2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

Tracking Hidden Agents Through Shadow Information Spaces

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

***Abstract*— This paper addresses problems of inferring the locations of moving agents from combinatorial data extracted by robots that carry sensors. The agents move unpredictably and may be fully distinguishable, partially distinguishable, or completely indistinguishable. The key is to introduce information spaces that extract and maintain combinatorial sensing information. This leads to monitoring the changes in connected components of the shadow region, which is the set of points not visible to any sensors at a given time. When used in combination with a path generator for the robots, the approach solves problems such as counting the number of agents, determining movements of teams of agents, and solving pursuit-evasion problems. An implementation with examples is presented.**

I. Introduction

Many important tasks involve reasoning about observations made while detectable bodies pass in and out of the field-of-view of moving sensors. Figure 1 shows several scenarios to which the concepts in this paper apply. In such scenarios, general problems include: 1) *counting* unpredictable people or robots that move in a complicated environment [1], [4], [9], [15], 2) *pursuing* an elusive moving target by sweeping through a complicated environment to guarantee detection or capture [3], [5], [8], [12], 3) *monitoring team movement* to ensure that sufficient numbers from each team are present in critical parts of an environment, and 4) *tracking* movements of agents to determine their possible locations [11], [14], [16]. For each of these and other tasks, numerous scenarios are possible depending on the types of robots, moving bodies, sensors, environments, and models that are given. Our paper is motivated by the following question: Is there a general way, regardless of the scenario, to systematically process all accumulated sensor observations and make inferences about where the moving bodies might be? Determining where bodies might be is like playing a complicated shell game[1]: We want algorithms that efficiently make perfect inferences about unobservable moving bodies.

[1]This is a gambling game, usually involving deception, in which the player must track the location of a hidden object as the dealer quickly shuffles several potential containers.

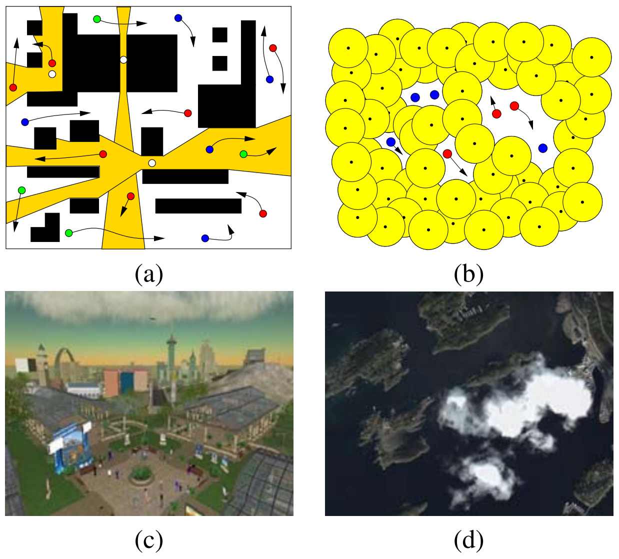

Fig. 1. a) Imagine three robots (white discs) that carry visibility sensors to detect and track the movements of agents in an indoor (complicated corridors) or outdoor (a campus of buildings) environment. There are seven connected white regions in which agents are out of view of sensors. b) Suppose there are holes in the coverage of an actuated sensor network, and we want to reason about possible movements of agents in the holes. c) In an urban setting, imagine tracking people using helicopters, and the field of view is frequently obstructed by buildings and landmarks. d) Consider using satellites to track boats on the sea while the land constrains their motions and moving clouds obstruct part of the view.

This paper addresses the general problem of inferring where unpredictable moving bodies, called *agents*, could be as they pass in and out of view. We only address the *passive* problem of making conclusions based on the motions of the sensors. In this sense, we present a *combinatorial filter*, which functions much in the same way as a *Kalman filter* or general *Bayesian filters*; however, our filter processes only combinatorial information. This provides useful inferences for humans or robots, and it is hoped that the filter also aids in the *active problem*, which involves deciding how to move robots and search the *information space* [6] to accomplish a given task. Regardless of the particular task or scenario, the combinatorial filtering problem amounts to processing critical changes in the *shadow region*, which is the portion of the environment that is not visible to any sensors at a particular instant. We construct an information space which accumulates constraints that arise as the connected components of the shadow region change. We introduce algorithms that encode the information state as a high-dimensional polytope and can efficiently answer queries regarding possible locations of agents by solving a *maximum flow* problem over an extended bipartite graph that is derived incrementally from the history of sensor observations.

To the best of our knowledge, there has been no attempt to reason about the movements of targets with the level of generality considered in our paper. The main assumption is that the connected components of the shadow region can be maintained. Beyond this, the environment may be two or three dimensional, known or unknown, and simply or multiply connected. There may be one or more robots or humans carrying sensors. The sensor field-of-view may be limited by any depth or angle. The agents in the environment may be indistinguishable, distinguishable, or partially distinguishable (for example, there are several teams). Figure 2 illustrates the high-level flow of concepts throughout the remainder of the paper.

II. Problem Formulation

A. Robots and visibility regions

Suppose that one or more *robots* move in a *world*, $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$. To avoid configuration-space complications, assume each robot is a point. Suppose that there are fixed obstacles in $\mathcal{W}$ that obstruct the robots. The obstacle boundary is assumed to be bounded and piecewise-analytic (to enable finite encodings). Let $F \subset \mathcal{W}$ denote the *free space*, which is an open set of possible robot positions. Let q denote the *configuration* of the robots; if there are k robots, then $q \in F^k$.

The robots carry sensors that "illuminate" a subset of F. Let $V(q)$ denote a closed *visible region*, when the robots are in configuration q. Let $S(q) = F \setminus V(q)$ denote the *shadow region*. If the robots move along a time-parameterized path $\tilde{q} : [0,t] \rightarrow F^k$, then the shadow region itself becomes time-parameterized: $S(\tilde{q}(t'))$ is obtained for every $t' \in [0,t]$. It is assumed that $V(q)$ and $S(q)$ behave nicely as q varies continuously; more precisely, arbitrarily small changes in q lead to small changes in $V(q)$ and $S(q)$ (this can be formalized using the Hausdorff metric). This enables the shadow components to be consistently labeled as q varies.

B. Moving agents

Let A denote a set of n *agents*. Each agent is a point that moves along a continuous, not-necessarily-known path through F. There are no given bounds on the maximum speeds of the agents. Let $r \in F^n$ be a $2n$-dimensional vector that specifies the agent positions.

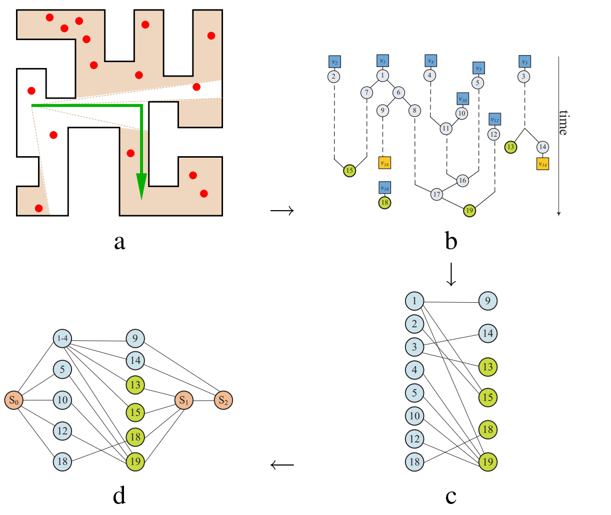

Fig. 2. a) A 2D environment and trajectory followed by a robot with omnidirectional visibility. b) A graph that encodes combinatorial changes in visibility. c) A bipartite graph, made by compressing information in (b). d) A resulting maximum-flow graph, used to answer questions about tracking or counting agents.

The vectors q and r yield the positions of all robots and agents. Therefore, let $x = (q,r)$ denote the *state*, and let $X = F^{k+n}$ be the *state space*.

To set up different levels of distinguishability, it will be convenient to assign team labels to the agents. Let T denote a set of m *teams*, with $m \leq n$. Let the mapping $l : A \rightarrow T$ assign a team to each agent. It is assumed that agents can be distinguished only if they belong to different teams. At one extreme, we may have $|T| = 1$, in which case all agents belong to the same team and are indistinguishable. At the other extreme, we may have $T = n$ and l is a bijection. In this case, the agents are all distinguishable. A convenient intermediate case is to assign *colors* to the agents. For example, $T = \{red, green, blue\}$. There might, for example, be 30 agents, with 10 reds, 15 greens, and 5 blues.

What does the sensor actually produce? Suppose that for any configuration q, the connected components of $S(q)$ are labeled with unique positive integers. As q changes, the components of $S(q)$ may undergo one of four critical changes: 1) two components merge, 2) one component splits into two, 3) a new component emerges, and 4) a component disappears.[2] Each critical change is called a *component event*. These events are based on inflection and bitangent crossings, and are described in more detail in [7], [13]. It is assumed that the component

[2]This includes a general position assumption, to avoid tedious cases in which three or more components are involved in a change.

labels remain constant unless there is a component event. Let $\mathbb{N}$ be the set of nonnegative integers. Let $h : X \to Y$ be the *sensor mapping*, in which $Y = \mathbb{N}^{m+1}$ is an *observation space*. An *observation* $y = h(x) \in Y$ is a vector of $m + 1$ integers. For i from 1 to m, the i^{th} component of y indicates the number of agents from the i^{th} team that lie in $V(q)$. At the instant in which an agent enters or exits $V(q)$, y_{m+1} gives the label of the component of $S(q)$ that is entered or exited; otherwise, $y_{m+1} = 0$. The instant at which an agent enters or exits $V(q)$ is called a *field-of-view event*.

C. Initial conditions

An initial condition is defined for each shadow component. The total number of agents, n, may or may not be known to the robot. For each component, we allow lower and upper bounds on any subsets T_i of T. Let l_i and u_i denote the lower and upper bounds associated, respectively, with T_i. Here are some examples. We might know initially that one component contains at least 2 and at most 5 red agents. In this case, $T_i = \{red\}$, $l_i = 2$ and $u_i = 5$. Alternatively, we might know that there are exactly 7 agents, with unknown teams. In this case, $T_i = T$ and $l_i = u_i = 7$. The completely unknown case becomes $T_i = T$, $l_i = 0$, and $u_i = \infty$.

Each component may have as many as k pairs of bounds (k may be different for each component). Let $T_1, \ldots, T_k$ denote the subsets of T chosen for constraints; we assume these subsets are disjoint. An initial condition for the component can be expressed as

$$\{(T_1, l_1, u_1), (T_2, l_2, u_2), \ldots, (T_k, l_k, u_k)\},$$

in which the l_i, u_i could be any nonnegative integer or positive infinity, and $l_i \le u_i$.

D. Possible Tasks

Several kinds of tasks can be defined based on the formulation. Suppose that the robots have moved along some trajectory. Possible tasks are:

- Determine the minimum and maximum numbers of red (or any other color) agents in one of the shadow components.
- Count the total number of agents in F.
- Determine whether one final shadow component contains any agents.
- Determine whether one final shadow component has more blue agents then red agents.
- Determine whether blue and red agents are separated into different shadow components.

III. The History Information Space

To address the tasks described in Section II-D, we carefully define and manipulate *information spaces* (for a general introduction, see Chapter 11 of [6]). We will use the abbreviations *I-space* for information space and *I-state* for information state. The basic idea will be to avoid precisely estimating the agent locations, and instead process the sensor data directly to solve tasks. To accomplish this, substantial effort is required to identify compact, combinatorial representations that preserve all information that is pertinent to solving tasks. In this section, we introduce the history I-space, in which an I-state contains initial conditions and the entire time-parameterized history of sensor observations.

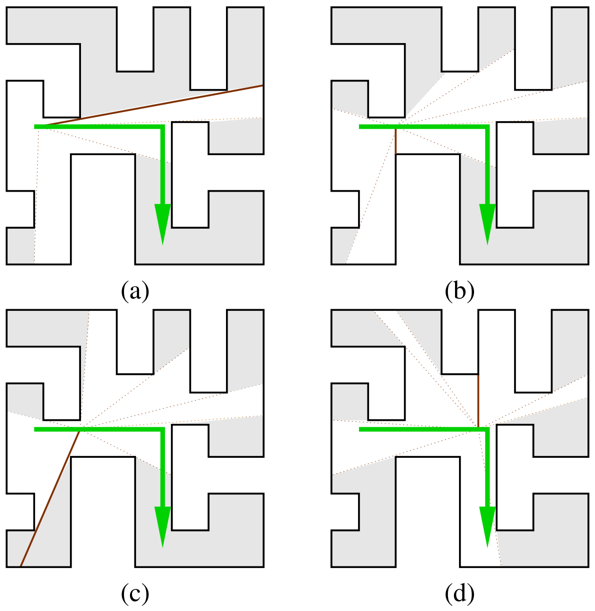

Fig. 3. In this simple example, a single robot carries a standard visibility sensor in a polygonal free space. The segment that emanates from the robot trajectory indicates where a component event occurs (only a few component events are shown). Agent trajectories are not shown. There are four possible types of component events: a) A shadow component *splits* into two. b) A shadow component *appears*. c) Two shadow components *merge*. d) A shadow component *disappears*.

After time evolves from time 0 to some time t, a perfect description of everything that occurred would be a *state trajectory* $\tilde{x}_t : [0, t] \to X$. It is impossible to obtain this, however, because the agent positions are generally unknown. Instead of the state trajectory, we are offered the *observation history*, $\tilde{y}_t : [0, t] \to Y$, which is a time-parameterized collection of sensor observations. Suppose temporarily that the robot configurations are always known (this assumption is removed in Section IV. Let η_0 denote the initial conditions from Section II-C.

The *history I-state* at time t is $\eta_t = (\eta_0, \tilde{q}_t, \tilde{y}_t)$, which represents all information available to the robots. The *history I-space* $\mathcal{I}_{hist}$ is the set of all possible history I-states, which is an unwieldy space that must be dramatically reduced if we expect to solve our tasks.

IV. The Shadow-Sequence Information Space

In this section, we construct a *derived I-space* $\mathcal{I}_{ss}$ which is obtained by a mapping from $\mathcal{I}_{hist}$ that discards information that is irrelevant to solving our tasks. Consider the information contained in $\eta_t = (\eta_0, \tilde{q}_t, \tilde{y}_t)$. In $\mathcal{I}_{ss}$, we retain η_0, but $\tilde{q}_t$ and $\tilde{y}_t$ are reduced in some way. The following reductions are made:

1) The configuration trajectory $\tilde{q}_t$ will be replaced by a graph structure that simply labels the connected components of $S(q)$ and tracks how they evolve. This information can be determined completely from the configurations, which are given by $\tilde{q}_t$. The particular configurations do not even need to be measured, provided that there is some alternative way to determine the shadow components (for example, they be inferred directly from depth discontinuities in a planar, simply connected free space [13]). The graph structure is updated only when a component event occurs, which means components split, merge, appear, or disappear (see Figure 3).
2) The observation history $\tilde{y}_t$ is compressed so that only changes in the observation need to be recorded. Every such change corresponds to a *field-of-view event*. The result is a sequence of distinct observations.
3) Precise timing information is discarded, except the order in which events occur. The component events and field-of-view events are interleaved according to their original time ordering. We make a general position assumption that no two events occur simultaneously.

The result is an I-state that lives in a derived I-space, $\mathcal{I}_{ss}$, which we call the *shadow-sequence I-space*.

One fact that should become clear during the presentation is that all of the concepts and methods from Sections III to V can be applied to the set of constraints on each T_i (recall from Section II-C) without consideration of constraints with respect to other subsets of T. We therefore present the concepts and algorithms for the case of a single team (i.e., $|T| = 1$). The approach then extends naturally to handle each T_i for i from 1 to k (we eventually obtain k independent max-flow problems, instead of one).

By considering one team only, it is simple to characterize the possible constraints for every initial shadow component and every shadow component that arises from an appearance component event, as shown in Figure 3(b). For the i^{th} component, we allow lower and upper bounds, represented as $v_i = (l_i, u_i)$, in which l_i and u_i are nonnegative integers (u_i may be infinite) such that $l_i \leq u_i$. For an initial shadow component, any possible v_i is allowable; however, it seems that for a component that arises from an appearance, it is only possible that

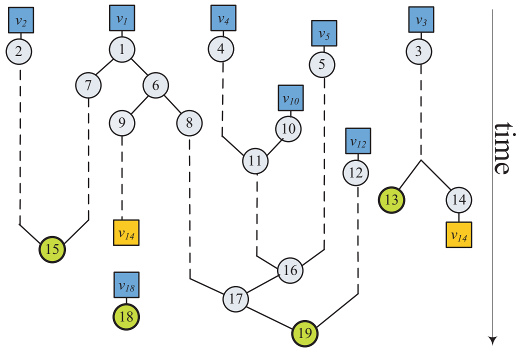

Fig. 4. A graph that accounts for the evolution of shadow components over time.

$v_i = (0, 0)$ (a new shadow component must be clear of agents).

We will divide field-of-view events into two classes to simplify the discussion. If k field-of-view events occur in which k agents enter a shadow component, say s_i, that was just formed by an appearance component event, then we associate $v_i = (k, k)$ with s_i. In the opposite direction, if k agents leave a shadow component s_i and in the next event s_i disappears, then $v_i = (k, k)$ is associated with s_i, and it is recorded that s_i disappeared. The remaining presentation up to Section V-C assumes that only these kinds of field-of-view events can occur (otherwise, our examples become cluttered). Section V-C then describes how the remaining field-of-view events are handled.

The shadow-sequence I-space $\mathcal{I}_{ss}$ will now be described using the example from Figure 3. A particular I-state in $\mathcal{I}_{ss}$ is illustrated in Figure 4. The square boxes indicate the upper and lower bounds $v_i = (l_i, u_i)$, which are associated with each shadow component that either: 1) existed initially (the square boxes at the top), 2) appears at a component event, or 3) disappears at a component event. The circles indicate particular shadow components. The I-state can be considered as a directed graph in which the vertices are the circles and all edges flow downward, which corresponds to the progression of time. The boxes simply show the information associated with some vertices. The edges indicate how the shadow components were involved in a split or merge component event. This association is important because we must account for possible movements of agents. For example, Figure 5 shows how an agent can move from one former shadow component to another after a merge component event.

Based on the information shown in the boxes in combination with the graph structure, the possible numbers of agents behind each final shadow component can be expressed as a polytope as follows. Every vertex (circle in Figure 4) is an nonnegative integer variable x_i, which

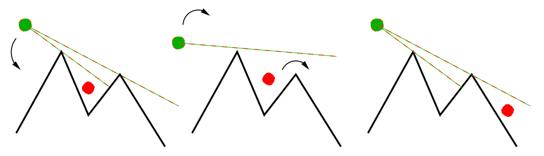

Fig. 5. A component event that enables an agent to relocate.

is associated to shadow component s_i. Every $v_i = (l_i, u_i)$ constraint represents the bounds $l_i \leq x_i \leq u_i$. Furthermore, every split and merge component event corresponds to a sum of variables. For example, in Figure 4, when s_6 splits into s_8 and s_9, we obtain an ILP constraint $x_6 = x_8 + x_9$. When s_4 and s_{10} merge into s_{11}, we obtain $x_{11} = x_4 + x_{10}$. Thus, a piecewise-linear description (polytope) of possible numbers of agents is obtained over the x_i variables.

Answering particular queries over the polytope becomes a kind of integer linear programming (ILP) problem. The general ILP problem is NP-hard [10], which might sound very discouraging; however, our particular ILPs have a special structure that enables them to be solved rather efficiently using maximum-flow algorithms. To arrive at this, however, we need to further compress the I-states. This brings us to the next section, in which a bipartite graph structure encodes I-states, and an I-space results that is even more compact.

V. Collapsing the Information Space onto a Bipartite Graph

A. *Information states as an augmented bipartite graph*

After carefully studying the I-states in $\mathcal{I}_{ss}$, it becomes apparent that the polytopes and associated ILPs that are obtained have some special structure. Any piece of information that originates in an initial shadow component or a shadow component that is generated by an appearance component event must eventually contribute to a final shadow component or a shadow component that disappeared. Furthermore, the information accumulated at any final component or a component that disappeared is composed entirely of information that originated from the initial components and ones that appeared from component events. These bilateral dependencies suggest that the information flow can be captured in a bipartite graph in which the left side indicates the sources of information and the right side indicates the destinations.

Figure 6 illustrates how to construct and incrementally maintain such a graph for each of the four possible types of component events. The bipartite graph contains weights on each of the vertices, which result from bounds of the form $v_i = (l_i, u_i)$. Initially, the graph contains a left and right vertex for every initial shadow component, and an edge between every corresponding pair. In fact, if there

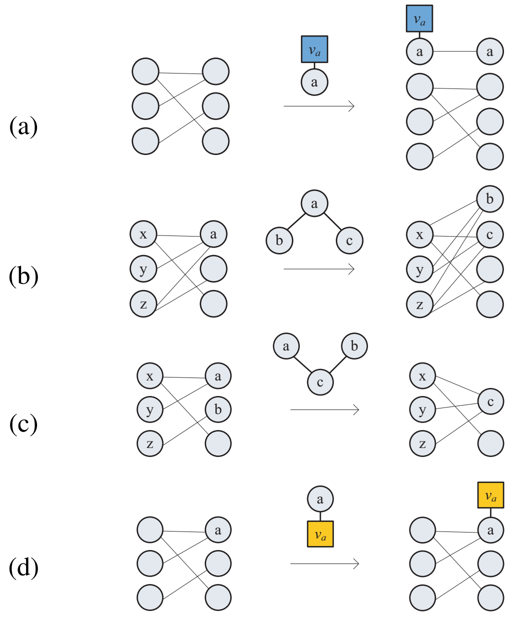

Fig. 6. Incrementally computing I-states in $\mathcal{I}_{bip}$: a) An appearance component event adds two vertices and an edge, with v_a associated with the left vertex. b) A split event splits a vertex and all edges pointing to that vertex. c) A merge event collapses two vertices into one and collapses their ingoing edges. d) A disappear event only associates v_a with the vertex on the right side.

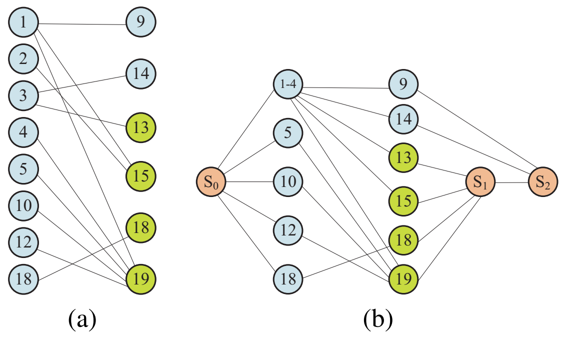

Fig. 7. a) An I-state in $\mathcal{I}_{bip}$, which is a bipartite graph. b) The corresponding maximum-flow graph for answering questions about agent movements. All edges are directed from left to right.

are k initial shadow components, then k copies of the construction shown in Figure 6(a) are made. Conceptually, this is equivalent to introducing k shadow components via appearance events. The four parts of Figure 6 show how the bipartite graph grows incrementally while the robots move along a configuration trajectory. At any time, the resulting bipartite graph, with associated weights on every vertex, is considered as an I-state in a new I-space denoted as $\mathcal{I}_{bip}$.

Figure 7(a) gives the bipartite graph for the example in Figure 3. Note that in general, particularly if there are many component events, the new graph is much smaller than the kind constructed in Figure 4. Note that the conversion causes some information loss: All internal nodes from the I-states in $\mathcal{I}_{ss}$ disappear as they are processed to construct the bipartite graph. Furthermore, the time-ordering information is no longer preserved in the bipartite graph. The information space $\mathcal{I}_{bip}$ is more compact than $\mathcal{I}_{ss}$, but it is nevertheless sufficient for solving our tasks.

B. Tracking agents as a maximum flow problem

We classify the bipartite graph vertices into three types. Vertices on the left are all from initial or appear-event shadow components, and are called *incoming* vertices. The vertices on the right that are from shadow components that have disappeared are called *inactive* vertices; all of these have fixed weights that correspond to the number of field-of-view events that occurred immediately prior to disappearance. The remaining right vertices correspond to shadow components that have not disappeared, and are therefore called *active* vertices.

Once the bipartite is constructed, the task of determining upper and lower bounds on final shadow components can be transformed into a *maximum flow* problem. A flow problem usually has source and sink vertices, and the edges have capacity and flow values. For each v_i, assume that $l_i = u_i$, which corresponds to a scalar weight $w_i = l_i = u_i$ that is associated with various vertices. This assumption will be lifted at the end of this section.

To transform our bipartite graph with vertex weights into a flow graph, introduce a source vertex S_0, with edges pointing to all incoming vertices. Also introduce a sink vertex, S_1, to which all inactive vertices point, and a second sink vertex, S_2, to which all active vertices and S_1 point (see Figure 7(b)). Note that shadow components from s_1 to s_4 are all collapsed into a single vertex here because we can equivalently assume that they are split from same v_i.

After obtaining the extended graph, the weights on vertices must be shifted to capacities on edges. Let $e(s,t)$ be an edge in the graph going from s to t, and denote the capacity and flow on that edge as $c(s,t)$ and $f(s,t)$, respectively.

Consider computing the minimum number of agents in a final shadow component, s_m. In this case, we want to determine the minimum flow through the corresponding active vertex in the graph. Every incoming vertex, i, now corresponds to a fixed number of agents, and the number is the capacity for $e(S_0,i)$. For every inactive vertex j, the capacity of $e(j,S_1)$ is w_j. By conservation of flow, the capacity of $e(S_1,S_2)$ is $c(S_1,S_2) = \sum c(S_0,i) - \sum c(j,S_1)$. To obtain the minimum for s_m, we want the least possible flow through $e(m,S_1)$; therefore, $c(m,S_1) = 0$. All other edges are assigned infinite capacity. After running the maximum-flow algorithm,

$$\min(f(m,S_1)) = \sum_i f(S_0,i) - \sum f(j,S_1) - f(S_1,S_2), \tag{1}$$

in which $f(j,S_1) = c(j,S_1)$. Equation (1) is correct because minimum flow at one edge of the graph is equivalent to flowing the maximum amount possible through the rest of the graph. Note that the particular maximum-flow algorithm is not critical; this is a classical problem and many efficient alternatives exist.

To instead compute the maximum number of agents possible for a final shadow component, s_m, let $c(m,S_1) = \infty$ and let $c(j,S_1) = 0$ for all other active vertices. Keep the rest of the graph the same as for the minimum case. The resulting $f(m,S_1)$ after running maximum-flow algorithm is the desired upper bound.

The procedure can be repeated for every active vertex. Note that the lower (or upper) bounds on final shadow components usually cannot be achieved simultaneously by actual agents because of the dependencies among the components.

Now we return to the more general case in which l_i and u_i are not necessarily equal. In this case, we simply replace v_i by l_i for determining minima and u_i for determining maxima.

C. Handling the remaining field-of-view events

In Section IV, we intentionally left some field-of-view events out of the discussion to simplify the explanation. Those events certainly need to be handled because agents may enter or exit the visibility region at any time. To account for those events, we can augment the bipartite graph as it is computed. For each agent appearance field-of-view event, we can treat it as if a component splits into two and then one of the two shadows disappeared, revealing a certain number of agents. We can treat each agent disappearance field-of-view event as if a new shadow component appears and merges into an existing one. Algorithmically, this effectively transforms field-of-view events into component events.

VI. Solving a Variety of Tasks

The maximum-flow method explained in Section V-B can be used to determine the minimum and maximum number of possible agents hiding in a specified shadow component. In addition to this, a variety of other tasks can be readily solved, and are covered briefly in this section. **Refining initial bounds:** Maximum flow computations can be used to refine the bounds given originally on the initial shadow components, based on information gained later. For example, if s_1 originally has a lower bound of 4, but it is then observed that there are 6 agents appearing from s_9, then there must have initially been at least 6 agents in s_1. Upper bounds can be refined similarly. To get a better lower bound for an initial shadow component, s_m, let $c(S_0, m) = l_m$ and let the remaining $c(S_0, i)$ assume the weights of their incoming vertices. The edges from inactive vertices weighted as before, and all $c(j, S_1)$ are infinite. Let $c(S_1, S_2) = \sum c(S_0, i) - \sum c(j, S_1)$. The refined lower bound is given by

$$l'_m = l_m + \sum_j c(j, S_1) - \sum_j f(j, S_1) \tag{2}$$

in which j ranges over the inactive vertices. To refine u_m, let $c(S_0, m) = u_m$, $c(S_0, i) = l_i$ for each $i \neq m$, and the remaining capacities be the same as in the lower-bound case. The refined upper bound is

$$u'_m = \sum_j f(m, j). \tag{3}$$

Counting: In this case, the total number n of agents is unknown. For determining n, the lower and upper bounds on each initial shadow component are specified as $v_i = (0, \infty)$. Using the algorithm from Section V-B, upper and lower bounds can be obtained on each component. If $l_i = u_i$ for each of these, then n has been determined. Otherwise, (2) and (3) give lower and upper bounds on n. Note that if F is not completely explored, then the upper bound remains at infinity.

Recognizing task completion: We might also want to recognize the completion of certain tasks. For example, in a wild animal preserve, it may be required that the total number of a species is verified periodically. This reduces to the problem of being given n and wanting to account for all of them. To verify the completion of this task, we can keep track of the lower bounds on the total number of agents, and if the number agrees with n, then the task has been accomplished.

Pursuit-evasion: Another task is pursuit-evasion, as defined in [3], [5], [8], [12]. Suppose there is a single evader and the task is to determine where it might be. In this case, $v_i = (0, 1)$ for each initial shadow component. In the end, there are three possibilities for each final shadow component: 1) $v_i = (0, 0)$ (the evader is not in s_i), 2) $v_i = (1, 1)$ (the evader is definitely in s_i), and 3) $v_i = (0, 1)$ (the evader may or may not be in s_i). Note that this is a passive version of the pursuit-evasion problem. We do not determine a trajectory that is guaranteed to detect the evader. In general, this problem is NP-hard [5]. Nevertheless, the calculation method proposed in this paper can be used with heuristic search techniques (or even human operators) to correctly maintain the status of the pursuit.

Multiple teams with bounds on single teams: Now suppose that there are multiple teams, $m > 1$. There are two important cases. For the first one, suppose that the bounds for each initial shadow component refer to individual teams. Thus, each T_i (refer to Section II-C) is just a single team. The task of determining lower and upper bounds for m teams can then be solved by defining m distinct single-team tracking problems, one for each team. If the task is to find a lower bound on all agents in a final shadow component, we simply sum lower bounds of all teams in that component to obtain the answer. This approach even works in the extreme case in which all agents are distinguishable.

Handling constraints across multiple teams: The second case for multiple teams allows T_i to contain more than one team. For example, an initial shadow component may have a constraint that there are between 3 and 7 agents that are red *or* blue. If we want to know the lower bound on red or blue agents, then we set red capacity from S_0 to the corresponding vertex in the flow graph to zero, and flow the reds through the graph to obtain a minimum of the red. We then do the same for blue. Adding those two numbers up gives the answer. Upper bounds can be computed similarly.

VII. Implementation

We implemented and tested the algorithms for a single robot that moves in a simply connected polygonal region in $\mathbb{R}^2$ using an omnidirectional visibility sensor. We chose this setting for simulation because the bitangents and inflections can be calculated so we could have an oracle for moving the agents around inside the free space and avoid simulating particular agent movements. Recall that the method works for any number of agents, any sensor models, and 2D or 3D worlds, as long as the shadow components can be maintained. We implemented the $O(VE^2)$-time Edmonds-Karp max-flow algorithm [2], in which V and E are the numbers of vertices and edges in the graph, respectively. The number V is the total number of initial, appearing, disappearing, and final shadow components. In the worse case, the bipartite graph may be complete, but in practice there are far fewer edges.

The program is written in Java 1.5. The byte code was run on Intel U2500 1.2GHz machine with 1GB RAM. For the free space in Figure 8(a), the trajectory generates 85 component events. We defined an oracle that randomly distributed 100 agents in the free space as the component events occur. This setting yields a bipartite graph that has 41 vertices and 60 edges. Calculating the lower and upper bounds for the 18 final shadow components for a single team took 0.1 seconds. The second free space, shown in Figure 8(b), has 385 component events, 491 total shadow components, 124 vertices in the bipartite graph with 339 edges. The example involves a million agents on 5 teams that intersperse. The bounds on 12 final shadow components for all 5 teams were computed in 2.5 seconds.

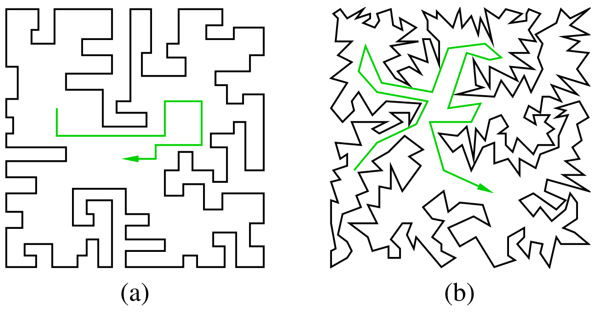
Fig. 8. Complicated examples that were used to test our approach. The given robot trajectories are shown.

VIII. Conclusion

We have introduced and solved a *combinatorial filtering* problem of tracking unpredictable agents that are mostly out of the range of sensors. The approach uses initial hypotheses on possible agent locations and reasons about how the shadow components evolve to make final conclusions. The resulting information space is collapsed into a bipartite graph, and queries are solved efficiently by solving a special maximum flow problem. The concepts are very basic and general. We therefore believe there is great potential for applying them to practical problems such as search-and-rescue, counting people, tracking movements in a building, and analyzing movement strategies for teams of robots or humans.

Many interesting open problems remain. The concepts apply to a wide variety of sensor models; however, the feasibility and complexity of maintaining the shadow components varies and should be studied in particular contexts. Also, the resulting information spaces may further simplify in some cases. For example, it seems that the derived I-spaces, $\mathcal{I}_{ss}$ and $\mathcal{I}_{bip}$, for a single robot in a simple polygon should be much simpler than that for numerous robots in a 3D free space that has holes.

Several open questions remain regarding weaker sensor models. For agents that lie in $V(q)$, we presently assume their team labels and entering/exiting shadow components are all observed. Examples of weaker models include: 1) a binary sensor that indicates only whether there is at least one agent in $V(q)$, 2) a counter that yields the total number of agents in $V(q)$, 3) uncertainty regarding which shadow component an agent entered or exited.

Acknowledgments: Yu is supported in part by a Siebel Fellowship. LaValle is supported in part by the DARPA SToMP program (DSO HR0011-07-1-0002). The authors sincerely thank Chandra Chekuri for a helpful discussion on ILPs and maximum flow.

References

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

[2] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972.

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

[5] 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.

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

[7] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–201, April 2001.

[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] X. Liu, P. H. Tu, J. Rittscher, A. Perera, and N. Krahnstoever. Detecting and counting people in surveillance applications. In *Proc. IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 306– 311, 2005.

[10] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[11] 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.

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

[13] B. Tovar, R Murrieta, and S. M. LaValle. Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518, June 2007.

[14] 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.

[15] 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.

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