Collaboration with Spiridon Bakiras (Dept. of Mathematics and Computer Science, John Jay College, CUNY) and Panos Kalnis (Dept. of Computer Science, National University of Singapore).

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 a 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. To this end, we introduce a novel method that transforms the tracking problem into a min-cost max-flow problem. We then utilize standard techniques that work in polynomial time with respect to the number of objects. Our 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, thus, scale well to a large object population.

1 Introduction

Recent advances in wireless communications and positioning devices have generated an enormous 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 spatio-temporal applications. This type of applications 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 underlying road network.

Nevertheless, the availability of such data at a centralized location raises some 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[1]. In the traffic monitoring system, for instance, 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 anonymized location information.

However, detailed trajectory data (i.e., coupled with object identifiers) can be very valuable in numerous situations. For example, a law enforcement agency trying to track a suspect that was seen in an automobile at a specific time instant, 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) [BSF88, Bla86] 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 a combinatorial explosion, since each new measurement may potentially be associated with any of the existing tracks. The reason for that, 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 standard techniques that work in polynomial time with respect to the number of objects. Our experimental results indicate that the proposed methods produce high quality results that are comparable with the state-of-

[1]Notice that assigning a random *id* to each object is not sufficient, since a user may be linked to a specific trajectory using some external information.

the-art MTT algorithms. In addition, our methods reduce significantly the computational cost and, thus, scale well to a large object population.

The remainder of the paper is organized as follows. Section 2 presents a formal definition of our problem, while Section 3 surveys the related work in this area. A detailed description and analysis of our algorithm is given in Section 4. In Section 5 we evaluate experimentally our methods, while in Section 6 we summarize our results and present some 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. We do not consider synchronized measurements, i.e., the time difference between consecutive timestamps $(t_{i+1} - t_i)$ is not constant. Each snapshot contains measurements from 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 relatively simple 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. Each of these points will be linked to a different trajectory, and thus 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 in a way, 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 $\{(x_1, y_1, t_1), (x_4, y_4, t_2), (x_7, y_7, t_3)\}$.

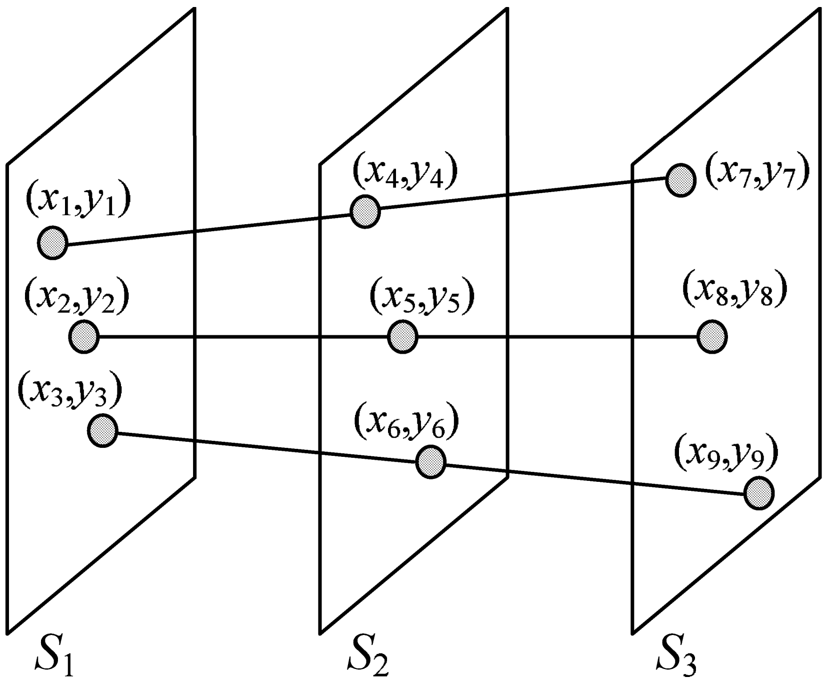


Figure 1: Example multiple target tracking

3 Related Work

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

NN techniques [Bla86] work in 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 function of the difference between the measured and predicted values. Among existing NN algorithms, the global nearest neighbor

(GNN) approach is the best solution. JPDA algorithms [BSF88] also work in 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 Ref. [LHB99]. 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 very good, leading to many false associations.

Reid's algorithm [Rei79] is probably the most representative method using multiple hypotheses. 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). While this process continues, unfeasible hypotheses are eliminated and similar ones are combined. Reid's algorithm produces high-quality results, but its computational and space complexity grows exponentially with the number of measurements.

To further reduce the complexity of the tracking process, clustering has also been considered as a viable solution. Ref. [CKPBS01, KNS04] 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.

Another interesting application of multiple target tracking is investigated in Ref. [KMCJ05], 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 all sets of observations that conform to that motion model.

Multiple target tracking has also been studied in the context of sensor networks. Ref. [OSS05] proposes a method where tracking is performed hierarchically, by forming groups around supernodes. Observations are first fused locally, and then transmitted to the corresponding supernode. In addition, tracks from different supernodes are combined. In Ref. [VGW05], only one sensor node is focused on each target at any time (leader node). However, leader nodes also take into account the possible existence of other targets. Ref. [SMK$^+$07] considers binary proximity sensors, i.e., sensors that produce a single bit of output ('1' when one or more targets are in the sensing range and '0' otherwise). The authors show how to count the number of distinct targets, given a snapshot of the sensor readings.

Finally, the idea of applying MTT techniques for the reconstruction object trajectories from anonymized data, was first introduced in Ref. [HG05]. In this work, the authors use five real GPS 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 propose an algorithm that modifies the actual dataset, so that, when at least two users are in close vicinity, their paths are crossed.

4 Tracking Algorithm

In this section, we discuss the details of our MTT algorithm. We begin in Section 4.1 with a brief overview of the min-cost max-flow problem in graph theory. In Section 4.2 we describe how to construct the graph from the history of location measurements, while in Section 4.3 we introduce the implementation of the algorithm. Finally, in Section 4.4 we analyze the computational complexity of our method.

4.1 Preliminaries

A *flow network* [CLRS01] 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 $u \in V - \{s, t\}$, 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 for every $u, v \in V$ the edge (u, v) has a cost $w(u, v)$, and we require $w(u, v) = -w(v, u)$.

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 2. Flow units are produced at the source s and consumed at the sink t. All the edges have a capacity equal to 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 a cost value $w(u, v)$ in the forward direction, and a value $-w(u, v)$ in the reverse direction. The rest of the edges have zero cost.

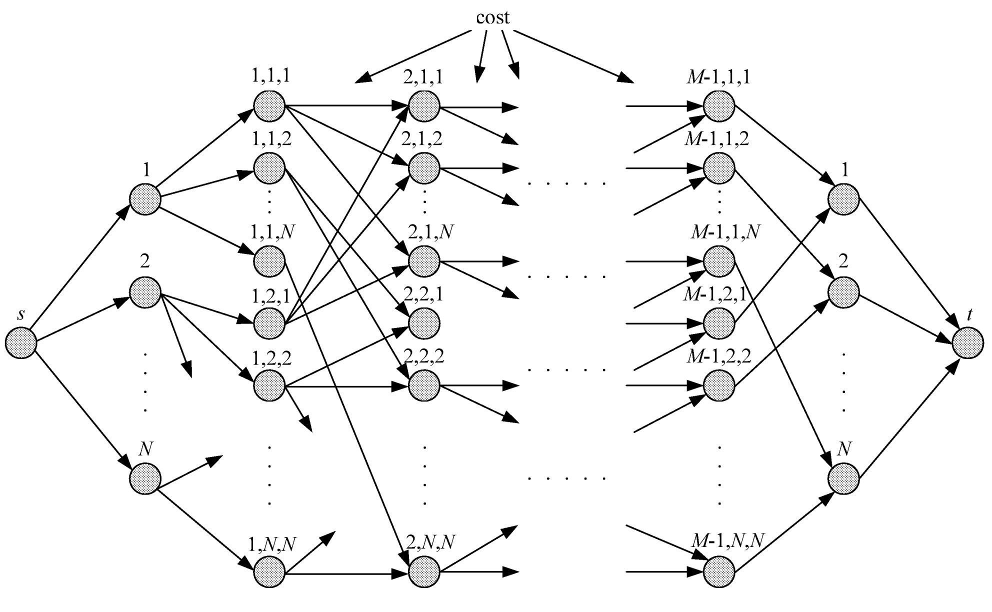


Figure 2: MTT flow network

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 a series of columns, containing N^2 nodes each. Notice that every node in these columns is identified by a triple (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 $(t_i, p_i, p_j) \to (t_{i+1}, p_j, p_k)$ represents a partial trajectory from three consecutive time-stamps $(p_i \to p_j \to p_k)$, where $p_i, p_j, p_k \in [1..N]$.

The cost value for the aforementioned edge is equal to the association error for the third measurement. As shown in Figure 3, 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 (the term $\frac{t_{i+1}}{t_i}$ is necessary, because we do not assume synchronized measurements). 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 Ref. [Rei79]. Notice 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.

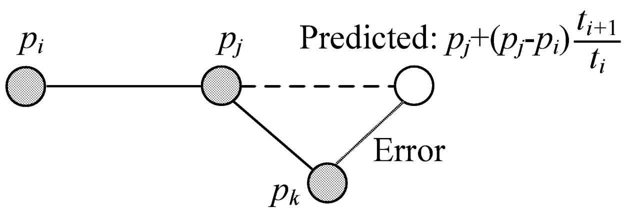


Figure 3: Association error

Finally, the N nodes connected to the flow sink (t) correspond to the last set of measurements, and indicate the final positions of the moving objets. To summarize, the total number of vertices in the flow network is $|V| = O(MN^2)$, while the total number of edges is $|E| = O(MN^3)$.

It is easy to notice, though, that the above flow network may lead to incorrect trajectories, by associating a single point with multiple tracks. For instance, if in the final solution we allow a positive amount of flow through edges $(1, 1, 1) \to (2, 1, 1)$ and $(1, 2, 1) \to (2, 1, 2)$ (Figure 2), then location 1 in timestamp 2 belongs to two different trajectories. In order to overcome this limitation, we 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*, and it is shown in Figure 4.

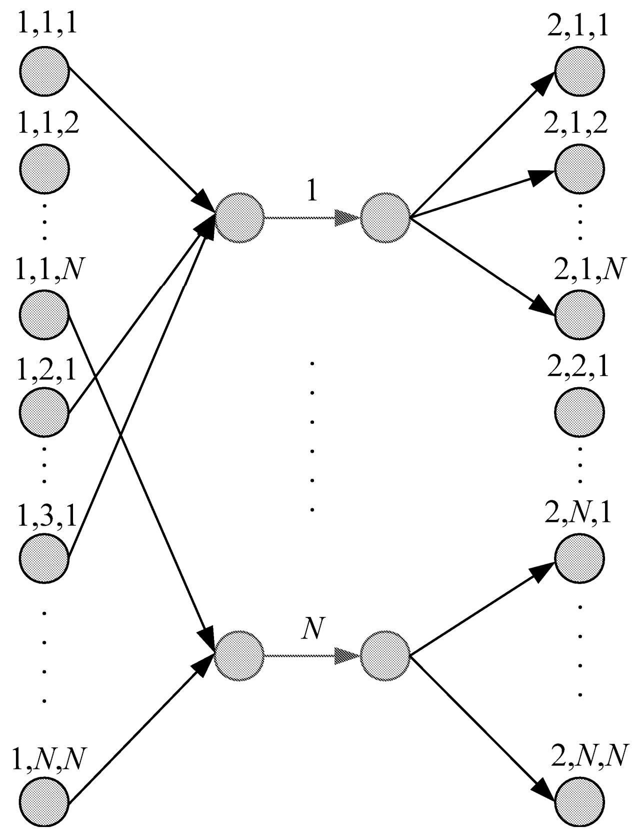


Figure 4: Flow network with N blocks

Although this approach solves the multiple assignment problem, it creates a new one, since each block consists of only $2N + 1$ edges; however, in the original network (Figure 2) there are N^2 edges with cost values associated with them. Therefore, we modify the block structure (as depicted in Figure 4), and replace each block with $2N$ vertices and N^2 edges. The result is shown in Figure 5. The edges connecting the two middle columns have the same cost values, as explained in the context of Figure 2. The remaining edges have a cost equal to zero, i.e., they do not affect the process of the min-cost max-flow calculation.

Another difference in the modified block structure compared to the rest of the flow network, is that we allow edges to run in the reverse direction

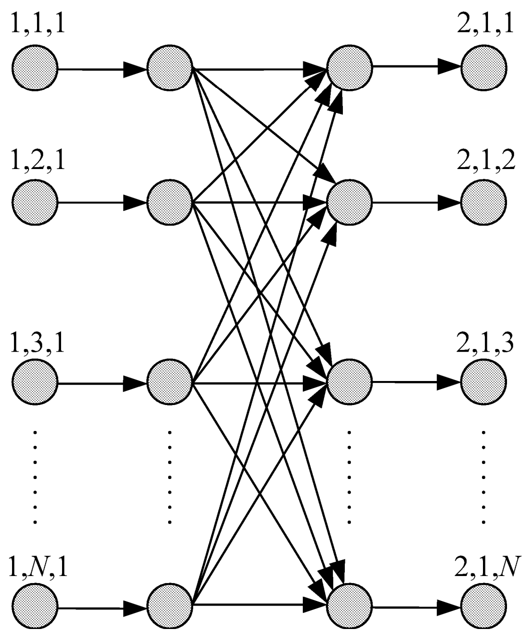
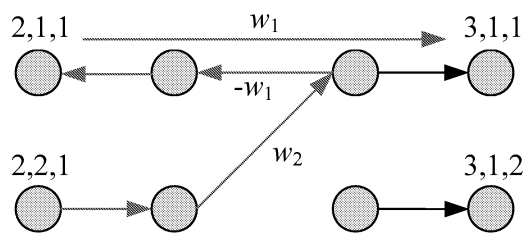


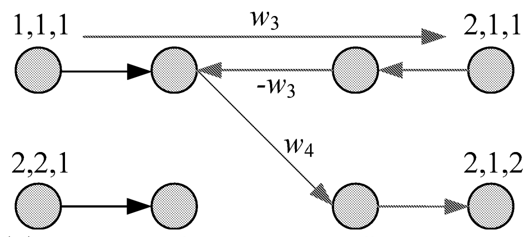
Figure 5: Modified block structure

(with a negative cost, as explained in Section 4.1). 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 unit, so any additional incoming flows have to be redirected backward.

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 w_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 $(2, 1, 1)$. Next, as shown in Figure 6(b), the incoming flow enters the block of the previous timestamp, where it is again directed to vertex $(2, 1, 2)$. Consequently, it is forced to select measurement 2 at timestamp 3 (instead of measurement 1), which results in two distinct trajectories.



(a) Redirection of a flow out of an active block



(b) The above flow selects a different location

Figure 6: Functionality of new block structure

4.3 The MTT Algorithm

We have a single source s that needs to send a total of N units of flow towards the destination t. Among all the 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 [AMO93]. It leverages the Ford-Fulkerson algorithm [CLRS01] 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 it is equal to $c(u, v) - f(u, v)$.

In the Successive Shortest Path Algorithm, instead of finding any augmenting path p, we find the path with the minimum cost (given the cost values of the edges on the residual graph). Since our flow network contains costs with negative values (inside the blocks of Figure 6), we need to utilize the Bellman-Ford algorithm [Bel58] for the shortest path calculations. This is not very efficient, though, as the Bellman-Ford algorithm has a complexity of $O(|V| \cdot |E|)$. In our network, this translates to $O(M^2N^5)$.

Instead, we use a well-known technique called vertex potentials, which can transform the network into one with non-negative costs (provided that there are no negative cycles). For every edge $(u, v) \in E$, where vertex u has a potential $p(u)$ and vertex v a potential $p(v)$, 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-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.

A coarse pseudo-code of our MTT algorithm is shown in Figure 7. It begins by constructing the detailed flow network (as explained in Section 4.2) 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 flow network, and the algorithm terminates after exactly N iterations. The resulting trajectories are returned by following each unit flow from s to t through the flow network.

4.4 Complexity

The computational complexity of the MTT algorithm shown in Figure 7, is directly related to the complexity of the underlying shortest-path algorithm. Theoretically, the fastest running time is achieved with Dijkstra's algorithm [Dij59], using a Fibonacci heap implementation for the priority queue. The complexity of Dijkstra's algorithm is $O(|V| \log |V| + |E|) = O(MN^2 \log(MN^2) + MN^3) \approx O(MN^3)$. Thus, the total running time (due to N iterations) is $O(MN^4)$.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

[2]If there are negative costs, Bellman-Ford must be run initially to remove them. In our case, however, we do not have negative costs before the first iteration, since none of the blocks are active.

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

Figure 7: The MTT algorithm

complexity of the Fibonacci heap structure has a large hidden constant, and a simple binary heap is often more efficient. The overall complexity of this method is $O(N(|V|+|E|)\log|V|) \approx O(MN^4\log(MN^2))$. An interesting approach, which works surprisingly well, is to utilize Bellman-Ford's algorithm for finding the shortest paths. Even though Bellman-Ford runs in $O(M^2N^5)$ for our flow network, in practice it works much faster, due to the "left-to-right" structure of the graph[3]. Furthermore, Bellman-Ford's algorithm works with negative costs as well, meaning that we do not have to maintain node potentials.

A final remark, regarding the running time complexity of the algorithm, concerns the complexity of accessing the edges of the flow network. In particular, the network presented in Section 4.2 is very strict and can be constructed automatically. Furthermore, the location of every node on the net-

[3]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.

work can be calculated in constant $O(1)$ time, using simple formulas. Also, we can identify the start and end node of every edge in the network in $O(1)$ time, given the unique number of this edge.

To conclude, the space complexity of our method is equal to amount of storage required to store the $|E|$ edges of the flow network (around 20 bytes for each edge). Therefore, the space complexity of our MTT algorithm is $O(MN^3)$.

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 Ref. [KNS04]. This approach works in low polynomial time with a complexity of $O(MN^2)$, and was shown to have the best performance among other MTT techniques in the detailed experimental evaluation of Ref. [LHB99]. Notice that we do not include Reid's MHT algorithm [Rei79] 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.

For generating the object trajectories, we randomly select a starting position and a destination for each object, which is uniformly distributed in a $[-10000, 10000]^2$ workspace. The object then follows a nonlinear trajectory (with some velocity) between the two points. Upon reaching the endpoint, a new destination is selected (that diverges slightly from the current trajectory) and the same process is repeated (with a different velocity).

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 *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 measurements 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.

Figure 8 shows the running time of the two methods as a function of the object cardinality. As expected, MCMF is much slower than GNN, but it

Parameter	**Range**
Number of objects (N)	10,30,**50**,100
Number of timestamps (M)	10,**50**,100,150
Object speed	slow,**medium**,fast,faster

Table 1: System parameters

improves considerably over the exhaustive MHT technique (which fails to terminate even in the simplest of cases). We expect that by employing some "divide-and-conquer" techniques (e.g., by forming clusters that may be solved independently of each other) we will be able to scale the proposed method to much larger datasets (similar to the methods used in Ref. [CKPBS01, KNS04]). We will explore this research direction as part of our future work.

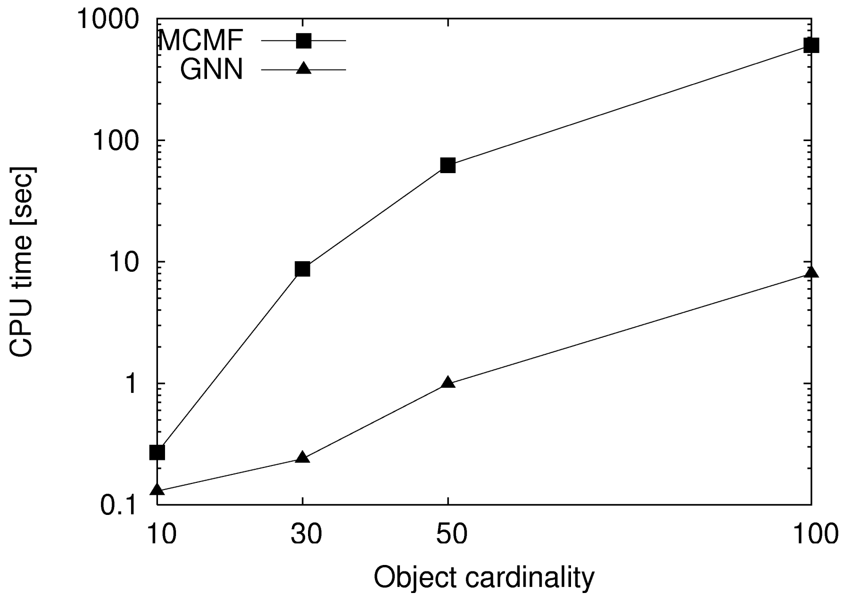


Figure 8: CPU time vs. object cardinality

The main advantage of our approach over single scan methods is depicted in Figure 9. This plot shows the accuracy of the trajectory reconstruction process, in terms of the percentage of correct associations. Even though GNN achieves a 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 the greedy approach utilized by

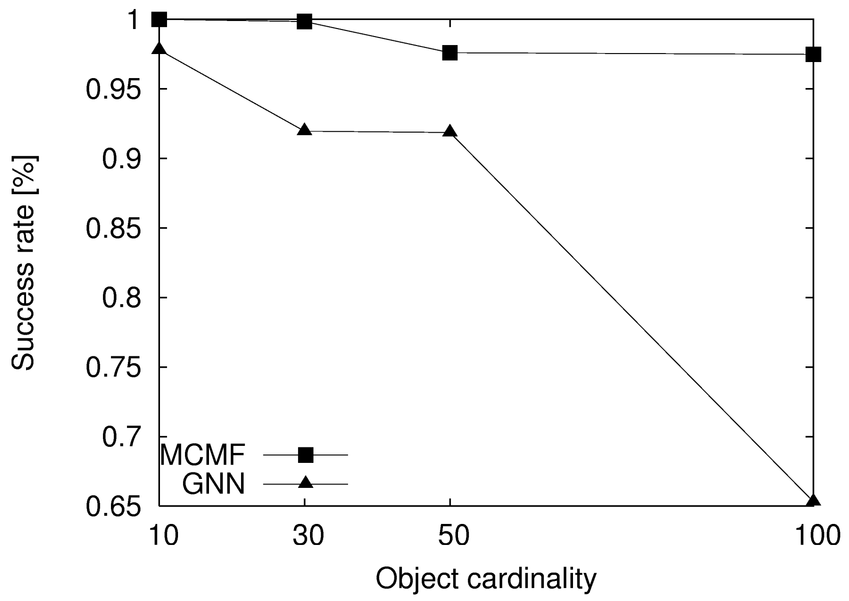


Figure 9: Success rate vs. object cardinality

GNN. Therefore, the results of GNN may be of little value in practice. On the other hand, MCMF is very accurate and maintains a constant success rate of over 98%.

A nice example that illustrates the ineffectiveness of GNN when dealing with crossing trajectories, is presented in Figure 10. In this example, the 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, both Reid’s algorithm and MCMF track the two objects successfully, since they minimize the error across all timestamps. The slight differences in the output between Reid’s algorithm and ours, is due to the filters that are used to smooth the trajectories (Kalman filter for Reid, as opposed to a simpler filter for MCMF).

Next, we investigate the impact of the history length on the running time of the two techniques. Figure 11 shows the CPU time for GNN and MCMF, as a function of M. Clearly, both algorithms scale linearly with M, which verifies the complexity analysis of Section 4.4.

Figure 12 illustrates the effect of M on the accuracy of GNN and MCMF.

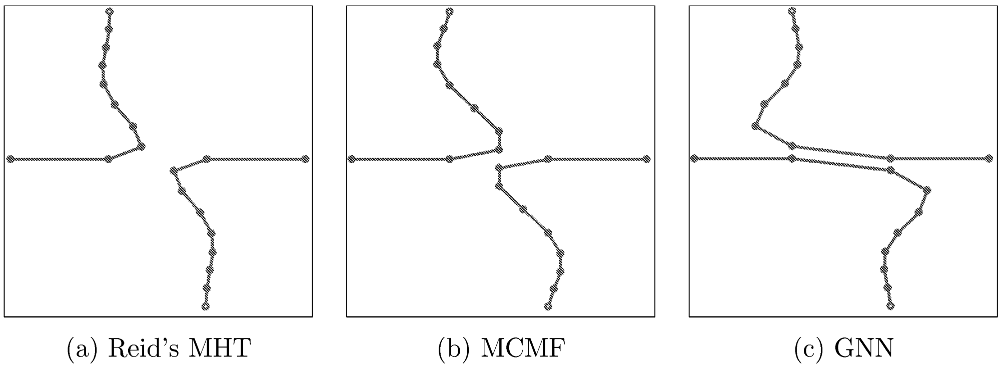

(a) Reid's MHT (b) MCMF (c) GNN

Figure 10: Trajectory reconstruction for different methods

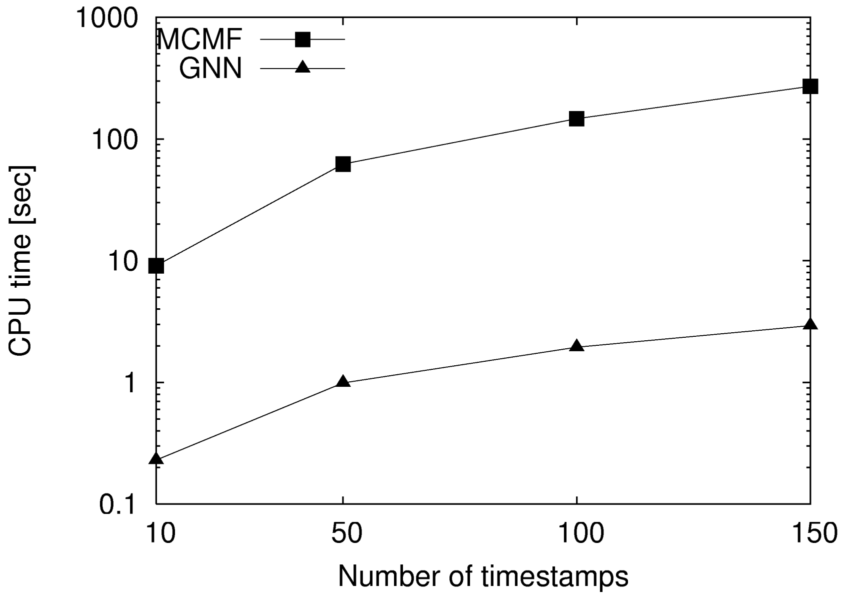


Figure 11: CPU time vs. number of timestamps

Both algorithms remain unaffected by the increasing history length, and maintain constantly a high level of success rate. MCMF is again better in terms of accuracy, which is almost 100% in all cases.

Finally, Figure 13 depicts the tracking performance of the two methods, as a function of the speed of the moving objects. Here, the superiority of MCMF is clearly illustrated. As the speed of an object increases, the successive

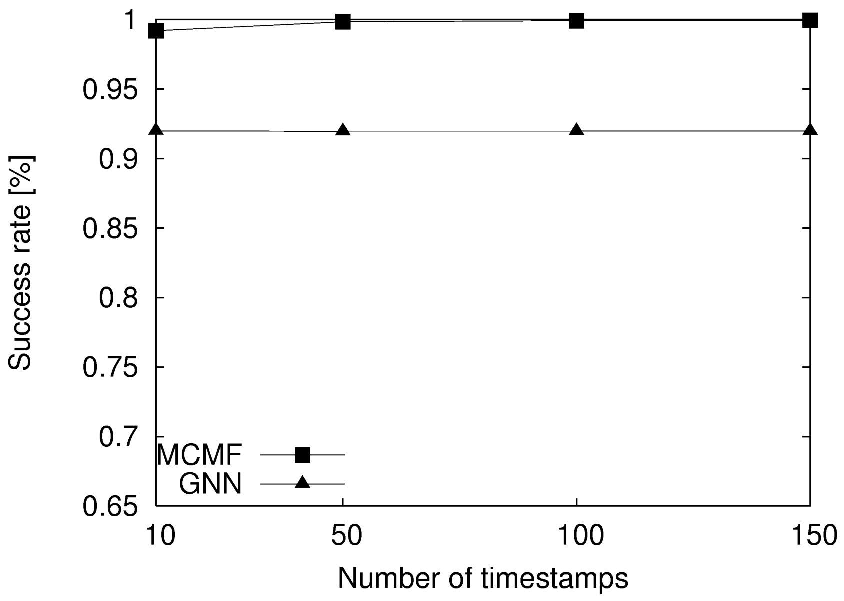


Figure 12: Success rate vs. number of timestamps

locations of its trajectory move further apart from each other. As a result, 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, 90% of the associations are incorrect. MCMF, on the other hand, is practically unaffected by the moving speed, and its success rate is close to 100%.

6 Conclusions

In this paper, we investigate the feasibility of applying multiple target tracking techniques in the context of anonymized trajectory databases. The complexity of existing methods that are based on exhaustive search, grows exponentially with the number of measurements and, thus, can not be applied to large databases. Although low polynomial algorithms exist that work well in practice, the quality of their results deteriorates significantly when the number of distinct trajectories is large. 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 $O(MN^4)$, where M is the number of timestamps and N is

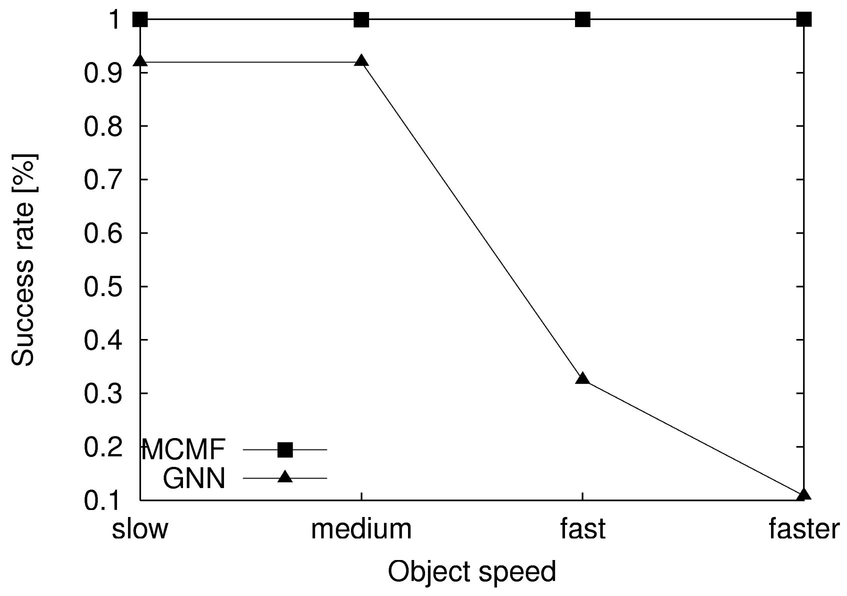


Figure 13: Success rate vs. object speed

the number of observations in each timestamp. Our initial results indicate that the proposed method produces very high-quality results, and is able to associate over 98% of the measurements with the correct trajectories in all cases.

In the future, we plan to extend our work in a number of directions. First, we will investigate the feasibility of our methods in a more realistic environment. In particular, we will consider a scenario 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 clustering, we will partition the tracking problem into a number of smaller sub-problems that can be solved more efficiently.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, 1993.

[2] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association.* Academic Press, 1988.

[3] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[4] S. S. Blackman. *Multiple-Target Tracking with Radar Applications.* Artech House, 1986.

[5] M. Chummun, T. Kirubarajan, K. Pattipati, and Y. Bar-Shalom. Fast data association using multidimensional assignment with clustering. *IEEE Trans. on Aerospace and Electronic Systems*, 37(3):898–913, 2001.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2nd edition, 2001.

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

[8] B. Hoh and M. Gruteser. Protecting location privacy through path confusion. In *IEEE International Conference on Security and Privacy in Communication Networks (SecureComm)*, pages 194–205, 2005.

[9] P. Konstantinova, M. Nikolov, and T. Semerdjiev. A study of clustering applied to multiple target tracking algorithm. In *Proc. International Conference on Computer Systems and Technologies (CompSysTech)*, pages 1–6, 2004.

[10] J. Kubica, A. W. Moore, A. Connolly, and R. Jedicke. A multiple tree algorithm for the efficient association of asteroid observations. In *Proc. ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 138–146, 2005.

[11] H. Leung, Z. Hu, and M. Blanchette. Evaluation of multiple radar target trackers in stressful environments. *IEEE Trans. on Aerospace and Electronic Systems*, 35(2):663–674, 1999.

[12] S. Oh, S. Sastry, and L. Schenato. A hierarchical multiple-target tracking algorithm for sensor networks. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 2197– 2202, 2005.

[13] D. B. Reid. An algorithm for tracking multiple targets. *IEEE Trans. on Automatic Control*, 24(6):843–854, 1979.

[14] J. Singh, U. Madhow, R. Kumar, S. Suri, and R. Cagley. Tracking multiple targets using binary proximity sensors. In *Proc. ACM International Conference on Information Processing in Sensor Networks (IPSN)*, pages 529–538, 2007.

[15] T. Vercauteren, D. Guo, and X. Wang. Joint multiple target tracking and classification in collaborative sensor networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 23(4):714–723, 2005.