Tracking Moving Objects in Anonymized Trajectories

Nikolay Vyahhi[1], Spiridon Bakiras[2], Panos Kalnis[3], and Gabriel Ghinita[3]

[1] Dept. of Computer Science, St. Petersburg State University, St. Petersburg, Russia
vyahhi@gmail.com
[2] Dept. of Mathematics and Computer Science, John Jay College, City University of New York
sbakiras@jjay.cuny.edu
[3] Dept. of Computer Science, National University of Singapore, 117590 Singapore
{kalnis, ghinitag}@comp.nus.edu.sg

Abstract. Multiple target tracking (MTT) is a well-studied technique in the field of radar technology, which associates anonymized measurements with the appropriate object trajectories. This technique, however, suffers from combinatorial explosion, since each new measurement may potentially be associated with any of the existing tracks. Consequently, the complexity of existing MTT algorithms grows exponentially with the number of objects, rendering them inapplicable to large databases. In this paper, we investigate the feasibility of applying the MTT framework in the context of large trajectory databases. Given a history of object movements, where the corresponding object *ids* have been removed, our goal is to track the trajectory of every object in the database in successive timestamps. Our main contribution lies in the transition from an exponential solution to a polynomial one. We introduce a novel method that transforms the tracking problem into a min-cost max-flow problem. We then utilize well-known graph algorithms that work in polynomial time with respect to the number of objects. The experimental results indicate that the proposed methods produce high quality results that are comparable with the state-of-the-art MTT algorithms. In addition, our methods reduce significantly the computational cost and scale to a large number of objects.

1 Introduction

Recent advances in wireless communications and positioning devices have generated significant interest in the collection of spatio-temporal (i.e., trajectory) data from moving objects. Any GPS-enabled mobile device with sufficient storage and computational capabilities can benefit from a wide variety of location-based services. Such services maintain (at a centralized server) the locations of a large number of moving objects over a long period of time. As an example, consider a traffic monitoring system where each car periodically transmits its exact location to a database server. The resulting trajectories can be queried by a user to retrieve important information regarding current or predicted traffic conditions at various parts of the road network.

Nevertheless, the availability of such data at a centralized location raises concerns regarding the privacy of the mobile clients, especially if the data is distributed to other parties. A simple solution that partially solves this problem is to anonymize the trajectory data, by not publishing the user *id*[1]. In the traffic monitoring system, for instance,

[1] Assigning a fake *id* does not guarantee anonymity, since a user may be linked to a specific trajectory using background knowledge (e.g., known home address as starting point).

the *ids* of the individual users are not essential for measuring the traffic level on a road segment. Therefore, the mobile users may not be willing to identify themselves, and may choose to transmit only their location, but not their *id*. Furthermore, anonymous data collection may be the only option in certain environments. For instance, in the traffic monitoring system the trajectory data may be collected by sensors that are deployed throughout a city. In this scenario, every vehicle that passes in front of a sensor automatically generates a measurement that contains no information regarding its identity.

Even though anonymization is important for protecting the privacy of mobile users, detailed trajectory data (i.e., coupled with object identifiers) are valuable in numerous situations. For example, a law enforcement agency trying to track a suspect that was seen in a car at a specific time, can certainly benefit from stored trajectory information. In this scenario, anonymization severely hinders the tracking process, since there is no information to link successive measurements to the same trajectory. A straightforward solution, given the similarity of the two problems, is to leverage existing methods that are used in radar tracking applications. Multiple target tracking (MTT) [1, 2] is a well-studied technique in the field of radar technology, which associates anonymized measurements with the appropriate object trajectories. This technique, however, is not practical. The reason is that every possible combination of measurements must be considered, in order to minimize the overall error across all trajectories. Consequently, the complexity of existing MTT algorithms grows exponentially with the number of objects, rendering them inapplicable to large databases.

In this paper, we investigate the feasibility of applying the MTT framework in the context of large trajectory databases. Given a history of object movements, where the corresponding object *ids* have been removed, our goal is to track the trajectory of every object in the database in successive timestamps. Our main contribution lies in the transition from an exponential solution to a polynomial one. To this end, we introduce a novel method that transforms the tracking problem into a min-cost max-flow problem. We then utilize well-known graph algorithms that work in polynomial time with respect to the number of objects. To further reduce the computational cost, we also implement a pruning step prior to the construction of the flow network. The objective is to remove all the measurement associations that are not feasible (e.g., due to a maximum velocity constraint). We perform an extensive experimental evaluation of our approach, and show that the proposed methods produce high quality results that are comparable with the state-of-the-art MTT algorithms. In addition, our methods reduce significantly the computational cost and scale well to a large object population.

The rest of the paper is organized as follows: Section 2 defines formally the problem, whereas Section 3 surveys the related work. A detailed description and analysis of our algorithm is given in Section 4. In Section 5 we evaluate experimentally our method. Finally, Section 6 summarizes the results and presents directions for future work.

2 Problem Formulation

Let $H = \{S_1, S_2, \ldots, S_M\}$ be a long, timestamped history. A *snapshot* S_i of H is a set of locations (measurements) at time t_i; the time difference $t_{i+1} - t_i$ between consecutive timestamps is not constant. Each snapshot contains measurements from

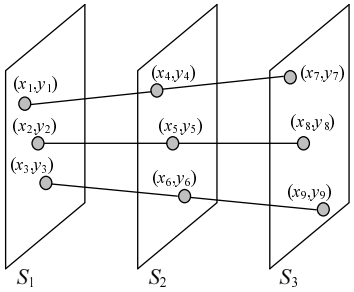


Fig. 1. Multiple target tracking (MTT) example

exactly N objects, i.e., we assume that (1) an existing object may not disappear and new objects may not appear during the interval $[t_1, t_M]$ and (2) the measurements are complete (there are no missing values). These assumptions may not hold in some cases, but our goal in this paper is to solve a restricted version of the problem. We plan to relax these constraints as part of our future work. Finally, we assume that the locations are anonymized, meaning that there is no object *id* that matches a certain location; any location measurement may correspond to any of the N objects.

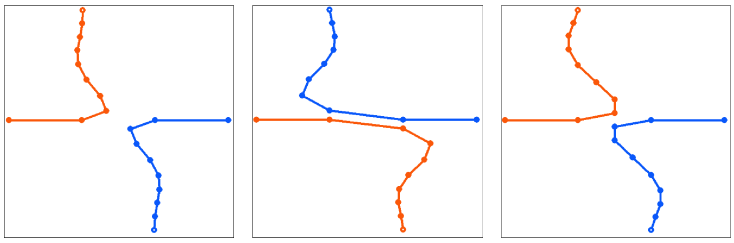
Given N objects and a history H spanning M timestamps, an MTT query returns a set of N trajectories, where each trajectory i has the form $\{(x_{i_1}, y_{i_1}, t_1), (x_{i_2}, y_{i_2}, t_2), \ldots, (x_{i_M}, y_{i_M}, t_M)\}$. Each triple in the above set corresponds to the location of the object at each of the M timestamps. To illustrate the significance of this result, consider the following scenario: A suspect was seen driving in the vicinity of his home address at time t_1. What a data analyst may want to do, is issue a range query and retrieve a set of points (i.e., measurements) that may be associated with the suspect (at time t_1). After the MTT query is resolved, each of these points will be the source of a unique trajectory that will identify possible locations of the suspect at subsequent timestamps.

Figure 1 shows an example MTT query with $M = N = 3$. Each line connecting two measurements in successive timestamps indicates that the two measurements belong to the same trajectory. The three trajectories are disjoint and are formed such that the overall error is minimized (the details of the error function are discussed in Section 4). Given the illustrated associations in Figure 1, the topmost trajectory is represented as $\{(x_1, y_1, t_1), (x_4, y_4, t_2), (x_7, y_7, t_3)\}$.

3 Related Work

Multiple target tracking has been studied extensively for several decades, and a variety of algorithms have been proposed that offer different levels of complexity and tracking quality. They can be classified into three major categories: nearest neighbor (*NN*), joint probabilistic data association (*JPDA*), and multiple hypotheses tracking (*MHT*).

NN techniques [2] require a single scan of the dataset; for every set of measurements (i.e., from one timestamp), each sample is associated with a single track. The objective is to minimize the sum of all distances, where the distance is defined as a



(a) Reid's MHT (b) GNN (c) Our algorithm

Fig. 2. Trajectory reconstruction for different methods

function of the difference between the actual and predicted values. Among existing NN algorithms, the best is the global nearest neighbor (*GNN*) approach [3]. JPDA algorithms [1] also require a single scan and, for every pair of measurement-track, the probability of their association is calculated as the sum of the probabilities of all joint events. An experimental evaluation of several NN and JPDA algorithms can be found in [3]. Even though some of these methods run in polynomial time (due to their greedy nature that minimizes the error at each timestamp independently), their tracking quality is not good, leading to many false associations.

Reid's algorithm [4] is the most representative of the MHT methods. Instead of associating each measurement with a single track, multiple hypotheses are maintained, whose joint probabilities are calculated recursively when new measurements are received. Consequently, each measurement is associated with its source based on both previous and subsequent data (multiple scans). During this process unfeasible hypotheses are eliminated and similar ones are combined. Reid's algorithm produces high quality results, but its complexity grows exponentially with the number of measurements.

An example that illustrates the superiority of multiple scan techniques over their single scan counterparts is presented in Figure 2. In this example, two objects move towards each other, until they "meet"; then, they suddenly change their trajectories and move at opposite directions. GNN makes the wrong track assignments when the objects are close to each other, just because these assignments happened to minimize the error at some particular timestamp. On the other hand, Reid's algorithm tracks the two objects successfully, since it minimizes the error across all timestamps. This figure also shows the output of our method, which exhibits an accuracy that is similar to Reid's algorithm but is able to run in polynomial time (as we will illustrate in the following sections). The slight differences in the output between Reid's algorithm and ours, are due to the filters that are used to smooth the trajectories (Kalman filter for Reid, as opposed to a simpler filter for our method).

To reduce the complexity of the tracking process, [5, 6] employ clustering. They group the set of measurements before forming the candidate tree, in order to remove unlikely associations. In this way, the problem is partitioned into smaller sub-problems that are solved more efficiently. Although this approach reduces the complexity, it still utilizes single scan techniques that are not accurate.

Another interesting application of multiple target tracking is investigated in [7], where the objective is to discover associations among asteroid observations that correspond to the same asteroid. The authors introduce an efficient tree-based algorithm, which utilizes a pruning methodology that reduces significantly the search space. However, their problem settings are different from ours, since (1) they assume that there is a given motion model that has to be obeyed, and (2) they are interested in returning those sets of observations that conform to the motion model.

Finally, the idea of applying MTT techniques for the reconstruction of trajectories from anonymized data, was introduced in [8]. The authors use five real paths and show that Reid's algorithm is able to associate the majority of the measurements with the correct objects. However, their objective is not how to efficiently track multiple targets, but rather how to enhance the privacy of the users through path perturbation. In particular, they modify the original dataset in such a way that Reid's algorithm is confused.

4 Tracking Algorithm

This section discusses the details of our MTT algorithm. First, we present a brief overview of the min-cost max-flow problem. Then, we explain how to construct the graph from the history of location measurements and present a pruning mechanism that reduces significantly the graph size. Finally, we discuss the implementation details of our algorithm and analyze its computational complexity.

4.1 Preliminaries

A *flow network* [9] is a directed graph $G = (V, E)$, where V is a set of vertices, E is a set of edges, and each edge $(u, v) \in E$ has a capacity $c(u, v) \geq 0$. If $(u, v) \notin E$, it is assumed that $c(u, v) = 0$. There are two special vertices in a flow network: a source s and a destination t. A *flow* in G is a real-valued function $f : V \times V \rightarrow \mathbf{R}$, satisfying the following properties:

1. **Capacity constraint**: For all $u, v \in V$, we require $f(u, v) \leq c(u, v)$.
2. **Skew symmetry**: For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.
3. **Flow conservation**: For all $u \in V \setminus \{s, t\}$, we require $\sum_{v \in V} f(u, v) = 0$. In other words, only s can produce units of flow, and only t can consume them.

The *max-flow* problem is formulated as follows: given a flow network G, find a flow of maximum value between s and t.

The *min-cost max-flow* problem is a generalization of max-flow, where:

1. For every $u, v \in V$ the edge (u, v) has a cost $w(u, v)$, and we require $w(u, v) = -w(v, u)$.
2. The flow conservation property of the flow network is replaced by the following *balance constraint* property: For all $u \in V$, $b(u) = \sum_{v \in V} f(u, v)$. Note that, $b(u)$ may have non-zero values for vertices other than the source or the sink. In other words, every node in the network may be a producer or consumer of flow units, as long as the following flow conservation condition is satisfied: $\sum_{u \in V} b(u) = 0$.

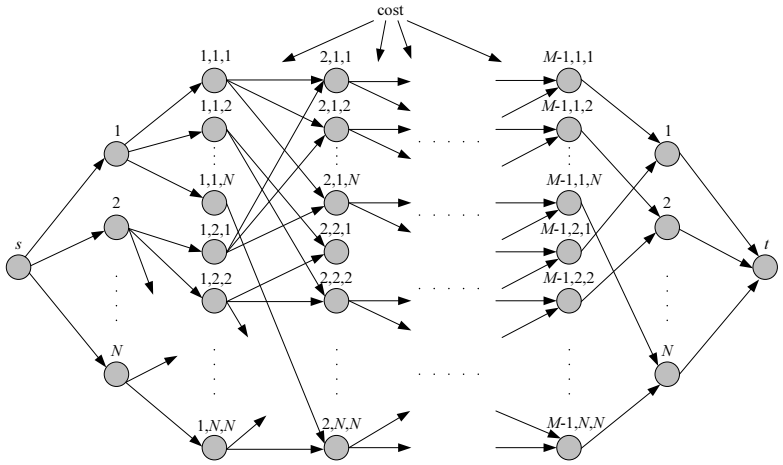


Fig. 3. Multi-target tracking (MTT) flow network

The cost of a flow f is defined as

$$cost(f) = \sum_{(u,v)\in E} w(u,v)f(u,v)$$

and the objective of the min-cost max-flow problem is to find the max-flow with the minimum cost.

4.2 Problem Transformation

A straightforward transformation of the MTT problem into a flow network is shown in Figure 3. Flow units are produced at the source s and consumed at the sink t; our objective is to send a total of N flow units from s to t, each one identifying a single object trajectory. All edges have capacity 1 in the forward direction, and 0 in the reverse direction. Also, every edge (u, v) in the middle of the network (as shown in the figure) has cost value $w(u, v)$ in the forward direction, and $-w(u, v)$ in the reverse direction. The rest of the edges have zero cost.

The N vertices that are directly connected to s correspond to the first snapshot of measurements (one vertex for each location). Following these vertices are series of columns containing N^2 nodes each. Every node in these columns is identified by a triplet (t_i, p_i, p_j), which has the following meaning: if a positive amount of flow runs through this node, then the underlying object moves from location p_i in timestamp t_i to location p_j in timestamp t_{i+1}. Consequently, edge $(t_i, p_i, p_j) \rightarrow (t_{i+1}, p_j, p_k)$ represents a partial trajectory from three consecutive timestamps ($p_i \rightarrow p_j \rightarrow p_k$), where $p_i, p_j, p_k \in [1..N]$.

The cost for the aforementioned edge is equal to the association error of the third measurement. As shown in Figure 4, if the first two measurements (p_i and p_j) belong to the same track, their values can be used to predict the next location of the object, based

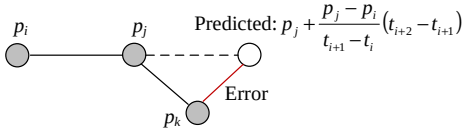


Fig. 4. Association error

on the assumption that objects move on a straight line with constant speed. Therefore, for every possible location p_k, we can calculate the error of associating this measurement with any of the existing tracks. This definition of error is also used in [4]. Note that our method minimizes the sum of errors across all trajectories (similar to multiple hypotheses tracking), as opposed to methods that work in a single scan. Finally, the N nodes connected to the sink t correspond to the last set of measurements, and indicate the final positions of the moving objects.

Observe that the above flow network may lead to incorrect trajectories, by associating a single measurement with multiple tracks. For instance, if in the final solution we allow a positive amount of flow through edges $(1,1,1) \rightarrow (2,1,1)$ and $(1,2,1) \rightarrow (2,1,2)$ (Figure 3), then location 1 in timestamp 2 belongs to two different trajectories. One way to overcome this limitation is to create a bottleneck edge (with capacity 1) for each measurement that only allows a single unit of flow (i.e., track) to go through. We call this structure a *block*. Figure 5(a) illustrates the (m,k)–block, i.e., the block associated with the k_{th} measurement of the m_{th} timestamp. Let us use the notation $p_{m,k}$ to identify that particular point location. Then, this block represents all partial tracks $p_{m-1,i} \rightarrow p_{m,k} \rightarrow p_{m+1,j}, \forall i,j \in [1..N]$. Since the capacity of the middle edge is equal to 1, only one of these tracks can be selected.

Every (m,k)–block, where $1 < m < M$ and $1 \leq k \leq N$, is characterized by the following matrix:

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,N} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,N} \\ \vdots & \vdots & \vdots & \vdots \\ c_{N,1} & c_{N,2} & \cdots & c_{N,N} \end{pmatrix}$$

where $c_{i,j}$ is the error in track $p_{m-1,i} \rightarrow p_{m,k} \rightarrow p_{m+1,j}$, i.e., the distance between the predicted location (based on the values of $p_{m-1,i}$ and $p_{m,k}$), and $p_{m+1,j}$. However, the block structure consists of only $(2N+1)$ edges, which are not sufficient to represent the N^2 error values that are included in matrix C. Therefore, we modify the aforementioned block structure, and replace the middle part of the block with $2N$ vertices and N^2 edges. The result is shown in Figure 5(b). The N^2 edges connecting the two middle columns have the cost values associated with matrix C, while the remaining edges have cost equal to zero, i.e., they do not affect the process of the min-cost max-flow calculation.

The difference of the modified block structure compared to the rest of the flow network, is that we need to manually route the flows inside the block in order to guarantee that only one flow unit goes through. Specifically, when a positive amount of flow runs through a certain block, that block is automatically marked as *active* and the identifier of the edge occupying the block is recorded. An active block may only output a single flow

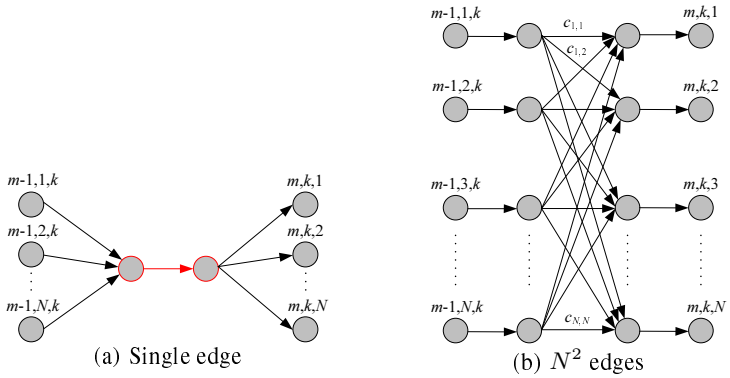


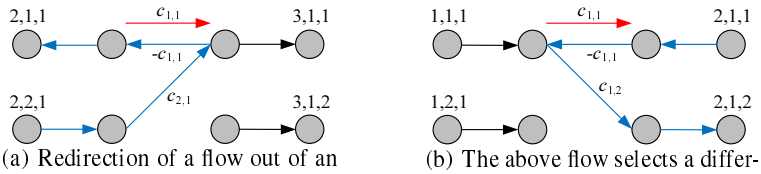
Fig. 5. Block structure for measurement $p_{m,k}$

unit, so an additional incoming flow has to be redirected backward in order to cancel the existing flow (hence the negative weight values on the reverse edges). In particular, a new flow is forced back through the reverse path of the existing flow, in order to select a new location in the previous timestamp. This is depicted in Figure 6(a), where the block is occupied by the flow with cost $c_{1,1}$. When a new flow enters from vertex $(2, 2, 1)$, it is only allowed to follow the path indicated by the arrows, which takes the flow in the reverse direction towards vertex $(2, 1, 1)$ and cancels the original flow. Next, as shown in Figure 6(b), the incoming flow enters the block of the previous timestamp, where it also cancels the flow with cost $c_{1,1}$ and then follows the path to vertex $(2, 1, 2)$. Consequently, it selects measurement 2 at timestamp 3 (instead of measurement 1), which results in two distinct trajectories.

There are N blocks in every timestamp, each one contributing $O(N)$ vertices and $O(N^2)$ edges to the overall network. Therefore, the total number of vertices in the flow network is $|V| = O(MN^2)$, whereas the total number of edges is $|E| = O(MN^3)$.

4.3 Improving the Running Time

Solving the min-cost max-flow problem requires multiple shortest path calculations on the MTT flow network (discussed in the next section). Therefore, the size of the network is crucial for maintaining a reasonable running time. In its current form, however, the size of the flow network becomes prohibitively large when the number of measurements increases. To this end, we propose a pruning technique that may reduce significantly the size of the network. Observe that any object can travel at most $R_{\max}$ distance between two consecutive timestamps. The actual value of $R_{\max}$ depends on (i) the maximum speed of the objects and (ii) the time interval between the two timestamps. Consequently, every measurement $p_{m,k}$ can only be associated with those measurements $p_{m+1,i}$, $\forall i \in [1..N]$, such that the distance between the two points is less than $R_{\max}$. We can leverage this constraint in order to reduce the number of vertices and edges inside each block. Specifically, if we assume that there are, on average, K



(a) Redirection of a flow out of an active block (block for $p_{3,1}$)

(b) The above flow selects a different location (block for $p_{2,1}$)

Fig. 6. Functionality of new block structure

feasible associations for any measurement $p_{m,k}$, the number of vertices in the flow network is reduced to $|V| = O(MNK)$, while the total number of edges is reduced to $|E| = O(MNK^2)$. This may result in significant savings when $K \ll N$.

4.4 The MTT Algorithm

We have a single source s that needs to send N units of flow towards the destination t. Among all feasible max-flows, we are interested in finding the one with the minimum cost. A very efficient method for solving the min-cost max-flow problem is the Successive Shortest Path Algorithm [10]. It leverages the Ford-Fulkerson algorithm [9] that solves the max-flow version of the problem. The Ford-Fulkerson algorithm starts with $f(u, v) = 0$ for all $u, v \in V$, and works iteratively by finding an *augmenting path* where more flow can be sent. The augmenting paths are derived from the *residual network* G_f that is constructed during each iteration. Formally, $G_f = (V, E_f)$, where $E_f = \{u, v \in V : c_f(u, v) > 0\}$. $c_f(u, v)$ is called the residual capacity and is equal to $c(u, v) - f(u, v)$. Note that when an edge (u, v) carries a positive amount of flow in the flow network, it will be replaced by edge (v, u) in the residual network (as shown in Figure 6). This means that the residual network may contain edges with negative weights, since $w(v, u) = -w(u, v)$.

In the Successive Shortest Path Algorithm, instead of finding an augmenting path, we find the path with the minimum cost (given the weight values of the edges on the residual graph). Since the flow network may contain weights with negative values, we need to utilize the Bellman-Ford algorithm [11] for the shortest path calculations. This is not very efficient, as the Bellman-Ford algorithm has worst-case complexity $O(|V| \cdot |E|)$. In our MTT network, this translates to $O(M^2N^2K^3)$.

Instead, we use a well-known technique called *vertex potentials*, which transforms the network into one with non-negative costs (provided that there are no negative cost cycles). For every edge $(u, v) \in E$, where vertices u, v have potential $p(u)$ and $p(v)$, respectively, the *reduced cost* of the edge is given by: $w_p(u, v) = w(u, v) + p(u) - p(v) \geq 0$. It can be proved that the min-cost max-flow problems with edge costs $w(u, v)$ or $w_p(u, v)$ have the same optimal solutions. Therefore, by updating the node potentials, we can utilize a more efficient shortest-path algorithm during the iterations of the Ford-

Algorithm **MTT(*H*, *M*, *N*)**
1. Construct flow network from H
2. **for** each $(u, v) \in E$ // Initialize flows
3. $f(u, v) = 0$
4. $f(v, u) = 0$
5. **for** each $u \in V$ // Initialize node potentials
6. $p(u) = 0$
7. **for** $i = 1$ to N
8. Find shortest path p from s to t in G_f
9. **for** each $u \in V$ // Update node potentials
10. $p(u) = p(u) + d(s, u)$
11. **for** each $(u, v) \in p$ // Augment flow across path p
12. $f(u, v) = f(u, v) + 1$
13. $f(v, u) = -f(u, v)$
14. **return** N trajectories

Fig. 7. The MTT algorithm

Fulkerson algorithm. Node potentials are initially set to zero[2], and are updated as follows: after the calculation of the shortest path, for every $u \in V$, $p(u) = p(u) + d(s, u)$, where $d(s, u)$ is the length of the shortest path from s to u.

The pseudo-code of our MTT algorithm is shown in Figure 7. It begins by constructing the flow network (as explained in Sections 4.2 and 4.3) from the history of measurements H. Then (lines 2-6), it initializes the flows and node potentials. At each iteration of the Successive Shortest Path Algorithm (lines 8-13), a single unit of flow is added to the network; the algorithm terminates after N iterations. The resulting trajectories are returned by following each flow unit from s to t through the flow network.

Before analyzing the computational complexity of our algorithm, we should briefly discuss a common problem that may occur in min-cost max-flow calculations. Due to the negative weights of some edges in the residual network, there is a possibility that negative cost cycles exist (we actually encountered this problem in our experiments). In this case, the shortest-path calculations can not be performed and the algorithm fails. Instead of terminating the algorithm when a negative cost cycle is detected, we implement a greedy approach that may generate non-optimal solutions. In particular, we (1) output all the tracks that are discovered so far (which might not be optimal), (2) remove all the vertices and edges associated with these tracks from the flow network, and (3) start a new min-cost max flow calculation on the reduced graph.

4.5 Complexity

The computational complexity of the MTT algorithm shown in Figure 7, is directly related to the complexity of the underlying shortest-path algorithm[3]. Theoretically,

[2] If prior to the first iteration of the algorithm there exist negative costs, Bellman-Ford must be invoked to remove them. In our case, however, we do not have negative costs before the first iteration, since the total flow inside the network is zero.

[3] The complexity of graph construction is $O(MN^2 + MNK^2)$ and can be ignored.

the fastest running time is achieved with Dijkstra's algorithm [12], using a Fibonacci heap implementation for the priority queue. The complexity of Dijkstra's algorithm is $O(|V|\log|V| + |E|) = O(MNK\log(MNK) + MNK^2)$. Thus, the total running time (due to N iterations) is $O(MN^2K(\log(MNK) + K))$. This corresponds to the main contribution of our work, i.e., a multiple hypotheses tracking algorithm that works in polynomial time, instead of exponential.

We have also experimented with other implementations of shortest-path algorithms, which produced similar, and in some cases better, running times compared to the aforementioned method. For instance, the computational complexity of the Fibonacci heap structure has a large hidden constant; therefore, a simple binary heap is often more efficient. The overall complexity is $O(N(|V|+|E|)\log|V|) \approx O(MN^2K^2\log(MNK))$. An interesting approach, which works surprisingly well, is to utilize Bellman-Ford's algorithm for finding the shortest paths. Even though the complexity of Bellman-Ford is $O(M^2N^2K^3)$, in practice it runs much faster for our flow network due to the "left-to-right" structure of the graph[4]. Furthermore, Bellman-Ford's algorithm works with negative costs as well, meaning that we do not have to maintain node potentials.

The space complexity of our method is dominated by the amount of storage required to store the $|E|$ edges of the flow network (around 20 bytes for each edge). Therefore, the worst-case space complexity of our MTT algorithm is $O(MNK^2)$.

5 Experimental Evaluation

In this section, we evaluate the performance of the proposed MTT algorithm, and compare it with a GNN implementation (using clustering) that is described in [6]. This approach works in low polynomial time with a complexity of $O(MNC^2)$ (where C is the average cluster size), and was shown to have the best performance among other MTT techniques in the detailed experimental evaluation of [3]. We do not include Reid's MHT algorithm [4] in this comparison, since it could not produce any results within a reasonable time limit. In the following plots, we use "*GNN*" to label the curves corresponding to the GNN approach, and "*MCMF*" to label our own algorithm.

We experimented on a real road map of the city of San Francisco [13], containing 174,956 nodes and 223,001 edges[5]. The original map was scaled to fit in a $[0, 10000]^2$ workspace. The trajectories are generated as follows: (1) We randomly select a starting node and a destination node (from the map) for each object. (2) Each object then travels on the shortest-path between the two points. At the first timestamp, the distance d_i covered by each object i is randomly selected between 0 and $R_{\max}$ (as defined in Section 4.3). At subsequent timestamps, the distance is adjusted randomly by $\pm 10\% \cdot R_{\max}$, while ensuring that it neither becomes negative nor exceeds $R_{\max}$. (3) Upon reaching the endpoint, a new random destination is selected and the same process is repeated.

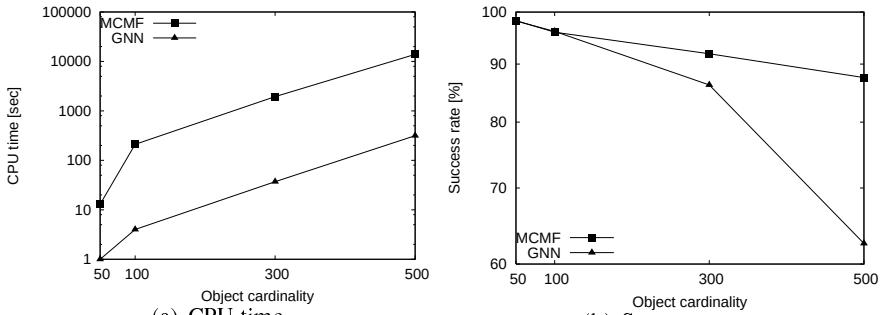
In each experiment we generate N random trajectories that are sampled for a period of M timestamps. We then run the corresponding MTT algorithms (without the object

[4] Actually, we also enhanced the functionality of Bellman-Ford's algorithm with a processing queue (for vertices), which reduces the $O(|V| \cdot |E|)$ complexity.

[5] This "network" corresponded to the map topology on which the objects move, and it has nothing to do with the "flow network" of our algorithm.

Parameter	Range
Number of objects (N)	50, **100**, 300, 500
Number of timestamps (M)	500, **1000**, 1500, 2000
Object speed ($R_{\max}$)	20, **40**, 80, 160

Table 1. System parameters



(a) CPU time (b) Success rate

Fig. 8. Performance vs. object cardinality

ids) and collect the resulting trajectories. These trajectories are compared to the original ones, where we measure the *success rate*, i.e., the percentage of successive triplets (as shown in Figure 4) that are associated with the correct trajectory. We use the CPU time and the success rate as the performance metrics. Table 1 summarizes the parameters under investigation, along with their ranges. Their default values are typeset in boldface. In each experiment we vary a single parameter, while setting the remaining ones to their default values. The total number of measurements varies from 50,000 to 500,000.

Figure 8(a) shows the running time of the two methods as a function of the object cardinality. As expected, MCMF is slower than GNN, but it improves considerably over Reid's algorithm, which is exponential to the size of the input and fails to terminate even in the simplest of cases. We expect that by employing "divide-and-conquer" techniques (e.g., by forming clusters that are solved independently of each other, similar to the methods used in [5, 6]) our algorithm will scale to much larger datasets.

The main advantage of our approach over single scan methods is depicted in Figure 8(b). This plot shows the accuracy of the trajectory reconstruction process, in terms of the percentage of correct associations. Even though GNN achieves lower running time, its accuracy deteriorates rapidly with increasing number of objects. This is due to the fact that more objects exhibit crossing trajectories, which confuses GNN (as shown in Figure 2). Therefore, the results of GNN may be of little value in practice. On the other hand, MCMF is very accurate and maintains a success rate of over 87%.

Figure 9 shows the CPU time for GNN and MCMF, as a function of the history length M. GNN scales linearly with M, while the slope of the curve for MCMF exhibits some variations. This behavior can be explained by the approximation that is discussed in the last paragraph of Section 4.4. When negative cost cycles are detected, the size

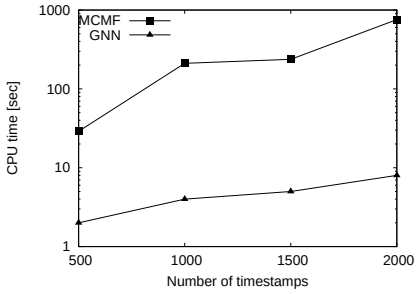


Fig. 9. CPU time vs. number of timestamps

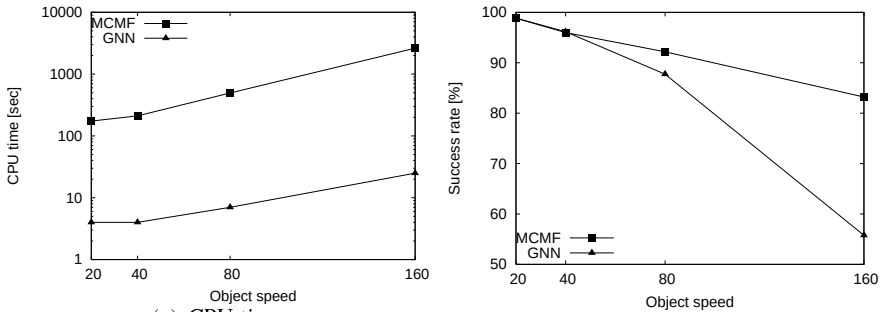
of the graph is reduced and subsequent iterations are executed faster. Consequently, the running time of our algorithm is also affected by the appearance of negative cost cycles. Note that the complexity analysis in Section 4.5 corresponds to the worst-case, i.e., when negative cost cycles never form. Regarding accuracy, both algorithms are unaffected by the number of timestamps.

Next, we investigate the effect of the object speed on the CPU time. As shown in Figure 10(a), both algorithms become slower as the speed increases. For GNN, this is due to the fact that clustering is less effective when the objects move faster. MCMF is also affected by the object speed, since the average number of feasible associations K for each measurement increases. Finally, Figure 10(b) depicts accuracy as a function of the speed of the moving objects. As the speed of an object increases, the successive locations of its trajectory move further apart from each other. Therefore, within a snapshot there may be many measurements that are closer to the object's previous location than the correct one. The greedy nature of GNN is not able to deal with that and, for high speeds, only 55% of the associations are correct. MCMF, on the other hand, is clearly superior; its success rate is always over 83%.

6 Conclusions

In this paper, we investigate the feasibility of applying multiple target tracking techniques in the context of anonymized trajectory databases. Existing methods are either very slow (i.e., the complexity is exponential to the number of measurements), or very inaccurate. The main contribution of our work lies in the novel transformation of the MTT problem into an instance of the min-cost max-flow problem. This transformation allows for a polynomial time solution in $O(MN^2K(\log(MNK) + K))$, where M is the number of timestamps, N is the number of measurements in each timestamp, and K is the average number of feasible associations for each measurement. Our initial results indicate that the proposed method produces very accurate results.

In the future, we plan to extend our work in a number of directions. First, we will investigate the feasibility of our method in complex scenarios where (1) new tracks may be initiated at random timestamps, and (2) location measurements may be lost due to errors on the wireless channel. Second, we will combine our methods with clustering, in order to further reduce the computational and space complexity. Specifically, through



(a) CPU time (b) Success rate

Fig. 10. Performance vs. object speed

clustering, we will partition the tracking problem into a number of smaller sub-problems that can be solved more efficiently.

References

1. Bar-Shalom, Y., Fortmann, T.E.: Tracking and Data Association. Academic Press (1988)
2. Blackman, S.S.: Multiple-Target Tracking with Radar Applications. Artech House (1986)
3. Leung, H., Hu, Z., Blanchette, M.: Evaluation of multiple radar target trackers in stressful environments. IEEE Trans. on Aerospace and Electronic Systems **35**(2) (1999) 663–674
4. Reid, D.B.: An algorithm for tracking multiple targets. IEEE Trans. on Automatic Control **24**(6) (1979) 843–854
5. Chummun, M., Kirubarajan, T., Pattipati, K., Bar-Shalom, Y.: Fast data association using multidimensional assignment with clustering. IEEE Trans. on Aerospace and Electronic Systems **37**(3) (2001) 898–913
6. Konstantinova, P., Nikolov, M., Semerdjiev, T.: A study of clustering applied to multiple target tracking algorithm. In: Proc. International Conference on Computer Systems and Technologies (CompSysTech). (2004) 1–6
7. Kubica, J., Moore, A.W., Connolly, A., Jedicke, R.: A multiple tree algorithm for the efficient association of asteroid observations. In: Proc. ACM International Conference on Knowledge Discovery and Data Mining (KDD). (2005) 138–146
8. Hoh, B., Gruteser, M.: Protecting location privacy through path confusion. In: IEEE International Conference on Security and Privacy in Communication Networks (SecureComm). (2005) 194–205
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 2nd edn. The MIT Press (2001)
10. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall (1993)
11. Bellman, R.: On a routing problem. Quarterly of Applied Mathematics **16**(1) (1958) 87–90
12. Dijkstra, E.: A note on two problems in connection with graphs. Numerische Mathematik **1** (1959) 269–271
13. Brinkhoff, T.: A framework for generating network-based moving objects. GeoInformatica **6**(2) (2002) 153–180