

**MULTI-TARGET TRACKING AND IDENTIFICATION BY A VECTOR OF SENSORS**

Sahin Cem Geyik and Boleslaw K. Szymanski

Center for Pervasive Computing and Networking & Department of Computer Science  
Rensselaer Polytechnic Institute, Troy, NY

**Abstract**—Diverse security applications often require monitoring of a narrow passage, such as an indoor corridor, a tunnel, a bridge; either to protect critical assets at the end of such a passage or to control the passage over it, or both. Often, sensors are arranged in a vector along such a passage and are capable of registering the crossing of a target but not its identity. In this paper, we consider coordinated tracking by such a vector of sensors where each sensor learns timings of objects passing at monitored spots from its predecessors. Hence, the problem we are solving could be formulated as an assignment or matching of target identities to arrival times of the targets at the subsequent sensors in a vector.

We introduce a cost function that is the sum of squares of the difference between each target's predicted and observed arrival time at each sensor. We use target's speed as computed from the arrival times of the target at previous two sensors to predict the time of arrival at the current sensor. In such a scheme, the simple matching algorithm that sorts predicted and actual arrival times in two separated lists and then matches both lists by positions minimizes this and similar cost functions for each sensor locally in  $O(n \log n)$  time. We have also developed more costly algorithms that yield higher quality global solution at the cost of communication, computation and memory.

In the paper, we evaluate analytically, and by simulations, different variants of this matching algorithm and their complexity and performance.

**I. INTRODUCTION**

A Wireless Sensor Network (WSN) consists of (i) a set of sensor nodes that measure the environment that the network monitors, (ii) one or more base nodes at which the measurements are collected, processed and interpreted, and (iii) relay nodes (that may be identical to the sensor nodes) which carry the information from the sensor nodes to the base node(s) [1]. Research on wireless sensor networks is often multidisciplinary as it involves many computer science and engineering areas, including networking, distributed computing, graph theory, machine learning, signal processing etc. Typical uses

of WSN's include various military and civilian applications, such as target tracking, environmental monitoring, product quality monitoring etc.

Target tracking has been a frequent focus of research on wireless sensor networks. Yet, many of the publications in this area deals with tracking of a single target in a monitored area [2]. Even if multiple target tracking is considered, it is often limited either in the number of targets that are allowed [3,4] or in trajectories that the targets can follow.

In this paper, we consider tracking of a constant flow of targets in which the speeds of the targets are determined from previous sensor measurements and are constantly updated using the most current sensor measurements. To assure broad applicability of our approach, we assume the simplest and most fundamental form of sensing [5] in which sensors are able to detect only the presence (but not the distance or bearing) of a single target within their sensing range [6]. We focus on a specific sensor deployment in a straight line vector of sensors. Such deployment is typical in many diverse security applications that require monitoring of a narrow passage, such as an indoor corridor, a tunnel, a bridge, either to protect critical assets at the end of such a passage or to control the passage over it, or both. In such applications, the physical constraints often limit traffic to a single path. For simplicity, in this paper, we assume unidirectional movements of targets along such a path, however, it is a matter of a straightforward extension to our scheme to make it cope with bidirectional traffic, as long as the targets do not change the direction of their movements.

**II. PROBLEM DEFINITION**

The system that we analyze consists of  $n$  sensors aligned along a straight path, with sensor  $s_1$  at the path's entry and sensor  $s_n$  at the path exit. The path is traversed by a set of targets. Each target enters the path at sensor  $s_1$  and leaves it at sensor  $s_n$ . Each passage through a sensing range of sensor  $s_i$  results in a measurement  $(i, t_{i,k})$  where  $i$  is the sensor identifier and  $t_{i,k}$  denotes

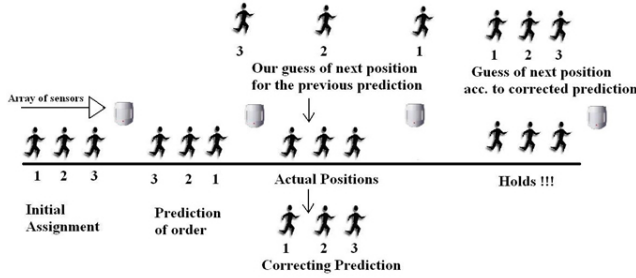


Fig. 1. Problem Definition

the time of the  $k$ -th target detection at sensor  $s_i$ . The overall diagram of the system is shown in Fig. 1.

At sensor  $s_1$ , starting with 1, we simply assign subsequent natural numbers as unique target identifiers to each measurement made by this sensor. We assume that such an assignment is possible by the ability of a sensor to distinguish entries of multiple targets at the same time. However, for the subsequent sensors, we need to assign the already issued target identifiers to each measurement made trying to correctly predict which target caused that measurement. Two cases need to be considered:

- 1) *online* target assignment, in which we need to assign target identifier to each measurement as it is being made using of course only partial information available, and
- 2) *offline* target assignment, in which the assignments are done after all measurements are made or accounted for.

Since the off-line assignment has more information available, it can make better assignments, yet, we found that quite good on-line assignments can be made much faster using the partial information available.

More formally, having  $m$  target identifiers issued at sensor  $s_1$  and subsequent  $(n-1)m$  measurements made at the remaining sensors, an assignment is a set of  $n-1$  one-to-one mapping  $a_i : \{1, 2, \dots, m\} \rightarrow \{1, 2, \dots, m\}$  for  $i = 2, 3, \dots, n$  such that  $a_i(j)$  denotes rank, under chronological order, of the arrival time of target  $j$  at sensor  $s_i$ . Hence, an assignment uniquely pairs each measurement  $t_{i,k}$  at sensor  $s_i$  with one target identifier, so if target  $j$  caused measurement  $t_{i,k}$ , then  $a_i(j) = k$ .

In an off-line assignment, all pairings are decided after time  $t_{n,m}$ , while in on-line assignment, pairing of measurement made at time  $t_{i,k}$  is made immediately at that time.

### III. PROPOSED SOLUTION

The basic idea of our solution is to predict the arrival time  $p_{i,k}$  of target  $k$  at the sensor  $s_i$ ,  $i > 2$  based on the speed with which this target passed from sensor  $s_{i-2}$  to sensor  $s_{i-1}$ . Having such a prediction, we want to assign actual measured target passing times at sensor  $s_i$  to targets in such a way that the sum of squares of the differences between the predicted and actual arrival times for all targets is minimized. More formally, we introduce the cost function and cast the problem as the following minimization problem:

$$\min \sum_{i=3}^n \sum_{j=1}^m (t_{i,a_i(j)} - p_{i,j})^2 \quad (1)$$

where

$$p_{i,j} = \frac{d_i}{d_{i-1}} (t_{i-1,a_{i-1}(j)} - t_{i-2,a_{i-2}(j)}) \quad (2)$$

In the above equation,  $n$  is the number of sensors in the sensor vector, while  $m$  is the total number of targets that passed through the system.  $d_i$  denotes the distance between sensors  $s_i$  and  $s_{i-1}$ . Often, this distance is the same for all sensors, in which case Eq. 2 simplify, by yielding the prediction that the passage time at the current pair of sensors will be the same as it was at the previous pair.

In summary, we are minimizing the sum of squares of deviations of actual times of arrivals from the predicted times of arrivals for all targets and over all sensors, except the first two. The arrival time for a target is predicted using the previous speed of the target.

Another possible cost function would sum up just absolute values of deviations of actual arrival times from the predictions, yielding the following cost function:

$$\min \sum_{i=3}^n \sum_{j=1}^m |t_{i,a_i(j)} - p_{i,j}|. \quad (3)$$

We will show later that our sorted matching solution is locally optimal for this function as well. Depending on application, either of the two cost functions could be useful, with the first cost function penalizing large deviations, whereas the second, the overall sum of deviations.

The problem defined by Eq. 1 is equivalent to the minimum cost online bipartite matching in which one set consists of the arrival time predictions and the other set consists of the actual arrival times. Edge costs consist of the square of differences between the two connected nodes. In the next subsection we will explain the minimum cost bipartite matching problem while the

following subsection will discuss the greedy solution to the posed problem.

### A. MIN-COST BIPARTITE MATCHING PROBLEM

The Min-Cost Bipartite Matching problem on the fully connected bipartite graph  $G = \{V, E = V_1 \times (V - V_1)\}$ , with  $|V_1| = |V|/2 = v$ , and cost function  $c : E \rightarrow R^+$  can be defined as follows:

$$\min \sum_{i=1}^v \sum_{j=1}^v c(\langle i, j \rangle) * m(\langle i, j \rangle) \quad (4)$$

where

$$m(\langle i, j \rangle) = \begin{cases} 1 & \text{if } \langle i, j \rangle \text{ is in the matching,} \\ 0 & \text{otherwise.} \end{cases}$$

and

$$(\forall 1 \leq i \leq v) \sum_{j=1}^v m(\langle i, j \rangle) = \sum_{j=1}^v m(\langle j, i \rangle) = 1.$$

The latter condition means that in a valid matching, each vertex belongs to exactly one edge in the matching. This general version of the min-cost bipartite matching problem can be solved in  $O(|V|^3)$  time using a modified version of Kuhn-Munkres algorithm [7,8].

Online version of min-cost bipartite matching problem asserts that one of the bipartite sets has incoming elements one-by-one in time and each of these should be matched at their time of arrival. Any online algorithm is evaluated using the competitive ratio which is the ratio of the worst case result of the given algorithm to the optimal solution. Optimal case is the one where we know in advance all the input and make decisions considering overall minimum (the general case of offline min-cost bipartite matching).

Min-cost bipartite matching in general case has a lower bound of  $2k-1$  on the competitive ratio. An online greedy algorithm may yield much worse competitive ratio of  $\Omega(2^k)$  [9]. Such a ratio arises if very costly vertices arrive in an unordered fashion when the cost function is actually a function of vertex values. In the next section we will show that, due to the nature of the input in a sensor system, a greedy algorithm achieves a local optimum for our problem.

### B. LOCAL OPTIMALITY

In this section we prove the local optimality of a greedy algorithm for the online bipartite matching problem defined by our assignment problem. Ordering the arrival time predictions and making a one-to-one sorted matching gives us the local optimal cost value, simply

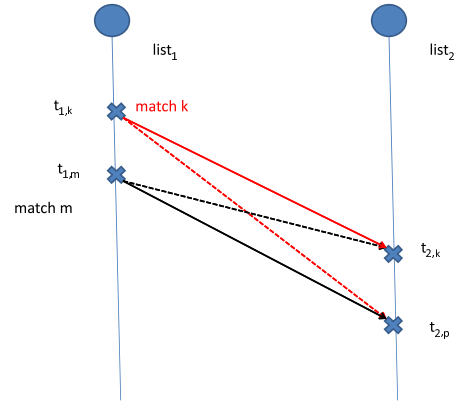


Fig. 2. Swapping Pairs of Elements Matched out of Order

because the second set of inputs, the actual arrival times, are already arriving in the chronological (sorted) order.

**Theorem 1.** *Given two lists of  $n$  numbers, sorting these two lists and matching their elements one by one gives the minimum sum of square of differences of the matched elements as well as the minimum sum of absolute values of their differences.*

*Proof:* We will proceed by contradiction. Let  $t_{l,r}$  denote element of list  $l$  with rank  $r$  in the sorted order of the elements of this list.

Suppose that we have the optimal matching of  $n$  elements on  $list_1$  and  $list_2$  in which at least one matched pair of elements  $\{t_{1,m}, t_{2,p}\}$  is out of order, meaning that  $m \neq p$ . Let  $t_{1,k}$  be the first element being part of such a pair, so  $t_{1,k}$  is the first element not matched with the equally ranked element  $t_{2,k}$  on the second list. Then, it must be matched with an element  $t_{2,p}$  for some  $p > k+1$ . Indeed, by definition  $p \neq k$  and all the second list elements with rank smaller than  $k$  are matched with the equally ranked elements on the first list, so also  $p > k$ . Hence, we have  $t_{2,k} < t_{2,p}$ . Let  $t_{1,m}$  denotes the element of the first list that is matched with element  $t_{2,k}$ . Again it must be, as shown in Fig. 2, that  $m > k$  because all elements on the first list with rank smaller than  $k$  are matched with the equally ranked elements on the second list. Hence,  $t_{1,m} > t_{1,k}$ . Let's consider now a matching in which element  $t_{1,k}$  is matched with element  $t_{2,k}$  and  $t_{1,m}$  is matched with  $t_{2,p}$  while all the other matchings of elements are left unchanged. The difference between the cost of the new matching and the original one for

| Order                                | Difference                 |
|--------------------------------------|----------------------------|
| $t_{2,k}, t_{2,p}, t_{1,k}, t_{1,m}$ | 0                          |
| $t_{2,k}, t_{1,k}, t_{2,p}, t_{1,m}$ | $2(t_{1,k} - t_{2,p}) < 0$ |
| $t_{2,k}, t_{1,k}, t_{1,m}, t_{2,p}$ | $2(t_{1,k} - t_{1,m}) < 0$ |
| $t_{1,k}, t_{2,k}, t_{2,p}, t_{1,m}$ | $2(t_{2,k} - t_{2,p}) < 0$ |
| $t_{1,k}, t_{2,k}, t_{1,m}, t_{2,p}$ | $2(t_{2,k} - t_{1,m}) < 0$ |
| $t_{1,k}, t_{1,m}, t_{2,k}, t_{2,p}$ | 0                          |

TABLE I  
ENUMERATION OF SIGN OF VALUES DEFINED BY EQ. 6

the cost function defined by Eq. 1 is:

$$(t_{i,a_i(j)} - p_{i,j})^2 = 2(t_{2,k} - t_{2,p})(t_{1,m} - t_{1,k}) \quad (5)$$

Since we showed that  $t_{2,k} < t_{2,p}$  and  $t_{1,m} > t_{1,k}$ , the difference is negative, so the new matching has lower cost than the original matching has, contradicting the assumption that the original matching was optimal.

When cost is defined by Eq. 3, the difference is

$$|t_{2,k} - t_{1,k}| + |t_{2,p} - t_{1,m}| - |t_{2,k} - t_{1,m}| - |t_{2,p} - t_{1,k}| \quad (6)$$

The proof by enumeration of all possible cases of relation between  $t_{1,k}, t_{2,k}, t_{1,m}, t_{2,p}$  that this difference is non-negative is shown in Table I.

There is a subtle difference between these two cases. For the first cost function, the minimum happens only for the sorted matching, while for the second cost function, the sorted matching is one of possibly several matchings that minimize this function.

In conclusion, whenever an arbitrary matching of two  $n$  elements lists is not sorted, by swapping two elements that are matched out of order, we achieve no higher cost. Thus, after all such swaps are made, we will end up with two sorted lists and the sorted matching without raising the cost of matching. ■

The disadvantage of this scheme is that by achieving a local minimum, it affects the predicted arrival times at the subsequent sensors and therefore it may not be able to achieve the global optimum. Fig. 3 shows an example of such a case. The sorted matching assigns the same speeds to both targets, and predicts their arrivals at sensor  $s_3$  at times that agree with the actual arrival times. For sensor  $s_4$ , the cost of the matching becomes  $(x-7)^2$ . Changing matching to accelerate target 1 between sensors  $s_2$  and  $s_3$  yields better prediction of its arrival at sensor  $s_4$ , yielding the cost function  $(x-8)^2 + 3$  which for  $x > 9$  (the case of  $x = 10$  is shown in Fig. 3) is smaller than  $(x-7)^2$ . Hence, in this case, the unsorted matching is less costly than the sorted one. Yet, this happens only when the total cost is high and therefore the relative difference between the two solutions is then small. Indeed, for any value of

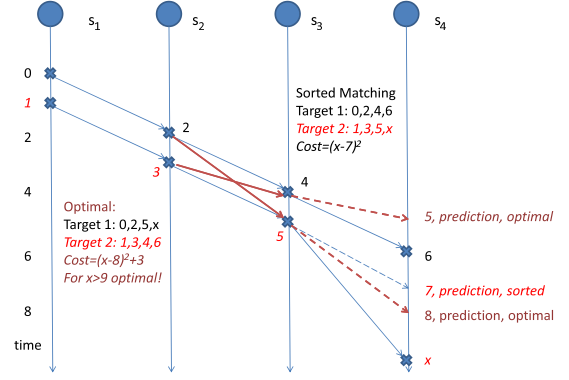


Fig. 3. Counter Example

$x$ , the ratio of the difference between the two solutions and the minimum of cost function is  $\frac{2(x-9)}{(x-8)^2+3} \approx \frac{1}{x-8}$ , so if we denote minimum value of cost function as  $c$  then the ratio is  $O(\frac{1}{\sqrt{c}})$ . In short, even if the sorted match is not optimal, the relative difference between its value and optimal one is in the order of the inverse of the square root of the optimal value.

### C. ALGORITHM

While it is straightforward to implement the sorted matching to match the actual arrival times with the predicted arrival times, the first interval creates a problem because we do not know the speeds of the targets. There are multiple possibilities and these possibilities are held in a data structure called *prediction*. This structure consists of lists of targets, their history and the predictions of their next location together with their predicted arrival times. Fig. 4 presents a centralized version of the algorithm. This implementation can be easily parallelized to make all the computations local to the sensor node that holds the relevant data.

The number of predictions can easily grow exponentially with the number of targets, as the new targets are created when they are detected by the first sensor. To limit the growth of the number of predictions, we eliminate predictions at the later sensors according to their costs. Our current heuristic rule deletes those predictions which have  $1 + \epsilon$  times higher cost than the current minimum cost among all predictions. While different values for  $\epsilon$  may be chosen, our experience indicates that it is best to set it as a function of the total number of targets that entered the system so far.

```

agent{
int next_stop; // The next sensor on this agent's journey
double previous_speed; // Speed of this agent for the previous interval
double previous_time; // Time of passage for the last sensor
double time_to_arrive; // Predicted time of arrival for next_stop
}
prediction{
agent list[MAX]; // Agent list for the prediction
double total_cost; // Total cost of prediction so far
} // Functions as a state table

list prediction_list; // List of prediction table that are kept in the system

For a signal at sensor k at time t :
if k = 0
  Arrival;
  Insert a new coming agent for each prediction;
endif
if k = 1
  for each prediction
    Multiply prediction for each agent with next stop of 2;
    // The reason for this is we have no speeds calculated
  endfor
endif
if k > 1
  for each prediction p
    agent a = agent m with minimum predicted arrival time to sensor k;
    p.total_cost += ( t - a.time_to_arrive )2;
    a.next_stop = k+1;
    a.time_to_arrive = t + ( distancek-1,k / (distancek,k-1 / ( t - a.previous_time ) ) );
    a.previous_time = t;
  endfor
endif
min_pred = with the minimum total_cost;
output matching of min_pred;
minimum_cost = min_pred.total_cost;
if(k > v) // v is a sensor which we use for cutting down number of predictions
  for each prediction p
    if p.total_cost > minimum_cost * (1 + f(minimum_cost))
      remove p;
    endif
  endfor
endif
endif

```

Fig. 4. Tracking Algorithm

#### D. OFFLINE VERSION

In this section, we examine the offline version of the problem. In this case, we have  $mn$  actual arrival times at  $n$  sensors where  $m$  is the number of targets that have crossed through the system. We will give a transformation of two different approaches to this problem to the minimum  $n$ -clique problem in a complete  $n$ -partite graph, where each partition holds  $m$  nodes [10]. We will also show how to assign a cost to the edges of this graph so that whichever clique is chosen, the cost of this clique will be equal to the cost of matching the corresponding arrival times for the single target. For notational simplicity, we assume that sensors are placed equi-distance from each other.

1) **CONSTANT SPEED MODEL:** The constant speed model can be summarized as follows. Given the start and finish times of a target at a vector of  $n$  sensors, match the middle points to the line that is constructed by connecting arrival times at sensors  $s_1$  and  $s_n$ . The question is if there are  $m$  non-conflicting (with distinct beginning and end points) lines with non-conflicting middle point matches that make the total minimized.

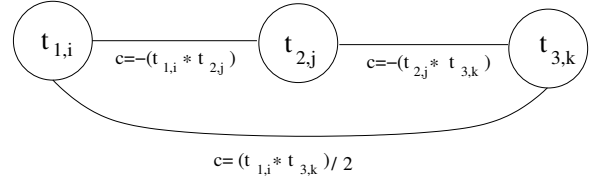


Fig. 5. Transformation for Constant Speed Model

Fig. 5 shows how to transform this problem to a graph setting where  $n=3$  (single sensor in the middle). The cost of choosing arrival time  $t_{2,j}$  with the beginning time  $t_{1,i}$  and the end time  $t_{3,k}$  is  $[t_{2,j} - \frac{t_{1,i} + t_{3,k}}{2}]^2$ . In this notation, the arrival time  $t_{p,q}$  is indexed by the sensor number  $p$  and the arrival (chronological) order  $q$  (so it is a rank of this arrival time in the ordered set of all arrival times at sensor  $p$ ).

As shown in Fig. 5, any 3 clique (triangle) in the graph has the cost  $[t_{2,j} - \frac{t_{1,i} + t_{3,k}}{2}]^2$  in this graph (without the independent parts of the equation which are related to only a single time-stamp, like  $t_{1,i}^2$ ). So the problem reduces to finding  $m$  disjoint 3-cliques. In  $n > 3$  case, the beginning and end nodes have edges with non-negative cost leading to each node in the middle. Costs between the middle nodes are set to zero (these edges are only added to have a *complete* graph).

2) **REALISTIC MODEL:** The realistic model is described by Eq. 1 and Eq. 2. The target speed crossing the distance between previous two sensors is calculated to approximate its arrival time at the third sensor. Fig. 6 shows the transformation made to the graph setting. In this case, a vertex has a nonzero edge to vertices associated with the two previous sensors. All other vertices have edges connecting them to this vertex with zero cost, just to have the *complete* graph.

As seen in Fig. 6, a vertex has an edge to a vertex associated with the previous sensor labeled with the cost equal to the sum of the two values associated with joined vertices. One comes from predicting the next arrival time by the previous sensor and another with its own prediction computed from the arrival times of the target at the previous two sensors. Cost of choosing  $t_{3,k}$  predicted by using  $t_{1,i}$  and  $t_{2,j}$  is  $(t_{3,k} - 2t_{2,j} + t_{1,i})^2$ . Choosing  $m$   $n$ -cliques have the total cost given in Eq. 1 (where the independent values, like  $t_{1,i}^2$ , are removed).

While our aim is to minimize the total cost of the cliques chosen, for convenience we will examine the case when we just take the negative of the edge costs and find set of cliques with the maximum cost. If we only had the edges between times associated with the

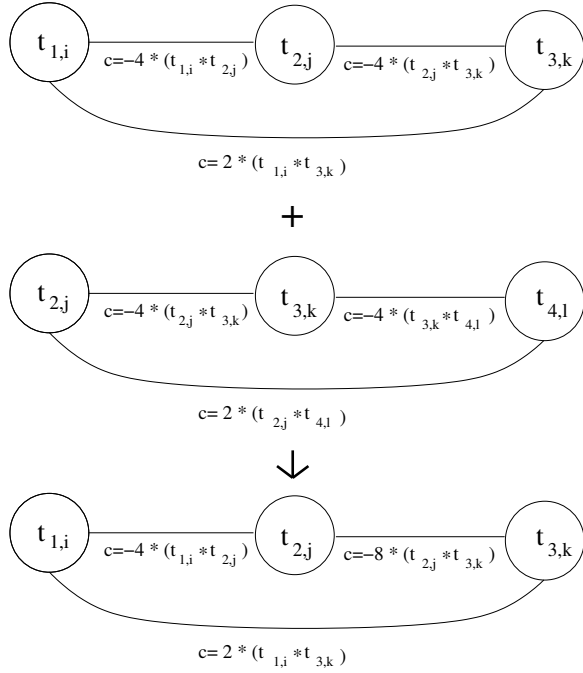


Fig. 6. Transformation for Realistic Model

neighboring sensors, then selecting the cliques in the sorted order would have given us the highest cost. Yet, since each arrival time has an edge with non-zero cost connecting it to the arrival time at the previous two sensors, this scheme does not always give us the optimal solution. However, we will show that such a case occurs in extreme conditions which is not really consistent with motion of the targets that we are tracking.

Suppose we have 7 sensors and we have already made a sorted matching, assigning the first target to the lowest arrival times at each sensor and the second target with the highest arrival times, as shown in Fig. 7. Suppose that we want to swap  $d$  with  $d'$  in terms of target assignment. For this to give us advantage (increase the cost), the following inequality must hold:

$$(d' - d)[4(c - c') + 4(e - e') - b + b' - f + f'] > 0$$

$$(b' - b) + (f' - f) > 4[(c' - c) + (e' - e)]$$

Hence, for the sorted matching to become non-optimal, there has to be a very large change in terms of speed. For the application of concern, target tracking, this is unlikely. It is also apparent that a 2-lookup is necessary to decide if a swapping of time-stamp assignment is beneficial. The two mentioned above reasons strengthen our conclusion that sorted matching gives optimal or near

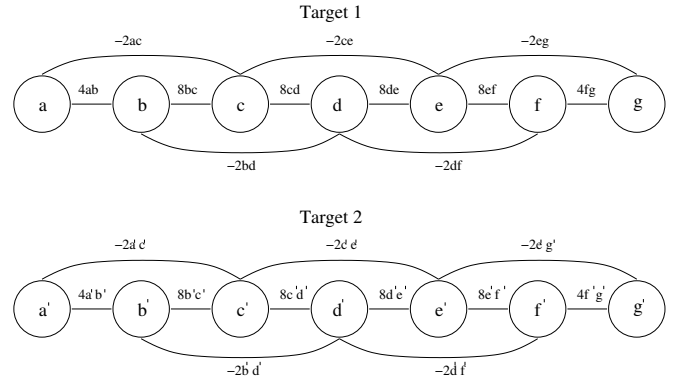


Fig. 7. Time-stamp Assignment to Targets

optimal results in both offline and online versions of the target tracking problem.

#### IV. SIMULATION RESULTS

Fig. 8 and Fig. 9 show the simulation results of the tracking algorithm discussed in section III.C.

We simulated  $n = \{21, 26, 41, 51\}$  sensors monitoring a tunnel of 200 meters (thus, yielding 10m, 8m, 5m and 4m constant distances between sensors, depending on value of  $n$ ). Each target begins its passage at sensor  $s_0$  and ends it at sensor  $s_{n-1}$ . Beginning speed of each target is uniformly distributed between 0.5 and 4.0 m/s and inter-arrival times of new targets are exponentially distributed with the mean value of 10.0 seconds. We have set running times to 1000 seconds and we took the average of 200 runs for each case. Change of speed in the graphs denote the absolute value of the range of change of the speed between to consecutive sensors in the vector. Hence, the actual change could vary from decrease or increase of the speed by the increment up to this absolute value.

As seen in Fig. 8, prediction accuracy (the ratio the number of correct matchings to the number of all matchings) is always above 55 % and reaches above 80 % when the speed changes range from -15% to 15%. It is important to notice that increasing the density of sensors in the vector lowers the distance between them and therefore decreasing the time deviation of expected and actual arrival times between two consecutive sensors. Hence, the quality of the prediction improves with the increased sensor density. As shown in Fig. 9, the cost grows with the square of the speed change, as expected. The total cost grows with the sensor density, however, the cost per sensor (prediction) decreases with it.

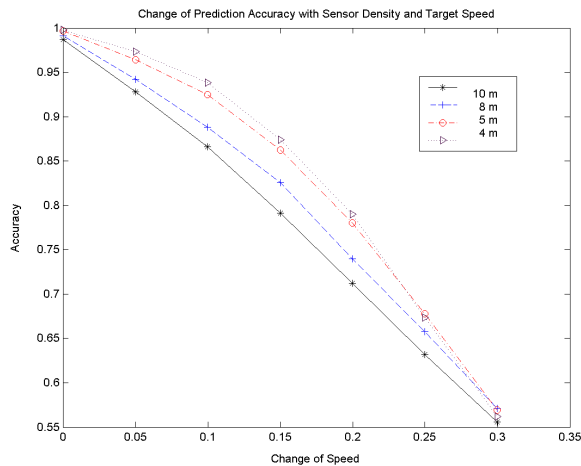


Fig. 8. Prediction Accuracy

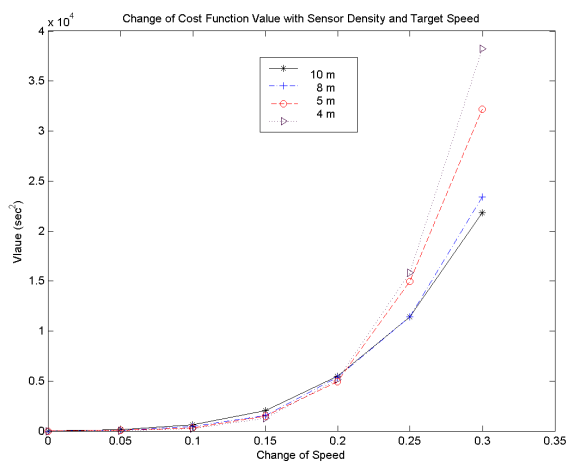


Fig. 9. Cost Function

## V. CONCLUSION

We have analyzed tracking of multiple targets along a one-dimensional path, a problem with many potential uses in both military and civilian applications. We have shown that sorted matching obtains local optimum at each sensor while achieving a near-optimal global solution. We have transformed the offline version of this problem to a min-cost k-clique problem in a k-partite graph and we showed that the sorted matching becomes sub-optimal only when large speed changes happen between two consecutive sensors. Since the distances between sensors are controlled at deployment and can be made small, this is unlikely to happen in realistic scenarios. This argument supports our conclusion that sorted matching is a robust and efficient, near-optimal solution to the presented problem. We have also imple-

mented a centralized version of the tracking algorithm and, using simulations, we have analyzed its performance in terms of prediction accuracy and the values of the cost function achieved for different ranges of change of target speeds as well as sensor density. An implementation of the version of this algorithm distributed to the sensors is an easy extension of this work and will be pursued by us in the future.

## REFERENCES

- [1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, *A Survey on Sensor Networks*, Computer Networks, 2002, pp. 393-422.
- [2] Chih-Chung Ke, Herrero, J.G. and Llinas, J., *Comparative Analysis of Alternative Ground Target Tracking Techniques*, Proceedings of the Third International Conference on Information Fusion, 2000, FUSION 2000.
- [3] Jercan, T., Dong Guo and Xiaodong Wang, *Joint Multiple Target Tracking and Classification in Collaborative Sensor Networks*, IEEE Journal on Selected Areas in Communications, pp. 714-723, April 2005.
- [4] Richard Brooks, David Friedlander, John Koch and Shashi Phooha, *Tracking multiple targets with self-organizing distributed ground sensors*, Journal of Parallel and Distributed Computing, Volume 64, Issue 7, pp. 874-884, July 2004.
- [5] Aslam, J., Butler, Z., Florin, C., Crespi, V., Cybenko, G. and Rus, D., *Tracking a Moving Object with a Binary Sensor Network*, Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, Los Angeles, California, USA, 2003.
- [6] Wang, Z., Bulut, E., and Szymanski, B., *A Distributed Cooperative Target Tracking with Binary Sensor Networks*, Proceedings of the IEEE International Conference on Communication (ICC'08), Cooperative Communications and Networking Theory, Practice, and Applications Workshop, Beijing, China, 2008.
- [7] Harold W. Kuhn, *The Hungarian Method for the assignment problem*, Naval Research Logistics Quarterly, 2:83-97, 1955.
- [8] J. Munkres, *Algorithms for the Assignment and Transportation Problems*, Journal of the Society of Industrial and Applied Mathematics, 5(1):32-38, 1957 March.
- [9] A. Meyerson, A. Nanavati, L. Poplawski, *Randomized online algorithms for minimum metric bipartite matching*, Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithm, Miami, Florida, pp: 954 - 959, 2006.
- [10] G. He, J. Liu and C. Zhao, *Approximation Algorithms for Some Graph Partitioning Problems*, Journal of Graph Algorithms and Applications, Vol. 4, no. 2, pp. 1-11, 2000.

This research was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.