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

Probabilistic Shadow Information Spaces

Jingjin Yu jyu18@uiuc.edu
Steven M. LaValle lavalle@uiuc.edu
Department of Computer Science
University of Illinois
Urbana, IL 61801 USA

Abstract— **This paper introduces a Bayesian filter that is specifically designed for counting targets that move outside of the field of view while performing a sensor sweep. Information space concepts are used to dramatically reduce the filter complexity so that information is processed only when the shadow region (all points invisible to the sensors) changes combinatorially or targets pass in and out of view. Previous work assumed perfect observations; however, this paper extends the approach to enable probabilistic disturbances. Practical algorithms are introduced, implemented, and demonstrated for computing the filter outputs based on realistic data.**

I. Introduction

Imagine a game of *hide-and-seek* (variations include *tag, tick, Cops and Robbers*) is being played. After the hiders conceal themselves (subsequent relocations are allowed), the seekers, usually having a map of the environment, start to search for the hiders. Most people who played the game as schoolchildren know that an effective search usually begins with the seekers checking places having high probabilities of containing a hider, from previous experience: a closet, an attic, a tall bush, and so on. After the most likely areas are exhausted, the next strategy is then to carry out a systematic search of the environment, possibly with some seekers guarding certain escape routes. Occasionally, during game play, some hiders may attempt to relocate themselves to avoid being found. While they succeed sometimes, they may end up being spotted by the seekers and instead getting found earlier.

Although a child's play, the hide-and-seek game requires fusion of previous experience and current observations to obtain estimation of the unobservable part of the world, upon which decisions can be made. The key ingredients of the game are common to a large, important class of problems in robotics including: 1) *counting* unpredictable people or robots that move in a complicated environment [1], [5], [14], 2) *pursuing* an elusive moving target by sweeping through a complicated environment to guarantee detection or capture [3], [6], [8], [11], 3) *tracking* movements of agents/targets to determine their possible locations [10], [12], [13], [16]. For many tasks [1], numerous scenarios are possibly depending on the types of robots, moving bodies, sensors, and environments that are given. However, all these problems involve reasoning about observations made while detectable bodies pass in and out of the field-of-view of moving sensors, with the help of cumulated statistical data (possibly gathered in the environment over a long period of time, similar to the cumulative experience in the hide-and-seek game). Just like estimating where the hiders are in the hide-and-seek game, we want an algorithm that maintains the distribution of the unobservable moving bodies in the environment.

In earlier work [15], we studied the nondeterministic version of the problem of tracking unpredictable agents moving in and out of view of moving sensors, presenting an efficient *combinatorial filter* solution to the *passive* problem of inferring the whereabouts of the agents based on the movements of the sensors. If desirable, one can easily turn the solution to the passive problem into a solution to an *active* problem of planning, such as sketching a strategy in hide-and-seek. This paper builds upon [15] to cover a more interesting and practical version of the problem, in which both agent distribution and field-of-view observations are assumed to be probabilistic. Inheriting the notion of *component event*s and *field-of-view event*s, which are critical to the analysis of the nondeterministic problem, we demonstrate how the agent distribution can be algorithmically and exactly propagated through the seven component and field-of-view event observations, given statistical data in the form of a *split rule* and *sensor statistics*. The Bayesian update based algorithm can be applied to cases with a manageable number of agents and events. When there are many agents and events such that the computation is overwhelmed, we approximate the exact algorithm with two heuristics, both *Monte Carlo* [9] in nature with the second also similar to *particle filtering* [2], to efficiently obtain the final agent distribution.

The contribution of the paper is twofold. Firstly, with the probabilistic extension, we offer a complete solution to the problem of reasoning about moving agents to the level of generality that, to the best of our knowledge, has not been attempted. With the assumption that *connected component*s are available, we allow the environment to be two or three dimensional, known or unknown, simply or multiply connected. We also allow the setting to be nondeterministic or probabilistic (for probabilistic version, we further assume that a couple of key statistics are available). Secondly, the probabilistic solution provides a general framework for treating a wide class of problems without tightly basing the solution on any specific distribution or statistical model, thus allowing great flexibility in adapting the framework to the

[1]For examples and additional detail, see full length paper at http://msl.cs.uiuc.edu/~jyu18/icra10/shadow.pdf

specifics of individual cases.

II. Problem Formulation

A. Free space, visible region and shadow region

Let k point robots move in a *world*, $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$. There may be obstacles in $\mathcal{W}$, which may move and/or change shape continuously, leaving $F_t \subset \mathcal{W}$ as the *free space*, which is an open set of possible robot positions that may evolve over time. The configuration for k robots is then $q_t \in F_t^k$ at time t.

The robots carry sensors that enable them to make observations in a subset of F_t. The sensors may assume various forms: They can be omni-directional infinite range sensors for robots carrying pan cameras, sensors with limited range for nodes in a sensor network, or fixed infrared beam sensors placed at passages in museums. Let $V(q_t)$ denote the closed joint *visible region* or *field-of-view* when the robots are in configuration q_t. Let $S(q_t) = F_t \setminus V(q_t)$ denote the *shadow region*. If the robots and obstacles move along a time-parameterized path over $[0, t_f]$, then the shadow region itself can be time-parameterized as well: $S(q_t)$ is obtained for every $t \in [0, t_f]$.

B. Shadow components and component events

It is assumed that $V(q_t)$ and $S(q_t)$ behave nicely as q_t varies continuously; more precisely, arbitrarily small changes in q_t incur small changes to $V(q_t)$ and $S(q_t)$, which can be formalized using the Hausdorff metric. This enables the *shadow component*s to be consistently labeled as q_t evolves over time. With the assumption, at any given $t \in [0, t_f]$, the shadow region $S(q_t)$ is composed of n (a finite number) shadow components, $s_1, \ldots, s_n$. As the system evolves, these disjoint, path-connected components will also evolve through one of four types of *component event*s: 1) Two components *merge*, 2) One component *split*s into two, 3) A new component *appear*s, and 4) A component *disappear*s. For an appear event, we assume that some agents (possibly zero) may move into the newly appeared component at the same time when the event happens, which is more general than the alternative assumption that a component appear event happens with zero agents in it. The same applies to the disappear event. The assumption is justified in practice: For example, a disappear event can be caused by a subset of the robots sweeping a shadow component to find out what is going on in the component; the sweep may report the exact number of agents or an agent distribution in that component. A general position assumption is assumed to avoid two tedious cases: 1) Four or more components are involved in a split or a merge, and 2) Two or more events happen at the same moment. A component retains its label unless it splits, merges, or disappears.

Components and component events are ubiquitous in robotics problems. One such scenario is that robots equipped with limited viewing angle, infinite range sensors move around in a known, simply connected polygonal environment[4]. Here, the component events are based on inflection and bitangent crossings. When the sensing range is limited, the system behaves more like a mobile sensor network. The components and component events for an ideal sensor network are illustrated in Figure 1. In reality, reliably detecting component events in wireless sensor networks is inherently hard due to unpredictable changes of the sensing nodes' sensing range. For example, cell phone signal strength at a fixed location can change due to weather and other environmental factors. Some systems of interest do not have component events at all, as when fixed beam sensors or security cameras are used. The shadow components remain fixed in these cases.

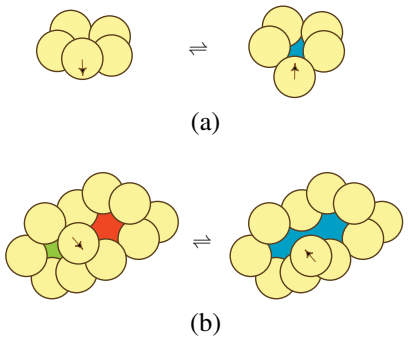

Fig. 1. Illustration of components and component events in an environment secured with watch lights (which also mimics an ideal mobile sensor network). The light cones' projections on the ground are approximated as yellow round discs. The arrow in the a disc indicate that light cone's moving direction: a) A component appears (left to right) and disappears (right to left). b) Components merge (left to right) and a component splits (right to left). Note that there is actually an unbounded component outside all the watch lights.

In this paper, it is assumed that the components and component events are either provided or can be efficiently obtained, with 100% accuracy. The assumption is made since the focus of the paper is to set up a common framework for a class of problems instead of covering every possible probabilistic variation. Modeling of imprecise component and component event observations will be addressed in future research.

C. Moving agents and field-of-view events

Let A denote a set of ℓ point agents. The speeds of the agents are irrelevant, provided that each agent moves along a not-necessarily known but continuous path in F_t. Let $r_t \in F_t^\ell$ be the vector that specifies the agent positions, the vector (q_t, r_t), their *state*, describes the positions of all robots and agents. As the agents move in and out of the sensors' field-of-view, they create the *field-of-view event*s. Component events and field-of-view events all together are called *critical event*s. For a given shadow component s_i, three field-of-view events are possible: 1) An agent *enter*s s_i from the field-of-view, 2) An agent *exit*s s_i into the field-of-view, and 3) Nothing happens at the boundaries between s_i and the field-of-view (for a period of time), or *null* event. Denoting these events e_e, e_x, e_n, respectively, the collection of possible field-of-view events for a component s_i is the set

$$E = \{e_e, e_x, e_n\}.$$

For each of these events, we assume that the sensors on the robots may correctly observe it or mistaken it for the other two events. For example, an enter event for a component may be reported by the sensor as enter, exit, or null event. That is, the sensor mapping is given by $h : E \to Y$, with Y being the set of *field-of-view observation*s

$$Y = \{y_e, y_x, y_n\},$$

in which y_e, y_x, and y_n are enter, exit, and null observations. Intuitively, the conditional probability of an observation given an event, $P(\mathbf{e} = e \mid \mathbf{y} = y), e \in E, y \in Y$, is affected by the sensor characteristics as well as the environment in which the observations are made. We assume that for a specific sensor and environment combination, the conditional probability distribution $P(\mathbf{e} \mid \mathbf{y})$ is known, possibly through data collection over a long period of time. As the field-of-view observation is concerned, some sensors may only generate the enter and exit observations explicitly, such as a sensing node in a sensor network that only senses agents passing through its sensing disc boundary. For detection beams, the field-of-view is a line segment, which causes two field-of-view events to happen consecutively (see Figure 2). Certain systems may not have field-of-view events at all; an instance is a pursuit evasion game in which the evader always avoids appearing in the pursuer's field-of-view. The game ends when an evader is found or when it is confirmed that no evader is in the environment.

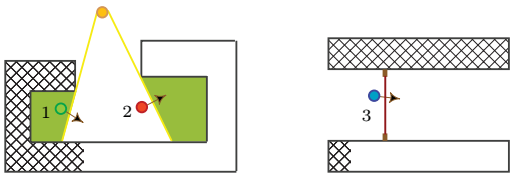

Fig. 2. Illustration of field-of-view events for an environment with obstructed visibility (left) and for an environment with detection beams (right). 1) An agent is about to exit a shadow into the field-of-view of the sensor (yellow dot). 2) An agent is about to enter a shadow from the sensor's field-of-view. 3) An agent is about to enter and exit the field-of-view of a beam sensor.

So far we have left undiscussed the agents in the sensors' field-of-view, as the sensors may or may not have an accurate count of these. We may view this information as a factor that affects $P(\mathbf{e} \mid \mathbf{y})$: A precise agent count in the field-of-view should improve the accuracy of field-of-view observations. We assume that agents are indistinguishable.

D. Tasks

Notation. We use s_i to denote the shadow component with label i, as well as the random variable for that component in the joint/multivariate distribution. For components $s_1, \ldots, s_n$, the joint distribution is then $P(s_1, \ldots, s_n)$, in which a specific entry is $P(s_1 = x_1, \ldots, s_n = x_n) \in [0, 1]$.

Given the initial joint probability distribution $P(s_1, \ldots, s_n)$ of agents in the n initial components plus the sequence of event observations that happen along the way, the main task of interest is to obtain the agent distribution in the m components existing at time $t = t_f$, $P(s'_1, \ldots, s'_m)$. The resulting joint probability distribution can then be used in solving a variety of decision making problems; for example, in a fire evacuation scenario, knowing the the expected number of people trapped in various parts of a building (possibly estimated through observations from infrared beam sensors or security cameras), firefighters can better decide which region of the building should be given priority when they look around. The expected number of people in each component can be easily obtained from the joint probability distribution.

III. Analysis of the Problem

A. Preliminary treatments of observation history

The distribution of agents in the shadow components does not change unless some critical event occurs. Therefore, we may safely ignore the period of time during which no critical events happen. As a first step in the analysis, we distill the critical event observations from the observation history. For some sensors, such as fixed beams, observations of the critical events are naturally stored in a sequential form, ready to be used. For others, additional computation may be required, as is the case with sensor networks: $V(q_t)$ and $S(q_t)$ must be continuously updated, from which the shadow components can be extracted and labeled, and the field-of-view observations can be associated with the labeled components thereafter. Thus, it is desirable to bring the observation data to a form that is succinct yet captures all the necessary information about critical events, regardless of the problem settings. To achieve this, we follow the *information space* formulation (see [7] for a general introduction) from the nondeterministic version of the problem [15], in which a time evolution of shadow components is captured. To account for field-of-view observations, we attach them to the components as they evolve. This preliminary treatment offers two advantages: 1) We obtain an abstract representation common to all problems involving component events and field-of-view events, and 2) It effectively compresses the *history information space* into a much more compact *observation sequence information space* that is combinatorial in nature, without loss of critical information. It may appear that time information associated with null observations is lost; this can be fixed by recording the time duration of a null event and introducing time parameter in the sensors' statistical model. Figure 3 gives a graphical illustration of a typical event observation sequence.

B. Effects of component events over agent distribution

Notation. When writing formulas and outlining algorithms, we sometimes shorten some of the random variable assignments to "..." on both the left hand side (LHS) and the right hand side (RHS). In such cases, the combined "..." on LHS and RHS denote the same terms. For example, in $P(\ldots) = P(\ldots, s_k, \ldots)$, the first "..." is equivalent to the next two "..." combined.

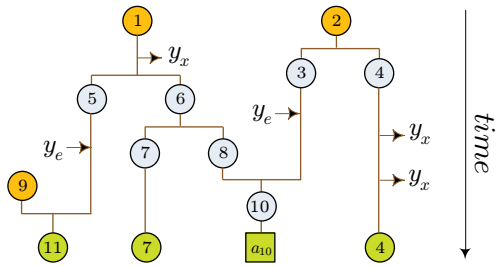

Fig. 3. A typical event observation sequence. The circles with numbers represent the shadow components; the labeled arrows represent the field-of-view observations of affect components. The square (with a_{10} in it) denotes that component s_{10} disappears to reveal a_{10} agents in it.

With observations of the critical events arranged in a sequential form, the next question is: How do these observations affect the probability distribution? Let us look at the component events first. Since they are observed with 100% confidence, there is no need to distinguish between events and observations for these. Among the four types of component events, the split and disappear event are the important ones and therefore are discussed before the others.

1) Split. When a component splits into two disjoint components, since the sensors can only observe the split event but not what happens within the components during the split, the probability masses in the newly spawned components cannot be predicted without further assumptions. In other words, the split event introduces more uncertainty. Solving this demands a *split rule*, either from supporting data or an oracle, that dictates how the originating component's probability mass should be redistributed. For example, statistical data may indicates that the number of agents in the child components are proportional to their respective volumes.

2) Disappear. When a shadow disappears, it will reveal how many agents are hiding behind it. This information can be used to update our belief about the agent distribution by eliminating some improbable distribution of agents. In particular, it can reduce the uncertainty created by split events. For example, say that a component s_i, having a_i agents in it (with 100% probability), splits into components s_j and s_k. The split rule may say that it is possible for s_j to have 0 to a_i agents and so does s_k. However, if s_k later disappears to reveal a_k agents in it and no other events happen to s_j and s_k, then s_j must have exactly $a_i - a_k$ agents in it. In general, assuming that component s_k disappears with an agent distriubtion $P(s_k)$, the update rule is given by

$$P'(s_1 = x_1, \ldots, s_{k-1} = x_{k-1}, s_{k+1} = x_{k+1}, \ldots,) \propto \sum P(s_1 = x_1, \ldots, s_k = x_k, \ldots, s_n = x_n) P(s_k = x_k),$$

in which the summation is over all joint probability entries of $P(s_1, \ldots, s_n)$ such that $s_k = x_k$. Normalization may be necessary.

3) Appear. An appearing component s_k, with distribution $P(s_k)$, can be joined with the rest via combining the independent distributions $P(s_k)$ and $P(s_1, \ldots, s_n)$:

$$P'(s_1 = x_1, \ldots, s_n = x_n, s_k = x_k) = P(s_1 = x_1, \ldots, s_n = x_n) P(s_k = x_k).$$

4) Merge. In this case, two probability masses are collapsed. We merely collect the joint distribution to form a single one,

$$P'(\ldots, s_k = x_k) = \sum_{x_i + x_j = x_k} P(\ldots, s_i = x_i, \ldots, s_j = x_j, \ldots),$$

in which s_k is the merged component from s_i and s_j. Table I gives such an example in which the original components are s_1, s_2, s_3 and s_2, s_3 merge form component s_4.

TABLE I

before merge	$P(s_1 = 1, s_2 = 1, s_3 = 4) = 0.2$ $P(s_1 = 1, s_2 = 2, s_3 = 3) = 0.2$ $P(s_1 = 1, s_2 = 3, s_3 = 2) = 0.2$ $P(s_1 = 2, s_2 = 1, s_3 = 3) = 0.2$ $P(s_1 = 2, s_2 = 2, s_3 = 2) = 0.2$
after merge	$P(s_1 = 1, s_4 = 5) = 0.2 + 0.2 + 0.2 = 0.6$ $P(s_1 = 2, s_4 = 4) = 0.2 + 0.2 = 0.4$

C. Effects of field-of-view events and observations over agent distribution

For field-of-view events, it is straightforward to see that an enter event should only affect the component being entered by increasing the expected number of agents in the component. For example, if there is a single component s and an enter event happens, we merely update $P(s = a_i) = p_i$ to $P(s = a_i + 1) = p_i$. On the other hand, an exit event does just the opposite and we change $P(s = a_i) = p_i$ to $P(s = a_i - 1) = p_i$. A complication shows up here: If component s_i splits into components s_j, s_k and an e_x event happens to component s_j, it suggests that it is impossible for s_j to have 0 agent before the e_x event. The affected probability mass needs to be removed and the remaining renormalized. The null event does not change the agent distribution.

Now, to propagate a probability mass through a field-of-view observation, y, we essentially update the entry into three pieces according to above rules, multiplying each resulting entries with the probability $P(\mathbf{e} = e_e \mid \mathbf{y} = y), P(\mathbf{e} = e_x \mid \mathbf{y} = y)$, and $P(\mathbf{e} = e_n \mid \mathbf{y} = y)$, respectively. If an enter event is not possible for the observation, the two entries left need to be renormalized.

D. A few agents versus many agents

As we try to make a general method for handling critical events without modeling under any particular distribution or split rule, another issue arises: The number of agents is discrete, but discrete joint probability distributions can use a large amount of space and time to process when there are a large number of agents, components, and events in the system. On the other hand, if we attempt to use nice discrete probability mass functions or continuous probability density functions for approximation, it becomes problematic when

there are only a few agents and events. In such cases, the loss of estimation accuracy may become unacceptably large after only a short sequence of events.

From the discussion above, we see that, in estimating the probability mass over the shadow components after a sequence of observations of critical events, there exists an intrinsic trade-off between accuracy and the amount of computation. We believe that deciding which representation to use should depend on the number of agents in the system and the number of events that happen, which can be partitioned into three categories:

1) There are a few agents and events, which requires high accuracy as computation is carried out. On the other hand, because the combinatorial choices are limited, this case can be treated with high fidelity, with the only assumption being good split rule and sensor statistics.
2) There are a few agents but a large number of events. Propagating the probability mass through many events will likely cumulate significant errors when there are only a few agents, unless an extremely reliable split rule and sensor statistics are available. If this is the case, then the method for the first category will work fine. Otherwise, a probabilistic approach may give results that are far off the true distribution; the nondeterministic approach [15] can be a better alternative here.
3) There are many agents in the system. If this is the case, we have more freedom in making simplifications without dramatically altering the outcome.

To offer an idea of what we mean by a few and many, our non-optimized Java implementation for the first category can handle tens of agents and events combined, beyond which the JavaVM may run out of memory after a long period of time (10+ minutes). The heuristics for the third category can handle up to a thousand of agents and events. In the next two sections, we give detailed analysis of the first and third categories.

IV. Accurately Propagating Probability Masses

When agents and events/observations are a few, to maintain an accurate agent distribution in the components as critical events happen, we want to keep the number of assumptions as few as possible. In this section, we only assume the bare minimum: 1) An accurate split rule is provided that governs how agents redistribute when a split event happens, and 2) The sensor statistics are provided ($P(\mathbf{e} \mid \mathbf{y})$ is known). It is hard to imagine any meaningful prediction can be made without these inputs. As events happen, the probability mass $P(s_1, \ldots, s_n)$, is updated according to a straightforward algorithm (Table III and IV) based on the analysis in section III, in which the *observation* data structure is defined in Table II.

As a demonstration, we work through an observation sequence simplified from Fig. 3, shown in Fig. 4, with the following assumptions: 1) Initially there are 2 agents each in component s_1, s_2, 2) The split rule is that each agent has 0.5 probability of going into each of the two split shadow components, 3) There is no null event or observation, with the true positive rate for any observation being $p = 0.9$, and 4) $a_5 = 1$ with probability 0.5 and $a_5 = 2$ with probability 0.5. The extra assumptions are made so that the calculation of the probability mass entries are limited (so that they can be listed in a table). The observation by observation processing is shown in the full paper, along with a graphical illustration. To verify the correctness of the outcome, Monte Carlo trials are also run, in which individual agents are propagated through the observation one by one. Since it is not an exact method, we leave the details of it to the next section. After 1000 successful random trials(this is the number of trials used for all Monte Carlo simulations in the paper), we get $P(s_4 = 0) = 0.079, P(s_4 = 1) = 0.154, P(s_4 = 2) = 0.767$, which matches well with our exact algorithm.

TABLE II

observation data structure used in Table III

t	event type, can be one of *appear, disappear, split, merge* component events and *enter, exit, null* field-of-view events
s_s	the originating component in a split event
s_{s1}	the first new component after a split event
s_{s2}	the second new component in a split event
s_{m1}	the first component in a merge event
s_{m2}	the second component in a merge event
s_m	the newly merged component
s_e	the newly appeared component from an appear event
n_e	the number of agents in an appear event
s_v	the disappearing component in a disappear event
n_v	the number of agents revealed in a disappear event

TABLE III

PROCESS_PROBABILITY_MASS
Input
initial agent distribution $P(s_1, \ldots, s_n)$
queue Q of observation sequence
split rule
sensor statistics $P(\mathbf{e} \mid \mathbf{y})$
Output
the agent distribution after all observations

foreach event *observation* o in Q
switch$(o.t)$
case *appear*:
update all $P(s_1 = x_1, \ldots, s_n = x_n) = p_j$ entries to $P(s_1 = x_1, \ldots, s_n = x_n, o.s_e = n_e) = p_j$
case *disappear*:
remove all joint distribution entries with $o.s_v \neq o.n_v$
renormalize the probability masses
case *split*:
add two new components $o.s_{s1}$, $o.s_{s2}$
split prob. mass in $o.s_s$ into $o.s_{s1}$, $o.s_{s2}$ by **split rule**
case *merge*:
add a new component $o.s_m$
collapse probability mass from $o.s_{m1}, o.s_{m2}$ into $o.s_m$
case *enter, exit, null*:
call PROCESS_FOV_EVENT
return the updated agent distribution

V. Efficiently Propagating Probability Masses

What if there are many agents and events in the system? For a slightly more complicated event observation sequence (Fig. 4), with 5 agents each in component s_1 and s_2 to

TABLE IV

PROCESS_FOV_EVENT
Input
agent distribution $P(s_1, \ldots, s_n)$
sensor statistics $P(\mathbf{e} \mid \mathbf{y})$
field-of-view observation $y \in \{y_e, y_x, y_n\}$
the affected component s_i
Output
the agent distribution after the observation

foreach $P(\ldots, s_i = x_i, \ldots) = p_j$ entry in the distribution
let $P'(\ldots, s_i = x_i + 1, \ldots) = p_j * P(\mathbf{e} = e_e \mid \mathbf{y} = y)$
let $P''(\ldots, s_i = x_i, \ldots) = p_j * P(\mathbf{e} = e_n \mid \mathbf{y} = y)$
if $x_i > 0$
let $P'''(\ldots, s_i = x_i - 1, \ldots) = p_j * P(\mathbf{e} = e_e \mid \mathbf{y} = y)$
else
normalize P', P'' such that $P' + P'' = p_j$
remove $P(\ldots, s_i = j, \ldots) = p_j$ entry
store entries P', P'' and P''' if applicable
return the updated agent distribution

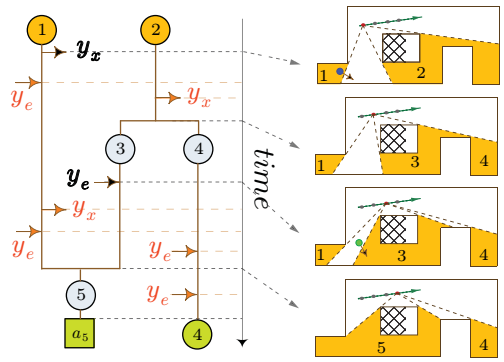

Fig. 4. A simple event observation sequence is generated (on the left, with only two field-of-view observations marked with bold faced font) when a robot carrying omni-directional, infinite range sensor follows the dotted path in a polygonal environment with a hole (the four figures on the right). The last event, disappearing of component s_5, is not shown on the right; we note that additional resource is needed to make s_5 disappear (say, a sub search team). A slightly more complicated sequence is also possible with six additional field-of-view observations (on the left, marked with lightened font).

start, 135 joint probability table entries are obtained before the merge step. To compare, the original sequence has only 10 entires. The probability mass entries increase rapidly because of the split events and the field-of-view events. For a split event, if the originating shadow contains up to n agents, then the number of probability mass entries can multiply by up to a factor of $n + 1$. For field-of-view observations, each has certain probability to be enter, exit, and null events, which may cause the number of probability mass entries to triple in the worst case. Therefore, as the number of agents and events increase, the space required to store the probability masses may grow exponentially. Since processing each observation requires going through all the entries, computation time will also explode, which means that the algorithm PROCESS_PROBABILITY_MASS will not work efficiently. On the bright side, as mentioned in section III, with many agents in the system, exact algorithms may not be necessary to guarantee accuracy.

A. Monte Carlo trials

Since our task is to probabilistically track the whereabouts of agents, sequential Monte Carlo methods appear to be a natural choice. As a first heuristic, we perform simple trials such that each trial starts with the initial distribution of agents. These agents are propagated through the event observations by querying a Monte Carlo simulator. During each trial, the outcome of simulation may contradicts an observation, in which case the trial is simply not successful and discarded. After certain number of successful trials are run, the final agent distribution can be obtained. For example, the mean of the number of agents in a final shadow component is simply the average of the number of agents in that shadow over all successful runs. We denote this heuristic simply as *Monte Carlo*. For simulations in this paper, we require 1000 successful trials. We also note that since the particular Monte Carlo simulation we perform in this paper do not depend on data, its result is probabilistically correct and therefore can serve as baselines for verifying results from other algorithms.

B. Improving the PROCESS_PROBABILITY_MASS algorithm

Observing that the computation is burdened by storing the sheer amount of probability mass entries when there are many agents and observations, an obvious simplification is to *resample* the entries and keep the important ones. For example, we may choose to retain the first 1000 probability mass entries of largest value. With each step of processing looking at each entry once, the processing time per step becomes a constant, albeit a large one. With this approximation, the earlier algorithm then runs in time linear in the number of observations. We denote this heuristic RT-X, with X being the number of entries to keep.

There is a clear similarity between this heuristic and particle filtering: Both begin with a discrete set of probability masses, push the set through an update rule, and resample when necessary. They also share the same weakness: If the key sample points with low probabilities are truncated, the end result may be severely skewed. Unlike in typical particle filter problems, the number of random variables in our problem keeps changing with split and merge events.

C. Simulation results

To test out the above mentioned methods, we run simulation with the observation sequence in Fig. 5. The sequence contains 14 component event observations. We also put in 32 field-of-view observations scattered along the sequence, which are not marked in the figure. Components $1, 2, 3, 11, 13, 16, 20$ are associated with $25, 22, 23, 9, 8, 15, 9$ agents (with probability 1), respectively, be them initial, appearing, or disappearing components. For performance measure, we look at the time for one run of the algorithm to complete, as well as the expected number of agents in individual components at the end (s_{17}, s_{18}, and s_{19}). The simulation program was developed adhering to the Java 1.6 language standard under the Eclipse environment.The

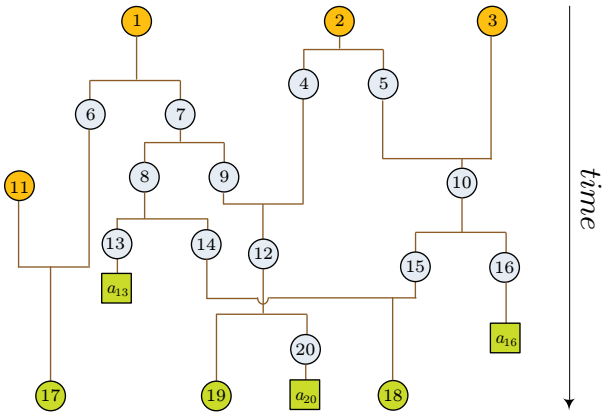

Fig. 5. An event observation sequence with a total 20 components in its life cycle. The field-of-view observations are not marked.

computation is performed on a workstation with an Intel Core 2 Quad processor running at 3.0 GHz. The JavaVM has a maximum memory of 1.5GB. Although our algorithm can be parallelized easily, no multi-threading is used in this implementation. The outcome is summarized in Table V.

TABLE V

Heuristic	s_{17}	s_{18}	s_{19}	t(s)
none, exact	out of memory after 10 mins			
Monte Carlo	18.26	22.25	11.85	43.2
RT-100000	17.78	21.83	12.34	66.3

The simulation shows that when no heuristic is used, the exact algorithm fails to run due to an out of memory error after 10 minutes. At the peak of its memory usage during the failed run, there are more than 2×10^7 probability mass entries. On the other hand, both heuristics take about a minute to finish; the RT-100000 result matches that of the Monte Carlo method.

VI. Conclusion and Future Directions

We have generalized the formulation and solution to the previously addressed *combinatorial filtering* problem of tracking unpredictable agents moving in and out of the field-of-view of moving sensors. After analyzing the problem, we provide both exact and efficient algorithms that handle the several possible scenarios depending on the number of agents and observations in a system. In solving the more general, probabilistic version of the tracking problem, a clear link is also established between combinatorial filtering and Bayesian filtering methods: The final agent distribution is in essence associating the combinatorial solution, a polytope structure, with appropriate probabilities. Viewing it from another angle, the *probabilistic shadow information space* extends naturally from its nondeterministic counterpart, adding merely additional dimensions to record probabilities.

In our problem formulation we have assumed that agents are indistinguishable. We note that it is also possible to work with agents at different levels of distinguishability (fully distinguishable, partially distinguishable or teams, indistinguishable) by introducing *team label*s over the agents. If distinguishability is desirable, the analysis and result from [15] can be applied here with minimal modifications by treating each team individually.

Our study of the probabilistic shadow information spaces is by no means the end of a storyline. On the contrary, it is meant to be the starting point of a general method that unifies and simplifies the treatment of the problem of tracking unpredictably moving agents. Further research is under way with the goal of providing more efficient algorithms as well as concrete example applications that demonstrate the correctness and practicality of the algorithms. In the end, we hope to provide a framework that could be used as a module in solving complex robotics problems.

Acknowledgments: Yu and LaValle are supported by NSF grant 0904501 (IIS robotics), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

References

[1] Y. Baryshnikov and R. Ghrist. Target enumeration via Euler characteristic integrals I: sensor fields. Preprint available on Internet, March 2007.

[2] A. Doucet, N. de Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, Berlin, 2001.

[3] B. Gerkey, S. Thrun, and G. Gordon. Clear the building: Pursuit-evasion with teams of robots. In *Proceedings AAAI National Conference on Artificial Intelligence*, 2004.

[4] B. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research*, 25(4):299–322, 2006.

[5] B. Gfeller, M. Mihalak, S. Suri, E. Vicari, and P. Widmayer. Counting targets with mobile sensors in an unknown environment. In *ALGOSENSORS*, July 2007.

[6] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. *International Journal of Computational Geometry and Applications*, 9(5):471–494, 1999.

[7] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.

[8] J.-H. Lee, S. Y. Shin, and K.-Y. Chwa. Visibility-based pursuit-evasions in a polygonal room with a door. In *Proceedings ACM Symposium on Computational Geometry*, 1999.

[9] N. C. Metropolis and S. M. Ulam. The Monte-Carlo method. *Journal of the American Statistical Association*, 44:335–341, 1949.

[10] J. Singh, R. Kumar, U. Madhow, S. Suri, and R. Cagley. Tracking multiple targets using binary proximity sensors. In *Proc. Information Processing in Sensor Networks*, 2007.

[11] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21(5):863–888, October 1992.

[12] B. Tovar, F. Cohen, and S. M. LaValle. Sensor beams, obstacles, and possible paths. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2008.

[13] C.-C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *International Journal of Robotics Research*, 26(9):889–916, 2007.

[14] D. B. Yang, H. H. Gonzalez-Banos, and L. J. Guibas. Counting people in crowds with a real-time network of simple image sensors. In *Proc. IEEE International Conference on Computer Vision*, volume 1, pages 122– 129, 2003.

[15] J. Yu and S. M. LaValle. Tracking hidden agents through shadow information spaces. In *Proceedings IEEE International Conference on Robotics and Automation*, 2008.

[16] F. Zhao and L. Guibas. *Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, San Francisco, CA, 2004.