2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

Solving the Continuous Time Multiagent Patrol Problem

Jean-Samuel Marier, Camille Besse and Brahim Chaib-draa

Abstract— **This paper compares two algorithms to solve a multiagent patrol problem with uncertain durations. The first algorithm is reactive and allows adaptive and robust behavior, while the second one uses planning to maximize long-term information retrieval. Experiments suggest that on the considered instances, using a reactive and local coordination algorithm performs almost as well as planning for long-term, while using much less computation time.**

Keywords: Multiagent, UAV, Online, Patrol.

I. Introduction

Unmanned Aerial Vehicles (UAVs) are a promising technology for many information gathering applications. As these devices become cheaper, more robust and have increased autonomy, we can expect to see them used in many different new applications such as surveillance and patrol missions. In such missions, the status of some sites must be monitored for events. If an UAV must be close from a location to monitor it correctly and the number of UAV does not allow covering each site simultaneously, a path planning problem arises: how should the agents visit the locations in order to make sure that the information about all locations is as accurate as possible?

In the last decade, several patrolling algorithms have been developed to this end. For instance, Santana *et al.* [7] proposed a graph patrolling formulation on which agents use reinforcement learning on a particular Markov Decision Process (MDP). They defined this MDP over a countably infinite state space, leading to long and costly computations. Their approach also assumes that agents communicate by leaving messages on the nodes of the graph, leading to unrealistic communication models. On the other hand, reactive algorithms such as the ant-colony approach from [4] have been shown to perform well in theory as well in practice. Such an approach also relies on simplistic communication models relying on so-called pheromones. The task of patrolling in an adversarial setting is studied in [1].

For the case where all locations are equally important, Chevaleyre [2] proved that the shortest Hamiltonian circuit is an optimal solution for a single agent. Moreover, some grid-based patrolling algorithms have also been explored [3] that give guarantees on the robustness in the event of UAV failures. However, as some locations may be more important than others, not visiting the less important ones from time to time might be advantageous.

In this paper, we use a model for the patrolling problem that was introduced in [5]. This model is focused on the *non-adversarial* setting. Contrary to previous approaches that tried to minimize the idleness of the vertices of the graph, it outlines the information retrieval aspect of the patrolling problem and the decay of the information value as the age of the information retrieved increases, thus forcing the agent to update its knowledge about nodes as frequently as possible. Our model has a continuous-time formulation, allowing for real-valued durations, asynchronicity and scenarios where durations are uncertain.

The focus of this paper is to evaluate algorithms that solve the patrolling problem under this formulation. Two algorithms are presented. A first reactive one that allows adaptive and robust behavior in case of UAV failure, online graph modification or any other non-stationarity in the model, and a second that tries instead to maximize long-term expected information retrieved from the nodes of the graph.

The remainder of this paper is structured as follows: section II presents the model used, section III presents the multiagent MDP framework and a formulation of the problem at hand, algorithms for solving the problem are presented in section IV, and section V presents experiments. Finally sections VI and VII discuss results and future work.

II. Problem Formulation

The patrolling problem has a graphical structure. Let us use V for the vertex set of that graph and E for its edge set. Let L be an $|V| \times |V|$ matrix, in which L_{ij} is a real number that represents the time required to go travel from i to j if $[i, j] \in E$ and is infinite otherwise. More generally, L_{ij} is a probability distribution. Each vertex i has a real *non-negative* importance weight, noted w_i. We note $\mathbf{w}$ the vector of all such weights. The graph embeds structure in the problem. If there is no structure at all, the graph is complete.

In patrolling literature such as [4], idleness is used as a performance measure. The idleness of vertex i, noted τ_i represents the time since the last visit of an agent to that vertex. The idleness is 0 if and only if an agent is currently at vertex i and $\tau_i^{t+\Delta t} = \tau_i^t + \Delta t$ if there are no visits to i in the time interval $(t, t + \Delta t)$. Because idleness is an unbounded quantity, a more suitable quantity is $k_i^t = b^{\tau_i^t}$, with $0 < b < 1$. We call this *freshness*. Since k_i^t is always in $[0, 1]$, it can be seen as the expected value of a Bernoulli random variable which has value 1 if vertex i is observed correctly and 0 otherwise. Thus, k_i^t is the probability that this random variable is 1 at time t. This idea of "correctness" is fairly general and means that agents have observed what they should have. At time t, k_i^t is the probability that the state of vertex i is known. The performance measure is a weighted (by $\mathbf{w}$) sum of $\mathbf{k}^t$.

All authors are with Department of Computer Science and Software Engineering, Laval University, Quebec, Canada. {marier,besse,chaib}@damas.ift.ulaval.ca

The probability evolves as $k_i^{t+\Delta t} = k_i^t b^{\Delta t}$ if there is no visit to i during time interval $(t, t+\Delta t)$. If an agent with noisy observations visits i at time t, idleness becomes 0 with probability $(1-a)$ satisfying $b < (1-a) \leq 1$. If n agents visit vertex i at time $t+\Delta t$ and that there is no visit since time t:

$$k_i^{t+\Delta t} = k_i^t a^n b^{\Delta t} + 1 - a^n. \quad (1)$$

To sum up, an instance of the patrolling problem is a tuple $\langle L, \mathbf{w}, a, b\rangle$, consisting respectively of the matrix L of edge lengths, the vector $\mathbf{w}$ of importance weights and parameters a (the probability that the idleness does not become 0 when an agent visits a vertex) and b (the rate at which k_i decays over time). Note that b could be different from one vertex to another and that a could be different for every agent-vertex pair. Throughout, we use a single a and a single b for clarity.

III. Multiagent Markov Decision Processes

This section casts the previous problem as a Multiagent MDP (MMDP). We assume that the problem state is fully observable, i.e. every agent has the same complete information to make its decision. Such problems are called MMDPs. The actions of each agent have a concurrent effect on the the environment and they are of different durations. Concurrency in decision processes is conveniently modeled with a Generalized Semi-Markov Decision Process (GSMDP), introduced by Younes *et al.* [8]. Such decision processes also generalize MMDPs to continuous-time with asynchronous events.

The state variables for this problem describe the position of each agent and the freshness of each vertex (as per equation (1)). If the total number of agents is N, the state space is

$$\mathcal{S} = V^N \times [0,1]^{|V|}. \quad (2)$$

Given some state $s = (\mathbf{v}, \mathbf{k}) \in \mathcal{S}$, v_i is the position of agent i and k_i is the idleness of vertex i. We use $s^t = (\mathbf{v}^t, \mathbf{k}^t)$ for the state and its components at time t.

At various time points, called decision epochs, the agents must choose an action. Every agent must be performing exactly one action at any given time. Furthermore, an agent may only change its action at its decision epoch. The actions available to an agent depend on the structure of the graph and its position: if an agent is at vertex v, it chooses its action from $\mathcal{A}_v = \{u : [v,u] \in E\}$. If an agent chooses action u from vertex v at time t^i, the next decision epoch for that agent occurs at time $t^{i+1} = t^i + L_{vu}$, and $v^t = v$ while $t \in [t^i, t^{i+1})$ and $v^t = u$ as soon as $t = t^{i+1}$. If L_{vu} is a probability distribution, so is t^{i+1} given t^i.

The problem is concurrent because the decision epochs of all agents can be interleaved arbitrarily. (Each component k_i of $\mathbf{k}$ evolves independently.) Equation (1) was defined in terms of two time epochs (t and $t+\Delta t$) and the number of agents (n). Let $\{t^j\}_j$ be the non-decreasing sequence of decision epochs and write n_i^j for the number of agents arriving at vertex i at time t^j. To simplify notation, let $\Delta t^j = t^{j+1} - t^j$. We thus have that

$$k_i^{t^{j+1}} = k_i^{t^j} a^{n_i^{j+1}} b^{\Delta t^j} + 1 - a^{n_i^{j+1}}.$$

The reward process R is defined in terms of $\mathbf{k}$. Specifically, the rate at which reward is gained is given by

$$dR = \mathbf{w}^\top \mathbf{k}^t \, dt. \quad (3)$$

The discounted value function for a GSMDP is defined by Younes *et al.* [8]. In our problem, it becomes:

$$V^\pi(s) \underset{(a)}{=} \mathbb{E}\left[\int_0^\infty \gamma^t dR\right] \underset{(b)}{=} \mathbb{E}\left[\sum_{j=0}^{\infty} \gamma^{t^j} \mathbf{w}^\top \mathbf{k}^{t^j} \frac{(b\gamma)^{\Delta t^j} - 1}{\ln(b\gamma)}\right] \quad (4)$$

where $\gamma \in (0,1]$ is the discount factor. Equality (a) is the definition of the continuous-time discounted value function and (b) is obtained by piecewise integration between decision epochs. The expectation is taken over action durations. The sequence of states and decision epochs encountered depends on the actions chosen by the agents, this is generally called a policy and is denoted by π. The problem is to chose policies for all agents that maximize this expectation.

A. Properties of the Value Function for the Patrol Problem

The value function defined in equation (4) has some properties that general decision problems do not have. All these properties are derived from equations (1) and (4). They generalize to more than two agents. We use $V^{\pi_1+\pi_2}(s)$ to denote the value of having agents 1 and 2 performing π_1 and π_2 jointly starting from state s.

Property 1: Non-negativity: $0 \leq V^\pi(s)$, for all π and s.

Property 2: Subadditivity: for all π_1, π_2 and s,

$$V^{\pi_1+\pi_2}(s) \leq V^{\pi_1}(s) + V^{\pi_2}(s).$$

Two agents in the same environment will not do as good as the sum of two agents in two identical environments. Equality is achieved when agents patrol disjoint sets of vertices. If at least one vertex is visited by both agents, the inequality holds. The value is not be impacted much if the time between the visits of two agents is large.

Property 3: Locality: $V^{\pi_1+\pi_2} \approx V^{\pi_1}(s) + V^{\pi_2}(s)$ if the time between the visits of the two agents is large.

Property 4: Lower bound: for all π_1, π_2 and s,

$$\max\{V^{\pi_1}(s), V^{\pi_2}(s)\} \leq V^{\pi_1+\pi_2}(s).$$

Two agents will never do worse than one agent. The worst case is attained when both agents visit the same vertices at exactly the same time. The reward gained is still greater than the reward gained by one agent.

IV. Solving the Patrolling Problem

While good performance with regard to the value function described in (4) is a desirable property, robustness and low computational cost are also interesting properties. For instance, consider the case of a fleet of UAVs patrolling an area. Robustness is important to deal with contingencies, such as vehicle failures, or the online reconfiguration of the patrol mission. Low computational costs are important to allow for cheaper hardware and lower energy consumption.

Online algorithms [6] are a class of algorithms that solve the problem at hand only for the current state. This is in contrast with offline algorithms that do so for all states. Solving for the current state (and a small number of other

states reachable from the current state) is typically easier than solving for all states. For this reason, we present online algorithms. Another useful property of algorithms is the so-called "anytime" property. An algorithm is anytime if firstly it can be stopped at any time (after initialization) and provide a useful result and secondly running it any longer does not degrade solution quality. This property is useful to meet planning deadlines.

A. A Reactive Algorithm

This algorithm relies on the use of augmented states $(s, \boldsymbol{\eta})$, where $\eta_i \in \mathbb{R}^+$ is the time remaining before the next decision epoch of agent i. At any time t, the next decision epoch happens at time $t + \min_i \{\eta_i\}$, at which point $\eta_i = 0, i \in \arg\min\{\eta_i\}$. When performing an action of duration $\Delta\ t$, the $\boldsymbol{\eta}$ component of the augmented state is updated as

$$\eta_i' = \tau(\eta_i, \Delta t) = \begin{cases} \Delta t & \text{if } \eta_i = 0, \\ \eta_i - \Delta t & \text{otherwise.} \end{cases} \quad (5)$$

While the agents never know the value of $\boldsymbol{\eta}$, these augmented states are nevertheless useful to describe algorithms. The available joint actions in an augmented state are

$$\prod_i \mathcal{A}(v_i, \eta_i), \text{ where } \mathcal{A}(v, \eta) = \begin{cases} \mathcal{A}_v & \text{if } \eta = 0, \\ \emptyset & \text{otherwise.} \end{cases}$$

Given an augmented state with possibly ongoing actions, algorithm 1 returns an action. The evaluation is made according to the immediate value of the reached state ($\mathbf{w}^\top \mathbf{k}$ at line 15). When an agent leaves a vertex, the value of η_i is taken to be the expected travel duration (in the loop at line 6). To account for the new action as well as for the currently ongoing actions, the state is updated until all the current actions have completed (line 12). The state update operation is denoted by $\tau(s, \Delta t)$. It leaves $\mathbf{v}$ unchanged and updates $\mathbf{k}$ and $\boldsymbol{\eta}$ according to equations (1) and (5).

The state update does not depend on the action, because the action being evaluated by EVALUATE had its effect included in the current state at line 8. Agents that have reached their destination are removed at line 11. If this was not done, agents would have to choose another action because they are at a decision epoch.

B. Anytime Error Minimization Search

The previous algorithm is myopic in the sense that it evaluates actions only up to a near horizon. It should be compared to an algorithm that looks further ahead. To this end, we propose a second algorithm that addresses this issue. It is based on Anytime Error Minimization Search (AEMS), an online algorithm introduced originally in [6] for Partially Observable Markov Decision Processes (POMDPs). It performs a heuristic search in the state space. The search proceeds using a typical branch and bound scheme. Since the exact long term expected value of any state is not exactly known, it is approximated using upper and lower bounds. AEMS guides the expansion of the search tree by greedily reducing the error on the estimated value of the root node.

Algorithm 1 REACTIVEALGORITHM

```
1: procedure REACTIVEALGORITHM(s)
2:     return arg max_{a∈A(s)} EVALUATE(s, a)
3: end procedure
4: procedure EVALUATE(s, a)
5:     (v, k, η) ← s
6:     for i such that η_i = 0 do
7:         Δt_i ← E[L_{v_i,a_i}]   ▷ Expected time for going from v_i to a_i.
8:         (v_i, η_i) ← (a_i, Δt_i)
9:     end for
10:    T ← max{η}
11:    while v, η are not empty vectors do
12:        (v, k, η) ← τ((v, k, η), min{η})
13:        Remove v_i, η_i from v, η for all i s.t. η_i = 0.
14:    end while
15:    return w^⊤ k
16: end procedure
```

While we are not in a POMDP setting, the greedy error reduction is useful. In our problem, actions have the same interpretation as in a partially observable setting. On the other hand, observations model the realizations of the stochastic travel durations. Let us recall briefly how AEMS works.

In AEMS, the error is defined using the upper bound and the lower bound on the value of some state. We have $L(s) \leq V(s) \leq U(s)$ where $V(s)$ is the actual value of s, and $L(s)$ and $U(s)$ are the lower and upper bounds respectively. Given some search tree $\mathbb{T}$, whose set of leaf nodes is noted $\mathcal{F}$, the bounds for the root node are estimated recursively according to

$$L(s) = \begin{cases} \hat{L}(s) & \text{if } s \in \mathcal{F} \\ L(s, \boldsymbol{a}) = \max_{\boldsymbol{a} \in \boldsymbol{\mathcal{A}}} R(s, \boldsymbol{a}) + \gamma L(\tau(s, \boldsymbol{a})) & \text{otherwise} \end{cases} \quad (6)$$

where $\tau(s, \mathbf{a})$ is the next state if action $\mathbf{a}$ is taken in state s. The upper bound, $U(s)$, is defined similarly and depends on $\hat{U}(s)$. $\hat{L}(s)$ and $\hat{U}(s)$ are problem-dependent heuristics such as those described in section IV-C.

An estimation of the error on the value of s is given by $\hat{e}(s) = U(s) - L(s)$. We are interested in expanding the state at the fringe of the search tree whose contribution to the error on the root node (s^0) is maximal. Since all states are not reachable with equal probability (depending on the policy), the contribution of any state s to the error on s^0 is approximated by:

$$\hat{E}(s^0, s^t, \mathbb{T}) = \gamma^t \Pr(h_0^t | s^0, \hat{\pi}) \hat{e}(s^t),$$

where t is the depth of s in $\mathbb{T}$, and $\Pr(h_0^t | s^0, \hat{\pi})$ denotes the probability of having history h_0^t (the sequence of joint actions and observations leading from s^0 to s^t), while following policy $\hat{\pi}$. The above term describes exactly the error whenever $\hat{\pi} = \pi^*$. We are now interested in $\Pr(h_0^t | s^0, \hat{\pi})$.

If $h_0^t = \boldsymbol{a}^0, \boldsymbol{o}^0, \boldsymbol{a}^1, \boldsymbol{o}^1, \ldots, \boldsymbol{a}^t, \boldsymbol{o}^t$, is the joint-action history for some sequence of states $s^0, s^1, \ldots s^t$, then

$$\Pr(h_0^t | s^0, \hat{\pi}) = \prod_{i=0}^{t} \Pr(\boldsymbol{a}^i = \hat{\pi}(s^i) | s^i) \Pr(\boldsymbol{o}^i | s^i, \boldsymbol{a}^i).$$

Since we do not know the optimal policy, a heuristic is:

$$\Pr(\boldsymbol{a}|s) = \begin{cases} 1 & \text{if } U(s,\boldsymbol{a}) = \max_{\boldsymbol{a}' \in \boldsymbol{\mathcal{A}}} U(s,\boldsymbol{a}') \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

Given a search tree $\mathbb{T}$, rooted at s^0, AEMS tells us that the next state to expand is

$$\tilde{s}(\mathbb{T}) = \arg\max_{s \in \mathcal{F}} \hat{E}(s, s^0, \mathbb{T}).$$

Each time a node $\tilde{s}$ is expanded, it is removed from $\mathcal{F}$, its children are added to $\mathcal{F}$ and the bounds of $\tilde{s}$ and its parents are updated. When an agent must choose an action, the action of maximum lower bound is chosen. This guarantees that in the limit of considering an infinitely large search tree, AEMS performs no worse than its lower bound.

C. Bounds for the Patrolling Problem

In order to use AEMS in the patrolling problem, we need lower ($\hat{L}(\cdot)$) and upper ($\hat{U}(\cdot)$) bounds for the value of states.

1) Lower Bound: A lower bound ($\hat{L}(\cdot)$) for the value of any state is the value of following any policy from that state. We use the reactive algorithm of section IV-A to generate a policy. The value is estimated by following this policy for finitely many decision epochs and assuming a zero reward from then to infinity. The accuracy depends on how far ahead in time the value is evaluated. From equation (4), the error in value incurred for stopping at time T (starting from time 0) is $\epsilon \leq -||\mathbf{w}||_1 \gamma^T / \ln \gamma$.

2) Upper Bound: An upper bound ($\hat{U}(\cdot)$) is usually obtained by relaxing problem constraints. We can thus upper-bound the value of a policy by assuming that agents are ubiquitous: they can be in more than one locations at the same time. Whenever an agent reaches a vertex, it instantaneously multiplies itself and starts heading to adjacent unvisited locations. This bound estimates the shortest time that a swarm of agents would take to cover the entire graph and estimates through the discount factor an upper bound on the maximum reward obtainable.

The upper bound is derived from equation (4). We assume that $k_i = 1$ as soon as an agent reaches i. For each vertex, this amounts to taking $a = 0$ and $b = 1$ when an agent reaches it. If the current time is 0 and the vertex i is first reached it at time T_i, an upper bound on the value is

$$V^+(\mathbf{k}^0, T) = \sum_i w_i \left[k_i^0 \frac{(b\gamma)^{T_i} - 1}{\ln b\gamma} - \frac{\gamma^{T_i}}{\ln \gamma} \right]. \quad (8)$$

D. Extension to Asynchronous Multiagent Setting

1) Extension to Continuous Time: In a discrete-time setting, the depth of a vertex is the number of observation edges between it and the root of the tree. This can be generalized by associating a positive real length to all such edges. (The discrete-time case is where all edges have unit length.) This allows representing distributions on the durations of the transitions: each observation edge corresponds to the duration of the action on the action edge that immediately precedes it in the tree. Since it is not possible to represent continuous probability distributions using this scheme, a suitable discretization of the durations must be used.

If $s' = \tau(s, o, a)$, equation (6) becomes

$$\begin{aligned} L(s,a) &= \max_{a \in A} R(a,s) \\ &+ \gamma^{-d(s,T)} \sum_{o \in O} \gamma^{d(s',T)} \Pr(o|s,a)\, L(s'). \end{aligned} \quad (9)$$

The upper bound $U(s,a)$ is changed similarly. These equations account for stochastic action duration when a realization of its duration is represented by o. These expectations do exactly the same thing as the original ones, i.e. taking the expectation with $o \sim \Pr(\cdot|s,a)$. The only difference is that $d(\tau(s,o,a), \mathbb{T}) - d(s, \mathbb{T}) \neq 1$, and that it depends on o.

2) Extension to Asynchronous Multiagent Setting: To extend to multiagent setting, it suffices to have a branch for each joint action and for each joint observation. This causes an exponential (in the number of agents) blowup in the branching factor of the tree. (Section IV-E attempts to mitigate this.) Asynchronicity is handled with state augmentation: different realizations of the durations have different augmented states. When a state must be expanded, actions and observations are added for agents for which $\eta = 0$.

E. Coordinating Agents

AEMS can be used to perform online planning for any subset of agents. However, it is unlikely that any agent can compute a joint policy, because the complexity is exponential in the number of agents. We thus coordinate agents *locally*. The notion of locality is motivated by property 3: agents interact more strongly when they may visit the same vertices within a small time interval. A total order among agents is defined. We say that an agent is greater than (resp. less than) another agent if it must choose its policy before (resp. after).

The agents compute their policy according to that order. Once an agent knows the policies of all greater agents, it proceeds to compute its policy, and then communicates it to the lesser agents. Whenever an agent selects its policy, it chooses the best policy given the policy of greater agents. This approach can be improved by using *altruistic* agents. Loosely speaking, such an agent does not consider only the value of its own policy. Instead it chooses the best sub-policy from a *small* set of joint policies: i.e. the joint policies for himself and a few lesser neighboring agents.

Algorithm 2 describes the procedure. Operator $<$ is the order among the agents. This order can change depending on time or on the problem state, but all agents must use the same order. The SOLVE(s, i, S, U) routine returns the best policy for agent i amongst the joint policies for agents $S \cup \{i\}$ starting from state s, given the policies of the agents in U. The contrast between policies of the agents in S and those in U is that the policies of the agents in U are considered to be fixed, whereas they are free variables for the agents in S. Parameter n controls how many lesser agents are considered at once by an agent choosing its policy.

There are $\binom{|L|}{n}$ ways to chose S in the loop at line 6. If each agent can choose among N policies, an agent only

has to consider $\binom{|L|}{n} N^{n+1}$ rather than $N^{|L|+1}$. In the case of the patrolling problem, the value of a policy found using algorithm 2 for some choice of n is never less than that found using $n-1$. It is so because of property 4 of section III-A.

This algorithm has the property that if the SOLVE routine is online and anytime, algorithm 2 also has these properties. The algorithm can be preempted at any time, but line 10 has to be executed nevertheless. To realize this, each agent has a deadline at which it must have chosen its policy. It is assumed that the communication mechanism is reliable (i.e. messages sent are received without modification and no messages are lost). Should an agent fail or a message be lost, its presence is ignored by its lesser agents.

We name C-AEMS (coordinated AEMS) this algorithm when it uses AEMS to implement its SOLVE sub-routine.

Algorithm 2 COORDINATIONBYPOLICIES

1: **procedure** COORDINATIONBYPOLICIES(i, s, n)
2: Let i be the unique identifier of the current agent.
3: Let $\hat{\pi}$ be a default policy.
4: $U \leftarrow \{j \mid i < j\}$
5: $L \leftarrow \{j \mid i > j\}$
6: **loop**
7: Choose $S \subseteq L$, such that $|S| = n$.
8: $\hat{\pi}_i \leftarrow$ SOLVE(s, i, S, U)
9: **end loop**
10: Communicate $\hat{\pi}_i$ and $\{\hat{\pi}_u \mid u \in U\}$ to agents of L.
11: **end procedure**

V. EXPERIMENTS

We compare the reactive and C-AEMS algorithms on many instances. There are two classes of experiments. The first class is on user defined instances. The second class of experiment is on random instances. Figure 1 shows the five instances of the first class that we used. The three small instances were chosen for their simplicity. The instances Map-A and Map-B are two classical instances of the patrolling literature [7]. All these instances were taken to have $c = 1$, $\mathbf{w} = \mathbf{1}$ and unit edges. We have used $\gamma = 0.95$ throughout. The initial state is $\mathbf{k} = \mathbf{1}$ for all experiments and the starting vertex is shown as filled circles in Figure 1.

In the random instances, both the vertex weights and the edge lengths were generated randomly. The vertex weights were sampled from a beta distribution with parameters $\alpha = \beta = 0.5$. The edge lengths were taken to be the length of the edges of a random graph. The graphs were constructed using Delaunay triangulation over a set of vertices sampled randomly on the unit square. The position of the vertices was generated using an iterative rejection sampling procedure that would resample the position of a vertex with high probability if was too close from another vertex. This procedure yields vertices that are more evenly spread around the unit square than uniform sampling. Figure 2 shows some examples of such randomly generated problem instances.

We have performed experiments with both deterministic or stochastic travel durations. The travel times for an edge of length μ were modeled as a non-negative truncated normal distribution of mean μ and variance $\sigma^2 = \nu\mu$. The degenerate case $\nu = 0$ shall mean deterministic durations. For C-AEMS, the distributions were discretized by splitting the mass in either one or three equal parts, and representing the mean of each with a point mass.

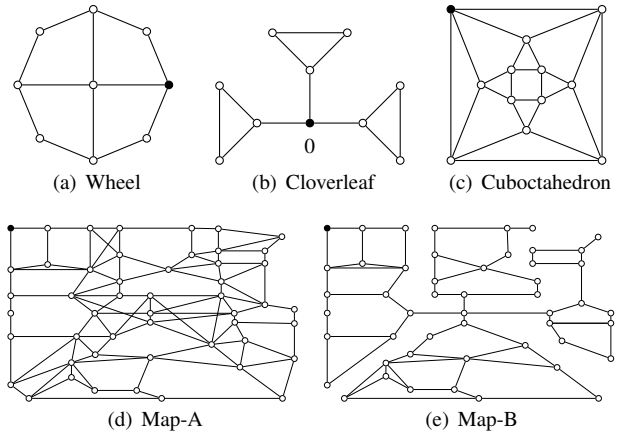

Fig. 1. Patrolling instances. Start vertex is filled, edges have unit length and vertices have unit weight unless otherwise noted.

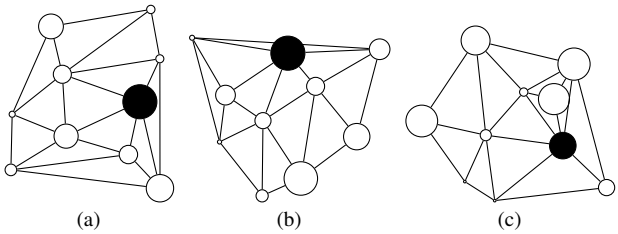

Fig. 2. Random instances. Start vertex is filled, edge length is proportional to travel time and vertex radius is proportional to the importance weight.

Results presented in Table I show the performance of the two algorithms for two agents. The steps column shows, for C-AEMS, the number of states that each agent was able to expand in the tree. This is typically determined by the time at which the algorithm is preempted to meet a planning deadline. The run time column shows the average CPU time used for each decision epoch. For the random instances, the results are the average for 10 instances.

VI. DISCUSSION

The experiments suggest that the two algorithms offer similar performance with regard to the objective that they are optimizing. In general, C-AEMS offers slightly superior performance. This was expected, because it is non-myopic and can escape local minima. For a small number of state expansions, AEMS does not perform as well as the reactive algorithm which is its lower bound. This is because the value is not well approximated when the search tree is small. The relatively good performance of the reactive algorithm is not surprising either, because it is in essence very similar to ant-colony approaches which are known to perform well [4]. In the case where the duration density was discretized as 3 point masses, the longer run time is explained by the increased branching factor. This also explains why, for the same

TABLE I
RESULTS FOR 2 AGENTS

	b	Steps	Total Reward	Average Run Time (s)
Wheel (50 time units)				
C-AEMS	0.9		163.5	$3.44 \cdot 10^{-1}$
Reactive			162.7	$7.66 \cdot 10^{-5}$
Clover (50 time units)				
C-AEMS	0.9		172.2	$2.68 \cdot 10^{-1}$
Reactive			171.9	$7.69 \cdot 10^{-5}$
Cuboctahedron (50 time units)				
C-AEMS	0.9		202.3	$1.39 \cdot 10^{0}$
Reactive			201.2	$9.06 \cdot 10^{-5}$
Map-A (200 time units)				
C-AEMS	0.95	200	5859.2	$2.13 \cdot 10^{1}$
Reactive			5788.5	$1.40 \cdot 10^{-4}$
Map-B (200 time units)				
C-AEMS	0.95	200	5608.5	$9.87 \cdot 10^{0}$
Reactive			5243.5	$1.11 \cdot 10^{-4}$
10 Random Instances ($\nu = 0$, 100 time units)				
C-AEMS	0.95	2	582.8	$5.08 \cdot 10^{-2}$
		5	583.2	$1.29 \cdot 10^{-1}$
		10	584.1	$2.68 \cdot 10^{-1}$
		25	584.9	$6.83 \cdot 10^{-1}$
		50	585.2	$1.40 \cdot 10^{0}$
Reactive			584.0	$4.70 \cdot 10^{-5}$
10 Random Instances ($\nu = 0.5$, one mass, 100 time units)				
C-AEMS	0.95	2	570.1	$5.29 \cdot 10^{-2}$
		5	571.0	$1.32 \cdot 10^{-1}$
		10	571.8	$2.40 \cdot 10^{-1}$
		25	572.9	$6.74 \cdot 10^{0}$
		50	573.6	$1.39 \cdot 10^{0}$
Reactive			574.1	$4.96 \cdot 10^{-5}$
10 Random Instances ($\nu = 0.5$, 3 masses, 100 time units)				
C-AEMS	0.95	2	570.6	$1.65 \cdot 10^{-1}$
		5	571.6	$3.94 \cdot 10^{-1}$
		10	571.8	$7.50 \cdot 10^{-1}$
		25	572.2	$2.11 \cdot 10^{0}$
		50	573.3	$4.29 \cdot 10^{0}$
Reactive			574.1	$4.96 \cdot 10^{-5}$

number of state expansions, the algorithm does not perform quite as well as it's counterpart without discretization.

On the other hand, the reactive algorithm and C-AEMS have their total run times differing by many orders of magnitude. It is interesting that a reactive algorithm, while much simpler can perform about as well on the selected instances. This suggests that on this model, there is not a great incentive to use a more computationally expensive algorithm such as C-AEMS. An important feature of the reactive algorithm is that since planning is quick, replanning is quick as well. This is important in practice, because the algorithm can be executed closer from the decision epoch.

Our experiments do not highlight very well how important fast replanning is. Typically, there is uncertainty about the time at which the agents will reach their destination. This uncertainty is reduced as the next decision epoch gets closer. If the replanning is long, such as in the case of C-AEMS, the initial state used for planning is likely to be less accurate. The planning algorithm can thus find a good policy for a state that no longer the current state. This plan is bound to be worse than expected if the state changes in the meantime. C-AEMS does not perform well in that respect. While it is an online anytime algorithm, the AEMS running on one agent must be preempted early enough to give the other agents enough time to compute their policy. Because the greatest agent has to preempt its planning process early, it is likely that it computes a plan for an inaccurate state. This inaccurate plan will then impact all the other agents as their policy depends on it. Because of this, it is expected that the performance of C-AEMS will not scale well to many agents.

AEMS spends much time evaluating its lower bound (evaluating the reactive policy up to a given horizon) even though our software implementation is efficient. We tried approximating the lower bound with Support Vector Machine regression. This offered similar performances (albeit much faster), the benefits will not scale to more agents. While the coordination algorithm we propose is a step in alleviating the complexity issues of multiagent planning, C-AEMS still has a complexity exponential in the number of observations (which depends on the particular discretization of duration outcomes) and exponential in the number of agents. Henceforth, while C-AEMS scales better in the number of agents than a joint AEMS would, it does not scale well to fine-grained discretizations of the travel durations.

VII. CONCLUSION

In this paper, we compared two online algorithms to solve a multiagent patrol problem with uncertain durations. Results suggest that the planning algorithm (AEMS) performs better than the reactive algorithm when its search space is small, such as the case where durations are deterministic. As the number of agents and duration discretizations increase, the time needed to run AEMS (and C-AEMS) also increases. In a setting where the time available to make each decision is limited, this also impacts the performance in value. Future work includes designing a multiagent approximate online planning algorithm that scales better with the number of agents and the number of discretizations in order to preserve the benefit of using a non-myopic algorithm.

REFERENCES

[1] N. Agmon, V. Sadov, G. A. Kaminka, and S. Kraus, "The impact of adversarial knowledge on adversarial planning in perimeter patrol," in *Proc. of AAMAS*, 2008, pp. 55–62.

[2] Y. Chevaleyre, F. Sempé, and G. Ramalho, "A theoretical analysis of multi-agent patrolling strategies," in *Proc. of AAMAS'04*, 2004.

[3] Y. Elmaliach, N. Agmon, and G. A. Kaminka, "Multi-robot area patrol under frequency constraints," in *Proc. of ICRA*, 2007.

[4] A. Glad, O. Simonin, O. Buffet, and F. Charpillet, "Theoretical study of ant-based algorithms for multi-agent patrolling," in *Proc. of ECAI*, 2008, pp. 626–630.

[5] J.-S. Marier, C. Besse, and B. Chaib-draa, "A Markov Model for Multiagent Patrolling in Continuous Time," in *Proc. of ICONIP*, 2009, pp. 648–656.

[6] S. Ross and B. Chaib-draa, "AEMS: An Anytime Online Search Algorithm for Approximate Policy Refinement in Large POMDPs," in *Proc. of IJCAI'07*, 2007, pp. 2592–2598.

[7] H. Santana, G. Ramalho, V. Corruble, and B. Ratitch, "Multi-agent patrolling with reinforcement learning," in *Proc. of AAMAS'04*, 2004.

[8] H. L. S. Younes and R. G. Simmons, "A formalism for stochastic decision processes with asynchronous events," in *Proc. of AAAI Work. on Learn. and Plan. in Markov Processes*, 2004, pp. 107–110.