Cyber Detectives: Determining When Robots or People Misbehave

Jingjin Yu and Steven M. LaValle

Abstract This paper introduces a problem of validating the claimed behavior of an autonomous agent (human or robot) in an indoor environment containing one or more agents, against the observation history from a sparse network of simple, stationary sensors deployed in the same environment. Following principles of dynamic programming, we partition the decision problem into incremental search over a sequence of connectivity subgraphs induced by sensor recordings, which yields efficient algorithms for both single and multiple agent cases. In addition to immediate applicability towards security and forensics problems, the idea of behavior validation using external sensors complements design time model verification.

1 Introduction

One night, a crime was committed in an office building with complex interior structure. The next morning, a few suspects were identified but none of them would come forward. Instead, all of them provided seemingly convincing stories that excused them from being present at the crime scene. Unknown to the suspects, however, the building's security system, composed of a set of sensors with different capabilities, had made a sequence of recordings of passing people. Knowing that the criminal among the suspects was lying, can we use the sensor recordings to help solve the crime?

Similarly, in computer science, robotics, and control, a frequently encountered problem is verifying that an autonomous system, be it a program or a robot, is performing as designed. For example, a service robot may plan a path to clean office rooms one by one. Due to internal (sensor/actuator/computing units malfunction-

Jingjin Yu
University of Illinois, Urbana-Champaign, e-mail: jyu18@uiuc.edu

Steven M. LaValle
University of Illinois, Urbana-Champaign, e-mail: lavalle@uiuc.edu

ing) or external factors (strong electromagnetic interference for example), the robot may mistake one room for another and fail to accomplish its task without knowing that it has failed. A robot or a system may also be compromised for malicious purposes, producing intentionally bogus records of its actual path to hide the fact. In such cases, it would be highly desirable if external monitoring could automatically determine that a robot has faltered.

In this paper, we introduce realistic abstractions of above problems and show that such formulations are computationally tractable. Specifically, one or more agents (robots or people) are assumed to move in an indoor environment, of which regions are monitored by external sensors (beam detectors and occupancy sensors). We assume that the agents are not aware of these sensors. From a story told by an agent, which is a sequences of places in the environment it has visited, and combined recordings of these sensors, we provide polynomial time algorithms (with respect to the complexity of the environment, the length of the story, as well as the length of the observation history) for the inference problem of whether the given story is consistent with the sensor recordings.

Our work takes inspirations from two active research topics in robotics and control. If one assumes that the behavior of a set of moving bodies is largely unknown, above problem becomes inferring various properties of these moving bodies with a network of simple sensors. Binary proximity sensors have been employed to estimate positions and velocities of a moving body using particle filters [3] and moving averages [17]. The performance limits of a binary proximity sensor network in tracking a single target are discussed and approached in [25], followed by an extension to the tracking of multiple targets [26]. The task of counting multiple targets is also studied under different assumptions [4, 15]. In these works, the sensor network's aggregate sensing range must cover the targets of interest at all times, which is much more difficult to implement than guarding critical regions of an environment. When only subsets of an environment are guarded, *word problems in groups* [12, 14] naturally arise. For the setup in which targets moving inside a 2D region are monitored with a set of detection beams, [28] characterizes possible target locations, target path reconstruction up to homotopy, and path winding numbers. In this domain, the surfacing of more interesting behaviors also induces an increase in complexity; few efficient algorithms exist. This prompts us to ponder: Can we do better if partial knowledge of a target's behavior is available? In viewing its resemblance to the questions asked in [3, 25, 28], our problem requires the design of a combinatorial filter, similar to those in [20, 29, 30]. These combinatorial filters are minimalist counterparts to widely known Bayesian filters [6, 9, 13, 21, 22, 27, 31].

On the other hand, if sensors external to moving bodies are ignored, one is left with the task of systematically verifying that the moving bodies do not have unexpected behaviors. Complex moving bodies such as robots are often modeled as hybrid systems. Existing verification techniques either address subclasses of hybrid systems or approximate reachable sets of such systems [2, 8, 10], because the problem of verifying a system with continuous state space and control input is generally undecidable [1]. In practice, this difficulty translates into the necessity of external measures to safeguard the unverified portion of a system. Alternatively, when high

level task specifications can be coded as General Reactivity(1) formulas [23], the task of composing controllers into verifiably correct hybrid automata can be carried out automatically using linear temporal logic [11, 18]. Even for such provably correct designs, malfunction can still occur due to sensor/actuator/computer errors. Keeping these systems in check again requires monitoring with external sensors.

The main contributions of this paper are twofold. First, using a sparse network of simple sensors to validate the claimed behavior of an autonomous agent introduces a new methodology that complements traditional system verification techniques such as [2, 8, 10]. We believe this is a necessary approach given that most verification processes focus on high level abstractions of an autonomous system, which only models simplified, ideal behavior. Second, applying principles of dynamic programming [5], we show that polynomial time algorithms exist for the proposed decision problems, providing insights into the structure of these detective game like problems. Moreover, the practical algorithmic solution may readily find its way in real world applications, such as system design/monitoring/verification, security, and sensor-based forensics.

The rest of the paper is organized as follows. Section 2 defines the two detective games we study in this paper. For the case in which a single agent triggers all sensor recordings, Section 3 extracts a base graph structure that captures the connectivity of the environment, which is subsequently broken down into pieces for the incremental search introduced and analyzed in Section 4. Section 5 extends the graphs and search algorithm to account for additional agents that are present in the environment[1]. Section 6 discusses many open questions and concludes the paper.

2 Problem Formulation

2.1 Workspace, Agents and Stories

Let the *workspace* $W \subset \mathbb{R}^2$ be a bounded, path connected open set with a polygonal boundary, ∂W. Let one or more point agents move around in W, carrying out unknown tasks. Every agent has a map of W and may move arbitrarily fast along some continuous path $\tau : [t_0, t_f] \to W$. The initial time t_0 and the final time t_f are common to all agents. Assume that we are interested in a specific agent x with a *story* of its own, which may be truthful or fictional. Since an agent may not always have an accurate estimate of its state, a truthful story is not necessarily what has happened. For example, a human can usually recall a (partial, possibly inaccurate) sequence of events after she has performed a task in an environment. In this iteration we let the story have a very basic form: A sequence of places in W that agent x has visited in increasing chronological order,

$$\mathbf{p} = (p_1, p_2, \ldots, p_n), \quad p_i \subset W,$$

[1] An interactive implementation of algorithms from Section 4 and 5 is available at `http://msl.cs.uiuc.edu/~jyu18/pe/cd.html`. A web browser with Java 1.5 support is required to access the page.

such that the unique elements of **p** are each a simply connected region with a polygonal boundary and pairwise disjoint. The set of all unique elements of **p** is denoted $\mathscr{C}_p$. We assume that for every $p \in \mathscr{C}_p$, x has accounted for all its visits to p in **p**. As an example, agent x may simply report "I went from room A to room B, then came back to room A, and eventually arrived room C, at which point I stopped."

2.2 Sensors and the Observation History

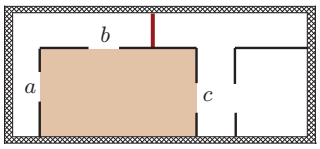

Fig. 1 A simple workspace with an occupancy sensor and a beam detector. The occupancy sensor guards the shaded area with three doorways a, b, and c. The beam detector guards the vertical line segment at the top.

Let a subset of the workspace W be guarded by a heterogeneous set of sensors. The placement of sensors in W is unknown to all agents. Among the commonly available sensors for surveillance, we focus on occupancy sensors and beam detectors. An occupancy sensor is assumed to detect the presence of an agent in a fixed, convex subset $s \subset W$. For example, a room may be monitored by such a sensor (the shaded area in Fig. 1). A data point recorded by an occupancy sensor o_i has two parts, an *activation*,

$$r_{oa} = (o_i, t_a),$$

and a *deactivation*,

$$r_{od} = (o_i, t_d),$$

in which t_a is the time when the first agent enters an empty s and t_d is the time when the last agent exits s. A beam sensor, on the other hand, guards a straight line segment, $\ell \subset W$, between two edges of ∂W (for example, the red line segment in Fig. 1). A data point of such a sensor, b_i, is recorded as an agent crosses ℓ, which can be represented by a 2-tuple:

$$r_b = (b_i, t).$$

A beam detector is deactivated right after activation. We further assume that when a beam detector is triggered by an agent, the agent must pass from one side of the beam to the other side. We denote the collection of all unique sensors in W as $\mathscr{C}_s$. With the introduction of occupancy sensors and beam detectors, we define the *observation history* simply as:

$$\mathbf{r} = (r_1, r_2, \ldots, r_m),$$

in which each $r_i = r_{oa}, r_{od}$, or r_b, is indexed by the time when it occurs, incrementally.

Both occupancy sensors and beam detectors are weak sensors in the sense that they cannot tell an agent's passing direction. In the example given in Fig. 1, a sensor recording of the occupancy sensor could imply that the agent enters and exits from any of the doorways a, b, or c. Similarly, when the beam detector is triggered, an agent could be passing it from left to right or in the other direction. These sensors certainly cannot distinguish among different agents. We choose to work with these two typical but weak sensors so that the algorithms we present apply to a wider range of sensors, provided that they are at least as powerful (although the algorithms may not take full advantage of these stronger sensors). For example, a video camera is a stronger occupancy sensor, capable of providing both passing direction and identification of agents.

Observation history for a single agent. Without loss of generality, we assume that the sensors' detection regions (field of view) are pairwise disjoint: When two or more sensors have overlapping detection regions, we may create virtual sensors by repartitioning these sensors' detection regions so that the virtual sensors have disjoint detection regions [20]. This implies that when agent x is the only agent in W, the activation of any sensor must be followed by the deactivation of the same sensor, with no other sensor activities in between. In particular, in the observation history for a single agent, there can be no other sensor activations or deactivations between one activation/deactivation of an occupancy sensor.

Observation history for multiple agents. When there are multiple (an unknown number) agents in the workspace, it is no longer reasonable to assume that all sensor recordings have activation-deactivation intervals that are pairwise disjoint. For example, one agent may pass a beam detector while another one occupies a room monitored by an occupancy sensor. Different occupancy sensors can also have overlapping intervals of activation. When multiple agents are present in W, we assume that all sensor activation and deactivation times are distinct, since the likelihood of simultaneous sensor triggering is very low.

2.3 The Verification Problem

Given W, $\mathscr{C}_p, \mathscr{C}_s$, agent x's story $\mathbf{p}$ and the observation history $\mathbf{r}$, we are interested in determining whether $\mathbf{p}$ is consistent with $\mathbf{r}$. For the comparison to make sense, we require that both $\mathbf{p}$ and $\mathbf{r}$ span the same time interval, $[t_0, t_f]$. That is, we must determine whether there exists an agent path containing the locations given in $\mathbf{p}$ in the specified order that triggers the sensor recordings given by $\mathbf{r}$. For the purpose of introducing algorithms, we use the example workspace given in Fig. 2 and let agent x's story be:

$$\mathbf{p} = (A, C, B, A, C). \tag{1}$$

In English, agent x told the story that it started in room A and went through room $C, B, A,$ and C, in that order. When there is a single agent in the workspace, we let

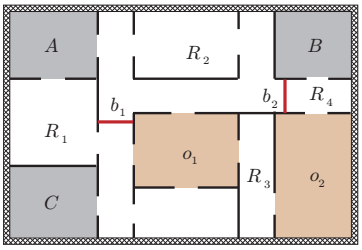

Fig. 2 A workspace with two beam detectors b_1, b_2, two occupancy sensors o_1, o_2, and three labeled rooms A, B and C. Thus, $\mathscr{C}_p = \{A, B, C\}$, $\mathscr{C}_s = \{b_1, b_2, o_1, o_2\}$. There are four connected components R_1 through R_4 when regions guarded by sensor range and rooms in agent x's story are treated as workspace obstacles.

the observation history be:

$$\mathbf{r} = ((b_1,t_1),(o_1,t_2),(o_1,t_3),(b_2,t_4),(o_2,t_5),(o_2,t_6)). \tag{2}$$

For this simple example, it is not hard to see that $\mathbf{p}$ is not consistent with $\mathbf{r}$: B can only be visited after agent x passes b_2 from left to right; however, after visiting B, either b_2 or o_2, o_1 must be triggered for x to visit A once more. When there are multiple agents in the workspace, we let the observation history be slightly different (with which $\mathbf{p}$ is consistent):

$$\mathbf{r} = ((b_1,t_1),(o_1,t_2),(o_2,t_3),(b_2,t_4),(o_2,t_5),(o_1,t_6)). \tag{3}$$

In the example, we have implicitly made the assumption that elements of $\mathscr{C}_p$ and elements of $\mathscr{C}_s$ have coverage regions that are pairwise disjoint; overlapping cases will be handled after the main algorithms are introduced.

3 The Connectivity Graph and Sensing Induced Subgraphs

Both occupancy sensors and beam detectors, when not triggered, act as obstacles that change the workspace connectivity. When a sensor is triggered, the part of the workspace blocked by that sensor is temporarily connected. To explore the structure from this intuition, we first build a *connectivity graph* G that captures the topological features of W. As we are only interested in finding a path[2] through $\mathbf{p}$ that is compliant with $\mathbf{r}$, we only need G to capture how elements of $\mathscr{C}_p$ are connected and how they are connected to the sensors, $\mathscr{C}_s$. Therefore, we treat these elements as vertices of G. Since there are two possible directions that an agent may pass a beam detector, two vertices are needed for each beam detector. A single vertex is needed

[2] In graph theory, a path does not visit one vertex multiple times. Therefore, the image of a continuous path, when discretized, becomes a *walk* in graph theory terminologies, since it may visit a vertex multiple times. In this paper, we abuse the term *path* slightly to denote both a continuous function $\tau : [t_0, t_f] \to W$ and the corresponding walk in a discrete graph.

for each element of $\mathscr{C}_p$ and for each location guarded by an occupancy sensor. For the example from Fig. 2, the collection of vertices is

$$V = \{A, B, C, o_1, o_2, b_{1u}, b_{1d}, b_{2l}, b_{2r}\},$$

in which b_{1u}, b_{1d} are the upper and lower sides of b_1, respectively (these two sides are naturally obtained if a beam is represented as two oppositely oriented edges, as commonly used in computations involving polygons). Similarly, b_{2l}, b_{2r} are the left and right sides of b_2.

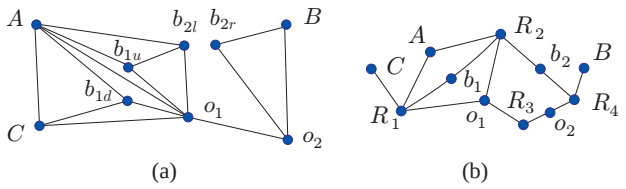

Fig. 3 a) The connectivity graph of the example given in Fig. 2. b) An alternative connectivity graph including connected components of W_{free} as vertices.

To connect the vertices, we need to obtain the connectivity of the workspace algorithmically, treating the regions occupied by elements of $\mathscr{C}_p$ and $\mathscr{C}_s$ as obstacles. Denote the workspace excluding these obstacles as W_{free}. Determining the connectivity of W_{free} is equivalent to finding the connected components of W_{free}. We call the subroutine that does this BUILDCONNECTIVITYGRAPH but omit the code since it is a fairly standard procedure [3]. Applying this procedure to our example yields the connectivity graph $G = (V, E)$ given in Fig. 3(a). We point out that there are other choices in constructing the connectivity graph. For example, following an (more natural) equivalence class approach, we may alternatively build the graph based on how regions R_1 through R_4 are connected (Fig. 3(b)). We may further treat sensors and rooms as directed edges. There are no fundamental differences between these choices for our purpose: Although the later two provide simpler graphs, slightly more sophisticated graph search routines would then be needed.

Algorithm 1 GETSUBGRAPH

Input: $G = (V, E)$, the start vertex s, $\mathscr{C}_p$, and goal vertices V_G
Output: $G' = (V', E')$, the part of G that is reachable from s

1: $V_C \leftarrow \mathscr{C}_p \cup \{s\}$
2: **return** GETREACHABLESUBGRAPH(G, s, V_C, V_G)

[3] One efficient way of doing this is to apply a cell decomposition procedure (see [19], Chapter 6), such as vertical cell decomposition [7], to W_{free} and then combine the cells that share borders. For more details, please refer to the extended version of the paper at `http://msl.cs.uiuc.edu/~jyu18/wafr10/full.pdf`

Algorithm 2 GETREACHABLESUBGRAPH

Input: $G = (V,E)$, s, V_C, V_G
Output: $G' = (V',E')$

```
1: for all edges (v_i, v_j) ∈ E such that v_i, v_j ∈ V_C do
2:     add (v_i, v_j) to E'        //  V' is also updated.
3: end for
4: G' ← CONNECTEDCOMPONENT(G', s)
5: if V_G is not empty then
6:     for all v_i, v_j such that v_i ∈ V', v_j ∈ V_G do
7:         if (v_i, v_j) ∈ E then
8:             add (v_i, v_j) to E'
9:         end if
10:    end for
11:    V' ← V' ∪ V_G
12: end if
13: return G'
```

With G constructed, we can now explore the extra information provided by the observation history: The relative timing of sensor recordings. This information essentially partitions G into different pieces at different time instances. In this section we focus on the case of workspace with a single agent. In the observation history given in (2), b_1 is the first sensor that is set off. This means that at the time right before t_1 when the sensor is activated, the agent must be at either b_{1u} or b_{1d}. During the time interval $[t_0,t_1)$, since b_2, o_1, and o_2 are inactive, they act as obstacles. The part of G that the agent may travel during $[t_0,t_1)$ is then given by G_1 in Fig. 4, in which A is the start vertex and b_{1u}, b_{1d} are the possible goal vertices. Vertex B does not appear in G_1 because it is not reachable. Similarly, we obtain the sub-

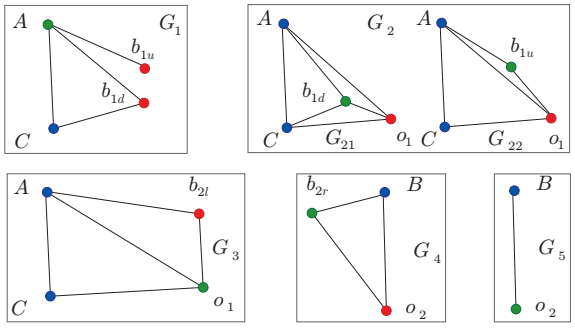

Fig. 4 The subgraphs of G induced by the sensor observation history. The green vertices are possible start positions and the red vertices are possible goal positions.

graphs of G during time intervals $(t_1,t_2),(t_3,t_4),(t_4,t_5),(t_6,t_f]$ as G_2 through G_5 in Fig. 4, respectively. Graph G_2 has two parts since there are two possible start vertices. Note that when the start and goal vertices in these subgraph correspond to

sensor vertices, they can be visited only once as the start vertex or the goal vertex. The pseudocode is given in GETSUBGRAPH (Algorithm 1). The algorithm calls the subroutine GETREACHABLESUBGRAPH(G,s,V_C,V_G) (Algorithm 2), which returns the part of G reachable from s, passing only vertices in V_C. If V_G is not empty, then a path from s must also end at vertices of V_G. We separate this subroutine since it will be reused. In Algorithm 2, subroutine CONNECTEDCOMPONENT(G,s) returns the connected component of G containing s. We note that, although it is possible to work with G directly instead of working with these subgraphs, they will be helpful in understanding the algorithm and in complexity analysis. Moreover, it can be a good heuristic to build these subgraphs to restrict search in problems with large workspaces.

The correctness of Algorithm 1 through 2 is by construction, which is straightforward to verify. We now give an estimate of the worst case performance of these algorithms. Let W_{free} have an input size of n_w, BUILDCONNECTIVITYGRAPH has time complexity $O(n_w^2)$. In subroutine GETREACHABLESUBGRAPH, the subroutine for obtaining connected components takes time linear in n_w [16]. The complexity is then decided by the for loop at line 6 and the membership check at line 7, which takes no more than $O(|V_G|n_w \lg n_w)$ in total.

4 Validating a Single Agent's Story Against an Observation History of a Single Agent

When there is a single agent in the workspace, every sensor recording is triggered by that agent. In this case, supposing that we have the subgraphs of G, the rest of the work becomes searching through these graphs, one by one, for a path that agrees with the agent's story. A straightforward approach is to connect one subgraph's goal vertices to the next subgraph's start vertices and perform an exhaustive search through paths to see whether there are matches. Such naive algorithms are not scalable, however, since every beam detector can require connecting the subgraphs in two ways (for example, G_{21}, G_{22} in Fig. 4). The number of search paths through the subgraphs is then exponential in the number of sensor recordings on average. In the worst case, breadth-first or depth-first search through all these graphs may take an exponential amount of time.

To organize the search more efficiently, we first connect the subgraphs to get a better understanding of the topology of the graph to be searched. To make the structure more explicit for search, we also make the subgraphs directed. Doing this to all sensing induced subgraphs of G yields the graph illustrated in Fig. 5. Denote this graph G_s. The problem of validating $\mathbf{p}$ against $\mathbf{r}$ becomes searching through G_s for a path $\mathbf{p}'$ such that, after deleting the vertices corresponding to sensors from $\mathscr{C}_s$, $\mathbf{p}'$ is exactly $\mathbf{p}$. We observe that, since G_s contains at most $2(m+1)$ copies of G, any element of $\mathscr{C}_p \cup \mathscr{C}_s$ cannot appear more than $O(m)$ times in G_s. This observation indicates it may be possible to apply the principles of dynamic programming to partition of the search problem into subproblems: Each subproblem is validating a tail $(p_i,\ldots,p_n)$ of $\mathbf{p}$, starting from a subset of vertices of G_s corresponding to p_{i-1}.

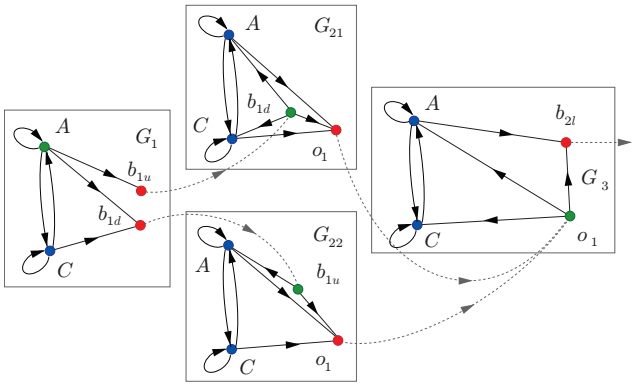

Fig. 5 Part of the composite graph G_s built from the connectivity graph G and sensor observation history $\mathbf{r}$.

The total number of subproblems per p_i is $O(m)$; if going from one subproblem to a smaller subproblem takes polynomial amount of time, then the total time spent on searching G_s is also polynomial.

This turns out to be the case for our problem. Before formally introducing the algorithm, we illustrate how it operates with the provided example. We write the agent's story compactly as $\mathbf{p} = ACBAC$. Since agent x starts in A, we are done with $p_1 = A$, leaving $CBAC$ to validate. For $p_2 = C$, it is possible to reach from A in G_1 to copies of C in G_1, G_{21} (passing b_{1u}, b_{1d}) , and G_3 (passing b_{1d}, b_{1u}, o_1). The copy of C in G_{22} is not directly reachable from A in G_1, passing only vertices from sensors. We may write the three subproblems as (For $P \in \mathscr{C}_p$, P_{G_i} denotes that the copy of P is from the subgraph G_i. For example, A_{G_1} denotes the copy of A from the subgraph G_1):

$$\begin{aligned} A_{G_1} C_{G_1} &\big| BAC, \\ A_{G_1} b_{1u} b_{1d} C_{G_{21}} &\big| BAC, \\ A_{G_1} b_{1d} b_{1u} o_1 C_{G_3} &\big| BAC. \end{aligned}$$

Since there are multiple subproblems for $p_3 = B$, going through these subproblems individually may introduce a factor of $O(m)$ per problem; there can be $O(m)$ subproblems, which will contribute a factor of $O(m^2)$ to the overall running time. To avoid this, we again use the sequential nature of G_s. Instead of processing each subproblem individually, we process all of them together, staged at each G_j. For our example, the first subproblem starts with the copy of C in G_1: It is possible to go through b_{1d}, b_{1u} and get to o_1. We now pick up the second subproblem and see that it is possible to go from C in G_{21} to o_1 as well. At this point, the first two subproblems collapse into a single subproblem. Going into G_3, we pick up the third subproblem. For the copy of C in G_3, since it must pass A to reach B, this subproblem dies; we are left with a single subproblem to reach B from b_{21} in G_3. Following above procedure, we obtain two subproblems after processing $p_3 = B$:

$$A_{G_1} b_{1u} b_{1d} C_{G_{21}} o_1 b_{21} b_{2r} B_{G_4} \Big| AC,$$
$$A_{G_1} b_{1u} b_{1d} C_{G_{21}} o_1 b_{21} b_{2r} o_2 B_{G_5} \Big| AC.$$

Note that we do not keep all valid paths in this search; doing so will require space exponential in m. After all of **p** is processed, if some subproblems survive, then **p** is consistent with **r**; any surviving subproblem also provides a feasible path.

Algorithm 3 VALIDATEAGENTSTORY

Input: G, $\mathbf{p} = (p_1, \ldots, p_n), \mathbf{r} = (r_1, \ldots, r_m)$
Output: **true** if **p** is consistent with **r**, **false** otherwise

```
 1: V_I ← {p_1}
 2: for j = 1 to m+1 do
 3:     initialize V_G as an empty set
 4:     if ISDEACTIVATION(r_j) then
 5:         continue
 6:     end if
 7:     if (j ≠ m+1) then
 8:         V_G ← SENSORVERTICES(r_i)
 9:     else
10:         empty V_G
11:     end if
12:     G_j ← GETSUBGRAPH(G, V_I, C_p, V_G)
13:     V_I ← V_G
14: end for
15: G_s ← CHAIN(G_1, ..., G_{m+1})
16: initialize V_s, V_s' as empty sets of two tuples
17: V_s ← {(p_1, 1)}        //    A two tuple is a vertex of G_s
18: for i = 2 to n do
19:     for j = 1 to m+1 do
20:         if (p_i, j) adjacent to (p_{i-1}, k) ∈ V_s for some k ≤ j then
21:             if i == n && j == m+1 then
22:                 return true
23:             end if
24:             add (p_i, j) to V_s'
25:         end if
26:     end for
27:     V_s ← V_s'; empty V_s'
28: end for
29: return false
```

The pseudocode is summarized in Algorithm 3. Subroutine ISDEACTIVATION(r) returns true only if r is the deactivation of an occupancy sensor. The subroutine SENSORVERTICES(r) returns the vertices of G induced by the sensor in a sensor recording r. The subroutine CHAIN$(\ldots)$ connects all input graphs sequentially based on sensor crossings, which is trivial to implement. In the code, we use (p_i, j) to denote the the copy of p_i in subgraph G_j. The correctness of VALIDATEAGENTSTORY follows from its construction based on dynamic programming, which we briefly corroborate. After each p_i is worked on, there are up to $O(m)$ subproblems since there are no more than $O(m)$ copies of p_i in G_s. Because the further observation

that G_s is sequential, the suproblems for each p_i can be processed in a single pass of G_s. We make $O(m)$ calls to GETSUBGRAPH, which takes $O(mn_w \lg n_w)$ total time (Since $|V_G| \leq 2$ in calls to GETSUBGRAPH). Going through the for loops, it is straightforward to get that the rest of the algorithm has complexity no worse than $O(n \cdot m \lg n_w) = O(nm \lg n_w)$. The worst case running time is then upper bounded by $O(m(n+n_w)\lg n_w)$.

As mentioned in the problem formulation, we have assumed that $\mathscr{C}_p$ and $\mathscr{C}_s$ do not overlap in $\mathbb{R}^2$. These are not included in the above algorithm to avoid complicating the presentation. What if some $p \in \mathscr{C}_p$ and $s \in \mathscr{C}_s$ do overlap? There are several subcases. If the regions of p, s coincide (for example, there may be an occupancy sensor in room A), this essentially breaks the problem into several smaller problems, to which the above algorithm applies. If $s \subsetneq p$, agent x then must go through p to reach s, in which case we can build G to make s a vertex connecting to p only. The same applies if $p \subsetneq s$. In the last case s, p partially overlap but do not include each other; we can treat s, p as three regions: $s \backslash p$ as an sensor, $p \backslash s$ as a room, and $s \cap p$ as a fully overlapping sensor and room (this case only happens to occupancy sensors, not beam detectors). This will split the verification problem into several subproblems, which may induce exponential growth in running time. However, the last case is not likely to often happen since occupancy sensors are usually placed to guard an entire room. We can also minimize such a problem by carefully placing the sensors.

5 Validating A Single Agent's Story Against an Observation History of Multiple Agents

When there are multiple agents (an unknown number) in the workspace, two complications arise. First, as mentioned in Section 2, when multiple agents are in the workspace, there can be many agents in the region monitored by an occupancy sensor during one of its activation-deactivation time interval. Effectively, this allows any agent to temporarily go through the region monitored by an activated occupancy sensor. This suggests that the recordings from occupancy sensors should only be treated as events that change the connectivity of the environment. That is, whether agent x is the agent that triggered the activation/deactivation of an occupancy sensor is not relevant. Second, agent x may not be responsible for all beam detector recordings. Instead, it may trigger any subsequence of sensor recordings. For an observation history sequence of length m, there are up to $O(2^m)$ possible subsequences; agent x may have triggered any one of these subsequences, but not the rest of $\mathbf{r}$.

To overcome these difficulties, we start by examining how the composite graph G_s changes. As analyzed above, only beam detector events need to be considered for agent x. For occupancy sensors, we maintain an active list as $\mathbf{r}$ is processed; additional processing is needed only when deactivation of an occupancy sensor happens. To illustrate the procedure for building the composite graph, we use $\mathbf{p}$ from (1) and $\mathbf{r}$ from (3). Starting with the first beam detector recording, (b_1, t_1), if agent x is responsible for it, then the reachable part before b_1 is crossed is the same as G_1 from

Fig. 4. To emphasize that this graph is built from t_0 to t_1, we denote it as G_1^0. The next two recordings in $\mathbf{r}$ are activations of o_1 and o_2. Since there are only activations, which only cause more locations of the environment to become reachable, we store these in the active occupancy sensor list and continue.

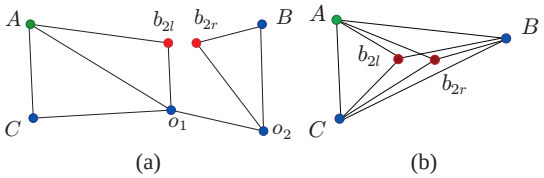

Fig. 6 a) One possible connectivity subgraph, G_4^0, during (t_4,t_5) when mulitple agents are in the workspace. b) Updated connectivity subgraph reflecting whether two vertices are reachable without triggering additional sensor recordings.

Algorithm 4 GETSUBGRAPHMULTI

Input: $G=(V,E)$, the start vertex s, $\mathscr{C}_p$, the active occupancy sensors O, and goal vertices V_G.
Output: $G'=(V',E')$, the part of G that is reachable from s.

1: $V_C \leftarrow \mathscr{C}_p \cup O \cup \{s\}$
2: $G' \leftarrow$ GETREACHABLESUBGRAPH(G,s,V_C,V_G)
3: **for all** $o \in (O \cap V')$ **do**
4: add to E' an edge between each pair of o's neighbors
5: remove o from G'
6: **end for**
7: **return** G'

For the next recording, (b_2,t_4), three subgraphs need to be built, one start from A, one start from b_{1u}, and one start from b_{1d}. Following the naming convention of G_1^0, these should have names G_4^0, G_4^{11}, and G_4^{12}, respectively. To build G_4^0, we need to keep vertices $\{A,b_{2l},b_{2r},o_1,o_2\}$. We also add $\{B,C\}$ since these are vertices of $\mathscr{C}_p$ that are reachable from $\{A,b_{2l},b_{2r},o_1,o_2\}$ without crossing additional sensors. This gives us the subgraph in Fig. 6(a). To facilitate searching, for each pair of neighbors of an active occupancy sensor, we add an edge between them and remove the occupancy sensor vertex, which yields the graph in Fig. 6(b). The pesudocode for building this subgraph, GETSUBGRAPHMULTI, is given in Algorithm 4. Assuming we have obtained G_4^{11} and G_4^{12} similarly, the next sensor recordings is (o_2,t_5), which corresponds to the deactivation of o_2. For this event, we need to create five subgraphs starting from $A,b_{1u},b_{1d},b_{2l},b_{2r}$ with names $G_5^0,G_5^{11},G_5^{12},G_5^{41},G_5^{42}$, respectively. Since during $[t_5,t_f]$, no new locations of G become reachable and no other beam sensor recordings happen, this part of $\mathbf{r}$ can be ignored. After connecting all these subgraphs based on sensor crossings, we obtain the composite graph G_s as illustrated in Fig. 7.

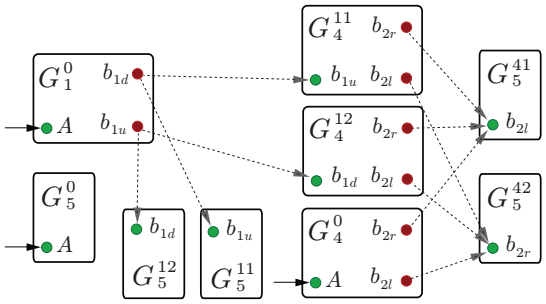

Fig. 7 Sketch of the composite graph G_s.

Before continuing with searching G_s, we make one observation from above graph building procedure: Since each sensor recording may cause up to $O(m)$ new subgraphs to be built, up to $O(m^2)$ subgraphs may be built altogether. This is the number of subgraphs in G_s since each subgraph appears once in G_s. To search through G_s for a path matching $\mathbf{p}$, the same strategy from VALIDATEAGENTSTORY can be applied. That is, a dynamic programming approach can be used in which a subproblem is a tail of $\mathbf{p}$ and a location in G_s. Since there are no more than $O(m^2)$ copies of p_i in G_s, there can be at most $O(m^2)$ subproblems after each p_i is processed. Similar to VALIDATEAGENTSTORY, during the processing of each p_i each subgraph only needs to be considered once. This limit the time complexity of searching G_s at $O(nm^2 \log m_w)$. To get the total time complexity, we need the time for building G_s, which is m^2 times the cost of the subroutine GETSUBGRAPHMULTI. The running time of GETSUBGRAPHMULTI is determined by the loop at lines 3 through 6, which takes $O(m_w^3)$ time. This yields the overall time complexity $O(m^2(n \log m_w + m_w^3))$. Since the algorithm operates much like VALIDATEAGENTSTORY, we omit the pesudocode.

6 Conclusion and Open Questions

We introduced a decision problem in which a story of an autonomous agent is validated against the observation history gathered by simple sensors placed in the same environment. We showed that sensor recordings act as temporary obstacles that constrain agents' movements in the environment, effectively creating a sequence of connectivity subgraphs of the environment. Based on this observation, we designed polynomial time algorithms that decide whether the agent's story is possible given the observation history and retrieves a possible path if one exists.

As a first attempt at a new problem, more questions are opened than answered. Some immediate ones are: What if the agent can only recall a partial story? In the multiple agent case, what if every agent's story is valid but the combined story is inconsistent? In letting the agents move arbitrarily fast, we only addressed one discrete aspect of a general problem in this paper. A natural next step is to work on

a kinematic agent model with bounded control inputs, which seems to require more careful analysis of the continuous state space and much richer behaviors. Since an agent's speed is limited, some plausible paths the agent could have taken will now be ruled out. Continuous state space also makes it possible to study optimality: a network of sensors can potentially detect whether an agent is performing its task most efficiently, in terms of time or distance traveled. This becomes more relevant as system designs become more and more complex; sometimes it is challenging to just get a feasible plan [11, 18]. Another interesting direction is to study optimal sensor placements for the detective task, which was discussed to some extent in [24, 28]. Furthermore, the story in this paper only contains "when" and "where". It is an intriguing open problem to check stories involving "what", "how", and "why"? Logic seems essential in investigating these elements of a story.

Probabilistic formulations of the story validation problem may also be fruitful to explore. The agents in this paper are assumed to be nondeterministic in that the algorithms treat all possible paths equally. In a real environment, however, a typical agent usually does have preferences when multiple choices are present. Models considering the transition probability of an agent from room to room could then uncover the most likely path(s) taken by the agent. If none of the feasible paths taken by the agent are probable, the agent could still be considered as misbehaving. Probability can also be introduced into sensor observation history. Sensors, no matter how reliable they are, may have false positives and false negatives. For example, a beam sensor may misfire (triggered by a mouse for instance) with small probability. This implies that sensors, unlike walls, may be better treated as "soft" obstacles.

Acknowledgments The authors thank Amir Nayyeri, Max Katsev, and the anonymous reviewers for helpful comments. This work was supported in part by NSF grants 0904501 (IIS Robotics) and 1035345 (Cyberphysical Systems), DARPA SToMP grant HR0011-05-1-0008, and MURI/ONR grant N00014-09-1-1052.

References

1. R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. In *Proceedings of the IEEE*, pages 971–984, 2000.
2. E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. pages 21–31. Springer, 2000.
3. J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. pages 150–161. ACM Press, 2002.
4. Y. Baryshnikov and R. Ghrist. Target enumeration via integration over planar sensor networks. In *Proceedings of Robotics: Science and Systems IV*, Zurich, Switzerland, June 2008.
5. R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
6. J. Castellanos, J. Montiel, J. Neira, and J. Tardós. The SPmap: A probabilistic framework for simultaneous localization and mapping. *IEEE Transactions on Robotics & Automation*, 15(5):948–953, 1999.
7. B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C. K. Yap, editors, *Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
8. P. Cheng and V. Kumar. Sampling-based falsification and verification of controllers for continuous dynamic systems. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2006.

9. H. Choset and K. Nagatani. Topological simultaneous localization and mapping (T-SLAM). *IEEE Transactions on Robotics & Automation*, 17(2):125–137, 2001.
10. A. Chutinan and B. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, January 2003.
11. D. C. Conner, H. Kress-gazit, H. Choset, A. A. Rizzi, and G. Pappas. Valet parking without a valet. In *IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2007.
12. M. Dehn. *Papers on Group Theory and Topology*. Springer-Verlag, Berlin, 1987.
13. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba. A solution to the simultaneous localisation and map building (SLAM) problem. *IEEE Transactions on Robotics & Automation*, 17(3):229–241, 2001.
14. D. B. A. Epstein, M. S. Paterson, G. W. Camon, D. F. Holt, S. V. Levy, , and W. P. Thurston. *Word Processing in Groups*. A. K. Peters, Natick, MA, 1992.
15. Q. Fang, F. Zhao, and L. Guibas. Counting targets: Building and managing aggregates in wireless sensor networks. Technical Report P2002-10298, Palo Alto Research Center, June 2002.
16. J. Hopcroft and R. Tarjan. Efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6):372–378, 1973.
17. W. Kim, K. Mechitov, J. Choi, and S. Ham. On target tracking with binary proximity sensors. In *ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 301–308. IEEE Press, 2005.
18. H. Kress-gazit, G. E. Fainekos, and G. Pappas. Wheres waldo? sensor-based temporal logic motion planning. In *Proceedings IEEE International Conference on Robotics & Automation*, pages 3116–3121, 2007.
19. S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Also available at http://planning.cs.uiuc.edu/.
20. S. M. LaValle. Filtering and planning in information spaces. Technical report, Department of Computer Science, University of Illinois, Oct 2009.
21. M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings AAAI National Conference on Artificial Intelligence*, 1999.
22. R. Parr and A. Eliazar. DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. In *Proceedings International Joint Conference on Artificial Intelligence*, 2003.
23. N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Proceedings Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer, 2006.
24. B. S. Y. Rao, H. F. Durrant-Whyte, and J. S. Sheen. A fully decentralized multi-sensor system for tracking and surveillance. *International Journal of Robotics Research*, 12(1):20–44, February 1993.
25. N. Shrivastava, R. Mudumbai, U. Madhow, and S. Suri. Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms. In *Proc. 4th Internat. Conf. on Embedded Networked Sensor Systems, 2006*, pages 251–264. ACM Press, 2006.
26. 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.
27. S. Thrun, W. Burgard, and D. Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Machine Learning*, 31(5):1–25, April 1998.
28. B. Tovar, F. Cohen, and S. M. LaValle. Sensor beams, obstacles, and possible paths. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2008.
29. 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.
30. J. Yu and S. M. LaValle. Tracking hidden agents through shadow information spaces. In *Proceedings IEEE International Conference on Robotics & Automation*, 2008.
31. J. Yu and S. M. LaValle. Probabilistic shadow information spaces. In *Proceedings IEEE International Conference on Robotics & Automation*, 2010.