Inferring Mobile Trajectories using a Network of Binary Proximity Sensors

Eunjoon Cho*, Kevin Wong*, Omprakash Gnawali†, Martin Wicke‡, and Leonidas Guibas†
{ejcho,kbw5}@stanford.edu, gnawali@cs.stanford.edu, wicke@eecs.berkeley.edu, and guibas@cs.stanford.edu

*Department of Electrical Engineering, Stanford University
†Department of Computer Science, Stanford University
‡Department of Electrical Engineering and Computer Science, University of California at Berkeley

***Abstract*—Understanding human mobility in an environment can be approached in many forms, one of which is to recover the underlying structure of user movement. In our work, we show that we can use a network of binary proximity sensors to detect paths between nodes and also extract highly popular trajectories users take. We show that with sufficient amount of these binary data, even with no prior knowledge of the location of these sensors, we can capture a correlation between the detection timestamps in the case where a physical path exists between any two nodes. Our algorithm also generates characteristics of the path, such as the distribution of transition times and volume. We further show that with sampling techniques we can estimate the underlying trajectories that generated the time stamps. We have tested our algorithm on a simulator and two sensor network deployments. We found that, despite the lack of position information about the sensor nodes, with timestamps alone our algorithm can accurately detect the trajectories and is robust enough to use in a real-world office building.**

I. Introduction

A network of binary proximity sensors has been shown to successfully track one or multiple targets. In these networks, the sensors report the time when they detect the mobile targets in their vicinity and using a centralized or a distributed algorithm estimate the current position of the target [1], [2], [3], [4].

The focus of our work is to infer higher order patterns in mobility in the network. Given the network, can we estimate the most popular routes the users take? We are interested in finding the mobility patterns, which we interchangeably call mobility structure, that emerge over much larger time scales (e.g., days or weeks) than what is typically considered in traditional target tracking.

Understanding and characterizing the long-term movement patterns has various applications. In the physical world, the movement of agents is limited by physical or social constraints. We can discover the extent to which those constraints influence the mobility at large. Such high level mobility patterns are also useful in wildlife monitoring or traffic modeling because they allow us to infer the salient flows in the network. The learned mobility patterns can help in solving optimization problems involving mobility, such as the layout of shopping malls and road networks, and the emergency exit locations.

One of the challenges in inferring long-term mobility patterns, or instances of trajectories in the short term as in target tracking, is the localization of the sensor nodes. The position of the mobile node is typically estimated with respect to the known position of the proximity sensors. GPS might not provide enough accuracy indoors and although there is a vast body of work in node localization (e.g. [5], [6]), it adds complexity to the software on the nodes. In this paper, we develop an algorithm that allows us to infer the long-term mobility patterns without the position information of the binary proximity sensors.

Furthermore, we infer the long-term mobility patterns with the additional constraint of unlabeled proximity sensor data. If the timestamps reported by the sensors also have a label that identifies the mobile agent, we can leverage the established body of work in recovering mobility pattern from labeled sensor data. Unfortunately, labeling the proximity timestamps with the identity of the mobile agent introduces complexity in the sensor network. The mobile agent would have to transmit or disclose its identity so the sensors can label the timestamps. Alternatively, the nodes can employ higher fidelity sensors such as cameras and use computer vision to identify the mobile agent and label the sensor data appropriately. In addition to the technical complexity, such approaches to sensor data labeling can also introduce privacy concerns.

We design an algorithm that can extract mobility structure and infer the specific user trajectories from sensor data that is not labeled. Our algorithm first detects link information between sensors by capturing the pairwise correlation of timestamps between nodes. It computes which pairs of nodes are connected by a path with high probability. Given the link information, the algorithm then recovers individual trajectories from the data using a Markov chain Monte Carlo (MCMC) sampling method. One of the most attractive features of our method is that the algorithm does not rely on any specific type of data. It can use timestamp data from any source, including data collections not initially designed to deliver mobility information, such as many legacy security systems.

With this approach, we have found that we can accurately extract mobility structure despite the lack of labels by analyzing a large number of observations. Our probabilistic framework allows us to extract detailed information about the mobility from previously unusable data. Despite the lack of any prior knowledge about the deployment setup of the

sensor network or the node positions, we can recover the mobility structure by capturing the trajectories at a high level of accuracy.

We validate the performance of our algorithm by using simulations and two sensor network deployments. We found that the likely trajectories and mobility patterns suggested by our algorithm matches the ground truth in the simulator and the mobility patterns we observe in our deployments.

Our contributions are:

- We design an algorithm to infer mobility patterns using a network of binary proximity sensors. The algorithm does not require the position of the sensors and works on unlabeled proximity sensor data.
- We show that the algorithm accurately recovers the mobility patterns using a variety of scenarios in the simulation.
- We deploy two sensor networks in office buildings and demonstrate that the algorithm is robust enough to infer the mobility patterns even in noisy real-world settings.

II. Related Work

Mobility patterns can be represented using geometric information, for example using clusters of free-form trajectories [7], [8]. Topological methods instead try to recover the topology of an embedding space [9], [10]. The latter assumes stronger constraints on the movement of agents, but can provide us with a higher level understanding on the structure of movements in the environment. While our method of finding links between nodes by correlation implicitly assumes geometric information (distances between nodes), our output in our link detection stage is purely topological: a graph encoding the physical connectivity between nodes.

Finding correspondence between sensors with limited information have been approached in various ways. Makris et al., [11], proposed a method to find connectivity between camera nodes by observing the correlation of event detections at each node. They define exit/entry zones in a field of view of each camera and then processes the information observed in the network to find connectivity between these zones. Their work provides an intuitive setup on finding correlation between nodes using the time stamps of observations. However, the use of an empirical threshold for accepting links can induce problems when the noise level is unpredictable. They also focus on capturing the correlation on the links independently, and do not extend their work in providing inference on the overall structure.

Non-parametric approaches have also been considered by Tieu et al., [12], where estimates are based on statistical dependence between camera sensor nodes. Statistical dependence is characterized by measures such as the internode transition times and the color histogram of objects. Tieu et al., show promising results on both simulated and real road networks. However, their algorithm depends on color histograms providing a weak labeling of the events.

Marinakis et al. have worked extensively on the problem of recovering topological information using binary sensor data [3], [4]. Their approach is to iteratively update a Markov network with trajectories sampled using the transition probabilities of the Markov model. A global iteration process optimizes parameters used in the Markov models while minimizing the complexity of the network model.

Although the basic setup is similar, the goal and approach is different from our work. Marinakis et al. focuses on recovering the underlying mobile topology of a sensor network, whereas our work focuses on estimating the actual trajectories the users take in the environment. Unlike Marinakis, we separate our algorithm into two steps where we first recover the link probabilities between sensors and use this information to assign trajectories on our time stamped data. We show that capturing characteristics of a link and exploiting this information frees us from having unnecessary assumptions about the number of users or a specific user behavior which is implied in Marinakis's work.

There is also a large field of work on target tracking using binary sensors, in which an accurate estimation of a user location is challenged by sparse sensor deployments and high mobility of the users [1], [2]. Problems with identity management also becomes a problem when multiple users are in the network [13], [14]. However, we emphasize that our work differs from general tracking problems in the sense that the objectives are different. We aim to extract the underlying mobility patterns of an environment by identifying the major user trajectories, while target tracking focuses on localizing a user's position by following and keeping track of a user's trajectory.

III. Problem Overview

We intend to understand the mobility of our environment given the sequence of timestamps at each sensor. A timestamp at a sensor indicates the time a user passed by the sensor. Thus our input is a 2-tuple, (t, n), where t is the time and n is the sensor node ID. With these input observations our objective is to identify the actual trajectories that generated these timestamps and assign a trajectory ID for each timestamp. We follow a two step procedure to accomplish this.

The first step is to identify links and its weights between nodes. The existence of a link indicates that a physical path exists between two nodes such that a user can walk between the nodes. The weight of a link, w_{ij}, is a multidimensional vector that expresses the characteristics of the physical path between the two nodes. Specifically, $w = (\mu, \sigma^2, v, \pi)$. μ is the average transition time users take to move between the two nodes, and σ^2 is the variance of the transition times. v is the volume: the number of times any user pass between the two nodes. π are the coefficients of the link signal and noise signal calculated by our mixture model. This is described in detail in Sec. IV. We measure the likelihood of the events given that a link actually exists, and show that with proper priors the actual probability of the link's existence can be calculated.

Our second step is to fit trajectories to our observed timestamps, i.e., to predict the underlying user trajectories that generated the timestamps at each sensor. We extend our input

2-tuple to a 3-tuple, (t, n, s), where s is the trajectory ID. For example, if we predict a user moved from node 1, 2, and 4 at times 10, 13, 18 respectively, we can assign this trajectory with an ID, i, and include (10,1,i), (13,2,i), and (18,4,i) in our 3-tuple. One benefit of our mixture model in the previous step is that it also provides the probability that a pair of timestamps between two sensors was generated by the same user trajectory. Thus using the previous step, we can measure the likelihood of a specific trajectory assignment on the timestamp observations. Due to the large combination of trajectory assignments possible, we use Markov Chain Monte Carlo (MCMC) sampling to search the optimal trajectory assignment on our observation.

IV. Link Detection

The first step in understanding the mobility of our environment is to find the correlation, if any, between the timestamps generated on a pair of nodes. For each node we have a signal corresponding to the time stamps of users passing by the specific node. Then, for each pair of nodes we generate a cross correlation histogram of the timestamp signals collected at each node. This histogram corresponds to the relative time difference of timestamp pairs between the two nodes. If a link exists, we can expect to find a correlated signal with an underlying distribution given by the transition times of mobile users passing by the two nodes.

A. Capturing correlation using a mixture model

Fig. 1a shows a cross correlated histogram of the timestamps between a sensor node pair. It can be seen there is a strong correlation around 5 seconds and another at 13. Our objective is to be able to capture these two correlations properly. Before we aim to do so we apply a simple window smoothing filter to smooth the cross correlated signal.

A correlation of timestamps is generally composed of two signals. One is the actual timestamp pairs that are generated by a user moving by the two nodes, which we from now on will refer to as link signals. The other is the noise signal, caused by a pairing of two timestamps on different user trajectories. For example in our correlation in Fig. 1a, a user A can pass by node 1 and 5 seconds later a user B can coincidently pass by node 2. The timestamp pair of these two events is accounted as noise in the histogram and should therefore be separated from the peaked link signal at 5 seconds. The level of noise in the correlated timestamps is dependent on the amount of simultaneous users in the environment. We show the effect of the noise level on the performance of our algorithm in Sec. VI.

We assume the transition time distributions to be normally distributed [11] and we use a variation of the well know Gaussian mixture model [15] to capture this correlation. For cases where the sensors are set far apart the transition time distribution has a longer tail and can be fitted with a more general Gamma distribution [12]. However, we find from our sensor deployment in two different office building settings (Sec. VI), that the Gaussian distribution is sufficient in capturing the correlation properly.

A naive fitting of the Gaussian mixture model, would easily fail in cases of high noise levels. We thus modify the model by fitting a combination of Gaussian distributions, $\mathcal{N}(\cdot)$, to capture the link signals and a uniform distribution, $\mathcal{U}(\cdot)$, to capture the noise signal. Eq. 1 shows the formulation of our model and we follow the standard method of fitting a mixture model by using Expectation Maximization (EM) to iteratively update the objective function and the parameters.

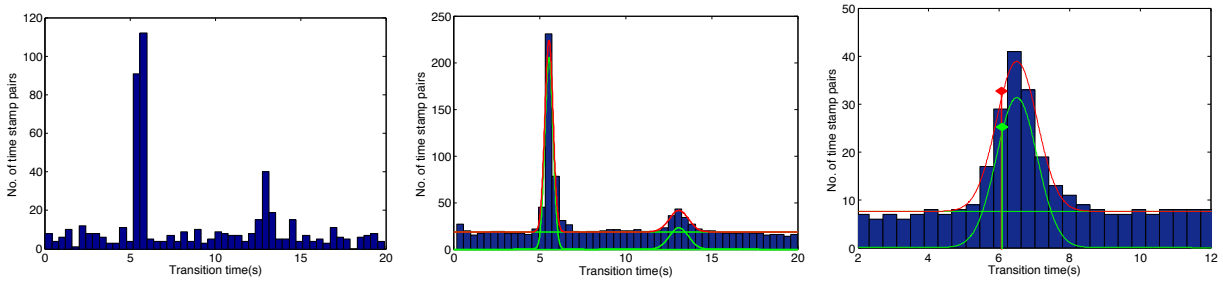
$$P(\tau \mid \mu, \sigma^2, \pi) = \sum_{i=1}^{N} \pi_i \mathcal{N}(\tau \mid \mu_i, \sigma_i^2) + \pi_{N+1}\mathcal{U}(\tau) \quad (1)$$

Here, the objective functions shows N Gaussian distributions corresponding to each link signal in the histogram, and the uniform distribution that captures the noise. τ corresponds to the time difference of a timestamp pair between two nodes, μ_i and σ_i^2 corresponds to the mean and variance in transition time of the i'th link signal. π_i, $i = 1, ..., N+1$ are the coefficients of the link signals and the noise signal. These coefficients define how strong a link signal is in comparison to the noise signal. We can also estimate the volume v of a link signal, i.e., the number of times users pass by these two nodes, by integrating the link signal. We can get a better estimation of the volume by trajectory fitting which is described later in detail (Sec. V).

Fig. 1b shows the result of our algorithm capturing two link signals and the noise signal correctly. One benefit of our model is that it can capture multi-modal distribution of a link. This is extremely useful in scenarios such as a city street deployment, where a clear bimodal distribution on the roads exist: pedestrians and cars. In general, for deployments inside a building, we only require one link signal to capture the transition times between two nodes. We continue to apply our mixture model on all pairs of nodes and find any correlation between them if they exist. It is important to note that a link does not necessarily indicate a direct path between the two nodes. Nodes A and C can be linked together by a node B in between, in which case our mixture model will still capture a correlation between A and C if a lot of users follow a trajectory A-B-C. More detail on how this impacts the accuracy of our trajectory assignments is discussed in Sec. V.

B. Link confidence

Although we now have a method to extract correlation between nodes, we still require a metric to evaluate our confidence on the link. One metric can be calculated by analyzing the strength of the link and noise signals. For example, if there is a link that only a few people traverse compared to the overall noise level, we would expect the peaks to be small in our cross correlated histograms. We need a method to evaluate the posterior probability of a link's existence given such observations. We apply Bayesian model comparison to compare our observations on two models: a model with a link signal $\mathcal{M}_L$ and a model without one $\mathcal{M}_{NL}$. Here $\mathcal{M}_L$ refers to



(a) (b) (c)

Fig. 1: The histogram of cross correlated timestamps between two nodes before (a), and after filtering (b). (b) shows the mixture model fit on top of the cross correlated histogram. The green line corresponds to the individual link and noise signals, and the red line corresponds to the combined signal. The mixture model gives the probability that a timestamp with τ(=6) second time difference comes from a user trajectory (c).

a link model where the distribution characteristics are specific to the one captured by our mixture model. Eq. (2) shows the Bayes factor between two models, where O is the total set of timestamp pairs between the two nodes.

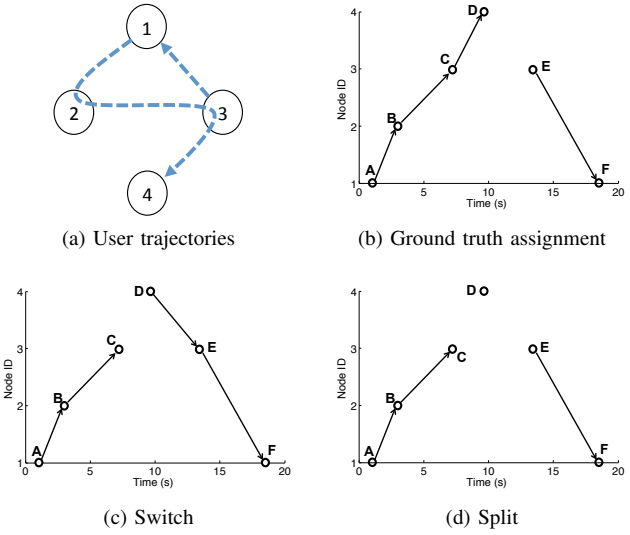
$$K = \frac{P(O|\mathcal{M}_L)}{P(O|\mathcal{M}_{NL})} = \frac{\prod_{i=1}^{N} P(\tau_i|\mathcal{M}_L)}{\prod_{i=1}^{N} P(\tau_i|\mathcal{M}_{NL})} \tag{2}$$

$P(O|\mathcal{M}_{NL})$ can be calculated by the uniform distribution and $P(O|\mathcal{M}_L)$ by the combined uniform and gaussian distribution fitted by our model.

The Bayes Factor K gives us a measure on how strong one model is over the other, and the higher this value is the more strongly the algorithm favors a model with a link over a model with no link. This metric is used later to evaluate the performance of our algorithm. We show in Sec. VI by introducing priors on $\mathcal{M}_L$ and $\mathcal{M}_{NL}$ we can also calculate the posterior probabilities of a link existence between two nodes. However, we have refrained from emphasizing too much on this because introducing these priors requires some prior knowledge of the network, and the overall posterior would be sensitive to these priors. In an event where we do have prior knowledge on our network, it is perfectly fine to multiply the additional prior ratio to give us an informative probabilistic score of a link.

C. Probability measure for trajectory fitting

A benefit of our mixture model is that it generates a method to calculate the actual probability a timestamp pair is associated with a link signal as opposed to a noise signal. Fig. 1c shows us how we can calculate the probability a timestamp pair with time difference 6 seconds is part of a link signal, i.e., generated by a user moving pass these two nodes. The probability will simply be the ratio of the link signal over the combined signal at τ=6 seconds. Thus the probability a timestamp pair at node i and j, with time difference τ, is generated by a specific user trajectory can generally be expressed as shown in Eq. (3).



(a) User trajectories (b) Ground truth assignment

(c) Switch (d) Split

Fig. 2: Two trajectories generated by users in a 4 node network (a), and the ground truth trajectory assignment (b). (c) shows a switch move made on timestamp D, and (d) shows a split move made on time stamp C.

$$p_{ij}(\tau) = \frac{\mathcal{N}_{ij}(\tau)}{\mathcal{N}_{ij}(\tau) + \mathcal{U}_{ij}(\tau)} \tag{3}$$

V. Trajectory Fitting

The idea of fitting trajectories on a sequence of observations has been discussed previously in literature [16], [4]. We propose a similar method extending on the probability measures found in our previous step (Sec. IV). Assuming we have N timestamps, we have a sequence of trajectory assignments $s = \{s_1, s_2, ..., s_N\}$ and a sequence of node assignments $n = \{n_1, n_2, ..., n_N\}$ for the sequence of timestamps $t = \{t_1, t_2, ..., t_N\}$. For example, the assignment in Fig. 2b will have a trajectory assignment of {'1', '1', '1', '1', '2', '2'}, and a node assignment of {1, 2, 3, 4, 3, 1} where '1'

is the first trajectory moving through nodes 1-2-3-4 and '2' is the second trajectory moving through nodes 3-1.

We defined the probability that a pair of timestamps at node i, j is part of a link signal, p_{ij}, in Eq. (3). We can find the probability of a trajectory assignment by multiplying the probability that it is a link signal, p_{ij}, if timestamps are assigned the same trajectory, and multiplying the probability that it is a noise signal, $1 - p_{ij}$, if timestamps are assigned a different trajectory. Eq. (4) shows the probability of a certain trajectory assignment on N timestamps.

$$P(s_1, s_2, ..., s_N) = \prod_{i=1}^{N} \prod_{j=1, j\neq i}^{N} Q(s_i, s_j) \tag{4}$$

where,

$$Q(s_i, s_j) = \begin{cases} p_{n_i n_j}, & if\ s_i = s_j \\ 1 - p_{n_i n_j}, & otherwise \end{cases}$$

Here we are implying independence on the link probabilities over multiple trajectories. Although this is not necessarily true, we show that this simple model is capable of extracting trajectories at a high level of accuracy. Details on the performance are shown in the experiment section (Sec. VI).

A. Sampling trajectory assignments

The possible number of trajectories assignments is too large for us to directly search for the optimal trajectory assignment. We use a Markov Chain Monte Carlo(MCMC) sampling method to sample trajectory assignments according to the underlying distribution. We initially start with each timestamp having its own trajectory ID. Then at each sample step we randomly select a timestamp and define two possible moves that alter from the current trajectory assignment: switch and split.

A switch move is when a trajectory assignment s_i changes to either its previous or next trajectory. For example, in Fig. 2c the trajectory assignment, s_D, is switched from trajectory '1' to trajectory '2'. A split move is when a trajectory is split into two or three trajectories. For example, Fig. 2d shows the 1st trajectory split into two trajectories, where the timestamp D is assigned a new independent trajectory. After this step $s_A = s_B = s_C =$'1', $s_E = s_F =$'2', and $s_D =$'3'. A trajectory would split into 3 trajectories if the selected timestamp was in the middle of a trajectory.

For each new trajectory assignment, $P(s'_1, s'_2, ..., s'_M)$, we use the Metropolis-Hastings algorithm [17], a method of MCMC sampling, and accept the sample according to an acceptance probability α.

$$\alpha = \min\left\{1, \frac{P(s'_1, s'_2, ..., s'_M)}{P(s_1, s_2, ..., s_M)}\right\} \tag{5}$$

The calculation for Eq. (5) can be greatly simplified since the components $Q(s_i, s_j)$ only change where a trajectory assignment has changed. For example, in case of the switch move in Fig. 2c, the probability ratio in Eq. (5) simplifies to $\frac{(1-p_{34})p_{43}}{p_{34}(1-p_{43})}$, since all other terms not related with timestamp C, D and E directly, will cancel out.

We keep track of the trajectory assignment that has the maximum assignment likelihood, $P_{MAX}(s_1, s_2, ..., s_N)$, and stop the sampling process if there is no change in the maximum trajectory assignment after a significant number of sample steps. This stopping criteria will depend on the size of our time sequence. In general, we find that 50,000 steps with no change is sufficient to find a near optimal assignment on a sequence of 10,000 timestamps. When the number of timestamps exceeds this, we divide the sequence into multiple sets and fit trajectories within each set separately. The fact that our algorithm is easily parallelized is a significant advantage for computational reasons. The fact that the algorithm can be divided into smaller sets is also beneficial in that the search for an optimal trajectory assignment can easily stop at a local minimum when the number of timestamps get large, and we are more likely to find a near optimal solution if the size of a set is smaller.

B. Increasing accuracy

We show later in our evaluation section that the algorithm performs at a surprising level of accuracy in estimating the trajectories, considering the lack of information we have for our inputs. We nonetheless discuss here, some of the reasons a false trajectory assignment can occur, and provide methods to overcome such faults.

False trajectory assignment occurs when there are a large number of simultaneous users in the environment. The clumped sequence of timestamp data can confuse the algorithm from choosing the optimal trajectory assignment. Moreover, since multiple users could be moving back to back on a similar trajectory, it might be insufficient to simply check the previous and next trajectory for our switch move. By exploring multiple trajectory candidates ahead and after the current selected trajectory it is more likely to search and find the optimal trajectory assignments on the timestamps. However, there is a tradeoff for this exploration in that the convergence time of our sampling procedure will increase.

Another scenario of false trajectory assignment can occur when a trajectory is assigned to two nodes that have an evident link signal, but is not necessary a direct link. For example nodes A and B can have a link but this does not necessarily mean there is a direct physical path connecting the two. Nodes A and B can be linked by another node C in between and not be connected directly, but still have a strong link if a lot of the users follow the trajectory A-B-C.

To overcome this situation, we can backtrack our fitted trajectories and eliminate any observations that are indirect links and redo our link detection. For example, for every trajectory A-B-C we fit, we can eliminate the observations between nodes A and C, and recalculate the signal and noise levels, π, from our mixture model. This updated probability measure will give us a more accurate estimate of when two timestamps can be directly linked as a sequence in a trajectory. We can update the trajectory assignments using these new set

of parameters, and the accuracy can be increased by iteratively repeating this step, which is sacrificed by the additional computation time.

VI. Evaluation

A. Evaluation Metric

In this section, we describe the metrics used to assess the performance of our algorithm. Since our algorithm is composed of two steps, we provide two metrics that evaluates the success at each step respectively.

1) Graph link detection: The purpose of using our mixture model on the cross correlated timestamps is two fold: to discover links between nodes, if they exist, along with its corresponding weights, and to find the probability that any pair of timestamps between two nodes are caused by a certain user trajectory. The later can be evaluated by assessing the quality of trajectory fitting. First, we focus on how well the algorithm can identify existing links, and predict the average transition time users take to traverse between two nodes.

The Bayes Factor, $K = \frac{P(O|\mathcal{M}_L)}{P(O|\mathcal{M}_{NL})}$, described in Sec. IV is a confidence indicator on the existence of a link between two nodes. In general, if a link does indeed exist one would expect the algorithm to assign a higher K value and thus weigh more towards the model with a link, $\mathcal{M}_L$. We calculate the average K values over all the direct links on the ground truth graph to assess the algorithm's confidence towards predicting links where they do indeed exist.

Our evaluation metric is geared towards avoiding any requirement of prior knowledge on the environment, but we can calculate the probability ratio on the existence of a link if we have prior knowledge on the ratio between the models. Since K is the likelihood ratio of our observations on the model with a link, $\mathcal{M}_L$, and without, $\mathcal{M}_{NL}$, we can estimate the posterior ratios of these two models if we have knowledge on the model priors.

$$\frac{P(\mathcal{M}_L|O)}{P(\mathcal{M}_{NL}|O)} = K \cdot \frac{P(\mathcal{M}_L)}{P(\mathcal{M}_{NL})} \tag{6}$$

The other metric for link detection is how accurately the algorithm detects the average transition times between nodes. To quantify this accuracy, we measure the mean squared error between the actual average transition time of users and the predicted one from our mixture model.

2) Trajectory fitting: We can assess the quality of our trajectories by identifying how many of the ground truth trajectories were correctly assigned by the algorithm. We can also weight the trajectories and give partial credits to a trajectory assignment by counting the overlap. For example, if 2/3 of the trajectory was properly assigned, the score would be 2/3 for that trajectory. Assuming N ground truth trajectories, a naive measure could be the following:

$$S_{naive} = \frac{1}{N}\sum_{i=1}^{N} \max_j \frac{\mathcal{L}(s_j^{fit} \cap s_i)}{\mathcal{L}(s_i)} \tag{7}$$

Here, s_i is the i'th ground truth trajectory, s_j^{fit} is the j'th fitted trajectory from our algorithm, and $\mathcal{L}(\cdot)$ is the length of a trajectory. We retrieve the j'th fitted trajectory that has most overlap on the i'th ground truth trajectory since the algorithm can assign multiple trajectories on a single ground truth trajectory.

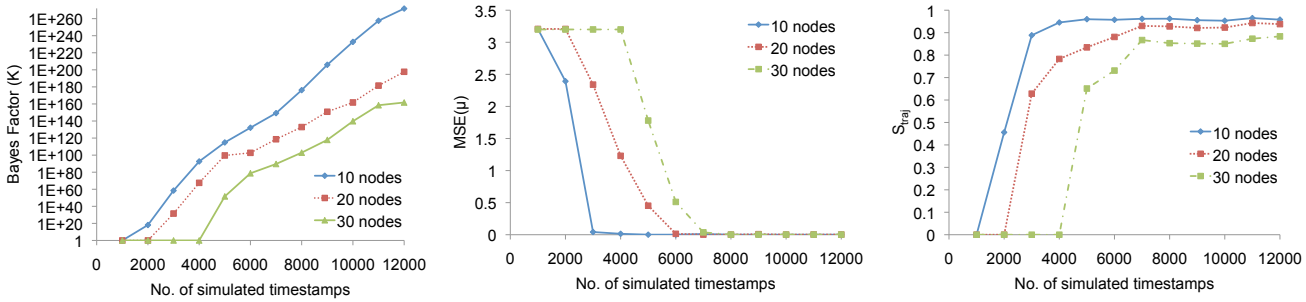
However, we see that this measure gives a perfect score on an assignment that fits a single trajectory to all the observations. Thus, we have a score that penalizes trajectory assignments that are longer.

$$S_{traj} = \frac{1}{N}\sum_{i=1}^{N} \max_j \frac{\mathcal{L}(s_j^{fit} \cap s_i)}{\max(\mathcal{L}(s_j^{fit}), \mathcal{L}(s_i))} \tag{8}$$

B. Simulation Experiments

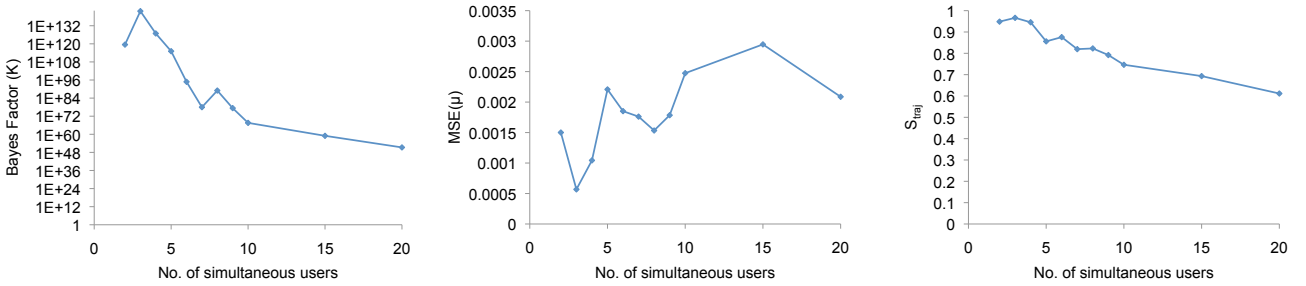
To assess the performance of our algorithm on ground truth data, we generated random planar graphs and collected timestamps at each node when a simulated user passes near the node. A user follows a modified random waypoint mobility model on the generated planar graph. We restrict the naive random waypoint model by not allowing agents to revisit its previous node by turning around. This is to prevent unlikely mobile behavior where a user can move back and forth in a corridor. Different users move at different speeds within a fixed range. A user continuously moves on the graph passing on average 3 to 4 nodes before pausing to indicate a single trajectory. After a pause of random length, the user begins another trajectory by randomly selecting a new node on the graph. The parameters used in the simulator are the number of nodes in the graph (N), the number of simultaneous users in the graph (P) and the total number of simulated timestamps (T). We evaluate the dependency of these parameters on our algorithm in the following sections.

1) Number of simulated timestamps: Fig. 3 shows how the performance metrics change depending on the number of timestamps input in our algorithm. Fig. 3a shows the average K values for different number of timestamps generated. Since the Bayes Factor is the likelihood ratio of all the observations on a link model, $\mathcal{M}_L$, to a no link model, $\mathcal{M}_{NL}$, the more observations there are that fit the link model the greater the Bayes Factor will be. We see also in Fig. 3b that the more timestamps we have the better our algorithm becomes at detecting links. We can also see that only a small number of timestamps is required to achieve a very high accuracy in detecting the transition times of a link. On average, we require 300 timestamp events per node to achieve a high quality estimation for μ. We also assess the trajectory assignment problem by varying the number of timestamps. Fig. 3c shows we only require a few timestamps to quickly achieve a near optimal score for trajectory fitting. This result demonstrates that we do not need a large number of data to provide high quality mobility inference. We later show in the results from the real deployments that a collection over a 2-week period in a normal office building environment is enough for accurate trajectory assignment.



(a) (b) (c)

Fig. 3: Algorithm performance with different the number of timestamps and number of nodes



(a) (b) (c)

Fig. 4: Algorithm performance with different number of simultaneous users

2) Number of nodes: Fig. 3 also shows the evaluation of our metric with different number of nodes in the network. We can, in general, see that the number of required timestamps increases for an accurate estimation, but the number of timestamps required per node is about the same. As long as there is sufficient amount of data between two nodes so that we can run the mixture model on a cross correlated timestamp data the algorithm performs well.

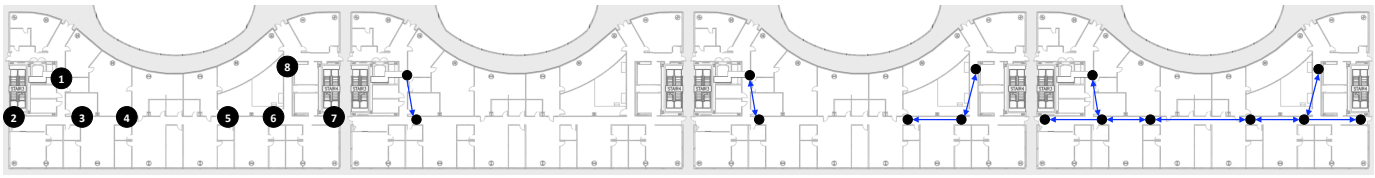
3) Number of simultaneous users: To understand the impact of the number of simultaneous users on link detection, we do a simulation on a 10 node network. We can expect the quality of link detection to drop as the level of noise will increase with more simultaneous users. Fig. 4 shows the result. We can see that the number of simultaneous users does not degrade link detection performance, and even with 20 simultaneous users in a 10 node network the link detection algorithm performs well, showing high K values and small MSE measures in μ. The number of simultaneous users, however, can have an effect in trajectory fitting algorithm since there are more choices of time stamps to choose from that have equally good fits. Fig. 4c shows that if there is less than one user for every two nodes in the network, we can achieve a trajectory estimation score of 0.8 and higher. This user density is much higher than a normal office environment, and we can add additional sensors if the building has a high user density.

C. Deployment Experiments

In this section, we describe the setup of our sensor network deployments in two office buildings and discuss the results.

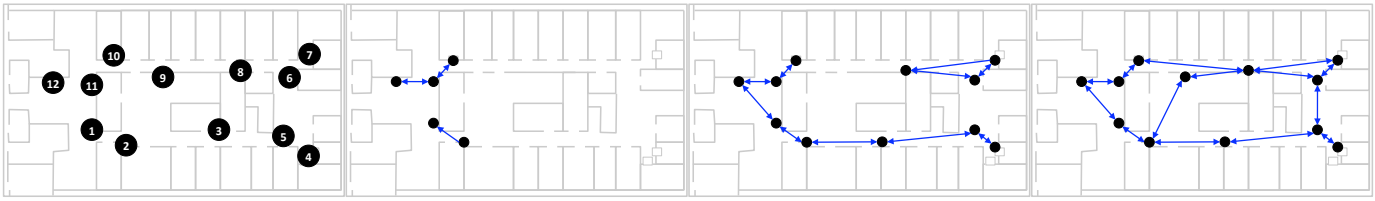
1) Hardware: We modified the TelosB [18] nodes by adding a low power Panasonic AMN41121 Passive Infrared (PIR) sensor to detect motion events. These sensors were further modified to restrict their field of view to a few degrees by using a tube with a non reflective coating. We mounted the sensors at approximately waist height to capture motion events when people walked past the PIR sensors. Since PIR sensors are more sensitive to objects moving across their field of view, we placed the sensors where people would more likely move perpendicular to the field of view of the sensor. These sensor motes were powered by a pair of high capacity lithium AA batteries. By combining a passive low power sensor, high capacity batteries, and the appropriate software, we were able to run our experiments for weeks at a time before having to replace the batteries.

2) Software: We wrote all of our software on the TinyOS [19] operating system. Our decision to use TinyOS was primarily due to the availability of a robust and low power networking stack. The motes use the CTP routing protocol [20] to send the sensor readings to the sink. We need accurate timestamps for the motion events, so we use the FTSP [21] time synchronization protocol to synchronize the times on our motes so that the timing error between the devices are kept



(a) Clark building deployment (b) Top trajectory (c) Top 5 trajectories (d) Top 20 trajectories

Fig. 5: Deployment in Clark building with discovered top trajectories



(a) Gates building deployment (b) Top 5 trajectories (c) Top 20 trajectories (d) Top 50 trajectories

Fig. 6: Deployment in Gates building with discovered top trajectories

within a few milliseconds. Lastly, we use BoX-MAC [22], a low power MAC for TinyOS. BoX-MAC puts the motes to sleep whenever there is no data transmission or reception so our devices predominately operate in an extremely low power sleep mode unless either a motion event is detected, or 200ms has elapsed since the mote was last awake. This combination of software allows the motes to run for around 4 weeks without changing the battery.

3) Deployments: Our first deployment was at Clark building where we mounted the nodes along the hallway and common area leading to the cubicles (Fig. 5). Our second deployment was at the B wing of the third floor of Gates building (Fig. 6). In this deployment, we mounted the nodes along the hallways and common areas. We instrumented all the entrances and exits from the deployment area so that we capture all the ingress and egress trajectories. These two deployments cover different hallway configuration and provide diverse test cases for our algorithm. We found that a two week period of data gathered on these networks were sufficient to generate the top trajectories used within the buildings. For the Clark building deployment we gathered a total of 7225 time stamps over 8 nodes during a period of 10 days. For the Gates building deployment we collected a total of 20969 time stamps over 12 nodes during a period of 14 days.

4) Network Performance: The performance of the network at Clark and Gates was similar so we only provide a summary of network performance for Clark. The network at Clark achieved an average delivery ratio of 91%, that is 91% of the packets sent by the sensor nodes were received by the sink and made available to the mobility inferencing algorithm. This delivery ratio is consistent with previously reported CTP results when it is run with a low power MAC. Because the mobility inferencing algorithm is robust to packet losses, we can still recover the trajectories reliably. The average path length was 1.47 hops.

5) Results from the Clark deployment: Results in Fig. 5 show the mobility structure recovered by overlaying the top trajectories fitted by our algorithm. Nodes 1 and 8 are the main entrance of this building floor and we can see the main trajectories start or end at these nodes. Each of these links between sensors are weighted with parameters $w = (\mu, \sigma^2, v, \pi)$ providing the characteristics of each link. Each link is a directed link, and we can see that the algorithm properly detects the one way exits at node 2 and 7 (one can open the door at these exits only from inside the building), by only being able to find trajectories pointing outwards at these nodes.

6) Results from the Gates deployment: We show a slightly more complicated deployment in Fig. 6 where the main entrance/exit is at node 12, and there are back door entrance/exit at node 6 and 5. Sensor node 9 is in front of an opening users can traverse to get to the other side of the floor. We can see that the algorithm properly captures all the movement through this opening. The absence of trajectories from 10 and 9 can be explained by users moving through another opening left to node 9.

To evaluate the accuracy of our algorithm in a real office environment, we generated ground truth trajectories at a time when no one else was on the building floor. Three users moved inside our deployment simultaneously following a sequence of 20 predefined trajectories. These trajectories were 2-10 nodes in length. At the end of a trajectory, the user would pause for approximately 10 seconds before starting a new trajectory. We later applied our trajectory fitting algorithm and measured how many of these trajectories the algorithm was capable of detecting. We looked into the specific time when we conducted our experiment within the 2 week period and compared the trajectories extracted from our algorithm to the actual trajectories the users took.

The main focus of this experiment was to evaluate the tra-

jectory fitting part of our algorithm, and to asses the accuracy of fitting trajectories once the link and average transition times were learned over some period of time, which was 2 weeks in our setup. Following our metric discussed above we obtained a trajectory score of $S_{traj} = 0.87$. This shows that the algorithm was accurate in trajectory fitting and is consistent with the results from the simulations.

VII. Conclusion

We have shown that a large amount of information can be extracted from extremely simple, unlabeled data. By analyzing the correlation between sensor inputs, we can find not only whether nodes are connected, but also information about the transition time distribution, volume and level of noise. We further use this to assign each timestamp with a trajectory number and predict highly popular trajectories in the environment. We show from our simulation results that our algorithm requires only a small amount of data to accurately predict the trajectories. Moreover, the fact that our algorithm can be easily parallelized benefits us with working on any additional data we can accumulate from the sensors. We also showed that the algorithm correctly predicts the trajectories in real deployments with noisy sensor readings and multiple people traversing the set of sensors.

In the future, we would like to explore the benefits of using weakly labeled data in our framework, and see how a few instances of labeled trajectories can increase our inference on the mobility patterns. Moreover, we aim to extend our framework to handle higher order probability models when fitting trajectories on the timestamps.

VIII. Acknowledgments

We gratefully acknowledge the support from the Stanford Army High Performance Computing Research Center grant W911NF-07-2-0027, ARO grant W911NF-10-1-0037, ONR grant N0001470710747, and the Samsung Scholarship.

References

[1] J. Singh, U. Madhow, R. Kumar, S. Suri, and R. Cagley, "Tracking multiple targets using binary proximity sensors," in *Proceedings of the 6th international conference on Information processing in sensor networks*, ser. IPSN '07. ACM, 2007, pp. 529–538.

[2] N. Shrivastava, R. M. U. Madhow, and S. Suri, "Target tracking with binary proximity sensors: fundamental limits, minimal descriptions, and algorithms," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, ser. SenSys '06. ACM, 2006, pp. 251–264.

[3] D. Marinakis and G. Dudek, "Topological mapping through distributed, passive sensors," in *International Joint Conference on Artificial Intelligence*, Hyderabad, India, Jan. 2007.

[4] ——, "Self-calibration of a vision-based sensor network," *Image and Vision Computing*, vol. 27, pp. 116–130, January 2009.

[5] H.-l. Chang, J.-b. Tian, T.-T. Lai, H.-H. Chu, and P. Huang, "Spinning beacons for precise indoor localization," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. ACM, 2008, pp. 127–140.

[6] K. Chintalapudi, A. Padmanabha Iyer, and V. N. Padmanabhan, "Indoor localization without the pain," in *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, ser. MobiCom '10, 2010, pp. 173–184.

[7] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Comput. Surv.*, vol. 38, no. 4, p. 13, 2006.

[8] J. Yu and S. LaValle, "Tracking hidden agents through shadow information spaces," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, May 2008, pp. 2331–2338.

[9] C. Niu and E. Grimson, "Recovering non-overlapping network topology using far-field vehicle tracking data," *Pattern Recognition, International Conference on*, vol. 4, pp. 944–949, 2006.

[10] B. Kusy, H. Lee, M. Wicke, N. Milosavljevi'c, and L. Guibas, "Predictive qos routing to mobile sinks in wireless sensor networks," in *Proc. of the International Conference on Information Processing in Sensor Networks (IPSN'09), to appear*, April 2009.

[11] D. Makris, T. Ellis, and J. Black, "Bridging the gaps between cameras," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 2, pp. 205–210, 2004.

[12] K. Tieu, G. Dalley, and W. Grimson, "Inference of non-overlapping camera network topology by measuring statistical dependence," in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 2, Oct. 2005, pp. 1842–1849 Vol. 2.

[13] J. Shin, L. J. Guibas, and F. Zhao, "A distributed algorithm for managing multi-target identities in wireless ad-hoc sensor networks," in *In IPSN 03: Information Processing in Sensor Networks*, 2003.

[14] S. Oh, L. Schenato, and S. Sastry, "A hierarchical multiple-target tracking algorithm for sensor networks," in *In Proc. of the International Conference on Robotics and Automation*, 2005.

[15] D. Titterington, A. Smith, and U. Makov, *Statistical analysis of finite mixture distributions*. Wiley, 1985.

[16] S. Oh, S. Russell, and S. Sastry, "Markov chain monte carlo data association for general multiple-target tracking problems," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1, Dec. 2004, pp. 735–742 Vol.1.

[17] W. Hastings, "Monte carlo sampling methods using markov chains and their applications," *Biometrika*, Jan 1970.

[18] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. IEEE Press, 2005, p. 48.

[19] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An operating system for wireless sensor networks," in *Ambient Intelligence*. Springer-Verlag, 2004.

[20] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection Tree Protocol," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*, November 2009.

[21] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *ACM SenSys '04*. ACM Press, 2004, pp. 39–49.

[22] D. Moss and P. Levis, "BoX-MACs: Exploiting Physical and Link Layer Boundaries in Low-Power Networking," *Stanford Information Networks Group Technical Report SING-08-00*, 2008.