Search for an Immobile Entity on a Network

Arun Jotshi and Rajan Batta
Department of Industrial & Systems Engineering and CMIF
University at Buffalo (SUNY)
438 Bell Hall, Buffalo, NY 14260.
U.S.A.

Submitted : February 2006, Revised : June 2006

Abstract – We consider the problem of searching for immobile friendly entities on an undirected network. The focus is on finding the first such entity. The search time is a random variable, whose probability density function (pdf) depends upon the path and also upon information about entity location. We seek a path choice that minimizes the expected search time. This problem differs from arc-covering problems, e.g. the Chinese Postman Problem (CPP), in the way that here the objective is not to find the minimum length tour that covers all the links at least once, but instead to minimize the expected time to find the entity. We introduce a heuristic algorithm to deal with the search process given that there is exactly one entity on the network and it is equally likely to be at any point on the network. We later prove that this path is also optimal for the situation when a number of entities are present and these entities are uniformly distributed.

Keywords: Search Problems, Chinese Postman Problem, Immobile Entities.

1 Introduction

We consider the problem of searching for an immobile friendly entity on a network. The network is assumed to be an undirected graph \( G(V,E) \), where \( V \) is the set of vertices (representing the road intersections) and \( E \) is the set of edges (representing the road links). An immobile entity cannot move, i.e., its position is stationary during the search process. An example of an immobile entity is an injured person trapped under debris of a collapsed building (rescue operation) or a mine placed by the enemy (military operation). By friendly we mean that the entity is not trying to evade the searcher. The entity is already in place, either by itself or planted by someone, but will not move. We assume in our analysis that the entity is of infinitesimal size, i.e., entity cannot be detected unless the searcher is on the exact location of the entity (zero discovery radius). We further assume that there is no prior agreement between the entity and the searcher about the search strategies they are going to use. Also there is no exchange of information or communication between the searcher and the entity. These assumptions are consistent with those used by other researchers in the field.

Actual search time consists of the travel time from the current location of the searcher to a chosen entry node on the network plus the search time to find the entity once the network has been entered. There are several objectives that might be appropriate for the search process. Let \( T_S \) (a random variable) be the time a searcher takes to find the entity (at that time both
will be at the same point). If there is no limit on the length of the search, then minimizing the expected search time ($E[T_S]$) is a reasonable objective function. With limited length of search (e.g. time till an entity will survive or constraints on the searcher itself), maximizing the probability of detection in a given time ($P[T_S \leq T]$) will be more appropriate. We focus on the minimization of the expected search time as our objective function.

We might have to cover the whole network, traversing every edge of the network at least once, since we have no a priori knowledge of exactly where the entities are but rather only have probabilistic knowledge of their whereabouts. This problem differs from the Chinese Postman Problem \(^1\)(CPP) in the way that here the objective is not to minimize total distance traveled but to minimize the expected time to find the entities (at which point of time we would not necessarily have traversed all of the links of the network). The entity we are considering is neither an evader nor does it cooperate in the search. We start by considering the case of a uniformly distributed single entity on the network. We later show that the search for the first entity is the same even when uniformly distributed multiple entities are present.

The problem surfaced while we were working on one of our projects — Dispatching and routing of emergency vehicles in a dynamic disaster environment using data fusion\(^2\) concepts \cite{1}. The precise location of a casualty is not known but instead we obtain from data-fusion, a region of interest and the entities are assumed to be within this region. A search delay was added to the actual response times for pick-up and delivery of casualties. The strategies developed in this paper help minimize this search delay. The number of entities is assumed to be given by a Space Poisson random variable, driven by the entity density and by the total length of links in the network (c.f. Larson and Odoni \cite{2}). Consider a homogeneous segment (link of the network) of length $l$ miles. By homogeneous we mean that the value of $\lambda$ is invariant throughout the region of interest. Let $\lambda$ be the average number of entities present per mile on this link. Then the number of entities that are present in the segment of length $l$ miles can be modeled as a Poisson random variable with mean $\lambda l$. Here the parameter $l$ (distance) plays a role directly analogous to $t$ (time) in a standard time Poisson process. For the Poisson model to be a reasonable one, the locations of entities must be consistent with Poisson type assumptions. Its implications are - number of entities found in non-overlapping intervals are mutually independent random variables (or equivalently, each entity is independent and uniformly distributed over the network) and as travel occurs, the probability of encountering an entity in the interval $[x, x + \delta x]$ is $\lambda \delta x$.

The structure of this paper is as follows. Section 2 is dedicated to literature review of the Search Theory area in general. We present some definitions, observations and a lemma in Section 3 that leads to our proposed approach and a heuristic algorithm for the case of a single entity discussed in Section 4. Section 5 suggests an alternate heuristic that does a partial enumeration of Eulerian tours. We present extensive results of the application of these two heuristics to randomly generated graphs in Section 6, while assuming that the search start node is not specified. In Section 7 we present a casestudy based on the Northridge area of Los Angeles and show our results assuming that the search start nodes are specified. Section 8 focuses on the case where there is more than one entity present. Section 9 is dedicated to conclusions and future research.

\(^1\)CPP tries to address the problem of finding a tour which traverses all the links of the network at least once and tries to minimize the number of repetitions of a link.

\(^2\)Data Fusion is a process dealing with the association, correlation, and combination of data and information from single and multiple sources to achieve refined position and identity estimates for observed entities, and to achieve complete and timely assessments of situations and threats, and their significance. Please see \cite{1} for more references.
2 Literature review

Search theory is considered to be one of the original disciplines within the Operations Research field. It emerged as a major part of Operations Research during World War II and developed as a separate field after the war. During the war, one important type of military operation was searching. Searches were conducted to locate the enemy, and to locate and recover one’s own lost or missing personnel or those of one’s allies. A typical search deals with the search process of finding an object called a target. Two scenarios have been considered in the literature depending on whether or not the target has its own motives. For the first scenario, the target can have two motives - target does not want to be found (Search Games resulting in Zero-Sum Games) or target wants to be found (Rendezvous Search Problem). The second scenario deals with a target that is not in a position to worry about when it is found. It is stationary and hidden according to a known distribution or is mobile and its motion is determined stochastically by known rules. Numerous applications of search theory are in the field of search and rescue operations, astronomy, industries (machine inspection and maintenance), mineral exploration etc. For a thorough understanding of the Search Theory subject, readers are referred to the books [3], [4], [5] and [6]. Depending upon the aim of the target, search problems can be classified into three categories as described in sections 2.1 through 2.3. Section 2.4 establishes the relationship between our problem and the Utilitarian Postman (UP) problem as suggested by Alpern in [7].

2.1 One sided search

In a typical one-sided search, the target has no motives of its own and the searcher is looking for the target. The target is simply stationary and hidden according to a known distribution or is mobile and its motion is determined stochastically by known rules. The classical problem of optimal search for a stationary target is mentioned in [3]. One of the earliest accounts of this type of problem is the Linear Search Problem (LSP). The LSP was first posed by Bellman [8]. Wallace [9] and Beck [10] worked on this problem, in which a searcher attempts to locate a randomly placed point \( x \) on the real line \( \mathbb{R} \) according to a known distribution function \( g(x) \). Searcher starts from a given point \( x_0 \) and follows a continuous path \( P \) at a constant speed, first in one direction (first step), then the other (second step) and so on until the searcher’s location first coincides with \( x \) at some time \( t \). The problem is to find the path that minimizes \( E[t] \).

Wallace considered the problem with the objective of minimizing average distance traveled before locating the object. He presented sufficient conditions on the distribution function \( g(x) \) for a solution to exist for this problem and these results were sharpened by Beck [10] for the attainment of the minimum (the same was presented independently by Wallace in [11]). It was not before 1970 when Beck and Newman [12] modified the LSP (introduced by Beck in [10]) by assuming that the probability distribution of the point sought by the searcher on a real line is not known to the searcher. As there is no a priori knowledge of the distribution, this problem essentially falls within game theory criteria with a minimax objective.

2.2 Search games

The second category consists of Search Games with a mobile or immobile hider who does not want to be found. Here the searcher and the entity have different objectives. Search Games are generally explained in terms of a two-person zero-sum game. Search games are studied under two main categories depending on whether the entity is immobile or mobile.

2.2.1 Immobile hider

In case of a search game involving an immobile hider, the searcher moves with maximal velocity (for optimality) and would like to capture the hider in minimum possible time. It was only
after 1970s, when Beck and Newman [12] modified the LSP by assuming that the probability distribution was not known to the searcher, that search games were studied in detail. Gal [13] presented a minimax solution for the game in which hider choses a real number and searcher seeks it by choosing a trajectory represented by a positive function. The game theory version of LSP was also studied by Fristedt and Heath [14] and Fristedt [15]. Gal [16] considered several extensions of the LSP, treating them from the point of view of game theory. Gal introduced the discrete version of a search game in [17] and a stochastic version in [18]. Gal [19] proved that the value of the search game with a given searcher starting point for an immobile hider on a network $G(V, E)$ is equal to half the total length of $G$ if and only if $G$ is Eulerian. 3. If the graph is not Eulerian some variant of CPP strategy has to be used (which is the aim of this paper). For any network $Q$, the value $v$ of the search game with an immobile hider, using such a strategy has to satisfy

$$\frac{\mu}{2} \leq v \leq \frac{\bar{\mu}}{2} \leq \mu,$$

where $\mu$ is the sum of all the links of the network and $\bar{\mu}$ is the length of the closed trajectory that visits all the points of $Q$ and has minimal length (as proved by Gal [19]). The lower bound is attained if and only if $Q$ is Eulerian and the upper value is attained when $Q$ is a tree. Also any minimal tour satisfies $\bar{\mu} \leq 2\mu$. The proofs are shown in [19] and [4]. Another important result proved by Gal was that for any network $Q$, the random Chinese postman tour4 finds any point $H$ in expected time not exceeding $\bar{\mu}/2$. Gal also presented his results of the search game on a tree [4]. Gal [20] showed that the random Chinese postman tour is an optimal search strategy if the graph is weakly Eulerian. Search games on a weakly cyclic graph were first solved by Reijnierse and Potters [21]. Kikuta [22] in his paper considered the search of an entity on a cyclic graph.

Stengel and Werchner [23] proved that the problem of finding the optimal search strategy for a general network is NP-hard if the length of the search path is arbitrary or if the searcher is free to revisit previously visited nodes. Gal conjectured in [5] that optimal strategies for searching of immobile hider on any network will never use trajectories that visit some (or part of) arcs more than twice. Solving the search game on a network with an arbitrary starting point for the searcher is an interesting problem which has not been investigated yet.

### 2.2.2 Mobile hider

The first presentation of a search game including a mobile hider was given in the Princess and Monster game described by Isaacs [24]. The monster $P$ (searcher) searches for princess $E$ (mobile hider), the time required being the payoff. The special case of this problem where the searcher and the evader move on the circumference of a circle was solved by Alpern [25] and Foreman [26]. Also a number of researchers have considered this problem of determining the number of searchers required to guarantee capture in a network (called search number). Megiddo et al. [27] proved that the problem of computing the search number is NP-hard for general graphs and solvable in linear time for trees.

Anderson and Aramendia [28] studied search problem on a network $Q$ with two players. They formulated the problem as an infinite-dimensional linear program and derived an algorithm for its solution. Work has also been done on search games on $k$ arcs, search on a circle, search on a figure eight shaped network, search in a maze, infiltration games and high low search. Readers are referred to [5] for a discussion on these types of search games. Even though search games on a network are difficult to solve, Alpern and Asic [29] provide a finite upper

---

3 A graph $G(V, E)$ is Eulerian (unicursal) when all its nodes have even degree.

4 A search strategy that encircles the Eulerian Tour equiprobably in each direction.
bound. Alpern and Gal [30] introduced the search problem when the motives of the entity are not known.

2.3 Rendezvous search

The third category is the *Rendezvous Search Problem*. Rendezvous search problems are characterized by the region of search, the player’s characteristics and prior agreements, if any, made about strategy co-ordination. It is a type of search game in which two players co-operate to minimize their meeting time. It has a min-min objective as compared to a min-max objective of a search game. The first discussion of a Rendezvous Problem appears in the book by Schelling [31]. He suggested that while searching in a region with features, one should always move to a focal point. In 1976 Alpern [32] introduced the Rendezvous Search Problem informally at a talk in Vienna. Two specific problems were suggested — Astronaut Problem and Telephone Problem (c.f. [33]). More rigorous description of this problem was first introduced by Anderson and Weber [34] on discrete location rendezvous and by Alpern on general continuous location rendezvous [35]. Most of the developments in this field are given in the article by Alpern [33].

Alpern [35] introduced the problems of rendezvous on the line. The symmetric version of this problem was introduced by Alpern [35]) and the asymmetric case was considered by Alpern and Gal [36]. The problem of rendezvous search on a graph is also not new. Alpern [35] presented elementary results regarding rendezvous on a network. Anderson and Weber [34] were the first to produce some results in rendezvous search theory on a graph. Alpern et al [37] considered the scenario of two agents placed randomly on nodes of a known graph. Alpern [38] considered the problem of rendezvous search on a labeled network called $H$-network (for definition please see [38]). Apart from the above, rendezvous search problems are classified under wide variety of other settings — rendezvous on an unlabeled, undirected or directed circle, rendezvous involving more than two players, rendezvous in higher dimensions and rendezvous evasion games.

2.4 Relationship with Utilitarian Postman problem

We note that our problem is essentially the same as the one that Alpern [7] independently introduced as the UP path problem, because such a path (in the postman setting) would minimize the average time a customer receives his mail. Such paths, for some special graphs and for all symmetric graphs, have been studied by Alpern, Baston and Gal ([39] and [40]) in the context of search games with immobile hiders. These papers deal with specific graphs and obtain precise results, while we are dealing with general graphs and present heuristics for finding the UP paths. The heuristics apply not only to friendly entities, but also in some cases to evaders. This paper also considers UP paths starting from a given node, which has not been considered before to our knowledge.

3 Preliminaries

Our main aim for this paper is to minimize the expected search time for finding the first entity given the fact that entities are uniformly distributed across the network. We divide our analysis into three cases, depending upon the available information about total number of entities ($m$) on the network. We solve for the best path under each of these separate circumstances. Without loss of generality, the searcher speed is assumed to be unity.

- **Case 1:** $m = 0$
  
  Expected time of showing there is no entity on the network.
Table 1 explains the notation and process of calculating expected search time.

**Definition**

Let $P$ be a path that covers all the edges of a given network $G(V, E)$ at least once. Let $E'(P)$ be the set of links on path $P$, with the $i^{th}$ link, $i(P)$, having a link length $l_{i(P)}$. The total number of links for this path $|E'(P)| \geq |E|$. This is because a searcher might have to traverse some links more than once to cover the whole network. Equality sign holds when the graph is Eulerian.

**Definition**

Let $i(P)$ be the $i^{th}$ link of path $P$. Let $Pr(i(P))$ be the probability of detection of the entity on link $i(P)$ when it is traversed. Further let $L_i(P)$ be the set of distinct links among the first $i - 1$ links of path $P$, with $L_1(P) = \phi$. Then, $Pr(i(P))$ is given by the expression:

$$Pr(i(P)) = \begin{cases} 0 & \text{if link } i(P) \in L_i(P), \\ \frac{l_{i(P)}}{L - \sum_{j \in L_i(P)} l_j} & \text{otherwise.} \end{cases} \quad (2)$$

Where, $L$ (sum of all the link lengths of the network) = $\sum_{i=1}^{E} l_i$. Note that whenever a link is traversed more than once, probability of detection on that link after the first traversal is zero since if we do not find the entity in first traversal, we will not find it in any subsequent traversals.

**Definition**

Expected search time given a path $P$ is given by the following expression:

$$E[T_S|P] = \sum_{i=1}^{|E'|} \left( \sum_{j=1}^{i-1} l_{j(P)} + \frac{l_{i(P)}}{2} \right) \times \prod_{j=1}^{i-1} \{1 - Pr(j(P))\} \times Pr(i(P)) \quad (3)$$

Consider the unit graph $G(V, E)$ shown in figure 1(a). Edge set $E = \{L_1, L_2, L_3, L_4, L_5, L_6\}$. Let path $P$ under consideration be $1 - 2 - 3 - 1 - 4 - 3 - 4 - 2$ with edge set $E' = \{L_1, L_2, L_3, L_3, L_4, L_5, L_6\}$. The process of calculating expected search time ($E[T_S]$) follows the Probability Detection Tree (as shown in figure 1(b)). Link length $l_{i(P)} = 1$ unit $\forall i(P) \in E'$.

Table 1 explains the notation and process of calculating expected search time.

---

5 Links and edges are used interchangeably throughout the paper.

6 Graph with all the link lengths equal to unity.
The solution methodology involves first checking if the graph is Eulerian or not. A non-Eulerian graph has a property that it has an even number of odd degree nodes (proved by Euler in 1736 [42]). Hierholzer in 1873 [43] addressed the problem of finding a Eulerian tour, which has been rediscovered by several other researchers since then. We briefly describe the algorithm presented by Edmonds and Johnson [41] for solving a CPP. To make a non-Eulerian graph Eulerian, solve a Minimum Weighted Perfect Matching in the graph, $G(V', E')$, defined over odd nodes. Edge set $E'$ contains edges (actual or derived) from each node $i$ to every other node $j$ ($i, j \in V'$) and these edges are the shortest distances between the node pairs. The matched edges in the matching $M$ are the edges that are added to the original graph to make it Eulerian. Then the algorithm to find a Eulerian tour is applied on this modified graph. A sketch of the algorithm is as follows:

- **STEP 1:** Starting from an arbitrary vertex $v$, trace a cycle by traversing the untraversed edges.
- **STEP 2:** If all edges have been traversed STOP. Else go to **STEP 3**.
- **STEP 3:** Trace another cycle starting from an untraversed edge incident to the cycle. Merge the two cycles using the suggested end pairing algorithm and go to **STEP 2**.

**Note** Given a Eulerian graph, its Eulerian tour is not unique. Hence, unlike the solution of a CPP where any Eulerian Tour will be acceptable as the optimal tour (since the objective is to minimize distance), it might not necessarily lead to an optimal minimum expected search time.
Note In order to search an entity there is no need to come back to the starting node (as in the case of a traditional CPP). Hence this is a case of an Open Chinese Postman Problem. An Open CPP does not have the same start and end nodes.

3.3 Observations and lemma

We now present some observations and a lemma that will help us develop our heuristic for solving our search problem.

Observation 1 The expected search time depends on the way we traverse the links of the Eulerian graph to complete the Eulerian tour.

Consider the graph shown in figure 2(a). We have two different Eulerian tours with different expected search times.

(a) Tour 1: 7-4-3-1-2-6-5-4-5-2-3-5-7-6-7.
Length of tour, \( L = 49 \) and expected search time, \( E[T_S] = 22.6333 \).
(b) Tour 2: 7-4-5-2-3-5-6-2-1-3-4-5-7-6-7.
Length of tour, \( L = 49 \) and expected search time, \( E[T_S] = 21.5952 \).

Observation 2 It is desirable to have repeated edges towards the end of the tour.

This intuitively makes sense because traversing traversing a link again cannot yield the discovery of an entity. Consider the unit graph \( G(V, E) \) as shown in figure 2(b). Graph \( G'(V', E') \)

(with shown link lengths) is the graph defined over the set of odd degree nodes (which in this case is \( \{1, 2, 3, 10\} \)). The optimal minimum weight matching for this graph is \( \{(1-2), (3-10)\} \). Since we are solving an Open CPP, we choose 3 (or 10) as the starting (or ending) node which results in edge \((1, 2)\) being traversed twice. We can find two different Eulerian tours with different expected search times:

a) Path 1: 10-7-6-3-4-1-2-5-4-7-5-8-9-10-6-4-2-1-3.
Length of path, \( L = 18 \) and expected search time, \( E[T_S] = 8.55882 \).

b) Path 2: 3-1-2-4-5-7-10-6-7-4-3-6-4-1-2-5-8-9-10.
Length of path, \( L = 18 \) and expected search time, \( E[T_S] = 8.73529 \).

Observation 3 For two paths with the same set of matched edges, the path that has lesser weight due to links that are in between the repeated links has a lower expected search time.
This follows from Observation 2. Consider the unit graph shown in figure 3. Minimum weight matching for this graph is \{(4, 5), (8, 7)\}. Consider the following two paths:

a) Path 1: 1 − 2 − 3 − 6 − 9 − 8 − 6 − 5 − 7 − 8 − 7 − 4 − 5 − 4 − 1 − 5 − 2 with $E[T_S] = 7.57143$.

b) Path 2: 1 − 5 − 6 − 8 − 7 − 5 − 4 − 1 − 2 − 3 − 6 − 9 − 8 − 7 − 4 − 5 − 2 with $E[T_S] = 7.21429$.

In path 1 we have two links between the repeated edges and in path 2 we have only one edge between the repeated edges and, hence, smaller $E[T_S]$.

**Lemma 1** The optimal minimum weight matching will not always lead to an optimal minimum search time.

**Proof** (By example)

Consider the unit graph shown in figure 4. The optimal minimum weight matching in graph $G'$ (defined over odd node set) is \{(8, 16), (2, 3), (4, 11), (19, 31), (22, 29), (33, 36), (38, 40)\} with weight=9. The total length of the Eulerian path in this case is 72 with an estimated search time of 32.992. Consider the sub-optimal matching \{(38, 40), (36, 33), (29, 22), (13, 11), (4, 3), (2, 8), (16, 24)\} with a weight of 11. The total length of the Eulerian path in this case is 74 with an estimated search time of 32.658 which is lesser than the one given above.

The difference in search times is due to the fact that with optimal minimum weight matching, the best possible Eulerian path achieved has more links in between two repeated edges and does not have repeated edges towards the end of the path. Forcing the repeated edges towards the end of the tour is made possible by using the sub-optimal matching. The methodology to find such a path is shown in our proposed heuristic. We note that the authors of [39] also point out the same result (as given by Lemma 1) while illustrating a distinction between a *Utilitarian Postman* (UP) path and a *Chinese Postman* (CP) path.
4 Heuristic

4.1 Proposed approach

Let $G(V, E)$ be the graph representing the region of interest where the entities are likely to be present. The searcher has to travel to some node of the network and then start the search process. The search process ends as soon as the searcher is at the same exact position as that of the entity. Even though the number of entry point choices is polynomial (equal to $|V|$), the possible number of tours (starting with a specific node) is not. Hence we limit our choices of possible entry nodes. Generally we will have a clear choice of the possible entry nodes, given by the boundary nodes of the region. Also the starting node should ideally be an odd degree node as required by the Open CPP. We do not consider a non-boundary node to start the search process. To reach such a node we have to first visit a boundary node (from the searcher’s current position), which means that the search process has already started. If all the boundary nodes are of even degree, we chose the closest one to the searcher and start the search. Hence, we pick a starting node $i \in n$ (where $n$ is the set of boundary nodes) and find the shortest path from the current location of the searcher to node $i$. Then we find a path that traverses all the links of the network at least once by applying the heuristic method developed below. We first present the heuristic method given that the start node is unspecified and then extend this heuristic for the case with a specific starting node. The approach is as follows:

- **STEP 1:**
  Check if the graph is Eulerian or not. If the graph is Eulerian, apply the Edmonds and Johnson’s Algorithm explained in [41] to find a Eulerian tour (optimal for this case).

- **STEP 2:**
  If the graph is non- Eulerian, apply the proposed heuristic to find a hamiltonian path over the odd nodes. The alternate edges of this path give the matching $M$ (not necessary optimal) to make the graph Eulerian. It also encourages the traversal of the repeated edges towards the end of the tour.

- **STEP 3:**
  Find Eulerian path $p^*$ by using results from the heuristic algorithm.

4.2 Heuristic when start node is not specified

**INPUT:** A non-Eulerian graph $G(V, E)$ with a set of odd nodes $V'$, where $V' \subseteq V$.

**OUTPUT:** A Matching $M$ (not necessarily optimal) that specifies the edges that are to be added to the original graph $G$ to make the graph Eulerian, and a path $T''$ which when added to the path found by merging the cycles of the modified graph (with edge set $\{E + M - T''\}$) gives a Eulerian tour with the repeated edges towards the end of the tour. The start node and the end node of the search process are also specified.

**PROCEDURE:**

- **STEP 0:**
  From graph $G(V, E)$ extract graph $G'(V', E')$ defined over the set of odd node $V' \in V$ as defined in section 3.2.

- **STEP 1:**
  Apply Minimum Spanning Tree ($MST$) algorithm over graph $G(V', E')$. Let the edges

---

7By boundary nodes we mean a small set of nodes that are closest to the current searcher location (on the outer perimeter of the network), preferably of odd degree.
in MST be denoted by the set \( E' \). If the MST is a Hamiltonian Path, i.e., degree of all vertices \( i \in V' \) is two except for two nodes \( x \) and \( y \) (with \( d(i) = 1 \)), STOP. Go to STEP 4. Otherwise, go to STEP 2.

**STEP 2:**
For all nodes \( i \) with degree \( d(i) > 2 \), delete an edge \((i, j)\) to decrease its degree and add another edge so that the increase in overall weight of MST is as small as possible. For any deleted edge \((i, j)\), the candidate edges that can be added to MST (replace the deleted edge in MST) are the ones that join two nodes with degree equal to 0 or 1 without forming any cycles. Out of all such candidate pairs, select the pair \((i, j)\) and \((k, l)\) as the deleted edge and added edge respectively, that results in the least increase in total MST weight. Ties are broken arbitrarily. This is done till \( d(v) = 2 \), \( \forall v \in V' \). Go To STEP 3.

**STEP 3:**
Let \( i = (p, q) \) be the first link if we traverse Path \( T' \) in the forward direction and let it be called Path A. Let \( j = (r, s) \) be the first link if it is traversed in the backward direction and let it be called Path B. Hamiltonian path \( T' \) will always have an odd number of edges (since it is obtained from a MST on an even number of nodes). Let \( l_i \) be the weight of link \( i \). If \( l_i = l_j \), delete either link \( i \) or link \( j \). Otherwise, let us define Forward Weight as the weight of all the links except for the last one in Path A, the alternate edges starting with link \( i \) being counted twice. Let Backward Weight be the weight of all the links except for the last one in Path B, the alternate edges starting with link \( j \) being counted twice. Hence,

Forward Weight = \( 2[l_i + (l_{i+2}) + (l_{i+4}) + ...] + [(l_{i+1}) + (l_{i+3}) + ...] \),

Backward Weight = \( 2[l_j + (l_{j+2}) + (l_{j+4}) + ...] + [(l_{j+1}) + (l_{j+3}) + ...] \).

If Forward Weight is greater than Backward Weight, then delete edge \( i \). Node \( p \) becomes the start node (or node \( q \) if it is the node with degree one) and node \( r \) becomes the marked node (or node \( s \) if it is the node with degree one). Else the roles of edges \( i \) and \( j \) as well as the corresponding nodes are reversed to obtain the deleted edge, start node and marked node. This results in Hamiltonian path \( T'' \) (over nodes \( V' - \{p\} \)). Let the set of edges of this Hamiltonian path be \( E'' \). Alternate edges starting from the edge incident to marked node are the edges belonging to the matching \( M \) (which might be sub-optimal). The matched edges are added to original graph \( G \), making it Eulerian. Go To STEP 4.

**STEP 4:**
Apply the algorithm of finding the Eulerian tour (open) as described in [41]. The procedure for finding an Open Eulerian tour is described later in section 5.1. The procedure is applied on graph \( G'' \). Graph \( G'' \) consists of the set of edges \( \{E + M - E''\} \). Starting from the search start node merge all the (traced) cycles into one. The tour ends at the marked node. If all the cycles are merged, i.e., all the edges from the set \( \{E + M - E''\} \) are traversed, STOP. Else, merge the cycle, not yet merged, into the path \( T'' \) at the node that is common to both. This might happen depending on the links we chose in defining the shortest path between two odd nodes. Then traverse the Hamiltonian path \( T'' \) starting from the marked node to complete the Eulerian path.

### 4.3 Improvements to the heuristic

To make the heuristic run faster, we suggest some improvements.
1. We modify STEP 1 and try to find a MST with minimum number of nodes with degree greater than two. We modify Kruskal’s Algorithm of finding MST to achieve this. Kruskal’s Algorithm works as follows:

   - **STEP 1**: Sort the edges of the graph \( G(V,E) \) in ascending order of the edge weights.
   - **STEP 2**: Let \( T \) be the spanning subgraph of \( G \) with \( E(T) = \emptyset \) and let \( i = 1 \).
   - **STEP 3**: If \( T + e_i \) is acyclic, set \( T ← T + e_i \). Set \( i ← i + 1 \). If \( i > |E| \), STOP. Otherwise, GO To STEP 2.

We modify the algorithm as follows:

If there are two edges \( e_i \) and \( e_j \) with the following properties:

a) both have same weight,

b) both when added to subgraph \( T \) do not create a cycle,

c) edge \( e_i \) when added to \( T \) results into \( d(k) \geq 3 \) for some node \( k \in V \) and edge \( e_j \) does not result into a node with \( d(k) \geq 3 \);

add edge \( e_j \) to the tree. This reduces the number of times STEP 2 is executed, whose running time is more than STEP 1. This improvement is incorporated in all the test runs presented in section 6 and section 7. We also suggest to specify a parameter \( α \) such that, for two edges \( e_i \) and \( e_j \), if \( l(e_j) = α.l(e_i) \) and properties (a) and (b) are satisfied add edge \( e_j \) to the tree.

2. We can modify STEP 2 by sorting the edges (for each node \( i \) with \( d(i) \geq 3 \)) that can be deleted, in decreasing order of the edge weights and storing them into a list \( L \). We then terminate the edge deletion and addition process by using the very first edge in the list for which we can find an edge that satisfies our criteria of addition. This might not be the optimal one, but will reduce the running time. So, depending on the willingness of the user, we may wish to introduce this modification. Let \( e \) be an edge both of whose end points belong to the set of nodes \( V'' \) with degree \( d(v) \geq 3 \) \( \forall v \in V'' \). We should give preference to this edge since by deleting this edge we are decreasing the degree of two nodes in one step.

4.4 Proof that the heuristic works

**Theorem 1** The heuristic will always result in a path that covers all the edges of the network at least once.

**Proof** Step 0 ensures that the complete graph \( G'(V', E') \) defined over odd nodes \( V' \subseteq V \) is extracted from the original graph \( G(V,E) \). Step 1 and Step 2 ensure that a minimum spanning tree is found and that MST is transformed into a Hamiltonian path \( T \) over the odd node set \( V' \). Step 3 deletes one of the end edges, \( e \), of path \( T \) and hence selects the end node, the start node and the marked node. It also gives us the matching \( M \). This Hamiltonian path \( T' (= T - e) \) is then transformed into walk \( T'' \) by converting the edges in \( G'(V', E') \) to the actual edges in \( G(V,E) \). \( T'' \) consists of edge set \( E'' \subseteq E \) and vertex set \( V'' \subseteq V \). Note that \( V' \subseteq V'' \). If the edges \( e' \subseteq T' \) belong to the matching \( M \), all the edges \( e'' \subseteq E'' \), that are due to edges \( e' \subseteq T' \), also belong to the matching \( M \). These matched edges are then added to the original graph resulting in an Eulerian graph \( G''(V,E'') \), where \( E'' = E \cup M \). Thus degree, \( d(i) \forall i \in V' \), is even for all nodes except for the start node and the end node (because all other odd nodes have been matched perfectly). Finally, we delete all the edges belonging to path \( T'' \) from the graph \( G''(V,E'') \) thus resulting in a graph with degree \( d(i) \forall i \in V' \) as even, except for the start node and the marked node. This is true because each node \( v \in T'' \) has even degree except for end
node and marked node ($T''$ is a walk from marked node to end node). When we delete the edges of this walk form the graph $G''(V, E'')$, degree of marked node (which is even) is reduced by one, degree of end node (which is odd) is reduced by one and the degree of rest of the nodes is reduced by two. This results in a graph $G(N, A)$ where each node has even degree except for start node and marked node. Step 4 applies end-pairing algorithm on this graph and solves it as an Open CPP. Thus a path $P$ exists that covers all the edges in graph $G(N, A)$. Append walk $T''$ to this path $P$ thus resulting in an open Eulerian tour that covers every edge of the original graph at least once.

4.5 Heuristic when start node is specified

For the case when we have a specified starting node, the same heuristic can be applied with a slight modification. Here, the forward and backward weights are defined over all the links of the path (paths including the last edge). If the graph is Eulerian, no changes are necessary. If the graph has two odd nodes and the start node is not one of them, add an edge (shortest path) joining these two nodes and apply the algorithm to find Eulerian tours. Else if the number of odd nodes is more than two, the graph over the odd node set is defined over all nodes $v \in V$ except for the specified start node if start node is of odd degree or else over all odd nodes. Apply the heuristic on this graph. End node is the node with degree one of the edge that would have been deleted while applying STEP 3 of the above heuristic (we do not delete it in this case). Rest of the procedure remains the same.

4.6 Solved examples

Next we present some solved examples to explain the working of this heuristic. The first example is the application of the heuristic on a general graph and the second one is application of the heuristic on the grid structured graph. We also present an example to solve the problem with a specified search start node.

4.6.1 General graph

Refer to the unit graph shown in figure 5(a). We will step through the heuristic to explain the procedure.

- **STEP 0:** Extract graph $G'(V', E')$ defined over the odd node set of the given graph. Link lengths are shown for the links with length greater than one.

![General graph to explain proposed heuristic](a)

![MST over $G'(V', E')$](b)

![Modified MST](c)

Fig. 5: Applying heuristic to general graphs

- **STEP 1:** Figure 5(b) shows the MST over the graph $G'$ (lines in full depict MST). Two nodes (1 and 16) have degree greater than two. Hence go to STEP 2.
STEP 2: Consider node 1. Edges that can be deleted are: (1 − 2), (1 − 7) and (1 − 19). If edge (1 − 7) is deleted there is no edge that can be added to MST without increasing the weight. But if we delete edge (1 − 7), we can add edges (2 − 7) or (7 − 19) without increasing the overall weight and without forming any cycles. Similarly, we can delete edge (1 − 19) and add edge (7 − 19). Let us delete link (1 − 19) and add edge (7 − 19). For node 16, we delete edge (16 − 17) and add edge (13 − 17) to complete STEP 2 of the algorithm. Now \( d(i) = 2 \) for all but two nodes and the Hamiltonian path \( T' \) is given by figure 5(c). Go to STEP 3.

STEP 3: Forward weight is equal to backward weight (15 units), so we can delete either edge 1 − 2 or 14 − 15. Let us delete edge 1 − 2. This results in a Hamiltonian path \( T'' \) with edges \( E'' \). Hence node 2 is the start node and node 1 is the end node. Node 14 is set as marked node. Edges (14 − 15), (16 − 13), (17 − 18), (19 − 7) are thus added to the original graph \( G \), as they form the matching \( M \).

STEP 4: Find the Open Eulerian tour that starts with the start node and ends at the marked node. The procedure is applied on modified graph \( G \) by traversing the edges from the set \( \{ E + M - E'' \} \). Starting at node 2 we trace 13 cycles and merge them into one (terminating the path at node 14). Then follow the Hamiltonian path \( T'' \) that terminates at node 1. The final Eulerian path given by the heuristic is: 2 − 3 − 4 − 5 − 9 − 14 − 15 − 23 − 22 − 21 − 20 − 19 − 11 − 6 − 10 − 19 − 12 − 18 − 20 − 17 − 21 − 16 − 22 − 15 − 9 − 13 − 16 − 17 − 18 − 13 − 12 − 9 − 8 − 12 − 11 − 8 − 4 − 7 − 3 − 6 − 2 − 1 − 10 − 11 − 7 − 8 − 5 − 14 − 15 − 16 − 13 − 17 − 18 − 19 − 11 − 7 − 6 − 1.

4.6.2 Grid structured graph

Grid graphs are specifically considered to work with our problem of finding casualties in a region of interest. We assume that the region can be subdivided into small cells and we will start our search process by moving from one cell center to another. As an example, we will apply our heuristic approach to the graph used to prove the lemma in section 3.3 (refer figure 4). Figure 6(a) shows the Minimum Spanning Tree after applying STEP 1 over the extracted graph \( G' (V', E') \) defined over set of odd nodes (obtained after applying STEP 0). Node 36 has degree greater than two. Hence we go to STEP 2. Figure 6(b) shows the modified MST where we delete edge (36 − 40) and we add edge (38 − 40). After applying STEP 3 we delete edge (19 − 31) and hence node 19 is the starting node, node 31 is the ending node and we set node 40 as marked node. Hence the path from node 40 to node 31 is the Hamiltonian path \( T'' \) and...
the edges in dashes (except for edge 19 – 31) are the edges that form the matching \( M \) and hence to be added to the original graph \( G \). Applying STEP 4 gives us the Eulerian tour.

### 4.6.3 Starting node specified

Refer to the unit graph as shown in figure 7(a). The shaded nodes are the nodes with odd degree. Let node 12 be the specified starting node. STEP 0 results into the graph \( G'(V', E') \) as shown in figure 7(b). Applying MST algorithm results in graph as shown in figure 7(c). There is no node with degree greater than two. Since forward weight is 10 and backward weight is 11, edge \((14 − 15)\) is to be deleted (but we do not delete it). Hence, node 12 is the start node, node 14 will be the end node and node 3 is set as the marked node. Edges \((3 − 5)\), \((8, 10)\) and \((11 − 15)\) are to be traversed twice. Go to STEP 4 and find Eulerian tour.

### 5 Alternate heuristic

In order to compute our solution quality, we need to compare our heuristic solution with the optimal solution. But obtaining the optimal minimum expected search time of finding the entity is not easy. One way to get the optimal solution is to do a total enumeration of all possible open Eulerian tours, which also is not so trivial. Hence, to compare the results of our heuristic, we propose a simple heuristic which basically tries to find \( k \)-different open Eulerian tours by doing a partial enumeration of the open Eulerian tours. We then compare our heuristic solution to the best, worst and average expected search times obtained from these \( k \)-different tours. Before describing the heuristic, we would like to draw the attention of the readers to the fact that given a non-Eulerian graph and a start node, the possible number of Eulerian tours depends on the possible number of different minimum perfect matchings and the possible number of Eulerian tours given a matching. For each start node we can have \( k_1 \) different matchings and for each matching we can have \( k_2 \) different Eulerian tours. Let us first restate briefly the algorithm for solving an Open CPP before describing the alternate heuristic.

#### 5.1 Algorithm for solving an Open CPP

The algorithm to find an open Eulerian tour for an Open CPP in case of an undirected graph can be briefly stated as follows:

- If the graph \( G(V, E) \) is Eulerian or has only two nodes with odd degree, no matching needs to be done. For the former case, simply apply the end-pairing algorithm given in [41] with any node as start node. For the later case, if the start node is one of the two odd nodes, apply the end-pairing algorithm. Else add an edge between the two odd nodes (shortest path between the two) and apply the end-pairing algorithm. Else if number of odd nodes is greater than two, introduce two new nodes \( n_1 \) and \( n_2 \) to the set of odd nodes and make the edge weights of these new nodes to all the original odd nodes less than zero. Also, give a very large weight to the edge between the two new nodes. Let this edge be \( l \). Let the graph defined over the odd nodes plus the two pseudo-nodes be \( G'(V', E') \).
• Next apply Minimum Weighted Perfect Matching algorithm (for general/non-bipartite graphs) on this new set of odd nodes. Let this matching be \( M \). It will always have two edges - edge \( l_1 = (n_1, O_1) \) and edge \( l_2 = (n_2, O_2) \), where \( O_1 \) and \( O_2 \) are the nodes belonging to the original odd node set. This is due to the negative weights on the edges from the two pseudo-nodes to all other original odd nodes. Delete these edges \( (l_1, l_2) \) from \( M \) and set \( O_1 \) and \( O_2 \) as the start node and end node.

• Expand this matching w.r.t the original links in graph \( G(V, E) \) and add the edges \( e \in M \) to \( G(V, E) \).

• Introduce a pseudo-edge between start node and end node with a very large weight (this edge will be deleted later to create an open Eulerian tour) hence making the graph Eulerian.

• Apply end-pairing algorithm given in [41] with the pseudo-edge as the first edge and finally delete this pseudo-edge from the Eulerian tour, thus giving an Open Eulerian tour.

5.2 Modified implementation of Open CPP algorithm

We now present the alternate heuristic used for finding \( k \)-minimum perfect matchings and \( k \)-Eulerian tours.

5.2.1 \( k \)-Alternate optimal minimum perfect matchings

We apply the Minimum Weight Perfect Matching algorithm for general graphs using LEDA library functions. LEDA is a C++ class library for efficient data types and algorithms which provides algorithmic in-depth knowledge in the field of graph and network problems (for more information about LEDA please refer to www.algorithmic-solutions.com/enleda.htm). All functions for computing minimum weighted perfect matchings using LEDA guarantee a running time of \( O(nm \log n) \), where \( n \) and \( m \) denote the number of nodes and edges, respectively. We obtain \( k \)-optimal minimum weighted perfect matchings on the graph \( G'(V', E') \) as follows:

Let \( M \) be the set of edges in the first matching. Delete one of the edge \( e \in M \) from \( G'(V', E') \) and apply the matching algorithm again. If we get a matching \( M' \) with weight equal to the matching \( M \), we have found the second optimal matching. Else edge \( e \) will always be in any optimal matching for \( G'(V', E') \) and we delete another edge from set \( M \) and apply the matching algorithm again and so on. If we do not find any edge that can be deleted from set \( M \) that gives a matching \( M' \) with weight equal to the matching \( M \), it implies that there is only one optimal solution. Otherwise, we keep on finding new matchings till we get the required \( k \)-optimal minimum weighted perfect matchings. Set \( M \) is always updated as the new set of matched edges at each step. We use \( k \) as a small number (generally \( k < 10 \) for our scenario).

5.2.2 Modified end-pairing algorithm for finding \( k \)-Eulerian tours

We use Edmonds and Johnson’s end-pairing algorithm to find Eulerian tours. We restate the algorithm here so that we can explain our modifications later.

• **STEP 0**: Let \( r \) be any node. Let both \( n_0 = n = r \). Let \( e_r \) be any edge incident to \( r \). Let \( e_1 = 0, e_2 = \infty \) and \( e_0 = e_r \). Initially all edges are unpaired. Set \( e = e_r \) and GO TO Step 1.

• **STEP 1**: Let \( n' \) be the node other than \( n \) incident to edge \( e \). If there is an edge \( e' \) with an end meeting \( n' \) which is not yet paired, Go To Step 2. Otherwise, \( n' \) must be equal to \( n_0 \). In that case, form the edge pairs \( (e_1, e_0) \) and \( (e, e_2) \) meeting node \( n_0 \). Go To Step 3.
• **STEP 2**: Pair the edges $e$ and $e'$ meeting $n'$. Set $n = n'$ and $e = e'$. Go To Step 1.

• **STEP 3**: Change $n_0$ to be any node which has at least one pair $(e_1, e_2)$ of edges meeting it and at least one unpaired edge also meeting it. Let $e_0$ be an unpaired edge meeting $n_0$. Set $n = n_0$ and $e = e_0$, and go to Step 1. If no such node $n_0$ exists, terminate.

As explained in section 5.1, the matching algorithm on graph $G'(V', E')$ will result in matching $M$ and will specify two nodes $O_1$ and $O_2$ as the start and end nodes. Hence, let $r = O_1$ in Step 0 of the algorithm and let $e_r$ be equal to the pseudo-link between $O_1$ and $O_2$ (as defined in section 5.1). Now, set $n' = O_2$ with any edge $e'$ incident to it that is not yet paired. Go to Step 2 and pair the two edges. Now at each subsequent iteration, we have a choice of edges that can be selected as edge $e'$ and depending on what edge we choose, we will have a different $n$ and hence a different $e'$ in the next iteration. This cycle continues and depending on what choices we make, we get a different path. We make use of a random generator function to randomly choose an edge out of the given candidate set and hence every time we use the algorithm we get a new tour (if one exists). Also the choice of $n_0$ in Step 3 is done using a random generator function to randomly choose one node to be $n_0$ from a given set of nodes satisfying the conditions in Step 3 of the algorithm. Hence if the algorithm is run $k$ times, we get $k$-different Eulerian tours.

### 6 Results: Search start node not specified

We now present some of the preliminary results obtained by applying our heuristic on 400 randomly generated graphs with non-negative weights. All the test runs were done on an HP Workstation (xw4100) with a Pentium 4 Processor and 1GB of RAM. In this section we assume that the start node is not specified and we let the heuristic decide its starting node.

#### 6.1 Solution quality and computational efficiency

We present our results for all the different runs we made, with each run having a different combination of number of matchings and the number of Eulerian tours. Let us call the heuristic presented in section 4.2 as **Heuristic A** and the alternate heuristic presented in section 5.2 (partial enumeration of Open CPP) as **Heuristic B**. The various runs made are:

- (a) 1 Perfect matching and 200 Eulerian tours.
- (b) 5 Perfect matching and 100 Eulerian tours.
- (c) 10 Perfect matching and 10 Eulerian tours.
- (d) 10 Perfect matching and 50 Eulerian tours.
- (e) 10 Perfect matching and 100 Eulerian tours.

The results shown here are based on tests done on 400 graphs with varying number of links ranging from 10 links to 700 links. A typical result showing the percentage increase of the best solution (obtained from Heuristic B) from that of heuristic solution (obtained from Heuristic A) versus the graph number (in increasing order of the number of links) is shown in figure 8(a) for the run with one matching and 200 Eulerian tours. Same is true for all other runs. Similarly figure 8(b) shows a typical result for percentage increase in running time of Heuristic B from that of Heuristic A versus the graph number (in increasing order of the number of links).

Comparison between **Heuristic A** and **Heuristic B** solutions is based on the percentage increase or decrease of **Heuristic B** solution from that of the **Heuristic A** solution. The running times (in clock seconds) of two heuristics are also compared based on the percentage increase or decrease of **Heuristic B** running time from that of the **Heuristic A** running time. The average percentage increase or decrease of all the test runs done for the case with no specified search.
start node is shown in table 2. The table shows the comparison of the best, the worst and the average solutions obtained by doing a partial enumeration (Heuristic B) with Heuristic A solution on an average basis for the 400 graphs. Running time comparison is shown as well. In the table, $M$ stands for matchings and $T$ stands for Eulerian tours. On an average the best solution obtained by doing the partial enumeration is approximately 4% higher than our heuristic solution. The worst solution of the enumeration on an average is 9% higher than our heuristic solution and the average solution of the enumeration is 7% higher than our heuristic solution. Also the running time for doing the partial enumeration is approximately 27 times more than the running time of our heuristic.

The comparison between the number of links of the graph and the running time of our heuristic (Heuristic A) is shown in figure 9(a). The running time is fairly constant for small and medium sized graphs but starts to increase with the increase in the number of links of a graph, as should be expected. Graph weights (half of total graph weight) when compared to the expected search times obtained by the heuristic are nearly equal as depicted by figure 9(b). It shows that the expected search times are linear in relation to half of the total graph weight. On an average, the heuristic solution is only 1.4% more than half of the total weight of the graph. This is because the number of odd nodes in most of these graphs is not too much hence we expect the expected search time to be close to half of the total graph weight.

### 7 Results : Search start node specified

In this section we consider the Northridge region in Los Angeles, CA as our casestudy. We assume the initial searcher location and accordingly get a candidate set of search start nodes. As described earlier in section 4.1, this set is limited to a small number. We got the road network of the region from Tele-Atlas(please refer to www.na.teleatlas.com). Each link in the
7.1 Solution quality and computational efficiency

The results shown here are based on tests done on 98 graphs (regions) with the number of links ranging from 400 links to 750 links. In this section, comparisons between Heuristic A and Heuristic B solutions are based on the percentage increase or decrease of Heuristic A solution from that of the Heuristic B solution. Running times (in clock seconds) of two heuristics are compared based on the percentage increase or decrease of Heuristic B running time from that of the Heuristic A running time. We first present the results when the running time improvement is not applied to the heuristic.

7.1.1 Heuristic without improved running time

The various runs made are:
(a) 3 Start nodes, 3 perfect matchings and 50 Eulerian tours.
(b) 5 Start nodes, 1 perfect matching and 200 Eulerian tours.
(c) 5 Start nodes, 2 perfect matchings and 200 Eulerian tours.

A typical result showing the percentage increase of the best expected search time (obtained from Heuristic B) from that of the best heuristic solution (obtained from Heuristic A) versus the graph number (in increasing order of the number of links) is shown in figure 10(a) for the run with three start nodes, three matchings and 50 Eulerian tours. Same is true for all other runs. Similarly figure 10(b) shows a typical result for percentage increase in running time of Heuristic B from that of Heuristic A versus the graph number (in increasing order of the number of links).

The average percentage increase or decrease of all the test runs done for the case with a specified search start node is shown in table 3. The table shows the comparison of the best,
worst and average Heuristic A solutions (due to various start nodes) to the best, worst and average Heuristic B solutions (due to start nodes and enumeration) obtained by doing the partial enumeration. The running time comparisons of the two heuristics is shown as well. In the table, $K$ stand for the number of search start nodes, $M$ stands for matchings and $T$ stands for Eulerian tours. On an average the best solution obtained from our heuristic is approximately 2.7% higher than the best solution obtained by doing the partial enumeration. The worst solution of the enumeration on an average is approximately 1.3% higher than our heuristic solution and the average solution obtained from our heuristic is approximately 0.15% higher than the best solution obtained by doing the partial enumeration. Also the running time for doing the partial enumeration is approximately 4 times more than the running time of our heuristic.

The comparison between the number of links of the graph and the running time of our heuristic (Heuristic A) for the first run (3 start nodes each with 3 matchings and each matching with 50 tours) is shown in figure 11(a). The running time is fairly constant till the number of links is around 400 links but starts to increase after that. Graph weights (half of total graph weight) when compared to the best expected search times obtained by the heuristic are shown in figure 11(b). The behavior is similar for the case of 5 start nodes each with 1 matching and 200 Eulerian tours as well as the case of 5 start nodes each with 2 matchings and 200 Eulerian tours. On an average, the heuristic solution is 28% more than half of the total weight of the graph for the first run with 3 start nodes (3M and 50T), 27% more for the second run with 5 start nodes (1M and 200T) and 27% more for the third run with 5 start nodes (2M and 200T).
When we apply the improvement suggested in section 4.3, the average percentage increase or decrease of all the test runs done for the case with a specified search start node is shown in table 4. The table shows the comparison of the best, worst and average Heuristic A solutions to the best, worst and average Heuristic B solutions obtained by doing the partial enumeration. Also the running time comparisons of the two heuristics is shown as well. Hence on an average the best solution obtained from our heuristic is approximately 6.5% higher than the best solution obtained by doing the partial enumeration. The worst solution of our heuristic, on an average, is approximately 3% higher than the enumeration solution and the average solution obtained from our heuristic is approximately 4.2% higher than the best solution obtained by doing the partial enumeration. The running time for doing the partial enumeration is approximately 10 times more than the running time of our heuristic. Also, the heuristic solution is 33% more than half of the total weight of the graph for the first run with 3 start nodes (3M and 50T), 32% more for the second run with 5 start nodes (1M and 200T) and 32% more for the third run with 5 start nodes (2M and 200T).

<table>
<thead>
<tr>
<th>Run Type</th>
<th>Compare Best</th>
<th>Compare Worst</th>
<th>Compare Average</th>
<th>Compare Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3K,3M,50T</td>
<td>7.187</td>
<td>2.605</td>
<td>4.318</td>
<td>493.44 ≈ 6 Times</td>
</tr>
<tr>
<td>5K,1M,200T</td>
<td>6.252</td>
<td>3.637</td>
<td>4.416</td>
<td>379.21 ≈ 5 Times</td>
</tr>
<tr>
<td>5K,2M,200T</td>
<td>6.5795</td>
<td>2.859</td>
<td>4.137</td>
<td>1913.916 ≈ 20 Times</td>
</tr>
</tbody>
</table>

Table 4: Average comparisons for the best, worst and average solutions when the running time of heuristic is improved

8 Case of multiple uniformly distributed entities

So far we have assumed that exactly one entity was present on the network. We now establish that the choice of path does not change when more than one entity is present, as long as the entities are uniformly distributed. We recall that, even when multiple entities are present, we seek a path that minimizes the search time for the first entity. We show that the same path minimizes $E[T_S|m = n]$ for $n = 1, 2, \ldots$, and hence conclude that this path minimizes $E[T_S]$ for the case when the number of entities is not known in advance (except for the fact that it is greater than zero).
Definition Let \( m \) be the number of entities present on the network. We assume that there are at least \( m_1 \) entities and at most \( m_2 \) entities on the network, i.e., we assume \( m_2 > m_1 \geq 1 \). Let \( \lambda \) be the overall rate (density) of entities for the entire network and \( L \) be the overall length of the network. The probability of \( m \) entities present on the network, \( P(m) \), is given by the expression:

\[
P(m) = \frac{(\lambda L)^m e^{\lambda L}}{m!} \left( 1 - \sum_{m=0}^{m_1-1} \frac{1}{m!} \right) + \frac{1}{m_1!} \sum_{m=m_2+1}^{\infty} \frac{1}{m!}.
\]

Then \( E[T_S] = E[T_S|m = 1] \times P(m = 1) + E[T_S|m = 2] \times P(m = 2) + \cdots. \)

Theorem 2 The path that is optimal for \( m = 1 \) case is also optimal for \( m > 1 \) case.

Proof (By Induction)

Let \( P_1 \) be the optimal path when \( m = 1 \), and \( P_2 \) be the path that is optimal for \( m = 2 \). For \( m = 2 \) case, let the entities be labeled \( X_1 \) and \( X_2 \). Let \( X \) be the first entity that is found by the searcher, i.e., \( X = \min(X_1, X_2) \). We have:

\( E[X_1|P_2] \geq E[X_1|P_1] \) (since \( P_1 \) is optimal for \( m = 1 \) case); if only \( X_1 \) is present.

\( E[X_2|P_2] \geq E[X_2|P_1] \) (since \( P_1 \) is optimal for \( m = 1 \) case); if only \( X_2 \) is present.

Since both \( E[X_1|P_2] \) and \( E[X_2|P_2] \) are greater than \( E[X|P_1] \) and \( X \) is first of \( (X_1, X_2) \) to be detected, it follows that \( E[X|P_2] \geq E[X|P_1] \).

Hence \( P_1 \) is optimal for the case \( m = 2 \).

Now let us assume that \( P_k \) is the optimal for the cases \( m = 3, \ldots, n \).

We have to prove that it is optimal when \( m = n + 1 \).

Let \( P_m \) be the path that is optimal for \( m = n + 1 \).

Let the entities present on the network be labeled \( X_1, X_2, \ldots, X_n, X_{n+1} \) and let \( X \) be the first entity found by the searcher, i.e., \( X = \min(X_1, X_2, \ldots, X_n, X_{n+1}) \).

Since it is true for \( k = n \) case, we can write for different combinations of \( n \) out of \( n + 1 \) entities;

\[
E[\min(X_1, X_2, \ldots, X_{n-1}, X_{n+1})|P_m] \geq E[\min(X_1, X_2, \ldots, X_{n-1}, X_{n+1})|P_1];
\]

if only \( (X_1, X_2, \ldots, X_{n-1}, X_{n+1}) \) were present.

\[
E[\min(X_1, X_2, \ldots, X_{n-2}, X_n, X_{n+1})|P_m] \geq E[\min(X_1, X_2, \ldots, X_{n-2}, X_n, X_{n+1})|P_1];
\]

if only \( (X_1, X_2, \ldots, X_{n-2}, X_n, X_{n+1}) \) were present.

\[
\vdots
\]

\[
E[\min(X_2, X_3, \ldots, X_{n-1}, X_n, X_{n+1})|P_m] \geq E[\min(X_2, X_3, \ldots, X_{n-1}, X_n, X_{n+1})|P_1];
\]

if only \( (X_2, X_3, \ldots, X_{n-1}, X_n, X_{n+1}) \) were present.

Therefore, \( E[X|P_m] \geq E[X|P_1] \), where \( X = \min(\min(X_1, X_2, \ldots, X_n), X_{n+1}) \) to be detected.

The theorem follows.

9 Conclusions and future research

In this paper, an attempt has been made to determine a good search strategy for finding an immobile entity on a network. Two separate cases were studied - one with a given search start node and another with an unspecified search start node. A heuristic was suggested that finds an open Eulerian tour covering all the links of the network at least once. This heuristic also forces the edges that are to be repeated towards the end of the tour. We also developed an alternate heuristic to do a partial enumeration of all possible Open CPP solutions. Our heuristic solution was compared to the best, worst and the average expected search times obtained by the enumeration. For the first case where the search start node is not specified,
on an average, the best solution obtained from the partial enumeration was 4\% more than the heuristic solution. Also, on an average, the clock time required to do the partial enumeration was 27 times more than the clock time required to get the heuristic solution. For the second case we used a real road network of the Northridge area in Los Angeles, CA and assumed an initial searcher location. Two different types of implementations of our heuristic were applied - one with and another without applying the running time improvement as suggested earlier. For the first type even though our heuristic solution, on an average, was 2.7\% more than the best solution obtained from the partial enumeration, the clock time required to do the partial enumeration was 4 times more than the time required for the heuristic solution. For the second type even though our heuristic solution was 6.7\% more than the best solution obtained from the partial enumeration, the clock time required to do the partial enumeration was 10 times more than the time required for the heuristic solution.

Future work should consider the search problem in which we wish to find the first \( n \) entities in a given region. A typical scenario is the case of a disaster environment with a number of casualties scattered around the quake hit region and emergency response vehicles trying to find these casualties to be picked up and delivered to hospitals. These vehicles will try to search for the first \( n \) casualties, where the value of \( n \) depends on the remaining capacity on-board the emergency vehicle. It is not clear whether the path that is best for the case where we are interested in searching only a single entity is also best for the case where we are interested in finding first \( n \) entities. Note that theorem 2 suggests that when we are trying to find only the first entity the path that is optimal for \( m = 1 \) case is also optimal for \( m > 1 \) case, whereas for this scenario, we are interested in finding the first \( n \) entities. Our initial findings show that it is not always the case. It might be better to re-optimize (to find the second entity) once the identity and the location of the 1st entity is known.

Acknowledgment

This work is being funded by the Air Force Office of Scientific Research via grant F49620-01-0371. We would like to thank three anonymous referees for their comments and suggestions. In particular, we are thankful to one of the referee for pointing out that similar work has been done by other researchers and also pointing out the differences between these efforts.

References


