world is not affected by camera actuations (the actuation of cameras does not change the way people use the space).

We assume that there is no observation error, thus an event that is taking place will be detected if it is being watched. However, zooming in on a region is at the expense of observing other regions covered exclusively by this camera.

We define the reward for time t as the sum of rewards over all monitored regions.

$$R(O_t) = \sum_{r=1}^{N} R(O_t^r). \tag{1}$$

The reward for each region is defined as follows:

$$R(O_t^r) = \begin{cases} \alpha & O_t^r = \text{ High Resolution Facial Image} \\ \beta & O_t^r = \text{ Low Resolution Facial Image} \\ 0 & \text{Otherwise.} \end{cases}$$

This representation expresses a large family of reward functions. It is inspired by the task of recognizing entities using pan–zoom–tilt enabled surveillance cameras. High resolution facial images enable more features to be extracted and thus increase the likelihood of a correct identification. Entities can still be recognized from a low resolution image, but with less certainty [7]. The choice of α and β defines the application utility associated with each observation. For example, if we are 10 times more likely to identify a person using a high resolution facial image as opposed to a low resolution we should set $\alpha = 10 * \beta$.

Consider the case where $\beta = 0$ and $\alpha > 0$. Collecting low resolution images serves absolutely no purpose to the application. A greedy approach where the scheduler collects only high resolution images is, sometimes, not optimal. Low resolution images provide a more accurate state representation. This, in turn, increases the likelihood of the scheduler to make the right call and satisfy the application needs.

The scheduler that we design will automatically choose the best action to take based on the specific $\alpha, \beta \in \mathbb{R}^+$ values given as input to the process. If, for example, $\alpha \leq \beta$ the scheduler should always stay zoomed out and not take the risk of missing a low resolution image. The "hardest" cases occur when $\alpha > \beta$. In this case, the scheduler must find the optimal balance between zooming in and staying up.

To summarize, A is the actuation plan specifying the PZT state of each camera (either zoomed-out or the region being zoomed into). The state transition model P captures the "motion semantics" of how people move in the monitored space; e.g., people tend to appear in spatially adjacent regions of cameras as they walk. In our model, actions do not directly affect the phenomena but the observation of it; $P_{A_t \in A}(X_{t+1}|X_t) = P(X_{t+1}|X_t)$. We write $O^r(X_t, A_t)$ for the observation at region r at time t given world state X_t and actuation plan A_t. $R(X_t, A_t)$ is the reward associated with being at state X_t while taking action A_t; we assume this decomposes into a sum of per-region rewards $R(X_t, A_t) = \sum_{r=1}^{N} R(X_t^r, A_t)$.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

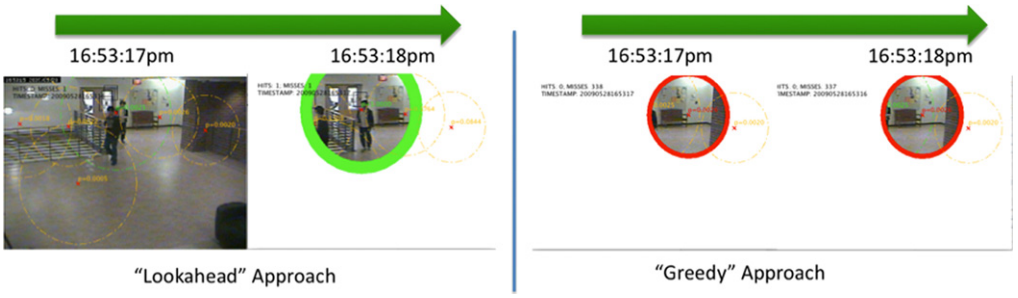


Fig. 3. Illustrating the benefits of looking ahead.

In this setting, we would like a system which schedules future actuations given a *partial observation* of the world, provided by the actuated camera network. It must trade-off actions that satisfy the immediate needs of an application versus those that maintain an accurate representation of the world. Specifically, due to physical limitations, each camera can be actuated to exist at a single PZT state. A zoomed in state allows the system to collect high resolution images at the cost of a limited field of view, which limits the system's knowledge of the monitored space. On the other hand, a low resolution image, while not providing a enough detail for face recognition, may still be used to obtain a coarse understanding of how people are moving within the monitored space, which in turn may help target the collection of high-resolution images in the near *future*.

2.5. The benefits of looking ahead

In order to illustrate the key idea behind the lookahead approach, consider Fig. 3. Where the scheduler takes into account the long term effects of its actions. It elects to take actions whose sole purpose is to reduce the uncertainty about the state of the world. The scheduler knows to zoom out with the understanding that this will "pay off" in the future, by zooming into the right region for a high resolution image. A greedy approach which would always choose to zoom into the most likely region, is more likely to miss events that are taking place. The two frames in Fig. 3 were extracted from a real execution of the scheduler with two different scheduling algorithms on the same recording of events. When executing our proposed approach, the scheduler was able to zoom into the right region after zooming out and seeing the activity taking place. When executing the greedy approach, the scheduler has missed both activities (high and low resolution) as it was zoomed into the wrong region.

2.6. A two second lookahead tree

In order to understand the "lookahead" approach let us consider a "toy example": Assume we have a network of a single camera with two zoom regions. X_t in this case is a two dimensional binary vector indicating the presence or absence of a face at time t in each of the regions. For the sake of presentation we assume that the two regions are independent and both regions transition based on the following model: $P(face\ \ face) = 0.65\ P(face\ \ no\ face) = 0.05$. Thus, the probability of seeing a face in the following time step given that we see a face right now is 0.65 and 0.05 if no face is observed. The reward function sets the constants to $\alpha = 1$ and $\beta = 0$, thus we are only interested in collecting high resolution facial images.

Assume that $X_t = (0, 0)$ is the current world state, and it is available to the schedule. Thus the scheduler was zoomed out at time $t - 1$ and was able to observe that there was no motion in either region. In Fig. 4 we illustrate the look ahead tree for two seconds. We compute all possible actions the scheduler can take and all possible states it can end up in. For example, if it decides to zoom out (marked as "UP") – the scheduler can observe one of four different outcomes – $(0, 0)$ = see nothing in either region or alternatively: $(0, 1)$, $(1, 0)$ and $(1, 1)$. If, however, the scheduler decides to zoom in region 1 (marked as C1) – the scheduler can reach only two possible states – $(0, 0.05)$ and $(1, 0.05)$. Either the scheduler observes motion in region 1 or not. For region 2—the scheduler does not get an observation, but based on the transition model, the scheduler will estimate the probability of motion as $P(face\ \ no\ face) = 0.05$. From this example we can see how the action that the scheduler chooses affects its observation of the world.

The expected reward of going "UP", taking into account two seconds of looking ahead is computed by taking into account the four possible ways the state of the world can evolve from state $(0, 0)$ and multiplying the probability of each state with its expected reward. For example, the transition $(0, 0)$–$(1, 1)$ has a probability value of $0.05 * 0.05$ and expected reward of 0.6675 when taking into account all eight possible outcomes that can follow from the system being in state $(1, 1)$ and are indicated as the small black dots.

The total expected reward of going "UP" is: $2 * 0.0475 * 0.6675 + 0.9025 * 0.05 + 0.0025 * 0.6675 = 0.112$ while the expected reward of zooming in to "C1" is: $0.05 * 1.6675 + 0.95 * 0.0809 = 0.16023$. Notice that the reward of going "UP" is greater than zero since the look ahead process also takes into account the future reward associated with an accurate state representation. In this case, the scheduler would choose to zoom into a region even when no activity is currently taking place. The risk/reward ratio of zooming out does not pay off in this example. Growing a similar tree where

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

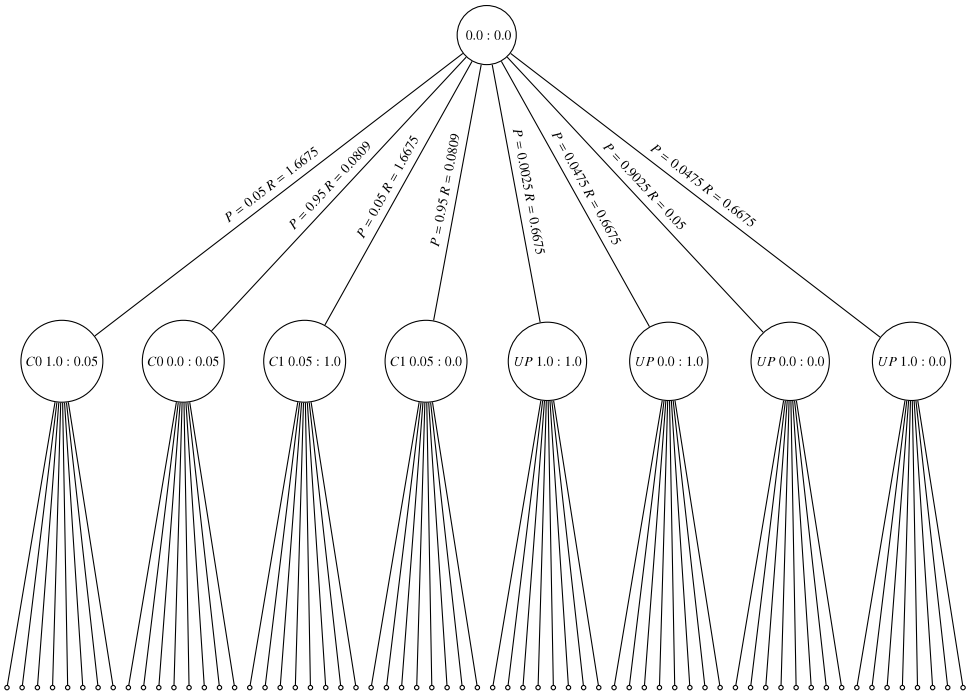


Fig. 4. Two second lookahead tree for our "toy example".

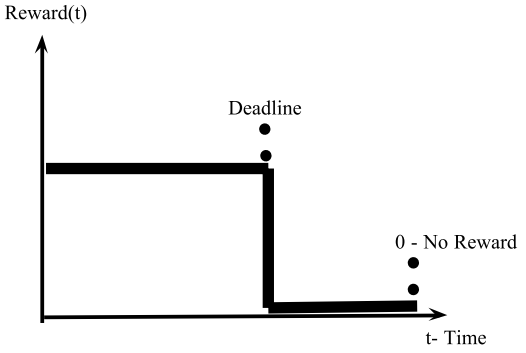


Fig. 5. The real-time challenge, after a short deadline, the utility of an action is zero.

$P(face\ \ face) = 0.8\ P(face\ \ no\ face) = 0.1$ would tilt the scales towards zooming up as the likelihood of being able to follow an activity after seeing it is high enough to be worth the risk of missing an event while being zoomed out "UP".

In this way we can compute the expected utility of all possible scheduler actions from a given world state. The optimal action for the scheduler is the action with the maximal expected utility. Growing a tree this way allows us to compute the expected utility of actions, however, it is not tractable (exponential with regions and time) and requires reconstruction for each state at real-time. This is a problem, since events that are taking place in the real world will only last for a short while, and will definitely not wait until the scheduler selects the optimal action. After the hard deadline, the activity is missed and utility of a correctly selected action is zero— Fig. 5.

The approach that we propose generates a policy table that approximates the action with the highest expected utility for all possible world states. Thus, the action selection happens in real-time, with very small latency while the hard computation parts of the problem are calculated offline.

2.7. An approximated POMDP Scheduler

We define a POMDP that naturally balances both long and short term benefits of the actuation problem. Our POMDP is a five-tuple (S, A, P, O, R) where S is the set of states of the physical system being monitored, A is the set of actions, P is the state transition model of the physical system, O is the state observation model, and R is the state-action reward function. Given the above POMDP, the goal is to find a policy π that specifies an action for each estimated world state so

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

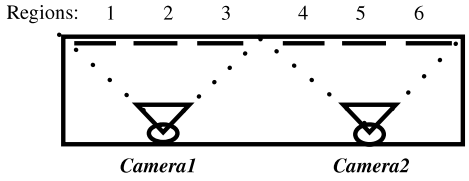


Fig. 6. "Toy" example camera network—long hallway continuously covered by cameras.

as to maximize the total expected reward over some finite horizon: $\sum_{t=1}^{T} E[R(X_t, A_t)]$. We refer to T as the "look-ahead" of the system; if $T = 1$, the system is myopic and will always actuate sensors to maximize its immediate reward. With larger T, an optimal policy will realize that it is better sometimes take actions to reduce uncertainty, as this will produce a more accurate estimate of world state which can help increase future rewards.

While discrete-state MDPs are solvable with dynamic programming, POMDPs are intractable [4].

Directly applying a POMDP solver is not tractable—

"Off-the-shelf" POMDP solvers are available[3] and they are guaranteed to find the optimal policy. However, they can only solve problems with state space containing about 10 states [8]. None of the available techniques can be applied directly, due to the scale of our problem. We propose one solution which is optimized for the camera actuation problem. In our case, the size of the state space (the space of our X_t states) is exponential compared to the number of regions—2^N. Thus, even for a modestly sized environment of half a dozen cameras with each covering 7 regions, the state space would be 2^{42} which is several orders of magnitude larger than the maximal state space solvable accurately.

This calls for significant optimizations in order to make the problem tractable. Our scheduler exploits the state transition characteristics of human motion in a building and the additive nature of reward functions (Eq. (1)) of pervasive applications.

One attractive approach is to cast a POMDP as a MDP whose state space becomes a "belief state", or the set of all probability distributions over world states. Unfortunately, one needs an exponentially-large number of gridded or quantized belief states. One of our primary contributions in this work is an adaptive gridding strategy that dynamically adds discrete states to an underlying belief-state MDP, so as to obtain a provably good approximation of the value (or expected reward) of each belief state and action.

3. Approach

Before we present the details of our proposed approach, let us first present a motivating example that will help the reader better understand the different algorithms and models, later presented in the paper. Consider a "toy example", where 5 cameras each with 5 zoom regions are aligned in a single hallway. Fig. 6 illustrates a hallway with 2 cameras and 3 zoom regions. Each camera can observe all its regions when zoomed out or only one when zoomed in.

The model for this example is defined as follows:

S is a vector of 25 bits, e.g., $S_i = (1, 0, \ldots, 0, 0, 0)$ indicating that a person is present in the leftmost region of the leftmost camera. A is $(A_1, A_2, A_3, A_4, A_5)$, where $A_i = (R_1, R_2, R_3, R_4, R_5, UP)$. For example, the action that zooms into the left most region in the first camera and stays up in all other regions is $A = (R_1, UP, UP, UP, UP)$. Given S and A as defined above, we can enumerate the action space: each region can be in one of six states A_i and the 5 cameras can choose an action independently of each other—$|A| = 6^5 = 7776$. Since the S_i is a vector of 25 bits, storing the transition probabilities between two states will require a table with $(2^{25})^2$ entries. Thus, explicitly modeling $P(S_{t+1}|S_t)$ is not tractable, even for this rather small example. The storage requirements of just storing the probability values will require 4096 Tb of space—clearly not tractable. Furthermore, using this model to construct a POMDP directly would require maintaining a probability value for each possible state the world can be in. Thus, at any given time, we would need to maintain a vector of 2^{25} dimensions. Although guaranteed to generate optimal actions, simply storing the required data is not tractable. We will not consider this approach further.

Let us first simplify the model. In reality, there is no need to store $P(S_{t+1}|S_t)$. What we really want is to store the interactions between camera regions—what is the likelihood of a person appearing in region i given they were present in region j a second ago. Inspired by that insight, the factored model approach, described in Section 3.1, requires storing a single probability value for each pair of regions. Reducing the size of the probability table to $25^2 * 5$ floats, or 1 kB of space (we multiply by 5 to account by a 5 s history).

3.1. Action selection alternatives

With the above tractable transition model, we move on to describe the strategies for scheduling actions. We assume that the reward for a high resolution (zoomed in) image is 1 and zoomed out image is 0. We would like to design a scheduler that continuously selects actions that have the highest expected utility.

[3] see http://www.pomdp.org.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

Table 1
Summarizing our approach to address different challenges.

Challenge	Approach	Subsection
Transition model is exponential $O(2^{NM})$.	Assume that people move independently and use the noisy OR model which is linear $O(N^2M)$ (as in [9]).	3.2
Scheduler belief state requires 2^N dimensions.	Assume that the scheduler belief state can be factored into a belief state for each individual region which reduces the belief state to N dimensions.	3.4
Looking ahead by growing a search tree is exponential in time $O(N^t)$	Represent only certain scheduler states and use dynamic programming (value iteration) $O(NT)$.	3.5
Hard to select which scheduler states to represent a priori.	Based on the properties of our reward function we bound the error and dynamically add points to the representation if the error is not acceptable.	3.7
Scheduling decisions must be made at real-time.	Partitioning the space by cameras so that only regions that are in direct competition be in the same partition. Pre-compute an approximated state action grid for each camera with constant space requirements.	3.9

N regions, M past states, T seconds lookahead.

3.1.1. Most likely approach

A naive approach for action selection is to always zoom into the region that has the highest likelihood of observing a person. The drawback of this approach is that it will never attempt to balance the need to reduce uncertainty with the need to maximize the reward. This balance is accomplished by the notion of "looking ahead" and considering the expected utility of each action based on probability weighted ways the world can evolve, see Fig. 4.

3.1.2. Tree based approach

Looking ahead by growing a search tree will require exploring $((6^5) * 25)^5 = 2.7 * 10^{26}$ nodes, at real-time—clearly not tractable.

3.1.3. Factored camera approach

To reduce the number of states explored, we propose to partition the action selection to computing the expected utility of each action for each camera separately. This will require considering $(6*5)^5 = 24$ M states in real-time. In our experiments, we found that the latencies for action selection for the tree based approach for 42 regions is close to 1 s. This latency is still not suitable for a real-time scheduler (see Fig. 12(a)).

3.1.4. Value iteration approach

To further optimize the action lookup in real-time, we pre-compute action tables. We do that by starting for assuming that the state of each camera can represented by $S = (S_1, S_2, S_3, S_4, S_5)$, where S_i is a finite set of probability values, for example 0, 0.1, 0.2, . . . , 0.9, 1. When an intermediate state needs to be accessed, it is approximated and the table entry is looked up. This approximated grid introduces unknown errors that affect the accuracy of the expected utilities computed.

3.1.5. Grid based approach

To introduce error bounds on the computed utilities and ensure that the scheduling actions are correctly selected, we propose an adaptive grid based method (Section 3.7). The key idea is that states are added to the grid based on their impact on the expected utility of actions. The tables are iteratively refined to meet the required error bounds. The smaller the desired error on the expected utility, the longer it takes to construct the table and the more space it will require in memory.

The rest of this section discusses in details our approach to address each of the above challenges, also summarized in Table 1.

3.2. State transition model

Our monitored space is divided into N disjoint regions. At (discrete) time t the observed phenomena is assumed to be in some state X_t. X_t is represented as a binary vector of length N for example, $X_t^i \in \{0, 1\}$ can be used to encode the presence of a person in region i at time t or lack of thereof.

Fig. 7 illustrates the conditional transition probabilities of the face appearing in each of the regions in a given camera given the person's current location.

3.3. Visualization of the motion semantics collected

To illustrate cross camera state transitions, consider the following camera layout as illustrated in Fig. 8(a). Camera 2 is deployed in a short hallway that leads to cameras 3 and 4. This deployment results in correlated activity between cameras 2, 3 and 4. Motion in camera 2 results in increased probability of motion at cameras 3 and 4. We plot the conditional probability

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

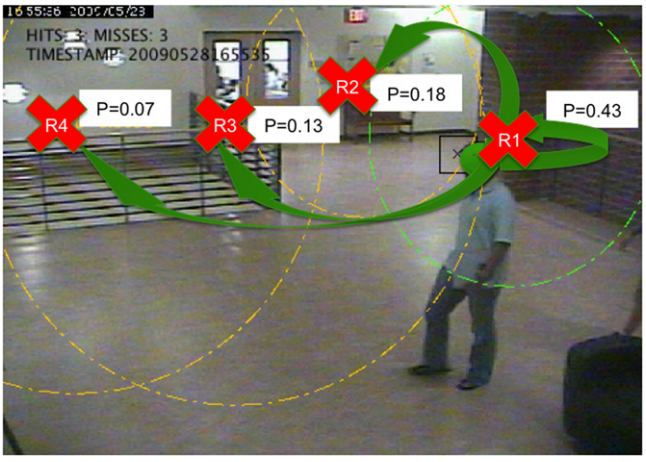
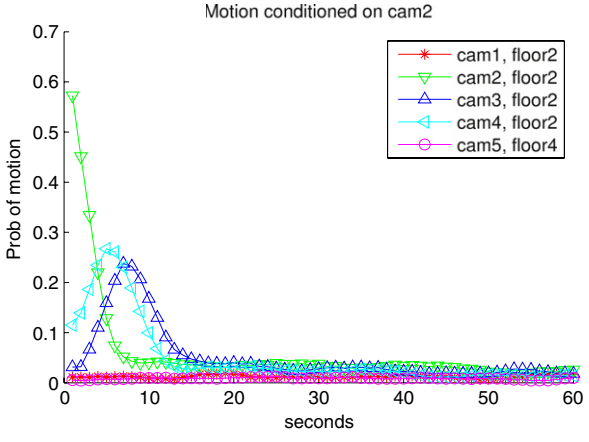
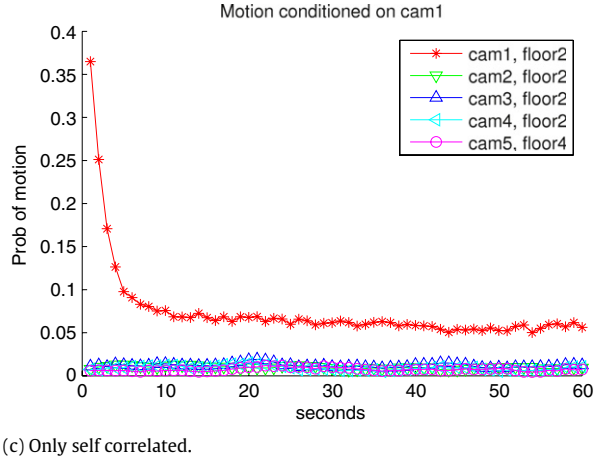


Fig. 7. Illustrating the conditional probability transition model.

(a) 2nd floor.



(b) Cross-correlated.



(c) Only self correlated.

Fig. 8. Physical instrumentation and visualization of motion semantics.
Source: Figures reprinted from [9].

of motion in all other cameras given that motion was observed at camera 2 in Fig. 8(b). Note the increased probability of motion in cameras 3 and 4, in the time flowing motion in camera 2. Another possible case, is when only self correlation is

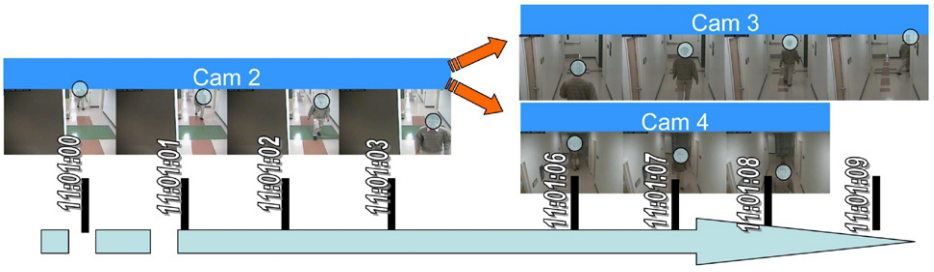


Fig. 9. A case illustrating the conditional correlation of motion.
Source: Figures reprinted from [9].

present as it is the case for camera 1, the conditional probability of motion in other cameras given motion in camera 1 as a function of time is plotted in Fig. 8(c). This semantic information is exploited by the scheduler in the following way: Given that motion appears in camera 1, it will estimate motion continuing in camera 1 with high probability. 6 s after motion was detected in camera 2, it will estimate motion in the correlated cameras (3, 4), with high probability. A case of such a motion pattern is illustrated in the image in Fig. 9.

The number of possible world states is 2^N, meaning that naive state transition models will not scale. We adopt the factored transition model of [9] which assumes that entities will move independently in a monitored environment:

$$P(X_t^j | X_{(t-M):(t-1)}^{1:N}) = 1 - \prod_{o=1}^{M} \prod_{i=1}^{N} (1 - \alpha_{ij}^o X_{t-o}^i) \tag{2}$$

where α_{ij}^o is the probability that a person moves to region j o-seconds later, given that they are currently at region i.

The above model can be derived from a noisy-OR assumption on spatio-temporal correlation. While simplistic, it captures many natural semantics about object movement (people tend to walk down hallways, wait at elevators, etc.) without explicit multi-object tracking, which can be difficult in unconstrained scenarios. [9] show that the factored-frontier algorithm [10], or one-pass belief-propagation, suffices for inference with such a model. The final algorithm can be intuitively thought of as a prediction filter (that predicts future states given the semantic model) followed by a correction stage (that re-weights predictions using new observations).

3.4. Approximating the POMDP belief states

POMDP solvers maintain an internal belief state about the external world. Because the number of distinct world states is exponentially large (2^N), representing distributions over such spaces would require a vector of (2^N) dimensions. Fortunately, the factored inference algorithm of the previous section suggests that one can approximate the probability over states as a product of marginal probabilities for each of the N regions. Hence we represent a belief state $\hat{X}_t$ as a N dimensional vector where each entry $\hat{X}_t^j$ is the probability of an event taking place in region R_j at time t.

Scheduling based on the estimated state $\hat{X}_t$—Given our best estimate of the state of the world—$\hat{X}_t$ we face a difficult problem: What should the course of action be for the next time step? One might suggest zooming into the most likely region in terms of the computed probability. In this case, the scheduler will always be zoomed in, as there will always be a region which has the highest probability. Furthermore, by following this approach, the probabilities will be computed based on a partial view of the space, as each camera is limited to viewing only the region it is zoomed in to. In a camera network of 6 cameras where each camera has 7 regions, there will constantly be $6 * 6 = 36$ regions out of the 42 for which the scheduler has high uncertainty. Optimally we should choose the action with the highest expected utility in the "long run", taking into account both the indirect reward associated with reduced uncertainty and the direct reward associated with satisfying the application requirements.

3.5. Value iteration

In this section, we temporarily make the simplifying assumption that approximated belief states $\hat{X}_t$ can be represented as discrete elements $s \in S$ such that $|S| = n$. In this case, our POMDP can be written as a MDP with discrete states, and the optimal policy π can be represented as a table with $O(n)$ entries whose entries can be computed in $O(nT|A|)$ with dynamic programming (where T is the horizon and A is the set of possible actions). The full algorithm, described in Algorithm 1, is known as value iteration. We refer the reader to [6,11] for complete details, but we briefly review it here as our solution builds upon it.

Algorithm 1 computes the maximal expected reward by performing a "backward walk" in time. At time T, the time has already ended (we started from 0, and performed a look ahead of T s). Thus the maximal expected utility is zero. At time $t \leq T - 1$ the process computes the expected reward of state s for each action $a \in A$ by taking into account all possible

Algorithm 1: Dynamic Programming Value Iteration

```
   Data: follow(s,a)- Returns the set of states reachable from state s taking action a.
 1 begin
 2     /* Initialization */
 3     for s ∈ S do
 4         for t ∈ 1..T do
 5             opt[s, t] ⟵ 0
 6     for s ∈ S do
 7         for t ∈ (T − 1)..1 do
 8             for a ∈ actions do
 9                 /* Compute the expected reward for a at state s when t seconds remain. */
10                 a_R ← 0
11                 for s_{t+1} ∈ follow(s, a) do
12                     a_R ← a_R + p(s_{t+1}|s) ∗ (R(s_{t+1}|a) + opt[s_{t+1}, t + 1])
13                 opt[s, t] ← max(opt[s, t], a_R)
```

scheduler belief states s_{t+1} that can follow from s when action a is taken and looking up their expected reward in the dynamic programming table. The immediate reward of being in state s_{t+1} while taking action a is $R(s_{t+1}|a)$. The value stored in the table is that of the action with the highest expected utility $E_{\max}[R(s)] = opt(s)$. The following subsections until 3.9 present our approach to discretize S, and find an approximate solution with quality guarantees.

3.6. Grid creation

In order to apply the above approach using a discretized set of beliefs states S, we start by including the zero and unity vectors and creating equally spaced grid points along probability vectors. However, future belief states s_{t+1} in line 11 in Algorithm 1 will likely not be present in the finite set S. One can use the closest element in S, but this introduces errors that are compounded over time. To address this, we describe an adaptive strategy for iteratively adding discrete states to keep the error within some tolerance.

3.7. Adaptive discretization with bounded error

We describe an algorithm for adaptively discretizing the state space S during Value Iteration such that the expected utility *opt* is computed within some user-provided tolerance. This bound is possible due to the additive and monotonic nature of the reward function in sensor actuation problems, which in turn lets us compute upper and lower bounds on *opt* during dynamic programming. We begin by defining two marginal belief state sets which will help us bound the error with a given state s:

$$FloorSet(s) = \{sf | \forall_j sf^j \leq s^j \text{ and } s \in S\}$$
$$CeilingSet(s) = \{sc | \forall_j sc^j \geq s^j \text{ and } s \in S\}.$$

The ceiling set of a state s contains all the approximated marginal belief states that have a higher or equal probability of an event across all regions. Similarly the floor set has lower or equal probabilities.

Lemma. $\forall_{sf \in FloorSet(s)} E_{\max}[R(sf)] \leq E_{\max}[R(s)]$.

Proof (*Sketch*). We model the space as a vector X_t where the goal is to actuate sensors to observe as many $X_t^i = 1$ as possible. Our approximate belief state is represented as marginal probabilities, each element in s_i represents the probability of an activity taking place in region r_i. Thus, reducing the probabilities in s to those in sf reduces the probability of transitioning to any state which contains $X_t^i = 1$ due to the additive nature of our model (see Eq. (2)).

We compute the maximal expected utility of s based on the utility of $\hat{s} \in follow(s, a)$ which we repeatedly replace either with a state $\hat{s} \in FloorSet(s)$ if $\hat{s}$ is not present in the grid, or the exact value if $\hat{s}$ is present. In any case, the expected utility that will be computed will be a lower limit on the true expected utility:

$$\forall_{sf \in FloorSet(s)} E_{\max}[R(sf)] \leq E_{\max}[R(s)]. \quad \blacksquare$$

A similar argument can be used to show that increasing the probabilities, results in an upper bound on the true expected utility:

$$\forall_{sc \in CeilingSet(s)} E_{\max}[R(sc)] \geq E_{\max}[R(s)].$$

This property allows us to bound the true utility:

$$opt[sf, k] \leq opt[s, k] \leq opt[sc, k].$$

Whenever we need to approximate state s_{t+1} we take the closest state in $FloorSet(s) - sf$ and the one in $CeilingSet(s) - sc$. Each entry in the dynamic programming table maintains four values: the expected utility when consistently approximating states using their floor set as well as the ceiling set and the actions that were used to achieve the highest utility for the floor and ceiling states.

Before the dynamic programming routine processes the entries for the next time step (as part of the for loop in line 7 of Algorithm 1) we verify that the error rates in the entries we just computed meet the desired error level. This allows us to upper bound the error by the following expression: $\epsilon \leq opt[sc, k] - opt[sf, k]$ If the upper limit on the error rate is below the desired error, the process terminates and returns the computed table. In the case where the error is above the user specified threshold, we must add states to our approximation of the state space—S and restart the process.

3.8. Adding states to the grid

If the error is above the desired threshold, we use the current grid to generate a new, finer grid by adding states based on their expected contribution to reducing the error. To accomplish this, we keep track of the states that were missing from the current grid and had to be approximated when s_{t+1} was computed (line 11 of Algorithm 1). We maintain a priority queue with the top k states (in our experiments we used $k = 5000$) ordered based on their Euclidean distance from the closest state currently in S or the priority queue. The reason we want to limit the number of states that are added in each iteration is to ensure that we do not increase the state space unnecessarily. Although there might be more than k states that were needed, once we add the top k states we might meet the ϵ required error bound. This allows us to control the growth rate of the state space such that it is within k states of the minimal size needed. Without this iterative selection process the state space explodes with states which are very close to each other and cause the process to exceed memory limits. The prioritization routine allows the process to find solutions for much smaller ϵ values. Moreover this allows the user to specify a timeout condition or grid size limit (as well as ϵ) which will return the best solution (perhaps exceeding ϵ error) that could be found within the time and space constraints. Fig. 11(a) illustrates the number of states that are needed for different ϵ values to compute a policy for a single camera with four regions. As the tolerated error rate decreases the number of states increases. The problem of finding an optimal scheduler remains not tractable, as a small ϵ will require an unbounded sized state space but solutions are iteratively improving and the processing duration and required state space can be controlled by changing ϵ. In order to illustrate the impact of the estimation accuracy of the number of states, we plot the number of states required to build the lookup table as a function of the desired estimation accuracy, Fig. 11(a). The estimation accuracy is captured by specifying the maximal error rate that is tolerable. When the error rate is exceeded, new states are added to the grid. Notice that the number of states needed for very high accuracy levels is still not tractable. From our experiments on real settings, we found that the number of states that are needed for the action selection policy to achieve comparable results to an exhaustive search is small. Note that for action selection purposes, not selecting the optimal action does not necessarily mean selecting an action of no utility. On the contrary, most likely the action selected is a very close "runner upper".

The required grid size needed in order to get relatively low error rates is manageable as illustrated by Fig. 11(a).

An improved initial grid—We applied an approach which allows us to gauge the need in states so that ones that are more likely to be accessed are included in the grid as follows: we first perform an offline execution of a tree based approach, with a lookahead of 2 s, computing the exact utility by growing a tree of all possible states and actions from a given scheduler state, for each camera. While the tree based approach is in execution we added points to the initial grid based on the same technique that we used to prioritize which states to add so that the error rates are within bounds. In this case, we used a priority queue to decide on which states to keep in the initial grid based on the states that are more likely to be accessed by the real-time scheduler. Saving all the states that could possibly be encountered is exponential, thus the priority queue approach allows us to effectively allocate our memory resources to where they are most needed. Note that the error bound guarantees are for points that are in the grid, thus, creating an initial grid this way reduces the errors introduced when points are looked up by the algorithm while at execution.

Efficiently approximating s_{t+1}—Whenever a state is approximated we find its nearest neighbor in its floor and ceiling sets. This is done efficiently by using a range query over a KD-Tree [12,13]. We start with S containing an equally spaced grid across all dimensions. For example, if we space each dimension according to $G = \{0, 0.5, 1\}$ of a four dimensional state space then $S = G \times G \times G \times G$. Whenever a state s_{t+1} is approximated we issue a range query between two points in the d-dimensional space—s_{t+1} and $Ceiling(s_{t+1})$. Where $Ceiling(s_{t+1})$ is the closest point to s_{t+1} that is also in $CeilingSet(s_{t+1})$. Such a point can be found efficiently by a binary search across each dimension. Since $Ceiling(s_{t+1}) \in S$ we are guaranteed a non empty result set in our range query, from which we select the nearest neighbor—sc. Similarly we select the closest point on the grid which is in s_{t+1}'s floor set sf. The closest of sc and sf is used to approximate s_{t+1} and $\epsilon \leq opt[sc, k] - opt[sf, k]$ is used to approximate the error associated with this approximation. This process is used to perform a lookup of a value in the grid both for the dynamic programming routine as well when an action is selected for a given state. KD-Range queries have time complexity of $O(\sqrt{|S|})$ and thus the process of approximating a given state to an existing grid point is extremely efficient. We report latency results in Fig. 12 (experimental setup is discussed in details in the experiments section). For the approximated POMDP approach, action selection is as fast as a table lookup. The more zoom regions in the monitored environment, the higher the latency of lookup. In a simulated environment of 100 regions, action lookup takes about 30 ms. Action selection for the tree based approach takes an order of 800 ms for 42 regions. This latency is no longer considered real-time. By the time the system decides where an activity of interest occurs, even when it is correct, it will be too late to

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

capture it as the environment keeps changing regardless of scheduling. The action selection in the approximated POMDP case, for 42 regions, takes an average latency of less than 2 ms.

3.9. Distributed real-time scheduling

At run-time, our scheduler updates its beliefs about the world using the *global* transition model—$\hat{X}_t = P(X_t^j | X_{(t-M):(t-1)}^{1:N})$. Beliefs are updated by standard prediction/correction equations (see Section 3.2) for temporal models. Notably, this model is *global* in that correlations across cameras are taken into account. However, approximating the optimal policy table (π) based on the estimated state $\hat{X}_t$ with small ϵ is still intractable for networks with a large number of cameras. This is because n, the number of discretized states required to achieve a low error, empirically tends to grow exponentially in T (expected, as solving POMDP optimally is not tractable). To address this limitation, we adopted a factored scheduling approach, where the *global* scheduling table is partitioned into non-overlapping *local* scheduling tables, see Fig. 13. In real-time, the scheduler is responsible for two tasks: first it must approximate the state of the world using the dynamic state transition model and its previous observations. Second, it must perform a lookup for the best action to take, per-camera based on the per-camera, pre-computed look ahead table. The first task requires $O(N^2)$ multiplications while the second is a single table lookup thus making the scheduler extremely lightweight and fast.

In our implemented system, cameras observe non-overlapping regions, and so it is natural to build a local policy table (or local scheduler) for each camera.[4]

Overall, real-time scheduling proceeds as follows: region-specific beliefs $\hat{X}_t^r$ are updated using prediction/correction filtering, which require (ON^2M) multiplications. The updated beliefs are then used to query a camera-specific policy table to return a scheduled action state for each camera. This query requires a single table lookup per camera $O(N)$, making the overall scheduler extremely lightweight and fast.

3.10. Multiple pervasive computing applications

Consider the scenario where multiple applications make use of different reward functions that may even change over time. For example, one may be interested in high and low-resolution during the day time, but interested in only high-resolution images at night (because low-resolution images under low light will be noisy). Assume we are given Q different applications. For any subset of active applications, one can compute the optimal policy using a reward function equal to the sum of each application-specific rewards. A naive approach would pre-compute an exponential (2^Q) number of policies corresponding to all possible subsets of active applications. We now describe an efficient algorithm for representing this set, making use of the additive property of our reward function. Let us augment our policy table from Algorithm 1 to store the expected utility of all actions for each state, and not just the best action. We then compute Q application-specific augmented policy tables. Given these Q tables, we compute policies for any subset of active applications by adding together "on-the-fly" the expected utility for each action-state, as described by Algorithm 2.

Algorithm 2: Approximate approach to actuate sensors for concurrent applications

```
Data: E_app[s, t, a]- Returns the expected reward for app taking action a from state s when t seconds remaining.
1  begin
2      best_action ← null
3      best_action_reward ← 0
4      for a ∈ actions do
5          action_reward ← 0
6          for app ∈ applications do
7              /* Compute the expected utility for a at state s when app applications executing. */
8              action_reward += E_app[s, t, a]
9          /* Save the action with maximal total reward. */
10         if best_action_reward < action_reward then
11             best_action_reward ← action_reward best_action ← a
12     return best_action
```

3.11. Cameras with overlapping regions

In certain cases, cameras can have overlapping regions. If this is the case, selecting actions for each camera independently might result in a sub-optimal actuation. For example, it chooses to zoom two cameras to the same region.

We suggest two ways to overcome this limitation. First, we can schedule several cameras at the same time and find the joint actions which have the highest expected utility. Second, we can select actions for cameras according to a pre-defined order and for each camera choose an action which has the highest expected utility given the actions that were already selected.

[4] Further approximations for overlapping regions are discussed in Section 3.11.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

The first alternative is guaranteed to find the best action to take, but is not tractable for most camera networks. The challenge is that the number of actions that are possible for a joint of K cameras where each camera has A actions is K^A. Furthermore, all regions that are covered by the K cameras need to be taken into account when performing the value iteration, and as a result we need to approximate a belief state of a larger dimensionality.

The second alternative is tractable for large scale camera networks and provides an approximated solution. In order to support this approach, the value iteration procedure needs to keep track of the expected utility, not only of the best action, but of all actions. This allows the real-time process to select actions according to Algorithm 3.

The main idea is to select the actions, considering the added utility of the action given all the actions that were selected so far. This will enable the scheduler to avoid having two cameras zooming into the same region, for example. Note that this will not take into account the long term effects of actions but only their immediate reward. The first alternative is guaranteed to find the optimal joint schedule taking both immediate reward as well as long terms reward of each (joint) action.

Algorithm 3: Selection actions for cameras with overlapping regions

Data: **ER-** Returns the expected reward associated with taking an action for a state for a given camera given all actions selected so far. **C-** Set of cameras with overlapping regions. **s-** returns the current state for a given camera. **actions-** returns the set of actions for a given camera.
Result: **Plan** of actions selected for each camera.

```
1  begin
2      Plan ← {}
3      for c ∈ C do
4          bestAction ← UP
5          bestActionScore ← 0
6          for a ∈ actions(c) do
7              /* Compute the expected reward for a at state s given Plan. */
8              utility ← ER(s(c), a, Plan)
9              if utility then
10                 bestAction ← a
11                 bestActionScore ← utility
12         Plan ← Plan ∨ bestAction
```

4. Experimentation

The main parameters of the our approach are space-partitions (the number of regions in a camera and number of cameras), the reward function, and the error tolerance ϵ for our POMDP solver. In terms of evaluation, there are three important metrics (1) total reward after running the scheduler (2) latency of discovering events, and (3) number of events of interest detected versus missed (i.e., precision and recall). We split our experimental results in those that (a) compare to standard scheduling baselines (Fig. 10) and (2) diagnose the impact of various parameters (Fig. 11).

Baseline comparisons: We have implemented different baseline alternatives for comparison: (1) Consider a scheduler that for each camera, cycles between a zoomed out configuration and zooming into the regions covered by it, one by one in Round Robin. (2) Cycles in Round Robin between zoomed in configurations only in Round Robin. (3) Stays "UP" in all cameras at all times. (4) Select an action that has the highest expected utility based on the predicted probabilities. (5) Our approximated proposed approach with look ahead of two seconds. (6) A Tree based approach computing the exact utility by growing a tree of all possible states and actions from a given scheduler state, for each camera. This brute-force solution requires searching over an exponentially large space, and so is useful for analysis though not practice. (7) An "Oracle" approach which selects the best camera actuation taking into account events taking place in the future. Note that the performance of the "Oracle" approach might not even be achievable since we do not have access to future events.

4.1. Implementation

Source code for all the experiments reported in this section is available online.[5] All of our experiments and algorithms are implemented in Java. In this section we will walk through the design of key modules that were used to implement and run the experiments and the algorithms presented in this paper.

Technical experimentation design—the most important module in the implementation is the process that builds the lookahead tables based on an approximated dynamic programming process. The file that implements this logic is: decision_maker/search/DynamicProgrammingLookahead.java

[5] https://code.google.com/p/sensoractuation/.

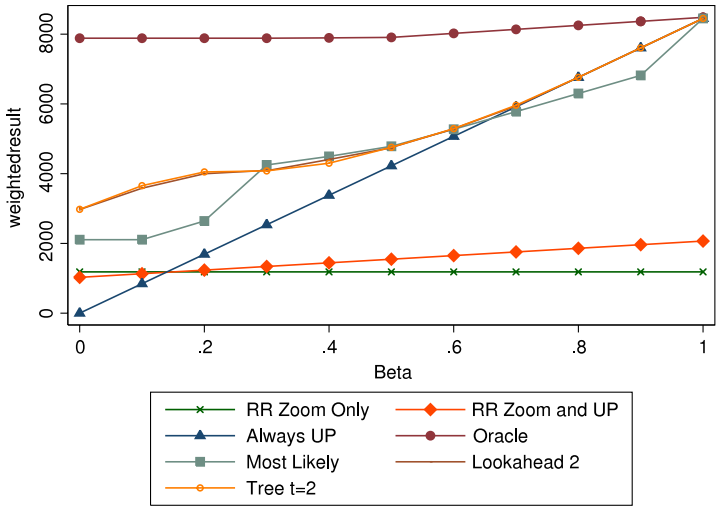
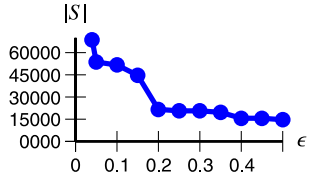
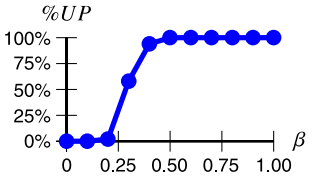


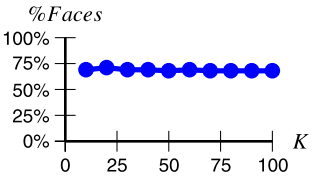
Fig. 10. Total reward for different alternatives, 7 zoom regions.



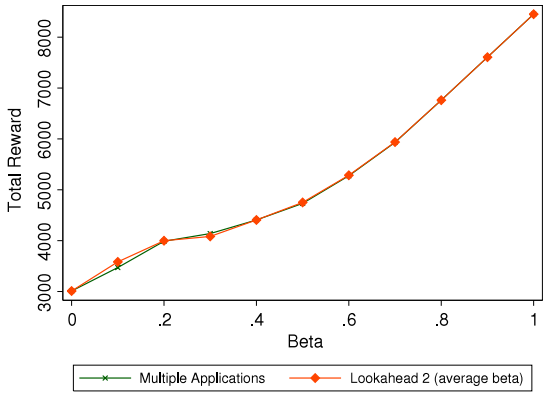
(a) Size of the state grid as a function of estimation accuracy (ϵ).



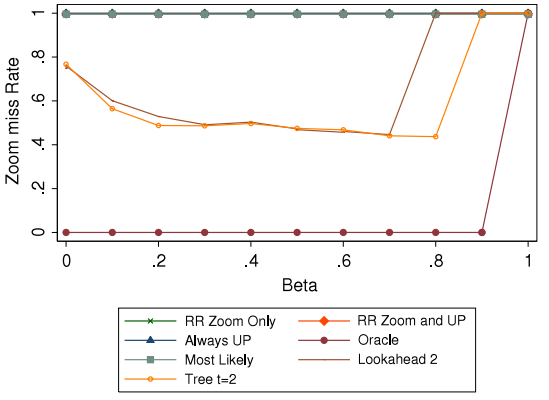
(b) % of states going "UP" as a function of β (when $\alpha = 1$).



(c) % of faces collected at high resolution as a function of the number of cameras.



(d) Evaluation of our approach to address multiple concurrent pervasive computing applications.



(e) Rate of actuations not resulting in a facial image.

Fig. 11. Illustrating the different aspects of the suggested approximated POMDP approach.

It does so by maintaining a data structure called ScoreArray, where ScoreArray[k][s] represents the maximal achievable score, reachable from state s when the current time is k. ScoreArray is updated according to Algorithm 1, using dynamic programming. In order to keep track of the error rate that is introduced in the process of generating this table, whenever a state is approximated, we store the floor and the ceiling values for that state.

At the end of each iteration of the dynamic programming step, we evaluate the maximal difference between the floor and ceiling values. If the error exceeds the user specified error bound, we update the grid, adding states that are farthest away from the existing points that have a representation. We re-evaluate the states to re-insert after every insertion, as inserting a state to the grid may change the distance function for the remaining states waiting to enter states.

The file that implements this logic is: decision_maker/search/States.java. States.java implements a grid based data structure that is stored as a KDTree. This enables the efficient execution of nearest neighbor queries. E.g.: given a state that is missing from the grid, what is the nearest neighbor currently in the tree, and what is the distance to that the neighbor.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

Simulation: to evaluate our algorithm against an optimal approach, we have implemented tree based search. The file that implements this logic is: decision_maker/search/TreeLookAhead.java. It explores the space of all possible states that can be reached from a given state, given a finite time horizon and calculates the highest expected utility for each action.

The actual simulation is implemented in a file named: simulation/Scheduler.java. The scheduler "replays" the dataset of images that contain the events as they took place in our monitored environment. For every time "tick" the scheduler only gets to "observe" the data for regions that it would see given the current action selection. For example, if camera 1 is zoomed into a region, all other regions are not available to the scheduler. The simulator then passes the data to the scheduler that selects an action for the next time "tick". We iteratively run the scheduler with different action selection algorithms, and different reward functions, computing different metrics such as latency and total reward.

4.2. Evaluation

Our evaluation dataset contains recordings of two full days of unscripted human activity.[6] The data was recorded by a network of 6 cameras covering a single floor in the computer science building. For each algorithm we evaluated, the data recorded was replayed exactly as it happened in real life, and the scheduler was given access only to data that it scheduled to collect and was evaluated based on its performance. We have divided each camera to $k = 7$ zoom regions. The more regions each camera has, the harder it is for the scheduler to select the right region to zoom in. The smaller the number of regions, the easier it is to "guess" a region that will contain a person. The reward for each region is defined as follows:

$$R(X_t^r, A_t) = \begin{cases} \alpha & \text{High-res face, } X_t^r = 1, r \in A_t \\ \beta & \text{Low-res face, } X_t^r = 1, r \notin A_t \\ 0 & \text{Otherwise.} \end{cases}$$

This representation expresses a large family of reward functions, intuitively presenting the utility of a high resolution image vs. a low resolution one. The evaluation was done by replaying the events that took place in the observed environment and computing the total reward that each alternative was able to achieve. For example, a single point in our evaluation graphs would be the total reward by Round Robin, for $k = 7$, $\alpha = 1$, $\beta = 0$. Varying β gives us a way to see how *RR* performs with different reward functions, and doing the same for the "Tree" approach allows us to compare the two scheduling alternatives.

The choice of α and β defines the application utility associated with each observation. Consider the case where $\beta = 0$ and $\alpha = 1$. Collecting low resolution images serves no purpose to the application. However, low resolution images provide information about the state of the system. This, in turn, increases the likelihood of the scheduler to make the right call and satisfy the application needs. In Fig. 10 we compare different scheduling alternatives in our camera network where each camera has seven possible zoom regions. For different application utility functions in which we increase the utility of a low resolution image, when the utility of a low resolution image is relatively high (over 0.3) the alternatives which are sensitive to the value of β are equivalent, basically reducing to a simple scheduler which constantly actuates all cameras to zoom out. For the "hard" cases, where the utility of a low resolution image is low, looking ahead for 2 s outperforms all other alternatives and accomplishes the highest total utility. Notice that our approximated approach performs very closely to the exact approach, but with real-time latencies.[7]

Diagnostic experiments: To illustrate the action selection process for different values of β where $\alpha = 1$, consider Fig. 11(b). We consider a single camera with 4 regions and solve the look ahead problem for 10 s ($\epsilon = 0.3$). We plot the fraction of states for which the best action is to zoom out (go "UP") as a function of β, the utility of a zoomed out event, where the utility of a zoomed in event is 1 ($\alpha = 1$). This table illustrates how the scheduler changes its behavior based on the characteristics of the reward function. The higher the utility of a low resolution image, the more likely the scheduler is to zoom out. The scheduler automatically selects the best action to take based on the specific $\alpha, \beta \in \mathbb{R}^+$ taking both long and short term benefits.

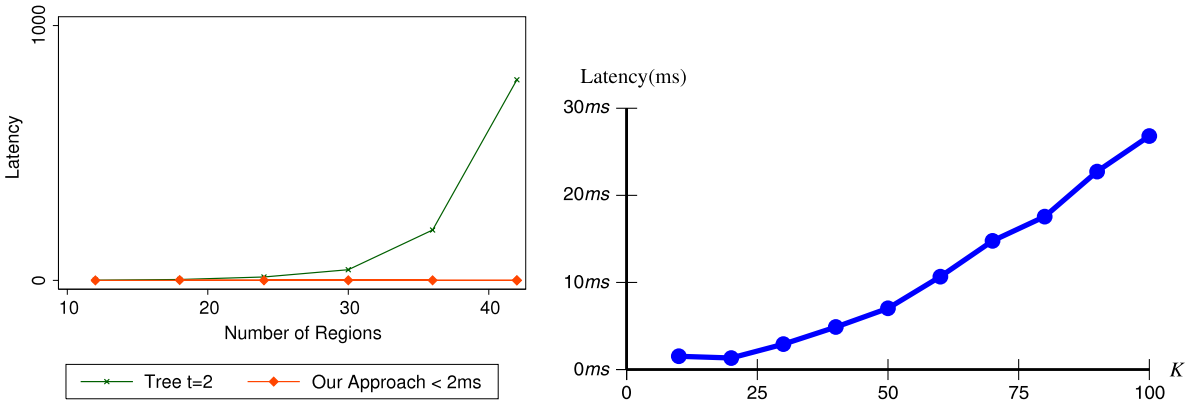
To illustrate the impact of computing the exact utility value on the latency of the scheduler, consider Fig. 12(a). We plot the latency in milliseconds of the total time that it takes the scheduler to actuate all cameras, when the decision process is performed by our approach and by a tree based approach (alternative 6 above). We vary the number of regions in each camera from 2 to 7 and thus the total number of regions in the camera network varies from 12 to 42. The latency of the tree based approach, even for a modest lookahead of 2 s, is not real-time and significantly higher than our approximated approach which takes less than 2 ms for the largest case of 42 regions.

In Fig. 11(e) we plot the rate at which a high resolution image was collected but there was no entity in the region. We compute the ratio of high resolution images with a face over the total number of high resolution images collected. We have defined the rate to be 1 if no high resolution images were collected. This experiment captures the need for a low rate of "false-positives". Since the captured observations are later processed by a heavy operator (e.g., face recognition), we would like to minimize the number of frames that it has to process. The lookahead approach both collects a larger total of observations of interest to the application and reduces the number of false positives as compared to the other alternatives.

[6] One day was used for training the model, and the other for evaluating the different schedulers.

[7] In the expectation tree lookahead should be optimal for all values of β. However, we evaluate our scheduler on a large but finite set of data. This causes small fluctuations.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014



(a) Latency—"optimal solution" vs. "approximate solution".

(b) Latency of scheduling as a function of number of cameras.

Fig. 12. Comparing scheduling latencies: approximated POMDP vs. tree approach.

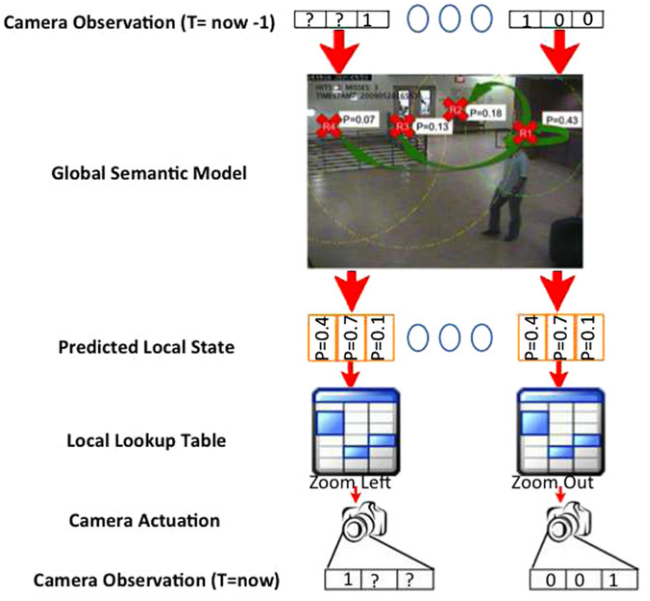


Fig. 13. Real-time scheduling using global semantics and local lookup tables.

Our approximated approach is comparable to the tree based approach and both lookahead alternatives vastly dominate other alternatives. As opposed to the tree based approach, our proposed approach is suitable for a sensor actuation setting, in which decision making has to be done in real-time latencies.

Multiple pervasive computing applications: To evaluate performance in the presence of multiple applications, we have created 11 scenarios in which there are 10 applications running in parallel each with a different reward function, as follows: (1) 10 apps with ($\alpha = 1, \beta = 0$). (2) 9 apps ($\alpha = 1, \beta = 0$) and 1 with ($\alpha = 1, \beta = 1$), and so on until scenario (11) in which we have 10 apps with ($\alpha = 1, \beta = 1$). The 11 cases are optimally solved by the following policies: (1) $\alpha = 1, \beta = 0$. (2) $\alpha = 1, \beta = 0.1$, and so on until (11) $\alpha = 1, \beta = 1$. This enables us to compare two alternatives: Alternative 1: solve using a single optimal lookahead table for all apps (e.g., 9 apps with $\alpha = 1, \beta = 0$ and 1 app with $\alpha = 1, \beta = 1$ solved optimally with $\alpha = 1, \beta = 0.1$). Alternative 2: as described by Algorithm 2, approximate the solution by using the individual lookahead tables of each app (e.g., 9 tables with $\alpha = 1, \beta = 0$ and 1 table with $\alpha = 1, \beta = 1$) and choose the action with the maximal expected utility. These 11 application reward values have been selected so that we could easily compare our approximation to a single optimal solution, in this case, alternative 1. In Fig. 11(d) we present our results. The results show that our approach is very close to the optimal solution, and thus validates that an approximated approach can be used to schedule multiple applications at the same time. Notice, that in the general case of the problem, simply reducing to a single lookahead table may not be possible and thus a principled approach to combine multiple applications is required.

Scalability in a large network: To examine performance as a function of network size, we have simulated a network of $K = 10$–100 cameras, monitoring a (very) long hallway. The figure below illustrates the case for $K = 2$ cameras.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

Each camera has 3 zoom regions and people walk in a single direction from left to right. We have simulated the walk trajectories generated by 10 people who arrive at the entry point of the pipeline according to a Gaussian distribution with mean of 100 s and standard deviation of 25 s. Each person walks all the way to the end of the hallway at their own walk speed, specified as the time it takes the person to walk from region to region, which is also a Gaussian with mean of 4 s and variance of 2 s. Our scheduling algorithm automatically learnt the transition probabilities from a generated script used as training data. We have experimented with an application utility interested in high resolution facial images ($\beta = 0, \alpha = 1$). We plot the latency of reaching a scheduling decision as a function of K in Fig. 12(b) and the percentage of recalled faces in Fig. 11(c). The latency for a network of $K = 100$ cameras is less than 30 ms and linearly increasing with the size of the camera network, which makes the scheduler a good fit for real-time applications. The recall of the faces remains high as the number of cameras increases. In the simulated results the recall of events is around 70% due to the fact that in about 1 of 3 cases the person will start moving and the scheduler will miss a single frame until it catches up in one of the following zoom regions.

5. Related work

This work is an extension of [14]. We have extended the introduction, problem statement, modeling, and our approach sections. We have added examples and figures that better illustrate the different alternatives and their relative complexity. Furthermore, in this version we address the problem of overlapping regions, propose a tractable algorithm based on our sensor actuation framework and present encouraging experimentation results. Finally, we have added a section describing the technical design we used to implement and evaluate our algorithms and run the experiments presented in this paper.

The most relevant theory to our work is on the tractability of POMDP solutions [15]. A number of exact value iteration algorithms have been proposed [6]. None of these alternatives can be used in our context since they are limited to a very small state space. For our needs we are not interested in finding the exact solution. Our approach avoids the exponential graph in state space by approximating the scheduler belief space using a single probability value for each region of the space, instead of a probability value for each state of the world. The difference is linear (in our case) vs exponential. Second, policy search methods have also been used to optimize POMDP solutions [16], their strength lies in the fact that they restrict their optimization to certain belief states (e.g., reachable states) [17–20]. Much like the value iteration techniques, policy search methods require the belief state to be fully represented and for any reasonably-sized camera network the belief state space is too large. Third, approximate value iteration algorithms were presented. These techniques consist mainly of grid-based methods [21–23]. They can solve larger problems (90 states [17]) by updating only values at discrete grid points. Our approach exploits the specific characteristics of sensor actuation problems that makes our approach practical for this problem setting. In our case, we cannot even represent the underlying MDP, which is required for the existing grid based methods. A dimensionality reduction technique for the belief space of the POMDP is proposed by [24]. Our approach can be seen as a dimensionality reduction with the advantage of preserving the additive property of the reward function which is used to estimate the error of our approximation and update our grid of represented belief states. [25] suggest a Monte-Carlo approximation for large scale POMDPs. They construct, online, a search tree of histories, and are thus not real-time. [26] focus on factored representations for discrete-state MDPs, while we use a POMDP.

A significant body of relevant work has recently emerged under the term "Network Distributed POMDP" [27,28]. ND-POMDP literature addresses the challenges that arise when two agents need to work in coordination. As an example, consider an application which is only interested in two frontal images of the same person from two different views at the same time (e.g., to reconstruct a 3D image). ND-POMDP can be used to find an optimal solution for this case. The state of the system as well as the system transition function is represented explicitly which renders DP-POMDP not tractable for the size of the problems that our approach addresses. As part of future work we are interested in addressing cameras with overlapping regions and joint camera reward functions. Applying techniques from ND-POMDP together with our approximation techniques seems like a promising direction.

To the best of our knowledge, we are the first to address the problem of actuating a large scale camera network based on unscripted human activities. We accomplish this by using a semantic model of building activities and approximating the expected utility of actions. As opposed to problem formulation in which sensors are selected given processing and resource constraints [9] we focused on sensor actuation wherein the physical constraints of the sensor prevents capture and propose a general framework for sensor actuation.

Sensor actuation has been previously studied in specific settings. E.g., in the context of *people and object tracking* there exists a large body of work, we point the interested reader to the following surveys [29,30]. The most relevant work is by [31] who suggests a POMDP approximation approach to address a single target tracking using a sensor network, where the goal is to conserve sensor battery life by querying only sensors that are likely to improve the location estimate of the target. Although the problem domain and formulation are similar, our approach differs from theirs in several key aspects: first, the scale of the problem in our case is different. [31] assumes that there is a single target in the system which makes the state space linear with the number of regions. In our case we track multiple targets, not just one, and address the exponential nature of the state of system as well as the state transition function. Second, [31] assumes that people walk according to a linear Gaussian model, while our formulation learns it directly. Third, in our case the policy is computed offline and actions are selected by an efficient lookup in a state-action map. In their case, the utility of action needs to be computed at real-time which makes the scheduler less agile and less likely to meet real-time deadlines, even for a single target.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

In the context of sensor networks other ways for modeling the problem were proposed, for example for energy efficient data collection from sensor networks, recent work by [32] applies a Q-learning [33] technique to allow each sensor to self schedule its tasks and allocate its resources by learning their utility in any given state. The main advantage of Q-learning is that it does not require a model of the environment. In our case, we would like to utilize the motion characteristics of our monitored space. Furthermore, due to the size of the state space we are prevented from learning the utility of different states individually.

In the context of multimedia applications, [34] suggest a scheduler for multimedia applications which supports applications with time constraints. It adjusts the allocation of resources dynamically and automatically scheduling real-time tasks and regulating their execution rates when the system is overloaded. This approach is similar in its motivation to our work, however, we utilize a semantic model and reason about the expected utility of actions.

In the context of video surveillance, [35] propose a new approach known as experimental sampling which carries out analysis on the environment and selects data of interest while discarding the irrelevant data. In the case of multiple surveillance cameras, the paper assumes that all video frames are can be collected for processing at any given time. The motivation is similar as out of the set of available images, the most important ones need to be selected. However, our problem is fundamentally different since we address the problem given physical constraints which prevent the system from accessing all video frames.

[36,37] address the problem of tracking when the estimates of location are not accurate, i.e., some of the generated estimates are erroneous due to the inaccuracy of the capturing sensor. The task in both papers was to reconstruct the actual trajectory of the object being tracked with statistical models, such as Kalman Filters by [38] which are utilized to find the most probable trajectory. [39] addresses the problem of being able to associate two images from two different cameras to the same person. The approach that they are using is to exploit semantic information to address the problem of inaccuracy feature based tracking algorithms that try to associate subjects seen at different cameras.

There is a large body of work on *schedule optimization* within the sensor modeling literature, see [40–43]. A common approach is to build a probabilistic model of object state and probe sensors that reduce the entropy, or uncertainly in state, at the next time frame. Our application is different in that, rather than reducing the uncertainty in building state, we want to collect images of events (even if, and indeed especially if, we are certain of their occurrence) for further high-level processing or forensic purposes. Direct modeling of the system state is difficult in our case because a large number of people move, interact, and enter/exit, making data association and tracking extremely challenging. We take a different approach and directly model the *environment* itself using a simple, but effective, sensor-correlation model.

Unlike these previous papers, this paper takes a different view. Specifically, it addresses the need to reason about the long term effects of actions by modeling the problem as a partially observable Markov decision process, and model the environment as an MDP. POMDP has been also studied extensively in the literature, but the scale of the problems that it can solve is orders of magnitude smaller than the problem we have at hand.

6. Conclusions and future work

We have developed a POMDP based framework to actuate sensors in a large scale sensor network. POMDP provides an elegant way of representing and reasoning about the partial knowledge about the environment. However, modeling the scheduling problem in sensor networks using POMDP introduces a scalability challenge as the state space of the environment is significantly larger than typical problems with similar formulations. The heuristics and approximations that we followed made our scheduler tractable. The real-time component is extremely lightweight and fast, and, most importantly there was significant improvement over alternative approaches. While our approach was evaluated in the context of camera based systems, the models that we have developed and the way that we use these models is much more general. For example, a very different type of sensor enabled application wherein loop sensors are used to measure traffic flows on various freeways/road networks, and the data captured is used to build applications such as route planning, traffic jam determination, etc. We can apply the techniques described in this paper as follows: The phenomena that we monitor would be the traffic flow at each sensor discretized to a finite set of flow states, e.g., average speed rounded to the nearest value in 10 miles per hour intervals. The phenomena semantics capture the nature in which traffic flows are correlated between different locations. The techniques that we have developed apply to this setting and can be used in order to schedule the data collection from the different sensors in order to detect an event of interest—slow traffic flow (i.e., speed of less than 10 miles per hour). The data collection process would increase the number of slow traffic flow events collected. Furthermore, we can reason about the expected effects of actions such as controlling the delay in traffic light signals located at entry points to highways and actuate the traffic light to reduce the chance of a traffic jam.

As part of future work we would also like to consider a hybrid push/pull approach that will benefit from the advantages of computation at the sensors while meeting the deadlines of a real-time system and supporting the dynamic nature of a surveillance task.

References

[1] Daniel Massaguer, Sharad Mehrotra, Ronen Vaisenberg, Nalini Venkatasubramanian, Satware: a semantic approach for building sentient spaces, in: Distributed Video Sensor Networks, Springer, 2011, pp. 389–402.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014

[2] Havard Rue, Leonhard Held, Gaussian Markov Random Fields: Theory and Applications, CRC Press, 2005.
[3] Jyotiprasad Medhi, Stochastic Processes, Wiley, Eastern New Delhi, 1982.
[4] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley & Sons, Inc., 1994.
[5] K.P. Murphy, A survey of POMDP solution techniques, Tech Report, 2000, p. 12.
[6] A.R. Cassandra, L.P. Kaelbling, M.L. Littman, Acting optimally in partially observable stochastic domains, in: Proceedings of the National Conference on Artificial Intelligence, 1995, pp. 1023–1023.
[7] W. Zhao, R. Chellappa, P.J. Phillips, A. Rosenfeld, Face recognition: a literature survey, ACM Comput. Surv. 35 (4) (2003) 399–458.
[8] G. Shani, R. Brafman, S. Shimony, Prioritizing point-based POMDP solvers, in: Machine Learning: ECML 2006, 2006, pp. 389–400.
[9] R. Vaisenberg, S. Mehrotra, D. Ramanan, Exploiting semantics for scheduling data collection from sensors on real-time to maximize event detection, in: Proceedings of SPIE, SPIE, 2009.
[10] Kevin P. Murphy, Yair Weiss, The factored frontier algorithm for approximate inference in DBNs, in: UAI'01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001, pp. 378–385.
[11] Lawrence R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[12] J.L. Bentley, K-d trees for semidynamic point sets, in: Proceedings of the Sixth Annual Symposium on Computational Geometry, ACM, 1990, pp. 187–197.
[13] A.W. Moore, An introductory tutorial on kd-trees, Extract from Andrew Moore's Ph.D. Thesis: Efficient Memory based Learning for Robot Control, 1991.
[14] Ronen Vaisenberg, Alessio Della Motta, Sharad Mehrotra, Deva Ramanan, Scheduling sensors for monitoring sentient spaces using an approximate POMDp policy, in: PerCom, 2013, pp. 141–150.
[15] P. Poupart, Exploiting structure to efficiently solve large scale partially observable Markov decision processes, Ph.D. Thesis, University of Toronto, 2005.
[16] A.Y. Ng, M. Jordan, Pegasus: a policy search method for large MDPs and POMDPs, in: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, 2000, pp. 406–415.
[17] J. Pineau, G. Gordon, S. Thrun, Point-based value iteration: an anytime algorithm for POMDPs, in: International Joint Conference on Artificial Intelligence, Volume 18, 2003, pp. 1025–1032.
[18] J. Pineau, G. Gordon, S. Thrun, Anytime point-based approximations for large POMDPs, J. Artificial Intelligence Res. 27 (1) (2006) 335–380.
[19] M.T.J. Spaan, N. Vlassis, Perseus: randomized point-based value iteration for POMDPs, J. Artificial Intelligence Res. 24 (1) (2005) 195–220.
[20] T. Smith, R. Simmons, Heuristic search value iteration for POMDPs, in: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2004, pp. 520–527.
[21] W.S. Lovejoy, Computationally feasible bounds for partially observed Markov decision processes, Oper. Res. (1991) 162–175.
[22] R.I. Brafman, A heuristic variable grid solution method for POMDPs, in: Proceedings of the National Conference on Artificial Intelligence, 1997, pp 727–733.
[23] G. Shani, R.I. Brafman, S.E. Shimony, Forward search value iteration for POMDPs, in: Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI, 2007.
[24] N. Roy, G.J. Gordon, S. Thrun, Finding approximate POMDP solutions through belief compression, J. Artificial Intelligence Res. (JAIR) 23 (2005) 1–40.
[25] D. Silver, J. Veness, Monte-Carlo planning in large POMDPs, in: Advances in Neural Information Processing Systems (NIPS), 2010.
[26] R. Givan, T. Dean, M. Greig, Equivalence notions and model minimization in Markov decision processes, Artif. Intell. 147 (1–2) (2003) 163–223.
[27] R. Nair, P. Varakantham, M. Tambe, M. Yokoo, Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs, Volume 20, AAAI Press, 2005, p. 133.
[28] P. Varakantham, J. Marecki, Y. Yabu, M. Tambe, M. Yokoo, Letting loose a spider on a network of POMDPs: generating quality guaranteed policies, in: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, ACM, 2007, pp. 1–8.
[29] A. Yilmaz, O. Javed, M. Shah, Object tracking: a survey, ACM Comput. Surv. (CSUR) 38 (4) (2006) 13.
[30] D.A. Forsyth, O. Arikan, L. Ikemoto, J. O'Brien, D. Ramanan, Computational studies of human motion: part 1, tracking and motion synthesis, Found. Trends Comput. Graph. Vis. 1 (2–3) (2005) 77–254.
[31] J.L. Williams, J.W. Fisher, A.S. Willsky, Approximate dynamic programming for communication-constrained sensor network management, IEEE Trans. Signal Process. 55 (8) (2007) 4300–4311.
[32] M. Di Francesco, K. Shah, M. Kumar, G. Anastasi, An adaptive strategy for energy-efficient data collection in sparse wireless sensor networks, in: Wireless Sensor Networks, 2010, pp. 322–337.
[33] C.J.C.H. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3) (1992) 279–292.
[34] J. Nieh, M.S. Lam, The design, implementation and evaluation of SMART: a scheduler for multimedia applications, in: Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles, ACM, New York, NY, USA, 1997, pp. 184–197.
[35] P. Anandathirtha, K.R. Ramakrishnan, S.K. Raja, M.S. Kankanhalli, Experiential sampling for object detection in video, in: Multimedia Content Analysis: Theory and Applications, 2009, p. 175.
[36] D. Schulz, D. Fox, J. Hightower, People tracking with anonymous and id-sensors using Rao-Blackwellised particle filters, in: Proc. of the International Joint Conference on Artificial Intelligence, IJCAI, 2003.
[37] E. Daeipour, Y. Bar-Shalom, An interacting multiple model approach for target tracking with glint noise, IEEE Trans. Aerosp. Electron. Syst. 31 (2) (1995) 706–715.
[38] R.G. Brown, P.Y.C. Hwang, Introduction to Random Signals and Applied Kalman Filtering, Wiley, New York, 1997.
[39] B. Song, A.K. Roy-Chowdhury, Stochastic adaptive tracking in a camera network, in: IEEE 11th International Conference on Computer Vision, 2007, ICCV 2007, 2007, pp. 1–8.
[40] J.L. Williams, J.W. Fisher III, A.S. Willsky, An approximate dynamic programming approach to a communication constrained sensor management problem, in: Proc. Eighth International Conference of Information Fusion, 2005.
[41] V. Isler, R. Bajcsy, The sensor selection problem for bounded uncertainty sensing models, in: Fourth International Symposium on Information Processing in Sensor Networks, 2005, IPSN 2005, 2005, pp. 151–158.
[42] V. Krishnamurthy, Algorithms for optimal scheduling and management of hidden Markov model sensors, IEEE Trans. Signal Process. 50 (6) (2002) 1382–1397. [see also Acoustics, Speech, and Signal Processing, IEEE Transactions on].
[43] V. Gupta, T.H. Chung, B. Hassibi, R.M. Murray, On a stochastic sensor selection algorithm with applications in sensor scheduling and sensor coverage, Automatica 42 (2) (2006) 251–260.

Please cite this article in press as: R. Vaisenberg, et al., Scheduling sensors for monitoring sentient spaces using an approximate POMDP policy, Pervasive and Mobile Computing (2013), http://dx.doi.org/10.1016/j.pmcj.2013.10.014