Inferring Complex Agent Motions from Partial Trajectory Observations

Finnegan Southey
Google Inc.

Wesley Loh
University of Alberta

Dana Wilkinson
University of Waterloo

Abstract

Tracking the movements of a target based on limited observations plays a role in many interesting applications. Existing probabilistic tracking techniques have shown considerable success but the majority assume simplistic motion models suitable for short-term, local motion prediction. Agent movements are often governed by more sophisticated mechanisms such as a goal-directed path-planning algorithm. In such contexts we must go beyond estimating a target's current location to consider its future path and ultimate goal.

We show how to use complex, "black box" motion models to infer distributions over a target's current position, origin, and destination, using only limited observations of the full path. Our approach accommodates motion models defined over a graph, including complex pathing algorithms such as A*. Robust and practical inference is achieved by using hidden semi-Markov models (HSMMs) and graph abstraction. The method has also been extended to effectively track multiple, indistinguishable agents via a greedy heuristic.

1 Introduction

Many interesting applications require the tracking of agents that move in a complex fashion through an environment in which only partial observations are possible. Examples include real-time strategy games, aerial surveillance, traffic, visual tracking with occlusion, and sensor networks. Related problems have been well-addressed in robotics and vision domains where approaches often start by assuming a known *motion model* that characterizes the movement in question. For example, Monte Carlo localization usually employs a motion model for short-term predictions, capturing simple movements via some tractable mathematical formulation. Unfortunately, in many applications, the movements of agents may be quite complex and involve long-term goals governed by sophisticated algorithms such as goal-directed path-planners. Traditional tracking approaches that rely on simple motion models may not prove effective. Furthermore, one may wish to draw inferences about the long-term behaviour of the agent rather than simply track its current location.

As an example, consider real-time strategy (RTS) games, a popular genre of commercial video games in which users control armies consisting of tens or even hundreds of units. The units are largely autonomous and the user provides only high-level instructions to most units. For instance, a unit can be commanded to move to a new location simply by specifying the destination. A pathfinding routine then computes the actual path to reach this destination and the unit travels there automatically. Another key feature of these games is the so-called *fog of war*—a limitation whereby the user can only observe activity in the immediate vicinity of their own units. This limited field of view means that enemy movements can only be observed when they occur close to an allied unit. This results in a series of short, disconnected observations of enemy movements. Additionally, most units of a given type (e.g., tanks) are indistinguishable from one another, so it may be uncertain whether or not two separate observations pertain to the same unit. Several interesting strategic questions can be asked in relation to these observations. How many enemy units are there? Which observations correspond to which unit? Where is an enemy when unobserved? Where was an enemy unit coming from and going to?

The method we propose and demonstrate facilitates inference using complex motion models in the form of a "black box". Given two points, the motion model need only return a path; the algorithm producing that path can be arbitrary and unknown. Our method infers answers to several of the above questions, employing abstraction to efficiently handle large numbers of possible paths and probabilistic models to robustly handle inaccurate motion models or noise in the observations. It is important to emphasize that the approach does not learn a model of the opponent's motion. Instead, we use existing motion models to infer complex behaviour. We demonstrate our results in a RTS-like context and discuss its use in real world domains.

2 Related Work

While there is a great deal of work on tracking, we are unaware of any work that employs complex black box motion models in the manner we describe here. The work of Bruce and Gordon on tracking human movements for robotic obstacle avoidance is close in spirit inasmuch as it works with the assumption that humans will follow optimal, and therefore potentially complex, paths [Bruce and Gordon, 2004]. An-

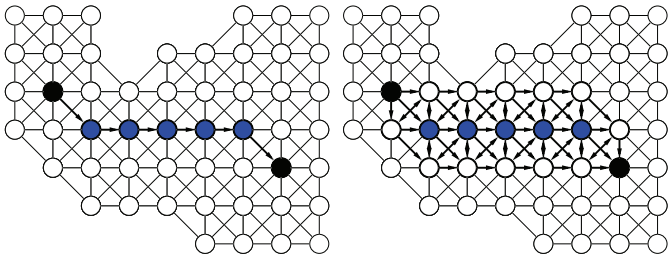
other closely related project is work on learning goal-driven human movements from GPS data [Liao *et al.*, 2004]. This work also employs hidden semi-Markov models and abstractions of the environment, although the details differ substantially. Similarly, the work described in [Bennewitz *et al.*, 2004] attempts to model complex human motion as a hidden Markov model learned from past observations. However, all of these examples relied on previous examples of movements and/or goals in order to learn models. Our work uses an existing but potentially very complex motion model and can be readily applied to new environments. The indistinguishable multi-agent task we identify here is very similar to that explored by [Sidenbladh and Wirkander, 2003] but they assume simplistic motion models. Work on multiple hypothesis tracking [Reid, 1979] and goal recognition also addresses similar issues. Finally, Bererton has applied a particle filtering approach to track human players in games to achieve realistic behaviours for game enemies [Bererton, 2004].

3 Formalism

Motion Models To formalize the discussion, we start with a known environment described by a graph $G = (V, E)$, where the vertices $V = \{v_1, \cdots, v_n\}$ correspond to a set of locations and the set of edges E indicates their connectivity. We assume the availability of a parameterized *motion model* $M : \theta, G \rightarrow p$, where θ are the parameters and p is a sequence of vertices from V that specifies a directed path through G. Specifically, we will consider *endpoint* motion models that take parameters of the form $\theta = (v_i, v_j)$, where $v_i, v_j \in V$ are endpoints. Figure 1 (a) shows an example graph and path. Any *target* we seek to track is assumed to be using M with some specific, but unknown, endpoint parameters. The model M is treated as a "black box" and the mechanism by which M generates paths is unimportant. We need only be able to query M for a path, given some endpoints.

The availability of such motion models for RTS is quite immediate as the game provides an algorithm for this purpose. Other applications might require constructing or learning a motion model (as in [Bennewitz *et al.*, 2004]), but in some domains reasonable models may already be available. For example, the various online mapping services can provide road paths between travel locations. Another avenue is to assume some cost-sensitive pathing algorithm such as A* search—a very reasonable choice in the case of RTS where most pathing algorithms are variants of A*. Note that most useful pathing algorithms require edge costs or other information beyond the graph itself. Any such extra information is assumed to be available along with the graph.

Observations Time is treated as advancing in discrete steps. At each time step t, the observations made are described by a set of vertex-result pairs, O_t. The observation results for a vertex v_i can be *negative* (nothing was seen there) or *positive* (a target was seen). For example, $O_3 = \{(v_2, +), (v_5, -)\}$ would mean that, at the third timestep, vertices v_2 and v_5 were under observation, and a target was seen at v_2 while no target was seen at v_5. All other vertices were not under observation. This arrangement allows for the fact that our observer(s) may be moving through the environment and/or have a varying field of view. We use $O_{1:t}$ to denote the sequence of observations made from timestep 1 to timestep t.



(a) (b)

Figure 1: (a) Path from an endpoint motion model. (b) Noisy version of the proposed path.

4 Inference with Complex Motion Models

We seek to infer details of target movements given our observations and the parameterized motion model. The overall approach we adopt is Bayesian inference. For now, we will consider only a single target and return to the multiple target case presently. We assume that the target is using the motion model M with two unknown parameters (the endpoints). If we can estimate these parameters, we can characterize the motion of the agent.

4.1 Prior and Posterior

We start by assuming a prior over endpoint parameters, $P(\theta)$, where θ is a pair specifying the *origin* and *destination* vertices. In many applications we may have well-informed priors. For example, in RTS games, targets typically move between a handful of locations, such as the players' bases and sources of resources. In all results presented here, we employ a uniform prior over endpoints, although our implementation accepts more informative priors. Much may be gained by exploiting prior knowledge of potential agent goals.

At any given timestep t, we compute the posterior distribution over endpoints, given all past observations, $P(\theta|O_{1:t})$. By Bayes' rule, $P(\theta|O_{1:t}) \propto P(O_{1:t}|\theta)P(\theta)$, giving an unnormalized version of the target distribution by computing the probability of the observations given the parameters. Note that for many purposes, the unnormalized distribution is perfectly acceptable, but normalization can be applied if necessary. We compute the full posterior distribution in our work here but it is straightforward to compute the *max a posteriori* estimate of the parameters to identify a "most probable" path. In the simplest scenario, $P(O_{1:t}|\theta)$ can be computed for all possible endpoints θ. We will address the obvious scaling issues with this approach shortly.

4.2 Hidden Semi-Markov Model

We must now explain how we will compute $P(O_{1:t}|\theta)$ for one particular setting of θ. The black box M is used to generate the *proposed path* p corresponding to θ. An obvious method for calculating the probability would be to perform exact path matching, testing to see whether the observations

$O_{1:t}$ exactly match what we would expect to see if the target traversed p. This results in a probability of 0 or 1. However, this approach is not robust. If our model M is not perfectly correct—perhaps because we do not have access to the true pathing routine but only an approximation—it will lead to problems. Also, in real-world domains we may have the additional problem of noisy observations, such as erroneously seeing or not seeing the target, or incorrectly identifying its exact location when seen. Finally, variable movement rates can result in observations inconsistent with p.

In order to compensate for these potential inconsistencies, we use p as the basis for a noise-tolerant observation model instead of using it directly. This model is a *hidden semi-Markov model* (HSMM) based on p. A HSMM is a discrete time model in which a sequence of hidden states $(S_1, \cdots, S_t)$ is visited and observations are emitted depending on the hidden state (see [Murphy, 2002] for a good overview). HSMMs extend the common *hidden Markov model* by allowing for a distribution over the time spent in each distinct hidden state. More precisely, a HSMM consists of:

- a set of hidden states (the vertices V in our case)
- a prior over initial hidden state, $P(S_1 = v_i)), \forall v_i \in V$
- a *transition function*—a distribution over successor states given that a transition occurs at time t, $P(S_{t+1} = v_i | S_t = v_j), \forall v_i, v_j \in V$
- an *observation function*—a distribution over observations, for each state, $P(O_t | S_t = v_i), \forall v_i \in V$
- a *duration function*—a distribution over the time to the next transition, for each state, $P(d_t | S_t = v_i), \forall v_i \in V$.[1]

For each proposed endpoint pair θ, we build a corresponding HSMM whose parameters (the above distributions) we will denote by $\tilde{\theta}$. We use the distribution of the observations given these $\tilde{\theta}$ parameter settings as a noise-tolerant, unnormalized estimate of the distribution over the original θ parameters, $P(\theta|O_{1:t}) \propto P(O_{1:t}|\theta)P(\theta) \approx P(O_{1:t}|\tilde{\theta})P(\tilde{\theta})$, where $P(\tilde{\theta}) = P(\theta)$. We will defer explanation of how we construct $\tilde{\theta}$ for now, and explain how we compute the probabilities first.

Inference Inference in HSMMs can be achieved using the *forward-backward* algorithm [Murphy, 2002]. Only the "forward" part is required to compute most of the distributions of interest. The forward algorithm computes the joint probability of the hidden state and the history of observations, $P(S_t, O_{1:t}|\tilde{\theta})$, for each time step t, which we will abbreviate by $\alpha_t : V \rightarrow [0, 1]$. Each function α_t is a function of earlier functions, $\alpha_s, s < t$, so the probability can be efficiently updated after each new observation. Setting $\alpha_1(v_i) = P(S_1 = v_i), \forall v_i \in V$, the remaining α's are computed as

$$\alpha_t(S_t) = \sum_d P(d|S_t) \left[\prod_{u=t-d+1}^{t} P(O_u|S_t) \right] \left[\sum_{S_j} P(S_t|S_j) \alpha_{t-d}(S_j) \right]$$

Note that the α's and the distributions used to compute them are all specific to a *single* HSMM, $\tilde{\theta}$. The conditioning on $\tilde{\theta}$ was omitted for readability. By computing the α functions for all $\tilde{\theta}$'s, we can obtain answers to some of our original questions, so we explicitly condition on $\tilde{\theta}$ hereafter.

Endpoint and Filtering Distributions One interesting strategic question was "where is the target going to and coming from". We can obtain the unnormalized distribution over endpoints by marginalizing out the hidden state variable S_t

$$\begin{aligned} P(\theta|O_{1:t}) &\propto P(O_{1:t}|\theta)P(\theta) \\ &\approx P(O_{1:t}|\tilde{\theta})P(\tilde{\theta}) = \sum_{S_t} P(S_t, O_{1:t}|\tilde{\theta})P(\tilde{\theta}) \end{aligned}$$

Another interesting question was "where is the target now"—the current hidden state of the target. This is often called the *filtering distribution* and can be computed as

$$\begin{aligned} P(S_t|O_{1:t}) &\propto \sum_{\theta} P(S_t, O_{1:t}|\theta)P(\theta) \\ &\approx \sum_{\tilde{\theta}} P(S_t, O_{1:t}|\tilde{\theta})P(\tilde{\theta}) \end{aligned}$$

4.3 Building the HSMMs

We still need to specify how to generate a HSMM $\tilde{\theta}$ from a proposed path p based on θ. In the simplest case, one could construct the transition function so it deterministically generates p, the observation model to give $0/1$ probabilities for (not) observing the target where p dictates, and the duration distribution to give probability 1 to a duration of 1. This is equivalent to the exact path-matching model. Importantly, by changing the distributions that make up the HSMM, we can relax the exact matching in several ways to achieve robustness. We describe such a relaxation for handling inaccurate black boxes and noisy real-world observations immediately below. However, relaxations can also facilitate approximations that are computationally motivated, such as the abstracted graph approximation described in Section 5.

Noisy Pathing

One way to deal with inaccuracies in M (i.e., differences between true paths and proposed paths) is to introduce some noise into the transition functions of the HSMMs. This creates a noisy version of the original model M. In our implementation, we introduce a small probability of straying from the exact path proposed by M to vertices neighbouring the vertices in the path, with a high probability of returning to vertices on the path. This creates a probabilistic "corridor" so

[1] If the observers are moving then the observation distribution changes over time (our implementation fully supports this). The transition and duration functions are stationary.

that small deviations from M will be tolerated. Figure 1 (b) shows an example of this idea.

There are many other possibilities for introducing noise (e.g., make the probability of error related to the edge weight of neighbours or distance from vertices in p). We explore only this simple transition noise here. Another, quite distinct, option is to introduce noise into the observation function instead. Uncertainty in real-world applications may come from M, or the observations, or both. Practically speaking, noisy transition functions and observations functions can serve to address either source of error (e.g., treating errors in proposed paths as though they were observation errors). We have not experimented with noisy observation functions but they might offer advantages by keeping the HSMM transition functions sparse, reducing memory usage and computation.

5 Abstraction

We can perform exact inference simply by considering all endpoint pairs. If the number of possible pairs is prohibitively high, we could approximate by Monte Carlo sampling or even employ a form of particle filtering. The former is fairly straightforward and the latter is certainly possible. However, we will not expand on these options here.

A complementary strategy for reducing computational cost is to abstract the original *base graph* G that describes the environment to obtain a smaller *abstract graph* $\hat{G} = (\hat{V}, \hat{E})$ that more coarsely models locations in the environment. An abstraction $\phi(G)$ maps every vertex in V to some member of the smaller set of abstracted vertices $\hat{V}$, thereby reducing the set of possible endpoints. This effectively scales down the number of HSMMs to store and compute. If the motion model M requires any additional information such as edge costs, these must be abstracted as well. In the work presented here we use *clique abstraction* [Sturtevant and Buro, 2005], which averages edge costs from G to produce edge costs for $\hat{G}$, but our implementation accepts any graph abstraction provided. For many purposes the resulting loss in precision is an acceptable tradeoff (e.g., it suffices to know that an enemy's goal is "near" some location).

Some flexibility is required to handle graph abstraction. First, each vertex in $\hat{V}$ corresponds to a set of vertices in V, some of which may have been under observation and the rest unobserved. We abstract our observations $O_{1:t}$ from the vertices in the base graph to observations $\hat{O}_{1:t} = \{\hat{O}_1, \ldots, \hat{O}_t\}$ such that we record a positive observation at an abstracted vertex if a target was seen in any of the corresponding base vertices under observation. These partially-observed abstract vertices introduce uncertainty when a negative observation is made; it is still possible that a target is in one of the unobserved base nodes. Fortunately, the use of HSMMs means that we can incorporate this uncertainty readily by changing the observation function. We use a simple and obvious model where the probability of a negative observation at some abstracted vertex, given that a target is actually at that vertex, is equal to the proportion of base vertices that are unobserved.

Similarly, the grouping of base vertices into a single abstract vertex means that, even if movements at the base level take unit time, movement through an abstracted node can take variable amounts of time. Again, we need only change the duration function for our HSMM to approximately account for this. We build this distribution for each abstract vertex by generating all base paths covered by the vertex and counting the number of paths of each length. We then assume a uniform distribution over paths through the abstract vertex.

The abstraction of graphs and observations may lead to discrepancies between abstracted observations and proposed paths, because the set of proposed paths is computed by M within $\hat{G}$, while a target's actual path is computed within G. As a result, there may be mismatches between the actual path and abstracted path due to details in M's algorithm and, particularly, the abstraction of extra information such as edge cost. On the other hand, abstraction can serve to mitigate minor differences between the true motion model and our assumed motion model M since many base level paths will conform to a single path in $\hat{G}$. The noisy pathing described earlier is one possible remedy for the former situation.

6 Multiple Targets

In general, we expect multiple targets in the world. However, as mentioned earlier, it may not be possible to distinguish one target from another. We therefore seek to establish which agents generated which observations. We use the following natural grouping of observations as a starting point. While a target is in view, it is assumed that it can be accurately tracked, and so all consecutive positive observations of a target are taken to be a single *trajectory*.[2]

We now wish to associate these trajectories with targets. Consider a set of integer "labels" $\{1, \cdots, m\}$, where m is the number of trajectories, and a labelling L that maps every observed trajectory onto the set of labels. In place of our earlier notation for observations, we here denote the trajectories associated with a single label j under labelling L, by $Y_L(j)$.[3] Conceptually, each label corresponds to a distinct target that generates the trajectories associated with that label. Some labellings will propose grouping a set of trajectories under one label that could not have been generated by a single target. Such labellings will have a zero posterior probability. Other labellings will be consistent but will differ in how well they explain the observations. Ideally then, we want to find a labelling that gives the best explanation—one that maximizes the posterior probability of the observations

$$\begin{aligned} & \max_L P(Y_L(1), \cdots, Y_L(m)|L)P(L) \\ \approx \; & \max_L \prod_{j=1}^{m} P(Y_L(j)|L)P(L) \\ = \; & \max_L \prod_{j=1}^{m} \sum_{\tilde{\theta}_i} P(Y_L(j)|L, \tilde{\theta}_i)P(\tilde{\theta}_i)P(L) \end{aligned}$$

[2] Tracking is easy for RTS but generally hard. Failure to identify trajectories will simply fragment what should be one trajectory into several, and so does not pose any insuperable problem.

[3] Computationally, α functions for the negative observations can be computed separately and shared by all positive trajectories, offering a considerable savings in computation and space.

where $P(L)$ is a prior over labellings. This formulation effectively assumes that the targets associated with the labels generate observations independently, an approximation we accept for the sake of reducing computation.

Maximizing over all possible labellings of the trajectories is still expensive, however, so we use the following greedy approximation. We start with the labelling L, which gives each trajectory a unique label (essentially claiming that each was generated by a separate target and that there are m targets), and then proceed iteratively. At each step, we select a candidate pair of labels j and k to *merge*, by which we mean that the trajectories of both labels are now associated with one of the labels. This gives us a new labelling, L' such that $Y_{L'}(j) = Y_L(j) \cup Y_L(k)$ and $Y_{L'}(k) = \emptyset$. Next, we compare the likelihoods of the labellings L and L', rejecting the merge if L is more likely, or keeping the new labelling $L \leftarrow L'$ otherwise. We then select a new candidate pair and repeat.

We heuristically order the candidate label pairs by computing the KL divergence of the endpoint distributions for all pairs of labels and selecting $\min_{j,k} KL(P(\theta|Y_L(j)) \parallel P(\theta|Y_L(k)) + KL(P(\theta|Y_L(k)) \parallel P(\theta|Y_L(j))$.[4] If a pair is merged, we update the KL divergences and repeat. If a pair is rejected for merging, we select the next smallest divergence, and so on. Merging stops when no candidates remain.

Intuitively, this heuristic proposes pairs of labels whose trajectories induce similar endpoint distributions (i.e., it prefers two hypothesized targets with similar origins/destinations given their separate observations). If the pair's observations were in fact generated by one target's path, then the merge should produce high posterior probability. However, it is also possible that there really were two targets and that when the observations are combined, an inconsistency is detected (e.g., two similar paths that overlap in time and so cannot be a single target). The divergence is simply intended to propose candidates in an order guided by the similarity of likely paths.

7 Experimental Results

We have implemented our approach on top of the HOG framework for pathfinding research [Sturtevant and Buro, 2005]. It provides the simulation, visualization facility, pathfinding routines and abstractions. The clique abstraction we use is hierarchical, providing a succession of abstractions, each abstracting the graph from the previous level. Abstraction level 0 is the original, unabstracted base graph, level 1 is an abstraction of that, 2 is abstraction of level 1, etc. The *baseline* case—no abstraction and a perfect motion model—obtains the true posterior distributions given the observations made. The usefulness of even the baseline case is necessarily limited by the available observations. If a target is never observed, or observed only at points in the trajectory that are consistent with a wide variety of paths, then the conclusions will be correspondingly vague.

Experiments were run to gauge the impact of the approximations involved in abstraction and the mechanisms used to handle noise. As we are unaware of any alternative method that performs inference with complex motion models of the kind we discuss here, our experiments are focused on showing that the approximations we use compare well to the baseline and improve on computational costs. We use four different maps for the most exhaustive tests. The first map, Penta, shown in Figure 2(a), is a 32×32 map created for our experiments specifically to demonstrate how endpoints can be accurately inferred. The other three maps, Adrenaline (65×65), Harrow (96×96), and Borderlands (96×96), are all maps found in the commercial RTS game "Warcraft 3"[5] by Blizzard Entertainment (see Figures 3, 4, and 5). Supplemental colour images and animations are available at http://ijcai2007.googlepages.com/.

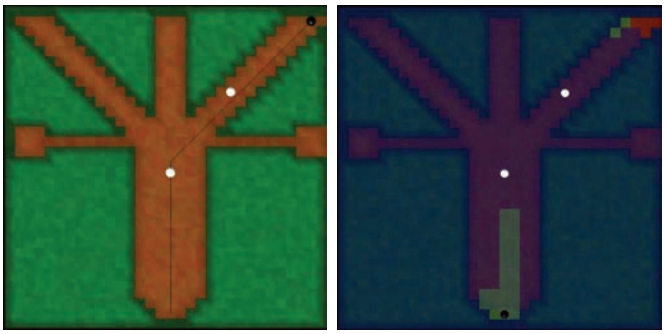


Figure 2: (a) Penta map showing target's path from at top right to bottom centre (b) Superimposed images of origin and destination posterior distributions

Within these environments, we place observers and targets. Observers have a 5×5 field of view centered at their location. Targets move using the PRA* pathfinding algorithm described in [Sturtevant and Buro, 2005], a fast variant of A* that generates sub-optimal paths similar to A* paths. We can test the effect of inaccurate models by comparing the use of PRA* for the modelling (a correct assumption) vs. A* (incorrect). In each trial, endpoints are randomly selected for the targets and observers are placed randomly along the paths, guaranteeing at least one observation. The simulation runs until all targets reach their destinations, plus an additional ten steps. Endpoint and filtering distributions are computed at every timestep and results averaged over 50 trials.

An example of the posterior distributions over the origin and destination of a target on Penta is shown in 2(b), ten time steps after the target (black dot) has completed the journey, viewed by two different observers (white dots) on the way. For compactness, the two distributions are superimposed—the shading at the top right is the origin distribution and the shading at the bottom center is the destination distribution.

To assess what might be called the "accuracy" of the method in the single target case, all paths are ranked according to their posterior probability at the end of the trial. We report the percentile ranking of the target's true path, averaged over all trials (higher is better). Results are reported for several abstraction levels, for the different modelling assumptions (correct: PRA* vs. incorrect: A*), and for noisy/non-noisy transition functions. This captures the penalty suffered by abstraction and an inaccurate model. We highlight the baseline case (no abstraction, no noise, and accurate model),

[4] KL divergence is not symmetric so we adopt the common symmetrized version.

[5] http://www.blizzard.com/war3/

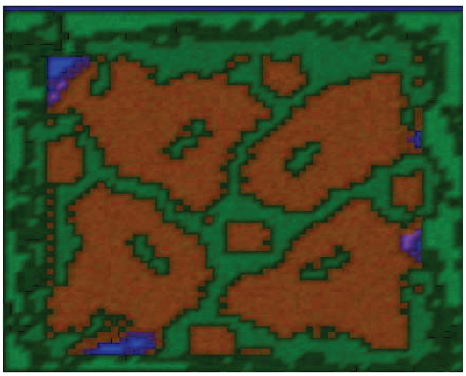


Figure 3: Adrenaline map

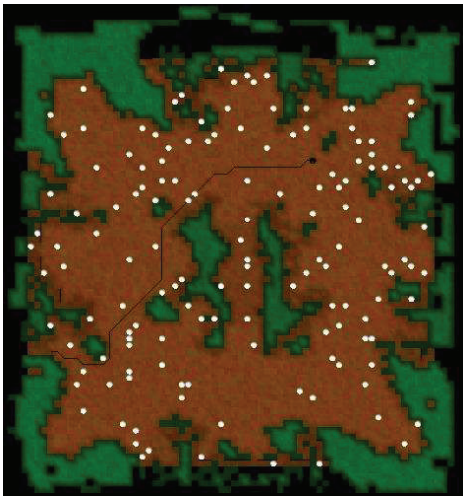


Figure 4: Harrow map with observers and a target

the best one can achieve given the observations. The number of vertices for each abstraction is also reported. We also report the average time per trial in seconds.

Results for the Penta map in Table 1 show high accuracy in all cases, but this map is very small. When the modelling assumption is correct (PRA* modelled by PRA*), the noisy transition function damages results, even at higher abstractions. The incorrect modelling assumption (PRA* modelled by A*) does further damage, but noisy transitions mitigate this slightly at the highest abstraction level. While all very similar, the averaged results do not capture the fact that a low percentile ranking on an individual run is often because the actual path was given 0 probability. Under noisy transitions, results tend to be smoother and failures less catastrophic.

The other three maps are too large to run the baseline or first level of abstraction. Having evidence from the Penta map

		Avg Time	PRA* by PRA*		PRA* by A*	
	$\|V\|$	per step (s)	Normal	Noisy	Normal	Noisy
No Abs	1024	0.4335	**99.99**	99.99	99.99	99.99
Abs 1	330	0.0486	99.99	99.99	99.99	99.99
Abs 2	123	0.0145	99.97	99.96	99.97	99.96
Abs 3	50	0.0134	99.89	99.89	99.89	99.89
Abs 4	18	0.0133	99.26	99.19	99.28	99.29

Table 1: Penta Map: Percentile of Actual Path

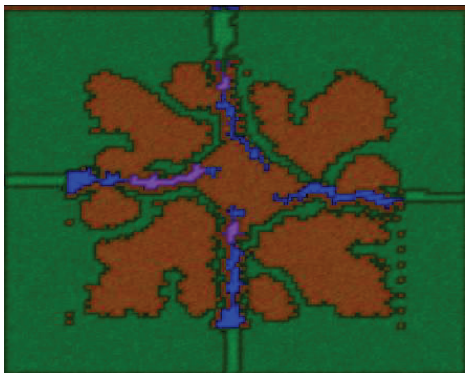


Figure 5: Borderlands map

		Avg Time	PRA* by PRA*		PRA* by A*	
	$\|V\|$	per step (s)	Normal	Noisy	Normal	Noisy
Abs 2	521	0.0305	99.99	99.99	99.99	99.99
Abs 3	216	0.0156	99.99	99.99	99.99	99.99
Abs 4	88	0.0130	99.95	99.94	99.97	99.96
Abs 5	40	0.0128	99.84	99.84	99.86	99.86

Table 2: Adrenaline Map: Percentile of Actual Path

that results at the second level are quite close to the baseline, we present Tables 2, 3, and 4 to evaluate performance on real RTS maps. In all cases, performance degrades gracefully as abstraction increases, with Harrow showing the greatest loss, probably due to its less structured layout. Note that even under the correct modelling assumption, the noisy transitions can mitigate the effects of abstraction in Harrow although noise typically damages higher abstraction levels. The same noise parameters are used for all abstractions and were never tuned. The level of noise is simply too high for the very crude Harrow abstractions where a path deviation of one vertex is comparatively large. These results show that effective inference can be performed even at fairly high abstractions.

Runtime improves dramatically for early abstraction levels but exhibits diminishing returns as the number of vertices grows very small. Currently the speed exceeds what is necessary for real-time tracking of a single target in RTS and more optimization is possible. The average runtime per step of the simulation is reported and these times reflect the costliest mode of running at a given abstraction (i.e., noisy transitions with A* modelling, which is the slowest mode).

Finally, we test the multiple target trajectory association by measuring how often trajectories are correctly identified as coming from the same target. In each case, two targets with random paths are observed by two observers. The association is computed at the end of the trial. Table 5 shows the

		Avg Time	PRA* by PRA*		PRA* by A*	
	$\|V\|$	per step (s)	Normal	Noisy	Normal	Noisy
Abs 2	807	1.2309	99.87	99.88	99.87	98.87
Abs 3	297	0.1941	99.50	99.52	99.50	99.53
Abs 4	110	0.0521	98.86	98.87	98.86	98.87
Abs 5	47	0.0284	98.31	98.25	98.29	98.26
Abs 6	19	0.0235	95.15	94.72	95.05	94.70
Abs 7	7	0.0223	93.39	92.73	93.39	92.73

Table 3: Harrow Map: Percentile of Actual Path

	$\|V\|$	Avg Time per step (s)	PRA* by PRA*		PRA* by A*	
			Normal	Noisy	Normal	Noisy
Abs 2	910	0.1052	99.99	99.99	99.99	99.99
Abs 3	346	0.0320	99.99	99.99	99.99	99.99
Abs 4	136	0.0231	99.98	99.96	99.98	99.96
Abs 5	59	0.0218	99.88	99.84	99.88	99.84
Abs 6	21	0.0215	99.58	99.58	99.58	99.58

Table 4: Borderlands Map: Percentile of Actual Path

	PRA* by PRA*		PRA* by A*	
	Normal	Noisy	Normal	Noisy
No Abs	94.58	92.41	82.37	89.00
Abs 1	92.77	89.46	77.81	86.63
Abs 2	86.57	84.18	82.98	82.37
Abs 3	79.94	76.40	75.08	72.95
Abs 4	67.26	65.49	68.39	64.74

Table 5: Penta Map: Trajectory association accuracy

percentage of correctly classified trajectories (i.e., correctly associated with any trajectories generated by the same target), averaged over all trials on the Penta map, giving overall high success rates. The results for Adrenaline, Harrow, and Borderlands in Tables 6, 7, and 8 reflect the single agent results. Harrow is clearly the most difficult, with the wide-open spaces allowing for many possible interpretations of observations. Interestingly, noise is more damaging to association results overall, suggesting than any "blurring" of goals can easily lead to misassociations. The robustness to abstraction seen on most maps shows that good associations are obtainable even at high abstractions unsuited to endpoint inference.

8 Conclusions

We have presented a method for inferring the movements and intentions of moving targets given only partial observations of their paths. The target can be governed by complex motion models incorporating long-term, cost-sensitive objectives. Given a black box M that describes the motion, a hidden semi-Markov model is constructed for each candidate path generated by M. Using HSMMs lets the system accommodate uncertainty in observations, paths, and the duration of each move. The implementation also supports inference on abstractions of the original map, allowing its use even on complex maps from real-time strategy games. The method has been tested using a complex motion model based on A* pathfinding. Finally, a greedy approximation has been proposed for associating observed trajectories with multiple, indistinguishable agents based on the posterior probability of joint observations and tested with good results. Future work includes Monte Carlo and particle filtering—allowing weak hypotheses to drop in favour of the stronger; better approximations for the multiple agent case; and parameterized motion models that include parameters other than the endpoints. Even without these improvements, this approach has demonstrated great potential for addressing complex applications involving agents with long-term goals.

	PRA* by PRA*		PRA* by A*	
	Normal	Noisy	Normal	Noisy
Abs 2	96.60	94.33	96.60	94.72
Abs 3	93.56	92.80	93.56	92.80
Abs 4	87.65	84.77	87.65	84.77
Abs 5	82.85	82.46	82.84	82.43

Table 6: Adrenaline Map: Trajectory association accuracy

	PRA* by PRA*		PRA* by A*	
	Normal	Noisy	Normal	Noisy
Abs 2	87.38	86.11	87.38	85.71
Abs 3	80.79	67.88	81.13	67.55
Abs 4	68.06	63.54	80.56	63.54
Abs 5	64.16	56.99	65.59	57.71
Abs 6	60.53	57.52	61.28	56.77
Abs 7	46.99	49.25	46.99	49.25

Table 7: Harrow Map: Trajectory association accuracy

	PRA* by PRA*		PRA* by A*	
	Normal	Noisy	Normal	Noisy
Abs 2	95.86	94.48	95.17	94.83
Abs 3	94.06	91.26	93.36	91.26
Abs 4	88.69	84.31	88.69	84.31
Abs 5	80.66	79.56	79.92	80.29
Abs 6	78.21	78.21	78.21	78.21

Table 8: Borderlands Map: Trajectory association accuracy

Acknowledgments

This research was funded by the Alberta Ingenuity Centre for Machine Learning. The authors offer warm thanks to Nathan Sturtevant for developing and abundantly supporting the HOG pathfinding platform used in this research.

References

[Bennewitz *et al.*, 2004] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant motion. *Intl. Jour. of Robotics Research*, 2004.

[Bererton, 2004] C. Bererton. State Estimation for Game AI Using Particle Filters. In *Proceedings of the AAAI 2004 Workshop on Challenges in Game AI*, Pittsburgh, 2004.

[Bruce and Gordon, 2004] A. Bruce and G. Gordon. Better Motion Prediction for People-Tracking. In *Intl. Conf. on Robotics and Automation (ICRA-2004)*, 2004.

[Liao *et al.*, 2004] L. Liao, D. Fox, and H. Kautz. Learning and inferring transportation routines. In *AAAI-2004*, pages 348–353, 2004.

[Murphy, 2002] Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, July 2002.

[Reid, 1979] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):84–90, December 1979.

[Sidenbladh and Wirkander, 2003] H. Sidenbladh and S. Wirkander. Tracking random sets of vehicles in terrain. In *IEEE Workshop on Multi-Object Tracking*, 2003.

[Sturtevant and Buro, 2005] N. Sturtevant and M. Buro. Partial Pathfinding Using Map Abstraction and Refinement. In *20th Conf. on Artificial Intelligence (AAAI-2005)*, 2005.