2011 IEEE International Conference on Robotics and Automation
Shanghai International Conference Center
May 9-13, 2011, Shanghai, China

Story Validation and Approximate Path Inference with a Sparse Network of Heterogeneous Sensors

Jingjin Yu
jyu18@uiuc.edu
Department of Electrical and Computer Engineering
University of Illinois
Urbana, IL 61801 USA

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

***Abstract*— Given a story from an agent (sensor outputs from a robot or a tale told by a human) and recordings from a spare network of heterogeneous sensors, this paper provides efficient algorithms that validate whether it is possible to reconstruct a path compatible with the sensor recordings that is also "close" to the agent's story. In solving the proposed problems, we show that effective exploitation of a unique finite automaton structure yields time complexity linear in both the length of the story and the length of the sensor observation history. Besides immediate applicability towards security and forensics problems, the idea of behavior validation using external sensors also appears promising in complementing design time model verification.**

I. Introduction

In [33], we introduced and solved a problem in which an agent's account (a *story*) of its path in an environment is validated against recordings from a sparse network of sensors deployed in the same environment. In that work, an agent's story is required to be error-free and has start/end time matching those of the sensor recordings'. Such formulations are restrictive for two reasons: 1. Even a truthful agent is likely to introduce errors in recalling a long story, especially when the agent is a human; 2. Requiring matching start/end time is hard to guarantee. Moreover, when an agent's story is not consistent with an observation history, the algorithms from [33] do not provide an alternative path for the agent that is "close" to the agent's story. In this paper, we provide highly efficient algorithmic solutions to address and remove all these limitations[1].

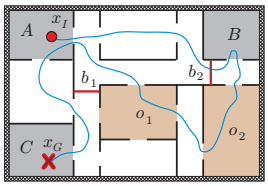

Fig. 1. A workspace with rooms A, B, C, occupancy sensors o_1, o_2 and beam detectors b_1, b_2. The curve connecting the start (circle) and goal (cross) locations is a possible agent path for the story A, B, A, C, which triggers the sensor recordings b_2, o_2, o_1, in that order.

Our study complements the verification of robotic system design. From the perspective of modeling, an agent can be viewed as a hybrid system: It goes from room to room to carry out tasks, changing its operation modes along the way. Since the verification of an autonomous system with continuous state space and control input is undecidable [2], it is desirable to have external measures to keep in check the unverified portion of such systems. Even for hybrid systems with provably correct designs, such as autonomous robots or cars [8], [15] based on specifying high level tasks using General Reactivity(1) formulas [22], a malfunction could occur due to mechanical errors or deliberate tempering. Therefore, automatically monitoring hybrid systems with external sensors can be an effective error correcting and safeguarding measure.

Sensor network based approaches have been applied to infer basic properties of moving bodies in its range. Networks of binary proximity sensors have been employed to track one or multiple moving bodies using various analytical tools [3], [14], [25], [26]. The task of counting multiple targets is also studied under different assumptions [4], [13]. In these works, the sensors' aggregate sensing range must cover the target(s) at all times, which we do not assume. When only subsets of an environment are guarded, *word problems in groups* [9], [12] naturally arise. For the setup in which targets moving inside a 2D region are monitored with a set of detection beams, target locations and path reconstruction up to homotopy were studied in [28]; few efficient algorithms are available.

In obtaining solutions to more general problems, it becomes clear that the story validation problem we raise can be transformed to the *string edit distance problem between a string and a regular language*, with first algorithmic solution appearing in [30]. Improvements on time and space requirements for algorithms solving such problems can be found in [31], [24], [19], [1]. For an overview of approximate string matching problems, see [20]. In viewing the resemblance of our problem to the questions asked in [3], [25], [28], our solution follows an *information space* approach [16], which requires the design of a combinatorial filter, similar to those in [17], [29], [32]. These combinatorial filters are minimalist counterparts to widely known Bayesian filters [5], [7], [11], [18], [21], [27].

The rest of the paper is organized as follows. Section II provides formulations of the three problems we study in this

[1]Additional scenarios can be found in the extended version available at http://msl.cs.uiuc.edu/~jyu18/icra11/full.pdf

paper. Section III briefly reviews the algorithmic solution to the base problem and apply it to solve the first of the three problems. Section IV relates our problem to the string edit distance problem. We then combine the gained insight with our special problem structure to provide algorithms for solving the other two problems with further efficiency gains, in Section V. We conclude with Section VI.

II. Problem Formulation

A. Workspace, Agent Stories, and Observation History

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 $\tau : [t_0 \ , \ t_f] \rightarrow W$. We are interested in a particular agent x which can be thought of as a *suspect*. Agent x provides a *story*, which is a sequence of locations it recalls along its path in increasing chronological order, $\mathbf{p} = (p_1, p_2, \ldots, p_n), p_i \subset W$. The story $\mathbf{p}$ can also be viewed as a string $p_1 \ldots p_n$ which is the notation we use most of the time. We assume that the unique elements of $\mathbf{p}$ are each a simply connected region with a polygonal boundary and pairwise disjoint. The set of all unique elements of $\mathbf{p}$ is denoted $\mathcal{C}_p$, which can be thought of as a set of rooms in W. To limit the number of questions that can be posed, we require that agent x starts from p_1 and ends in p_n. Unless otherwise stated in a particular problem, it is assumed that agent x recalls in $\mathbf{p}$ locations on its path correctly.

Let a subset of the workspace W be guarded by a set of occupancy sensors and beam detectors. The placement of sensors in W is unknown to the agents. An occupancy sensor o_i 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 (o_1, o_2 in Fig. 1). A data point recorded by o_i has two parts, an *activation*, $r_{oa} = (o_i, t_a)$, and a *deactivation*, $r_{od} = (o_i, t_d)$. Here 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 detector b_i, on the other hand, guards a straight line segment, $\ell \subset W$, between two edges of ∂W (b_1, b_2 in Fig. 1). A data point of such a sensor 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.

If we gather all sensor recordings (of the types r_{oa}, r_{od}, r_b) during a time interval $[t_0, t_f]$ and order them by time incrementally, an *observation history* is obtained. This sequence is unique since it is reasonable to assume that no two recordings happen at the exact same time. As we do not assume that an agent provides the exact time when it visits a location, the time in the sensor recording is also relative. Therefore, when we compose the observation history of the sensors, we may discard the time element of each sensor recording, keeping only their relative order. A simplified observation history can be written as (each s_i corresponds to the region covered by a sensor): $\mathbf{s} = (s_1, s_2, \ldots, s_m), s_j \subset W$. Similar to $\mathbf{p}$, we can write $\mathbf{s}$ as a string. The unique sensor regions from $\mathbf{s}$ are denoted as $\mathcal{C}_s$. As justified in [33], we assume that any two elements of $\mathcal{C}_p \cup \mathcal{C}_s$ are disjoint (in W).

B. Validation Problems

Given the above setup, we want to know whether a story is consistent with a sensor observation history. For example, if an agent starts from A in the workspace from Fig. 1, it cannot end up at B without triggering any sensor recordings. Our earlier work addressed the following validation problem (as well as a multiple agent extension):

Problem 1 (Exact Story, Matching Time Intervals) *Let there be a single agent in a workspace. The agent provides a story* $\mathbf{p}$ *for the time interval* $[t_0, t_f]$. *During the same time interval, the sensors in the workspace produce an observation history,* $\mathbf{s}$. *Validate whether* $\mathbf{p}$ *is consistent with* $\mathbf{s}$. *Extract a feasible path if the story is consistent.*

The formulation of Problem 1 is quite restrictive in at least two ways. First, requiring the story and the observation history to span the exact same time interval $[t_0, t_f]$ may not be always possible. For example, the sensor network may be inactive at some point due to daily operation schedule or system maintenance. We capture and generalize this case with the following problem:

Problem 2 (Exact Story, Mismatching Time Intervals) *Let there be a single agent in a workspace. The agent provides a story* $\mathbf{p}$ *for the time interval* $[t_0, t_f]$. *The sensors in the workspace produce an observation history,* $\mathbf{s}$, *for a time interval* $[t'_0, t'_f]$ *that overlaps with* $[t_0, t_f]$. *Validate whether* $\mathbf{p}$ *is consistent with* $\mathbf{s}$. *Extract a feasible path if the story is consistent.*

The second restriction is that a truthful agent, be it human or robot, may inevitably make mistakes in recalling a story. This scenario translates into the formulation below (For two strings/sequences $\mathbf{p}, \mathbf{p}'$, the expression $\mathbf{p} \leq \mathbf{p}'$ or $\mathbf{p}' \geq \mathbf{p}$ denotes that $\mathbf{p}$ is a subsequence of $\mathbf{p}'$):

Problem 3 (Partial Story, Matching Time Intervals) *Let there be a single agent in a workspace. The agent provides a story* $\mathbf{p}$ *for the time interval* $[t_0, t_f]$. *During the same time interval, the sensors in the workspace produce an observation history,* $\mathbf{s}$. *Let* $\mathbf{p}' \geq \mathbf{p}$ *be a sequence with elements from* $\mathcal{C}_p$. *Validate whether there exists a* $\mathbf{p}'$ *that is consistent with* $\mathbf{s}$. *If such a story* $\mathbf{p}'$ *exists, find one of shortest length.*

Problem 3 motivates a more general case: The agent, even with best effort, may add locations it has not visited (insertion), miss locations it has visited (deletion), or report some locations it has visited incorrectly (substitution). Calling each of the three possibilities to introduce an error as an *edit*, we obtain the following problem:

Problem 4 (Error in Story, Matching Time Intervals) *Let there be a single agent in a workspace. The agent provides a story* $\mathbf{p}$ *for the time interval* $[t_0, t_f]$. *During the same time interval, the sensors in the workspace produce an observation history,* $\mathbf{s}$. *Let* $\mathbf{p}'$ *be a sequence with elements from* $\mathcal{C}_p$. *Find a* $\mathbf{p}'$ *that is consistent with* $\mathbf{s}$ *such that* $\mathbf{p}$ *can be obtained from* $\mathbf{p}'$ *with the least number of edits.*

III. Baseline Algorithm and the Case of Mismatching Time Intervals

In this section we review the baseline algorithm for Problem 1, with the example from Fig. 2. Only the most essential elements from [33] are reproduced; readers may check that paper for a thorough explanation. After the recapitulation, we apply the algorithm to solve Problem 2.

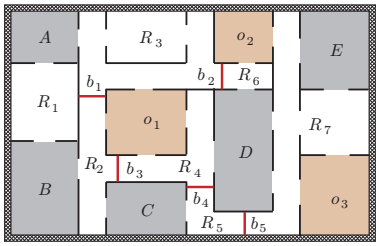

Fig. 2. A workspace with five beam detectors b_1 through b_5, three occupancy sensors o_1 through o_3, and five labeled rooms A through E. Thus, $\mathcal{C}_p = \{A, B, C, D, E\}$, $\mathcal{C}_s = \{b_1, b_2, b_3, b_4, b_5, o_1, o_2, o_3\}$. There are seven connected components R_1 through R_7 when regions guarded by sensors and rooms are treated as workspace obstacles.

A. Baseline Algorithm

When regions covered by sensors and rooms are not occupied, they act as obstacles. In the example, these obstacles partition the workspace into seven connected components, R_1 through R_7. Assuming that the workspace, rooms, and sensors are given to us as edge lists, we apply a cell decomposition procedure [6] to extract these connected components, which is then transformed into a graph structure that captures the connectivity of the rooms and sensors. We call this graph G the *connectivity graph*, which extracts all necessary information needed for validating an agent's story. The connectivity graph for our example is given in Fig. 3. Each beam detector has two vertices in the graph to represent its two sides; these are marked differently from other vertices.

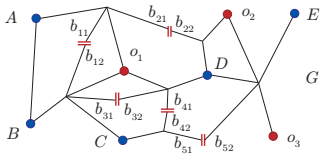

Fig. 3. The connectivity graph G for the example given in Fig. 2.

With the construction of the connectivity graph G, Problem 1 effectively becomes graph search. At any given moment, only part of G is available, depending on where the agent is and which sensor is active. For story $\mathbf{p} = p_1 \dots p_n$ and observation history $\mathbf{s} = s_1 \dots s_m$, since the search must march through both strings forward, a position in the two strings and a location in the graph G define a subproblem that is just the same as the initial problem. For example, a subproblem may be validating $p_i \dots p_n$ against $s_j \dots s_m$ starting from room E. There is clearly only a polynomial number of subproblems; if the time it takes to go from one subproblem to a smaller one is also polynomial, then the overall search is efficient in input size.

It turns out that this is indeed the case for Problem 1. For illustration, suppose that the story is $\mathbf{p} = ABDEC$ and the sensor observation history is $\mathbf{s} = b_1 b_3 o_2 o_2 b_4$. First, we construct a graph based on G and $\mathbf{s}$ that captures all possible agent paths for a fixed $\mathbf{s}$. Since agent x starts at A and must first pass b_1, before b_1 is triggered, the part of G that is available is G_1 (see Fig. 4(a)). Similarly, the next part of G available, after agent x's passing of b_1 but not b_3, is G_2 (see Fig. 4(b)). For all such G_j's, agent x must pass through them sequentially. Observing this, G_j's are chained together to get a composite graph G_s. For example, b_{11}, b_{12} in Fig. 4(a) are connected to b_{12}, b_{11} in Fig. 4(b), respectively.

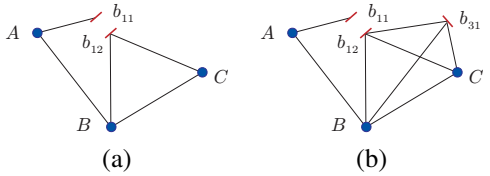

Fig. 4. a) The available subgraph of the example from Fig. 2 when b_1 is the first sensor recording. b) The available subgraph of the example from Fig. 2 for the recording $b_1 b_3$.

Algorithm 1 ValidateAgentStory

Input: W_{free}, $\mathbf{p} = (p_1, \dots, p_n)$, $\mathbf{s} = (s_1, \dots, s_m)$
Output: **true** if $\mathbf{p}$ is consistent with $\mathbf{s}$, **false** otherwise

```
G ← BuildConnGraph(W_free)
for j = 1 to m + 1
  G_j ← SubG(G, j == 1?p_1 : s_{j−1}, j = m + 1?nil : s_j)
G_s ← Chain(G_1, ..., G_{m+1})
initialize V_s, V_s' as empty sets of two tuples
V_s ← {(p_1, 1)}    //    A two tuple is a vertex of G_s
for i = 2 to n
  for j = 1 to m + 1
    if (p_i, j) adjacent to (p_{i−1}, k) ∈ V_s for some k ≤ j
      if i == n&&j == m + 1
        return true
      add (p_i, j) to V_s'
  V_s ← V_s'; empty V_s'
return false
```

The rest of the algorithm is dynamic programming much like Dijkstra's algorithm[10] with a total complexity of $O(nm \lg n_w)$, in which n_w is the size of input W_{free}; the details can be found in [33]. The pseudocode is summarized in Algorithm 1. The subroutine BuildConnGraph(W_{free}) returns the connectivity graph G given an edge list representation of W_{free}. The subroutine SubG(G, v_1, v_2) returns the available part of G starting from v_1 and ending at v_2. Finally, Chain$(\dots)$ connects all input graphs sequentially. The time spent on these portion of the algorithm is $O(mn_w \lg n_w + n_w^2)$. If ValidateAgentStory returns true, which means $\mathbf{p}$ is consistent with $\mathbf{s}$, a possible path can be retrieved via backtracking the search process. We now apply Algorithm 1 to Problem 2.

B. Mismatching Time Intervals

Problem 2 has several subcases, depending on how $[t_0, t_f]$ and $[t_0', t_f']$ compare. Assuming that t_0, t_f, t_0', t_f' are pairwise different, the following six cases are possible: 1) $t_0 < t_f < t_0' < t_f'$; 2) $t_0 < t_0' < t_f < t_f'$; 3) $t_0' < t_0 < t_f < t_f'$;

4) $t_0 < t'_0 < t'_f < t_f$; 5) $t'_0 < t_0 < t'_f < t_f$; and 6) $t'_0 < t'_f < t_0 < t_f$. For the first and last cases, $[t_0, t_f]$ and $[t'_0, t'_f]$ are disjoint. In these cases, the sensors cannot possibly observe the agent during $[t_0, t_f]$; nothing needs to be done about $\mathbf{p}$ (as long as $\mathbf{p}$ does not conflict with itself). We are not yet done, because we still need to check whether an empty story is consistent with $\mathbf{s}$. This can be done using VALIDATEAGENTSTORY (The search portion only takes $O(m \lg n_w)$ time).

The second case is $t_0 < t'_0 < t_f < t'_f$, which means that a later portion of agent story overlaps with an earlier portion of the sensor observation history. This can be handled by running VALIDATEAGENTSTORY algorithm n times. For the i-th run, the story is $(p_i, \ldots, p_n)$ and the sensor observation history is $\mathbf{s}$. If any of the runs returns true, then the story is valid; otherwise the story is inconsistent. We note that the search can be arranged more efficiently by working on the same p_i's of different runs at the same time; thus, the time complexity for this case remains $O(nm \lg n_w)$.

For the third case, $t'_0 < t_0 < t_f < t'_f$, the story $\mathbf{p}$ may start in the middle of G_s. VALIDATEAGENTSTORY can be easily modified to handle this: Instead of starting in G_1, we now allow the search to start at (p_1, j) for all applicable j's. If p_n is ever reached in the search, the modified algorithm should report that $\mathbf{p}$ is consistent with $\mathbf{s}$. The time complexity again remains the same. Following along the same lines, we can decide cases four and five.

IV. THE COMPOSITE AUTOMATON STRUCTURE

Although solution to Problem 2 is a relatively straightforward extension of the VALIDATEAGENTSTORY algorithm, it is not immediately clear whether the approach applies to Problem 3, 4, and more general cases. Part of the difficulty comes from the composite graph G_s: It appears to have more structure than a standard directed graph with $O(mn_w)$ vertices. On the other hand, we observe that VALIDATEAGENTSTORY operates much like an automaton in the sense that it processes $\mathbf{p}$ sequentially and either accepts or rejects. This prompts us to ask: Can we turn G_s into an automaton and apply results from Automata Theory to tackle our problem? We answer the first part of this question in this section and delay the second part to the next.

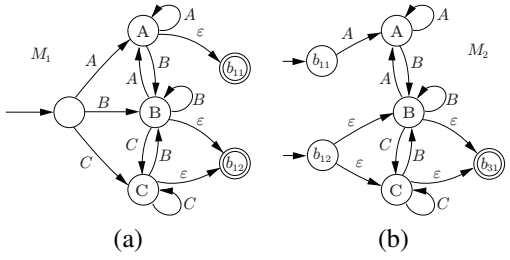

Fig. 5. a) The corresponding NFA when b_1 is the first sensor crossed. b) The NFA for consecutive sensor recordings $b_1 b_3$.

The conversion of a connectivity graph G and a observation history $\mathbf{s}$ into finite automata is relatively straightforward. For each pair of consecutive sensor recordings s_i, s_{i+1} (except when $i = 0, m$), we isolate the part of G that can reach vertices of s_{i+1} from vertices of s_i and convert it to an automaton. When $i = 0$, we let the start state of the automaton transit to all vertices in $\mathcal{C}_p$ and when $i = m$, we let all vertices in $\mathcal{C}_p$ be acceptance states. These automata are then chained together to give a composite automaton. As an example, the subgraphs from Fig. 4(a), (b) yield the NFAs given in Fig. 5(a), (b), respectively.

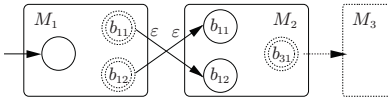

Fig. 6. Sketch of the composite automaton for the observation history string $b_1 b_3 o_2 o_2 b_4$.

For an observation history $\mathbf{s}$ of length m, $m+1$ automata are obtained. Denote these automata M_1 through M_{m+1}. For implementation purpuse, we connect all of M_{m+1}'s states to a single acceptance state F via an ϵ transition. It is straightforward to observe that a story $\mathbf{p}$ is consistent with $\mathbf{s}$ if and only if the story string can be partitioned into pieces $P_1, \ldots, P_{m+1}$ such that P_i is accepted by M_j. Alternatively, if we connect M_1 through M_{m+1} together to get a *composite automaton*, M (Fig. 6), then $\mathbf{p}$ must drive M from start state to F. With minimal effort, VALIDATEAGENTSTORY can be modified to work with the composite automaton M, which will be able to resolve Problem 1 and 2, keeping the dynamic programming framework intact. Although it does not make the algorithm more efficient, the approach provides an alternative interpretation of these problems: Searching $\mathbf{p}$ through G_s now becomes simulating M over $\mathbf{p}$, making these problems tests of whether a string belongs to a regular language.

V. SOLVING GENERAL PROBLEMS

For Problem 1 and 2, only a few stories are checked against an observation history $\mathbf{s}$. This is no longer the case for Problem 3 and 4 since an infinite number of possible stories must be checked in the process of solving these problems. This is where the automaton structure comes in: If we simulate a story $\mathbf{p}$ over an automaton, we know that on seeing p_i, there are a set of fixed states the automaton can be in. As long as we maintain these finite number of states, an infinite number of string patterns can be handled. In fact, existing results from Automata Theory have completed part of the job for us: Problem 3 and 4 can be viewed as the *string edit distance problem between a string and an automaton* (with the string being $\mathbf{p}$ and the automaton being M), for which efficient algorithms exist.

Our job is not done, however. Exisitng algorithms assume that the string $\mathbf{p}$ and the automaton M are the inputs. In our problem, the automaton M is an intermediate input built from $\mathbf{s}$ and G_s; thus, we may be able to do better. In this section, we explore how the sequential nature of M allows us to subdivide Problem 3 more effectively. After solving Problem 3, we sketch at high level how the same structure helps solve Problem 4 as well.

A. Partial Story

We continue to work with the example from Fig. 2 and assume that the story is $\mathbf{p} = ABDEC$ and the observation

history is $\mathbf{s} = b_1 b_2 o_2 o_2 b_4$. Suppose that we obtain automata M_j's as well as M from $\mathbf{s}$ already. With the analysis from Section IV, Problem 3 becomes finding a shortest $\mathbf{p}' \geq \mathbf{p}$ such that $\mathbf{p}'$ is accepted by M. The problem may seem a bit daunting at first glance: An infinite number of $\mathbf{p}'$ needs to be examined, as long as $\mathbf{p}' \geq \mathbf{p}$.

However, any consistent $\mathbf{p}'$ must drive the composite automaton M to an accepting state. If we assume that some $\mathbf{p}' \geq \mathbf{p}$ is one shortest such that is accepted by M, then it must have the format $\mathbf{p}' = \omega_1 A \omega_2 B \omega_3 D \omega_4 E \omega_5 C \omega_6$, in which ω_i's denote the parts missing from $\mathbf{p}$. Since $p_1 = A$, we search M and find shortest string from the start state of M to all copies of A in M (We may denote the copy of A from M_j as (A, j)); there are up to $m+1$ of these. Denoting these strings as $\sigma(1, 0, j)$. For each j there may be multiple such strings of the same shortest length; in this case any of these will do. By the principle of dynamic programming, $|\omega_1| + 1 = |\sigma(1, 0, j)|$ for some j. Moving to $p_2 = B$, for each (B, k) let us denote a shortest string taking M from some (A, j) to (B, k) as $\sigma(2, j, k)$. Among these $\sigma(2, j, k)$'s for a fixed k, we need to get one such so that $\sigma(1, 0, j) + \sigma(2, j, k)$ is a shortest. Again, the principle of dynamic programming tells us that for some j, k, $|\omega_1| + 1 = |\sigma(1, 0, j)$ and $|\omega_2| + 1 = \sigma(2, j, k)|$. Following this method, if some simulation branches remain after all of $\mathbf{p}$ are exhausted, then a consistent $\mathbf{p}'$ exists and one can be extracted via backtracking the search process.

Algorithm 2 VALIDATEPARTIALSTORY

Input: $\mathbf{p} = (p_1, \ldots, p_n, F), M, M_1, \ldots, M_{m+1}$
Output: **true** if M accepts some $\mathbf{p}' \geq \mathbf{p}$, **false** otherwise

```
initialize 2D array L as array of ∞'s
L(0,1) ← 0
for i = 1 to n + 1
  for j = 1 to m + 1
    ℓ ← ∞
    for each start state S_k of M_j
      t ← {min_{j'} SHORTESTLEN((p_{i-1}, j'), S_k)}
      ℓ ← min{ℓ, t + SHORTESTLEN(S_k, (p_i, j))}
    L(i,j) ← min{ℓ, SHORTESTLEN((p_{i-1}, j), (p_i, j))}
if L(n+1, m+1) ≠ ∞
  return true
return false
```

With above analysis, it becomes clear that the insight enabling the reduction of a factor of m in VALIDATEAGENTSTORY also applies here. That is, in finding the shortest string containing $p_1 \ldots p_i$ and taking M from the start state to (p_i, k), we do not need to look at (p_{i-1}, j) for all $j \leq k$. Instead, we only need to know the shortest string that takes $(p_{i-1}, j), j \leq k-1$ for some j, to the start state(s) of M_k; the rest of the search is limited to M_k. Since searching inside M_k for shortest path can be done with Dijkstra's algorithm in $O(n_w^2)$, the overall running time for the search part of validating a partial story is $O(nmn_w^2)$. The pesudocode is described in Algorithm 2. Note that we append to $\mathbf{p}$ an element F, which is the acceptance state of M. Element $L(i, j)$ of the 2D array L stores the length of the shortest string that drives M to the state (p_i, j) and contains $p_1 \ldots p_i$ as a subsequence. The subroutine SHORTESTLEN(a, b) returns the length of the shortest string that takes M from state a to state b. As discussed, we can obtain SHORTESTLEN$((p_{i-1}, j'), S_k)$ incrementally.

B. Story with Errors

We now move to Problem 4. On one hand, to allow insertion, deletion, and substitution of story string $\mathbf{p}$, we need to know the shortest edit distance to reach all states of M for each p_i, instead of knowing only the shortest distance to states (p_i, j). Denote this distance $D(p_i, T)$ in which T is a state of M and let $D(p_i, S, T)$ be the shortest edit distance from state S (of M) to T on p_i, we obtain the recursion

$$D(p_i, T) = \min_S \{D(p_{i-1}, S) + D(p_i, S, T)\}.$$

For a general automaton of Q states, $D(p_i, S, T)$ requires $O(Q^3)$ ($O(m^3 m_w^3)$ for our problem) computation time using a modified all pairs shortest path algorithm [30]. In our case, since $D(p_i, S, T)$ is ∞ for $S \in M_k, T \in M_j$ when $j < k$, we may subdivide the calculation of $D(p_i, T)$ further, staged at each M_j. Suppose T is an internal state of M_j (not start/end states), $\{S_j\}$ are the start states of M_j (there are at most two of these), then the shortest edit distance from $S \in M_{j-1}$ to T, passing some $\{S_j\}$, can be obtained as

$$\min_S \{D(p_{i-1}, S) + \min_{S_j} \{\min_{S_j} \{D(p_i, S, S_j) + D(\epsilon, S_j, T)\}, \min_{S_j} \{D(\epsilon, S, S_j) + D(p_i, S_j, T)\}\}\}.$$

We can regroup the formula as

$$\min_{S_j} \{\min_S \{\min_S \{D(p_{i-1}, S) + D(p_i, S, S_j)\} + D(\epsilon, S_j, T), \min_S \{D(p_{i-1}, S) + D(\epsilon, S, S_j)\} + D(p_i, S_j, T)\}\}.$$

Hence, instead of $O(m^3 m_w^3)$ time, the sequential structure of M cuts the processing time per p_i to $O(m m_w^3)$. On the other hand, with the introduction of deletion and substitution, a matching story is always possible for Problem 4, as long as the sensor recordings are self-consistent (That is, the language of M is not empty): In the worst case, we can change $\mathbf{p}$ to the shortest string accepted by M via substitution, followed by insertion if $\mathbf{p}$ is too short and followed by deletion if $\mathbf{p}$ is too long. If we denote the length of the shortest string accepted by M as n', then a $\mathbf{p}'$, accepted by M and closest to $\mathbf{p}$, cannot have length more than $\max\{n, n'\}$. This is also the maximum number of edits necessary.

At this point, we adapt the transducer construction procedure from [1] to our problem and denote this transducer U. Our transducer can be viewed as $(n+1)(m+1)$ automata (each is of the form M_1 from Fig. 5(a) and is denoted M_j^i, $1 \leq i \leq n+1, 1 \leq j \leq m+1$) chained together, with transitions between the automata only from M_j^i to M_j^{i+1} or M_{j+1}^i. An additional structures in our transducer is that it is directed in two directions. This means that searching U can be partitioned into searching individual M_j^i's and then move forward on a 2D grid. For Problem 3, finding $\mathbf{p}'$ with smallest number of edits is equivalent to finding accepting string of U with with the smallest cost. This is then a single source shortest path problem on U. As said, for each i, we carry out the search from M_1^i to M_{m+1}^i,

which takes time $O(m \cdot m_w \cdot m_w \lg m_w)$. Doing this for all i then takes time $O(nmm_w^2 \lg m_w)$. In contrast, preprocessing along takes $O(m^3 n_w^4)$ in [30] (Recall that m, n are of comparable lengths). Our result is also slightly better than the (more general) algorithm presented in [1], which has time complexity $O(nmm_w^2 \lg(mm_w))$ in this context.

VI. Conclusion and Future Directions

In this paper, we studied the problem of validating an agent's story, which may be partial or contain errors, against the observation history of external sensors. In addition to deciding the validity of an agent's story in time linear in both the length of the story and the length of the observation history, our algorithms produce a path compatible with the sensor observations that is also closest to the agent's story in terms of string edit distance, whenever such a path exists.

Many intriguing questions remain. Focusing on the discrete setting, the next natural question appears to be the validation of multiple agents' combined story: It is possible that several agents' stories are valid when validated individually but problematic when put together. On the other hand, if we consider differential constraints for the agents, two immediate qualitative implications arise. First, since an agent's speed is limited, the exact time of sensor recordings, otherwise neglected, becomes relevant in filtering out long paths between sensor locations. Second, it becomes possible to measure how far away an agent's behavior deviates from the optimal behavior (In terms of distance traveled, for example). Besides these immediate extensions, another interesting direction is to study optimal sensor placements for the detective task, which was discussed to some extent in [23], [28]. Finally, we have only studied part of the spatial and temporal features of a story. With logic, it may become possible to interpret more challenging agent behavior such as "how" and "why".

Acknowledgments 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] C. Allauzen and M. Mohri. Linear-space computation of the edit-distance between a string and a finite automaton. In *London Algorithmics 2008: Theory and Practice*, 2008.

[2] 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.

[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] 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.

[6] 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.

[7] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (T-SLAM). *IEEE Transactions on Robotics & Automation*, 17(2):125–137, 2001.

[8] 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.

[9] M. Dehn. *Papers on Group Theory and Topology*. Springer-Verlag, Berlin, 1987.

[10] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] S. M. LaValle. Filtering and planning in information spaces. Technical report, Dept. of Comp. Sci., University of Illinois, Oct 2009.

[18] 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.

[19] E. W. Myers and W. Miller. Approximate matching of regular expressions. *Bulletin of Mathematical Biology*, 51(1):5–37, 1989.

[20] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[21] 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.

[22] 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.

[23] 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.

[24] D. Sankoff and J. B. Kruskal. *Time Wraps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley, Reading, MA, 1983.

[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] R. A. Wagner. Order-n correction for regular languages. *Communications of the ACM*, 17(5):265–268, 1974.

[31] R. A. Wagner and J. I. Seiferas. Correcting counter-automaton-recognizable languages. *SIAM Journal on Computing*, 7(3):357–375, August 1978.

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

[33] J. Yu and S. M. LaValle. Cyber detectives: Determining when robots or people misbehave. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, 2010.