Finding The Most Navigable Path in Road Networks

Ramneek Kaur · Vikram Goyal · Venkata M. V. Gunturi

Received: date / Accepted: date

Abstract Input to the Most Navigable Path (MNP) problem consists of the following: (a) a road network represented as a directed graph, where each edge is associated with numeric attributes of cost and "navigability score" values; (b) a source and a destination and; (c) a budget value which denotes the maximum permissible cost of the solution. Given the input, MNP aims to determine a path between the source and the destination which maximizes the navigability score while constraining its cost to be within the given budget value. The problem can be modeled as the arc orienteering problem which is known to be NP-hard. The current state-of-the-art for this problem may generate paths having loops, and its adaptation for MNP that yields simple paths, was found to be inefficient. In this paper, we propose five novel algorithms for the MNP problem. Our algorithms first compute a seed path from the source to the destination, and then modify the seed path to improve its navigability. We explore two approaches to compute the seed path. For modification of the seed path, we explore different Dynamic Programming based approaches. We also propose an indexing structure for the MNP problem which helps in reducing the running time of some of our algorithms. Our experimental results indicate that the proposed solutions yield comparable or better solutions while being orders of magnitude faster than the current state-of-the-art for large real road networks.

Keywords Spatial Networks · Road Networks · Routing

1 Introduction

The problem of finding the most navigable path takes the following as input: (a) a directed graph representation of a road network where each edge is associated with a cost (distance or travel-time) value and a navigability score value; (b) a source and a destination and; (c) a budget value. Given the input, the objective is to determine a path between the source and the destination which has the following two characteristics: (i) the sum of navigability score values of its constituent edges is maximized and, (ii) the total cost (in terms of distance or travel-time) of the path is within the given budget. In other words, MNP is a constrained maximization problem.

This problem finds its applications in navigation systems for developing nations. Quite often, street names in developing countries are either not displayed prominently or not displayed at all. In such cases, it becomes difficult to follow the conventional navigation instructions such as *"Continue on Lal Sai Mandir Marg towards Major P Srikumar Marg"*. In such nations, it is desirable to travel along a path which is "easily identifiable" by a driver. For instance, consider the example shown in Figure 1. Here, Path 1 is the shortest path between KFC in Cannaught Place (point A) and the Rajiv Chowk metro station (point B). The path has a travel time of 10 minutes. However, this route is potentially confusing due to lack of prominently visible sites and easily identifiable turns. To navigate on this route, one would have to

Ramneek Kaur
IIIT-Delhi, New Delhi, India
E-mail: ramneekk@iiitd.ac.in

Vikram Goyal
IIIT-Delhi, New Delhi, India. E-mail: vikram@iiitd.ac.in

Venkata M. V. Gunturi
IIT Ropar, Rupnagar, India. E-mail: gunturi@iitrpr.ac.in

heavily rely on a very accurate GPS system (potentially expensive) operating over a good quality map with no missing roads, both of which may be non-trivial for a common traveler in developing countries.

In contrast, consider Path 2 from A to B in Figure 1 with a travel time of 12 minutes. This route involves going past some popular sites like the hotel Sarvana Bhavan and the hotel Radisson Blu Marina, and then taking the first right turn that follows. Given the challenges associated with transportation and navigational infrastructure in a developing nation setting, one may choose Path 2 even if it is 20% longer than the shortest path. This is because it is easier to describe, memorize, recall and follow. This option would be even more amenable if the driver is not well versed with the area.

Fig. 1: Problem illustration

Furthermore, travelers who are not comfortable using navigation systems while driving, senior citizens for instance, generally look up the route suggestions before starting their journey. Such travelers tend to memorize the route based on the sites en-route to their destination. These travelers would thus be benefited by our concept of most navigable paths which are potentially easier to memorize. The concept of most navigable paths would also help drivers of two-wheelers (predominant in developing countries) by suggesting routes which are easier to follow, as it can be difficult to follow step-by-step instructions on a screen while driving two-wheelers. To the best of our knowledge, both Google maps (`www.google.com/maps`) and Bing maps (`www.bing.com/maps`) do not have the option of navigable paths. The notion of "navigability" can also be customized for different sets of user requirements. We discuss some more notions of navigability in Section 2.2 and show that our algorithms can be generalized to consider these different notions.

1.1 **Computational challenges**

Finding the "Most Navigable Path (MNP)" is computationally challenging. The MNP problem is formalized as the Arc Orienteering Problem (AOP) (a maximization problem under constraints) which is known to be an NP-hard combinatorial optimization problem [2, 11]. The AOP problem can easily be reduced [26] to the Orienteering Problem (OP) which is also NP-hard [10, 12, 18]. Another factor which adds to the complexity of the MNP problem is the scale of real road networks, which typically have hundreds of thousands of road segments and road intersections.

Challenges in adapting a minimization problem: It is important to note that a maximization problem such as the MNP problem *cannot be trivially reduced into a minimization problem* by considering the inverse of the navigability scores. Even without the budget constraint, the MNP problem involves maximizing the sum of navigability scores of the output path. Mathematically, it is equivalent to maximizing the sum of n parameters, $s_1, s_2, \ldots, s_n$ (s_i denotes the score of edge e_i in a path); *which is not equivalent* to minimizing the sum of their inverses $\frac{1}{s_1}, \frac{1}{s_2}, \ldots, \frac{1}{s_n}$. Figure 2 shows an example illustrating this fact. In the Figure, the path $s \to a \to b \to d$ has a higher navigability score. However, when we consider the inverses of the navigability score and adopt a minimization based approach, we would get the path $s \to c \to e \to d$ as the solution, which is incorrect.

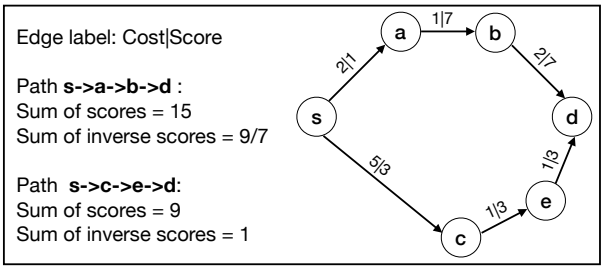


Fig. 2: Example illustrating challenges of using inverses of navigability scores.

To summarize, one cannot trivially generalize algorithms developed for shortest path problems (e.g., [7, 9, 13, 17]) to get an optimal solution for the MNP problem. For the same reason, algorithms developed

for k shortest loop-less paths problem [14, 20, 28] and route skyline queries [16, 25] can also not be trivially adapted to solve the MNP problem optimally. This holds true despite the fact that they provide an opportunity to incorporate the budget constraint by making small changes to their algorithm. Basically, during the path exploration in [14, 16, 20, 28], any candidate path whose cost increases beyond the budget is pruned. Note that k would be set such that we obtain at least one path within our designated budget. We would broadly refer [7, 9, 13, 14, 16, 17, 20, 28] as minimization based approaches.

Note that in addition to lack of mathematical equivalence in the objective functions, the minimization based approaches do not achieve the following desirable property that an ideal algorithm for the MNP problem should have: if a user is willing to spend more time (i.e., more budget), then he/she should be suggested paths with even higher scores that were not valid otherwise. This means that the score of returned path should increase monotonically with *budget* value. The minimization based approaches, when adapted for the MNP problem, do not guarantee such solutions.

Despite the shortcomings of the minimization based approaches, we adapted the advanced route skyline computation (ARSC) algorithm proposed in [16] as a representative sample of that class of algorithms for experimental comparison with our proposed approaches. However, our experiments showed that the proposed algorithms outperform the adaptation in terms of solution quality.

1.2 **Limitations of related work**

The existing algorithms for the AOP problem (or the OP as AOP can be reduced to OP [26]) can be divided into three categories: exact, heuristic and approximation algorithms. Exact algorithms have been proposed in [2, 8]. However, these algorithms cannot scale up to any real world road networks. For instance, algorithm proposed in [2] takes up to 1 hour to find a solution for a graph with just 2000 vertices.

There have been several works which proposed heuristic algorithms for the AOP [24, 27] and OP [5, 23] problems. A core requirement of these algorithms is pre-computation of *all-pairs shortest paths* of the input graph. This pre-computation step is necessary for ensuring their scalability (as also pointed out by Lu and Shahabi[19]). However, it is important to note that in any realistic scenario, urban networks keep updating frequently, for e.g., roads may be added, closed or heavily congested (due to repair or accidents) etc. Thus, any real-life system for the MNP problem working on large-scale urban road maps cannot use these techniques which require frequent computation of all-pairs shortest paths.

To the best of our knowledge, the only heuristic algorithm that does not pre-compute shortest paths is [19]. However, it generates paths with loops. We adapted their solution for the MNP problem. However, our experimental results indicated a superior performance of our algorithms. Our understanding is that their algorithm is more suitable when there are very few edges with a non-zero navigability score value.

Gavalas et al.[11] propose an approximation algorithm for the OP problem where edges are allowed to be traversed multiple times, a relaxation not suitable for our problem as it would be pointless to drive unnecessarily in a city to reach a destination. Similarly, approximation results were proposed in [3, 4, 6, 21, 22]. But they would also need to pre-compute all-pairs shortest paths for efficiency, which as discussed previously is not suitable for the MNP problem in typical real-world scenarios.

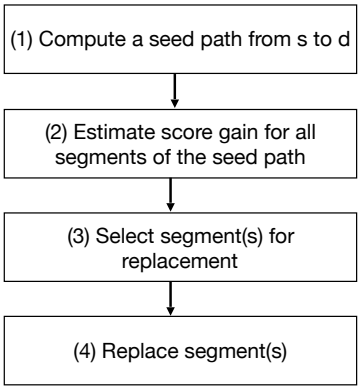


Fig. 3: Proposed framework

1.3 Proposed Solution Framework

Our overall framework for solving the most navigable path problem primarily consists of four steps (as shown in Figure 3). Firstly, we compute an *initial seed path* from s to d. In the second step, we estimate the potential gain in path score that can be achieved upon replacing each segment of the seed path with a new segment. In the third step, we select a subset of segments of the seed path for replacement with the goal of improving the navigability score of the solution. Lastly, the segments selected in the third step are replaced, and while this is being done, we ensure that the total cost of the resulting path is within the budget value.

1.4 Contributions

A preliminary version of this paper appeared in [15]. In [15], we put forth the problem of finding the most navigable path and proposed the overall solution framework shown in Figure 4. We used shortest paths for computing the initial seed path (first step), and used a modified bi-directional breadth first search for computing the potential increase in score values (second step). We proposed an algorithm called *OPTD* for step 3 (details in Section 3.3.3). This algorithm made some simplifying assumptions such as ignoring the budget constraint while computing the solution. In [15], the budget constraint was applied as a post processing step. Lastly, we also proposed an index as a heuristic. In this work, we make the following new contributions over our conference submission:

1. Propose new algorithms for different modules in our solution framework proposed (refer Figure 4).
   (a) We explore a new technique for computing the seed path in Step-1.
   (b) We propose two new algorithms for the Step-3 called *OPTF* and *SUBOPTF*.
2. Add semantic richness to the notion of navigability by extending our notion of navigability to include parameters such as the number of turns in the path, the navigability scores on the road intersections, etc.
3. Conduct more thorough experimental evaluation
   (a) We conduct our experiments on a much larger road network (of New York, USA) which contained 264,346 nodes and 733,846 edges.
   (b) We implemented all the newly proposed techniques and compared them against an adaptation of a route skyline algorithm [16] and the current state of the art [19] most relevant to our problem setting.
   (c) We introduce new experiments by considering different parameters.
      – We studied the effect of score values (of edges) on the performance of algorithms. To this end, we assigned the navigability scores in two ways: (a) random distribution (conference submission), (b) normal distribution.
      – We studied the effect of different parameters on the solution quality and running time in a detailed fashion.
      – Lastly, we also consider selecting the same seed path, and applying the improvement algorithms of our proposition and that of the state-of-the-art, and compare their performance.
4. Conduct a case study based on a real-life scenario in a developing nation that proves the effectiveness of our notion of navigability and the proposed algorithms

The rest of the paper is organized as follows. We cover the basic concepts and formally present the problem definition in Section 2. The proposed framework is discussed in detail in Section 3, followed by a description of the proposed algorithms in Section 4. In Section 7, we discuss the experimental evaluation. Finally, Section 8 concludes the paper.

2 Basic concepts and problem definition

Definition 1 Road network: A road network is represented as a directed graph $G = (V, E)$, where the vertex set V represents the road intersections and the edge set E represents the road segments. Each edge in E is associated with a cost value which represents the distance or travel-time of the corresponding road segment. Each edge is also associated with a score value (≥ 0) which represents the navigability score of the corresponding road segment. We refer to an edge with a score value > 0 as a navigable edge.

Definition 2 Path: A path is a sequence of connected edges $< e_1\, e_2 \ldots e_n >$. For this work, we consider only simple paths (paths without cycles).

Definition 3 Segment of a path: A sequence of connected edges, $< e_i\, e_{i+1} \ldots e_j >$, denoted as S_{ij}, is a segment of the path $P =< e_1\, e_2 \ldots e_n >$ if $1 \leq i \leq j \leq n$. $S_{ij}.score$ denotes the sum of scores of all the edges in S_{ij}. Likewise, $S_{ij}.cost$ denotes the sum of costs of all the edges in S_{ij}. $S_{ij}.start$ and $S_{ij}.end$ denote the first and last vertices of S_{ij}.

For example, $< (s,a)(a,b)(b,d) >$ is a path in Figure 2. $S_{23} =< (a,b)(b,d) >$ is a segment of this path. $S_{23}.score = 14$ and $S_{23}.cost = 3$. All edges of this path are navigable edges.

2.1 Problem definition

Input consists of:
(1) A road network, $G = (V, E)$, where each edge $e \in E$ is associated with a non-negative cost value and a navigability score value.
(2) A source $s \in V$ and a destination $d \in V$.
(3) A positive value *overhead* which corresponds to the maximum permissible cost allowed over the cost of the minimum cost path from s to d. In this paper, we refer to the term $(overhead + cost\, of\, the\, minimum\, cost\, path\, from\, s\, to\, d)$ as the *budget*.
Output: A path from s to d
Objective function: $Maximize\ path.score$
Constraint: $path.cost \leq budget$.

2.2 Practical considerations while using MNP in real life

While using MNP problem in real life one would have to first define the notion of "navigablity" and then assign the navigability scores to road segments accordingly. Though the score values may be subjective, a rule of thumb could be followed. Scores could be assigned over a range (e.g., 0-15) where higher values (e.g., 10-15) are given to roads which satisfy the stated notion of navigability to a greater extent, and lower values (e.g., 0-5) otherwise. Following are sample use-cases which define the notion of "navigability" in some typical developing country scenarios.

Consider a case where a user wants to take a route which has the maximum number of easily identifiable sites (to ease the following of path) on its constituent road segments, while constraining the total cost to be within the budget. This can be done by assigning higher scores to edges (e.g., 10-15) containing unique/popular sites like a well-known temple or a prominent building, and lower scores (e.g., 1-5) to edges with sites like petrol pumps and ATMS. A road segment with no such easily identifiable site may be given a score value of zero.

One can also implement a slightly different interpretation of navigability by assigning scores based on the quality of road segments. This is often a requirement in developing nations where the quality of roads is usually not good. High navigability scores (e.g., 10-15) can be given to roads which are wider and/or have less number of potholes. In contrast, narrow roads and high number of potholes can decrease the navigability score of a road. Consequently, the most navigable path would now mean a route which ensures smoothest drive to the destination, while constraining the total cost to be within the budget.

Lastly, in some cases, a user may want to minimize the number of left and/or right turns in addition to choosing a path with easily identifiable sites. As discussed previously, road segments with easily identifiable sites can be modeled by giving high navigability scores to the corresponding edges in the graph. Priorities for turns can be incorporated by expanding each intersection into multiple nodes and adding edges that represent all the turns. Later, score can be assigned to these new edges (explained in detail in Section 5). For example, if one prefers straight (i.e., no turns) routes, then at each road intersection, we can assign more score (e.g., 14) to the edge that represents a straight route. The edges representing left or right turns would get a low navigability score (e.g., 5). Consequently, the resulting routes would have fewer number of turns. In Section 5, we describe our approach to generalize the algorithms proposed in the upcoming sections to consider turn based notions of navigability.

3 Proposed framework

Figure 4 shows the modules of the proposed framework along with the techniques proposed for each module. We now explain each of these techniques in detail.

3.1 Computing the initial seed path

We have the following two ways to compute the initial seed path between our given source and destination nodes:

1. **Shortest Path Seed:** We use the shortest path (based on the cost parameter) between our source and destination node as the initial seed path.
2. **Vicinity Based Seed:** In this approach, we first select all the edges which have the following two properties: (a) score value is greater than zero and, (b) they lie within the elliptical vicinity of our source and destination nodes. We call this set as $NavEdges_{sd}$. Elliptical vicinity of a source s and destination d is defined as the set of all edges which lie in an ellipse drawn with s and d as its two focii and *budget* as the length of the major axis. Note that in our implementation we use our proposed *Navigability Index* for determining these edges efficiently. Details on the Navigability Index are presented in Section 3.2.2. Following creation of the set $NavEdges_{sd}$, we randomly pick an edge (u, v) from this set and compute the following shortest paths (based on the cost attribute). The first one being between s and u, and the second one between v and d. Let $\mathcal{P}$ be the path (between s and d) created by concatenating these two shortest paths (denoted as $s \rightsquigarrow u \rightarrow v \rightsquigarrow d$). $\mathcal{P}$ is selected as the seed path if its cost lies within the budget value, otherwise, the same process is repeated until we get a valid seed path.

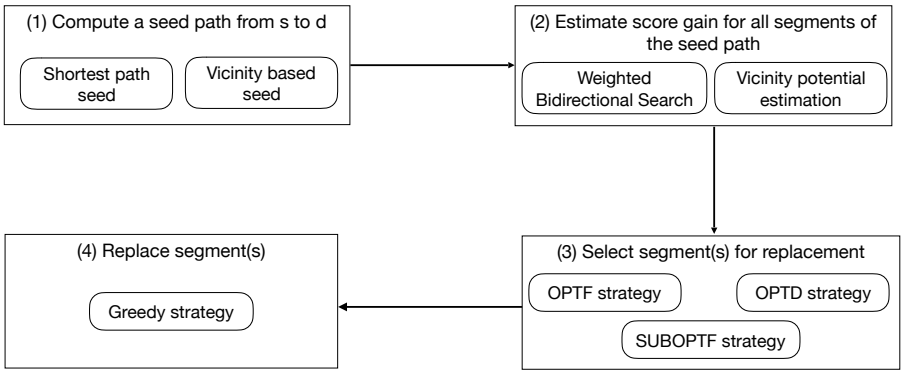


Fig. 4: Proposed framework

3.2 Estimating score gain for all segments of the seed path

The second step of our framework entails estimation of score gain for all segments of the seed path. For this, we propose the following two procedures: (i) the *weighted bi-directional search* and, (ii) Using our novel *Navigability Index* to estimate the score improvement potential of a segment. These procedures are described in detail next. Section 3.2.1 details the weighted bi-directional search procedures. We describe our *Navigability Index* based approach in Section 3.2.2.

3.2.1 ***Weighted Bidirectional Search (WBS)***

Input to the WBS algorithm consists of the following: the input road network, the initial seed path, a specific segment (S_{ij}) of the initial seed path, and a budget value (B'). The goal of the algorithm is to determine a replacement (S'_{ij}) for S_{ij} such that the following criteria are satisfied:

1. The new segment (S'_{ij}) has a higher score value than the input segment.
2. The resultant path from the source to the destination, obtained after replacing S_{ij} with S'_{ij}, is simple (i.e., no loops).
3. The total cost of the new segment S'_{ij} is within B'.

The WBS algorithm employs a bi-directional search to determine the replacement S'_{ij}. The forward search starts from the first vertex of the input segment S_{ij}, whereas the backward search starts from the last vertex of S_{ij}. In each iteration of the WBS algorithm, the forward search determines the *best-successor* of the current tail node (denoted as F_{tail}) of the partial segment it is developing. This is done by processing the out going edges of F_{tail}. In contrast to the forward search, the backward search determines the *best-predecessor* of the current tail node (denoted as B_{tail}) of the partial segment it is developing. This is done by processing the incoming edges at B_{tail}. At the beginning of the algorithm, F_{tail} is initialized to the $S_{ij}.start$, whereas B_{tail} is initialized to $S_{ij}.end$. We now provide details on computation of the best-successor and best-predecessor.

Determining the *best-successor* of F_{tail}: Given the current tail node of the forward search frontier F_{tail}, and the target node (current B_{tail}), the algorithm computes the *Forward Navigability Potential*

(Γ^f in Equation 1) of all the *outgoing neighbors u of* F_{tail}. Following this, the neighbor with the highest Γ^f is designated as the best-successor of F_{tail}. Algorithm 1 details this process.

$$\Gamma^f(u, F_{tail}, B_{tail}) = \frac{1 + score(F_{tail}, u)}{cost(F_{tail}, u) + D_E(u, B_{tail})} \tag{1}$$

In Equation 1, D_E denotes the Euclidean distance[1] between the outgoing neighbor u (of F_{tail}) and the current tail node of backward search B_{tail}. As per Equation 1, neighbors of F_{tail} which are closer to B_{tail} (i.e., lower Euclidean distance), and involve edges with high navigability score values and low cost values, would get a higher value of Γ^f. Algorithm 1 chooses the neighbor which has the highest value of Γ^f.

Algorithm 1 Best-Successor of F_{tail} (G, F_{tail}, B_{tail})

Input: A road network G, F_{tail} and B_{tail}
Output: Best-Successor of F_{tail} and its gamma value (Γ^f_{best})

1: **for all** OutNeighbors u of F_{tail} **do** ▷ only the unvisited Outneighbors
2: Compute $\Gamma^f(u, F_{tail}, B_{tail})$
3: **end for**
4: Best-Successor of $F_{tail} \leftarrow$ OutNeighbor u of F_{tail} with highest Γ^f

Determining the *best-predecessor* of B_{tail}:

Analogous to the computation of Γ^f, the *Backward Navigability Potential* (Γ^b) is computed using Equation 2. Here, we consider the incoming edges of B_{tail}. The neighbor with the highest Γ^b is designated as the best-predecessor of B_{tail}.

$$\Gamma^b(v, B_{tail}, F_{tail}) = \frac{1 + score(v, B_{tail})}{cost(v, B_{tail}) + D_E(v, F_{tail})} \tag{2}$$

Note that if we consider the turn based notions of navigability scores (e.g., a driver preferring route with minimum turns), then Equation 1 and Equation 2 need to be adapted to include these aspects. This changes are detailed in Section 5.

It is important to note that both the forward and the backward searches have "moving targets." In other words, the value of B_{tail} in Equation 1 (and F_{tail} in Equation 2) would change as the algorithm proceeds. To this end, the WBS algorithm employs a design decision to help in quick termination (while not sacrificing on the navigability score). The algorithm first moves the frontier (forward or backward) whose next node to be added [2] comes in with a higher value of Γ. The rationale behind this design decision is the following: a node with higher Γ can imply one or more of the following things: (1) closer to the current target; (2) higher navigability; (3) lower edge cost. Needless to say that all these circumstances are suitable for the needs of the WBS algorithm. After advancing the selected search, WBS re-computes the next node to be added to the other search frontier before it is advanced. For instance, if in the first step the node added by the forward search had higher Γ, then WBS would first advance the forward search frontier by updating its F_{tail} to its best-successor. Following this, best-predecessor of the backward search is re-computed based on the new value of F_{tail}. After that, the backward search is also advanced by updating its B_{tail} to its best-predecessor. Note that in any particular iteration of WBS, both the forward and the backward searches are advanced.

Putting together forward and backward searches: Algorithm 2 puts together our proposed forward and backward searches along with a *termination condition* and a mechanism to *collect candidate solutions* during the course of the execution. We now describe both these aspects of the WBS algorithm.

As one can imagine, a natural termination condition for the WBS algorithm would be meeting of the forward and the backward searches i.e., both the searches color the same vertex. Algorithm 2 uses this as the primary termination condition as indicated in the while loop on line 3 of the pseudo-code. In addition to this, the algorithm also terminates, if at any stage, the total cost of the partial paths constructed so far by the forward and the backward searches happens to be greater than the available budget B'. This termination clause is indicated in lines 23–25 of Algorithm 2.

[1] If the edge costs represent travel-times, then a lower bound on the travel time may be used. This can be computed using the upper speed limit of a road segment.

[2] best-successor in case of forward and best-predecessor in case of backward

Algorithm 2 Weighted Bidirectional Search (G, P, S_{ij}, B')

Input: A road network G, a path P, a segment S_{ij} of P, a budget value $B^{'} = B - P.cost + S_{ij}.cost$, **Output:** A new segment $S^{'}_{ij}$ to replace S_{ij}

1: $F_{tail} \leftarrow S_{ij}.start$, $B_{tail} \leftarrow S_{ij}.end$
2: Mark F_{tail} as colored by forward search and B_{tail} as colored by backward search
3: **while** Forward and backward searches do not color a common node **do**
4: **if** F_{tail} and B_{tail} are connected via a 1-hop or 2-hop path **then**
5: $P_{cand} \leftarrow S_{ij}.start \rightsquigarrow F_{tail} \rightsquigarrow B_{tail} \rightsquigarrow S_{ij}.end$
6: **if** $P_{cand}.cost \leq B^{'}$ **then**
7: Save P_{cand} in the set (Ω) of candidate segments
8: **end if**
9: **end if**
10: Compute Best-Successor of F_{tail} & its Gamma, Γ^{f}_{best} (using Algorithm 1)
11: Compute Best-Predecessor of B_{tail} & its Gamma, Γ^{b}_{best}
12: **if** $\Gamma^{f}_{best} > \Gamma^{b}_{best}$ **then**
13: $F_{tail} \leftarrow$ Best-Successor of F_{tail} ▷ Move the forward search
14: Compute Best-Predecessor of B_{tail}
15: $B_{tail} \leftarrow$ Best-Predecessor of B_{tail} ▷ Move the backward search
16: Mark F_{tail} and B_{tail} as colored by their respective searches
17: **else**
18: $B_{tail} \leftarrow$ Best-Predecessor of B_{tail} ▷ Move the backward search
19: Compute Best-Successor of F_{tail} (using Algorithm 1)
20: $F_{tail} \leftarrow$ Best-Successor of F_{tail} ▷ Move the forward search
21: Mark F_{tail} and B_{tail} as colored by their respective searches
22: **end if**
23: **if** $(S_{ij}.start \rightsquigarrow F_{tail}).cost + (B_{tail} \rightsquigarrow S_{ij}.end).cost > B^{'}$ **then**
24: Break
25: **end if**
26: **end while**
27: **if** Forward and backward searches have colored a common node **then**
28: $P_{cand} \leftarrow$ Reconstructed path between $S_{ij}.start$ and $S_{ij}.end$ by following the *Best-Successors/Best-Predecessors*
29: Save P_{cand} in Ω
30: **end if**
31: Return the segment in Ω with highest navigability score

During the course of the algorithm, it collects several candidate solutions in a set called Ω (lines 4–9 and lines 27–29 in Algorithm 2). At termination, WBS returns the solution having the highest navigability score. The primary reason to collect these candidate solutions being that the forward and the backward searches may not always meet during the course of the algorithm. WBS collects the candidate solutions in the following two ways:

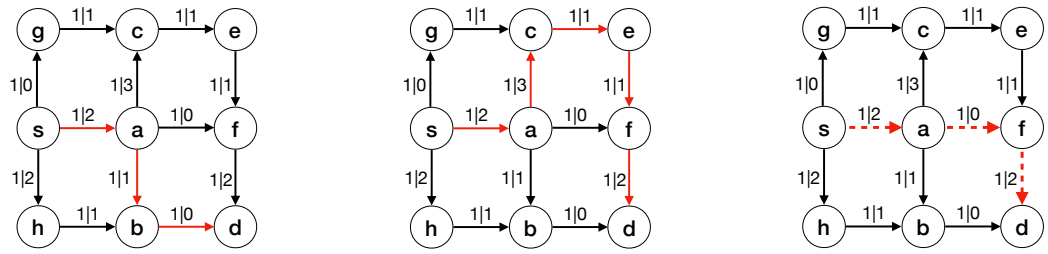
1. At any time during the exploration, if the current F_{tail} and B_{tail} are connected through either a direct edge or two edges then, the segment formed by concatenating this direct edge (or two edges) with the current partial segments formed by forward and backward search is saved as a candidate solution in the set Ω. This is done only if the total cost of this candidate solution is less than the budget B'. This case is illustrated in lines 4–9 of the algorithm.
2. Trivially, if the two search frontiers meet, the partial segments formed by the forward and the backward search are concatenated to create a candidate solution between the first and last nodes of the original segment S_{ij}.

Example: Consider the graph representation of a road network in Figure 5(a). Each edge e_i is labeled with attributes $cost_i|score_i$. We consider the shortest path from s to d marked in red. This path has a score value 3 and a cost value 3. Let the total *budget* value be set as 5. If segment $< (a, b)(b, d) >$ of this path is given as an input to the WBS algorithm, the value of B' is $5 - 1 + 2 = 3$.
Iteration one: $F_{tail} \leftarrow a$ and $B_{tail} \leftarrow d$. Since F_{tail} and B_{tail} are connected via a 2-hop path $(a \rightarrow f \rightarrow d)$ with cost value 2 and score value 2, we add this path to the set Ω. Therefore, $\Omega = \{a \rightarrow f \rightarrow d\}$. The Γ^{f} values are computed as shown in Equations 3, 4 and 5. Therefore, the Best-successor of a is c, and $\Gamma^{f}_{best} = 1.236$.

$$\Gamma^{f}(c, F_{tail} = a, B_{tail} = d) = \frac{1+3}{1+\sqrt{5}} = 1.236 \tag{3}$$

$$\Gamma^{f}(f, F_{tail} = a, B_{tail} = d) = \frac{1+0}{1+1} = 0.5 \tag{4}$$

$$\Gamma^{f}(b, F_{tail} = a, B_{tail} = d) = \frac{1+1}{1+1} = 1 \tag{5}$$



(a) Shortest path (b) Path returned (c) Candidate path

Fig. 5: Example illustrating the working of WBS. Legend: "Cost of edge|Score of edge"

Similarly, the Γ^b values are computed as shown in Equations 6 and 7. Best-successor of d is f, and $\Gamma^b_{best} = 1.5$. Since, $\Gamma^b_{best} > \Gamma^f_{best}$, we move the backward search first, i.e., $B_{tail} \leftarrow f$. The Γ^b values are now recomputed taking into account the new value of B_{tail}. The recomputed values are given as: $\Gamma^f(c, a, f) = (1+3)/(1+\sqrt{2}) = 1.656$, $\Gamma^f(f, a, f) = (1+0)/(1+0) = 1$, and $\Gamma^f(b, a, f) = (1+1)/(1+\sqrt{2}) = 0.828$. $\therefore F_{tail}$ is set to c.

$$\Gamma^b(f, B_{tail} = d, F_{tail} = a) = \frac{1+2}{1+1} = 1.5 \tag{6}$$

$$\Gamma^b(b, B_{tail} = d, F_{tail} = a) = \frac{1+0}{1+1} = 0.5 \tag{7}$$

Iteration two: $F_{tail} \leftarrow c$ and $B_{tail} \leftarrow f$. The Best-successor of both c and f for this iteration is e. Both searches color the same node in this case. The reconstructed path ($a \rightarrow c \rightarrow e \rightarrow f \rightarrow d$) with cost value 3, is added to Ω, i.e. $\Omega = \{a \rightarrow f \rightarrow d, a \rightarrow c \rightarrow e \rightarrow f \rightarrow d\}$. The algorithm terminates, and the segment with the highest Γ value in Ω, i.e. $a \rightarrow c \rightarrow e \rightarrow f \rightarrow d$, is returned as the solution.

Time complexity analysis: The number of vertices visited by WBS is $O(|V|)$. No vertex is visited twice, and each vertex v that gets colored leads to computation of $degree(v)$ number of navigability potential values. Thus, the time complexity of WBS is $O(|E|)$. Here, $|V|$ denotes the number of nodes and $|E|$ denotes the number of edges in the input graph.

3.2.2 ***Estimating the score improvement potential using Navigability Index***

Unlike the WBS algorithm described in the previous section, the approach proposed in this section *does not involve computing an actual replacement* (S'_{ij}) for estimating the potential for getting a higher navigability score (within the budget) when the given segment S_{ij} is replaced. Instead, we estimate the potential by computing a novel metric called as the *Vicinity Potential* (VP) for segment S_{ij}. A higher vicinity potential implies greater potential in finding a replacement which has higher navigability score (than S_{ij}). The Vicinity Potential of a segment is computed using our proposed *Navigability Index*. Details of this approach are presented next.

Given a segment S_{ij} and a budget value B', the vicinity of S_{ij} is defined as the area bounded by the ellipse with focal points as $S_{ij}.start$ and $S_{ij}.end$. The length of its major axis is B'. The properties of an ellipse allow us to claim the following: any path between $S_{ij}.start$ and $S_{ij}.end$ of length $\leq B'$ would not include any edge lying completely outside or intersecting this ellipse.

Vicinity Potential (VP) of a segment: The VP value of a segment S_{ij} is defined as the average of navigability scores of all the navigable edges lying in the *vicinity* of S_{ij}. Equation 8 presents this formally[3]. Recall that only the edges having a navigability score value > 0 are referred to as navigable edges (denoted in Equation 8 as $'ne'$).

$$VP(S_{ij}, B') = \frac{\sum_{ne \in ellipse(S_{ij}.start, S_{ij}.end, B')} ne.score}{Count(ne \in ellipse(S_{ij}.start, S_{ij}.end, B'))} \tag{8}$$

This definition is based on the intuition that more the number of highly navigable edges in a segment's vicinity, the higher would be the probability of finding a segment to replace it. We now introduce an indexing structure, the Navigability Index, which we use to estimate the VP value of a segment.

Navigability Index for computing the VP values: To compute the VP value of a segment, one needs to obtain the set of edges that are contained in its vicinity. This computation can be made efficient

[3] If edge costs represent travel-times, then the travel-time based budget can be converted to a distance based budget using the upper speed limit of a road segment.

using our *navigability index*. Navigability Index is similar to a regular spatial grid. Each cell in this index stores two numeric values: *sum* and *count*. The *sum* value of cell (x, y) is set to the sum of scores of all navigable edges contained in the rectangle bounded between cells $(0, 0)$ and (x, y). The *count* value of cell (x, y) is set to the number of navigable edges contained in this rectangle. Given these, the *sum* and *count* values of any rectangle in the grid bounded between cells (i, j) and (m, n) can be computed using Equations 9 and 10. Here, $sum(x, y)$ and $count(x, y)$ respectively denote the *sum* and *count* values of cell (x, y).

$$Sum = sum(m, n) - sum(i - 1, n) - sum(m, j - 1) + sum(i - 1, j - 1) \tag{9}$$

$$Count = count(m, n) - count(i - 1, n) - count(m, j - 1) + count(i - 1, j - 1) \tag{10}$$

The proposed index structure computes the VP value for any segment in $O(1)$ index lookups, irrespective of the order of the grid index used. This is done as follows: to compute the VP value of S_{ij}, we take the spatial coordinates of $S_{ij}.start$ and $S_{ij}.end$ as the foci of the ellipse with the length of major axis= B'. Next, we compute the grid aligned minimum bounding rectangle (MBR) of this ellipse. The VP value of S_{ij} can then be computed by plugging the bottom-left (i, j) and upper-right (m, n) coordinates of this MBR in Equations 9 and 10. This makes the computation much faster. The idea of this index structure was inspired by the work in Aly et al.[1].

3.3 **Selecting segment(s) for replacement**

Recall that the third step of our framework entails selection of segment(s) for replacement. In this section, we describe three strategies to select a set of segments for replacement from the seed path. Two of these strategies (OPTF and SUBOPTF covered in Section 3.3.1 and Section 3.3.2) require the *score-gain* (*sgain*) and *cost-gain* (*cgain*) values for each of the segments S_{xy} of the initial seed path. Here, *sgain* of a segment S_{xy} is defined as the difference in the navigability score values of S_{xy} and its replacement S'_{xy}: $S_{xy}.sgain = S'_{xy}.score - S_{xy}.score$. *cgain* is also defined in an analogous way: $S^{'}_{xy}.cgain = S'_{xy}.cost - S_{xy}.cost$. As one can imagine, for using *OPTF* and *SUBOPTF* strategies in module 2 (refer overall framework in Figure 4), we need to use only the weighted bi-directional search algorithm in module 1. This is because, the navigability index based approach (discussed in Section 3.2.2) does not give us the *cgain* values while computing the score gain estimates.

In contrast to the *OPTF* and the *SUBOPTF* strategies, our third strategy *OPTD* requires only the *sgain* values for each of the segments S_{xy} of the initial seed path. Thus, we can use both the weighted bi-directional search and navigability index based approach for determining the potential score values of all the segments in the initial seed path.

The rest of this section is organized as follows. We first describe the overall problem of selecting segments for replacement and illustrate its challenges. Following this, we describe our proposed strategies *OPTF*, *SUBOPTF* and *OPTD* in Section 3.3.3, Section 3.3.2 and Section 3.3.3 respectively.

Selecting a set of segments to replace from a given seed path is non-trivial. This is because of the following three reasons: (a) segments chosen for replacement may have common edges, (b) budget constraint and, (c) replacements of the chosen segments may overlap.

Example: As an instance of challenges (a) and (b), refer to Table 1. The table illustrates a sample scenario on replacing segments of an initial seed path $< e_1\, e_2\, e_3\, e_4 >$. The ten possible segments of this path are shown along with their sample *sgain* and *cgain* values. Here, an entry (0,0) implies that no solution was found by WBS for that segment. In this example, we can either replace the segment $< e_2\, e_3 >$ or the segment $< e_1\, e_2\, e_3 >$. Replacing both would not be possible as e_2 is common to both segments.

Table 1: Set of all segments of path $< e_1\, e_2\, e_3\, e_4 >$

Segment	$(sgain, cgain)$	**Segment**	$(sgain, cgain)$
$< e_1 >$	(7,5)	$< e_2\, e_3 >$	(16,10)
$< e_2 >$	(7,5)	$< e_3\, e_4 >$	(15,10)
$< e_3 >$	(5,8)	$< e_1\, e_2\, e_3 >$	(15,18)
$< e_4 >$	(0,0)	$< e_2\, e_3\, e_4 >$	(0,0)
$< e_1\, e_2 >$	(15,12)	$< e_1\, e_2\, e_3\, e_4 >$	(10,15)

In addition, if the allowed overhead was 15 then, the segments $< e_1 >$ and $< e_2\, e_3 >$ can be collectively replaced. Whereas, segments $< e_1\, e_2 >$ and $< e_3 >$ cannot be collectively replaced, as their *cgain* value is 20. We now formalize this idea using the concept of a feasible and disjoint set of segments corresponding to a path.

Feasible and Disjoint Set of Segments (FDSS): FDSS of a given path P, is a set of segments such that no two segments in the set share an edge, and $P.cost + \sum_{\forall S \in FDSS} S.cgain \leq B$. In our previous example, $< e_1 >$ and $< e_2\, e_3 >$ form an FDSS. As expected, the central goal would be to determine an FDSS which results in the highest increase in navigability score.
Computational structure of the FDSS problem: The FDSS problem can be seen as an advanced version of the 0/1 knapsack problem where certain items are not allowed together (i.e., segments having common edges). For solving FDSS, one can first enumerate all sets of disjoint segments and then, run an instance of 0/1 knapsack on each set of disjoint segments. Basically, each set is generated by cutting a path at k unique locations ($0 \leq k \leq$ #edges in the path - 1). However, this technique would have two computational challenges: (a) knapsack problem is known to be NP-hard, (b) a path with l edges would have 2^{l-1} unique sets of disjoint segments. We now describe our three proposed strategies for segment selection.

3.3.1 ***Optimal strategy for FDSS selection (OPTF)***

Our first strategy for segment selection computes the optimal solution to the FDSS problem. A careful examination of the FDSS problem reveals that solving for the optimal solution involves overlapping subproblems. We use this insight in our DP based solution to compute the optimal FDSS.
Selection of FDSS: We compute the solution in a bottom up manner starting with subproblems of size 1, where the size of a subproblem represents the number of edges in the corresponding segment. We denote the set of feasible solutions for the segment having edges numbered i through j as F_{ij}. This set has pairs of *sgain* and *cgain* values corresponding to all possible ways of replacing a segment within the given budget. The sets representing solutions of subproblems of size 1 are initialized as $F_{ii} = \{(S'_{ii}.sgain, S'_{ii}.cgain)\}$. $F_{(i+1)i}$ is initialized as $\{(0,0)\}$. Equation 11 gives the set of feasible solutions for subproblems of size $>$ 1.

$$F_{ij}(i<j) = \{(s_x, c_x) : s_x = S'_{ik}.sgain + s_y \text{ and } c_x = S'_{ik}.cgain + c_y \,\forall (s_y, c_y) \in F_{(k+1)j}, c_x \leq B\} \cup \{(S'_{ik}.sgain, S'_{ik}.cgain)\} \cup F_{(k+1)j}, \text{where } i \leq k \leq j \quad (11)$$

Here, the variable k denotes the edge number after which the first cut would be placed. For each possible first cut, we check if combination of S'_{ik} (solution found by WBS for segment towards the left of the cut) and solutions in $F_{(k+1)j}$ (set of feasible solutions computed by the algorithm for segment towards the right of the cut) are feasible. If this combination is feasible, the new $(sgain, cgain)$ pair is added to the set F_{ij}. Additionally, both S'_{ik} and solutions in $F_{(k+1)j}$ are added to F_{ij}. The option for having no cut is considered when $k = j$. The optimal solution for the initial seed path containing l edges (numbered 1 through l) is given by Equation 12.

$$FDSS_{opt} = \{(s_x, c_x) : (s_x, c_x) \in F_{1l}, c_x \leq c_i, s_x \geq s_i \,\forall (s_i, c_i) \in F_{1l}\} \quad (12)$$

Example: Consider the *sgain* and *cgain* values given in Table 1 for path $< e_1, e_2, e_3, e_4 >$. The set of feasible solutions for subproblems of size 1 are initialized as: $F_{11} = (7,5), F_{22} = (7,5), F_{33} = (5,8)$ and $F_{44} = (0,0)$ (refer Figure 6). To compute F_{12}, we would consider the first cut being made either after edge e_1 ($k = 1$) or after edge e_2 (corresponds to no cut, $k = 2$). For $k = 1$, we check if combination of S'_{11} and solutions in F_{22} are feasible. Recall that S'_{11} refers to the solution found by WBS for segment S_{11} in Step-1 of the framework (as shown in Table 1). The new $(sgain, cgain)$ pair is given as (7+7,5+5)=(14,10). Since this pair is feasible, we add this to F_{12} along with the pair (7,5). For $k = 2$, we check if S'_{11} and solutions in F_{32} are feasible. Since F_{32} is initialized as (0,0), we add to F_{12} the pair (15,12). Thus, the set of FDSS for subproblem $< e_1, e_2 >$ is $\{\{< e_1 >\}, \{< e_2 >\}, \{< e_1 >, < e_2 >\}, \{< e_1, e_2 >\}\}$. Proceeding in this manner we compute F_{14}, which gives us the optimal solution as (29,20).

i \ j	1	2	3	4
1	(7,5)	{(14,10),(15,12),(7,5)}	{(19,18),(23,15),(14,10), (12,13),(20,20),(15,18)}	{(19,18),(23,15),(29,20), (20,20),(15,18),(10,15)}
2		(7,5)	{(12,13),(16,10),(7,5), (5,8)}	{(12,13),(16,10),(22,15)}
3			(5,8)	{(5,8),(15,10)}
4				(0,0)

Fig. 6: Illustration of example for OPTF and SUBOPTF strategies

This method for optimal FDSS selection computes solutions to the distinct subproblems only once. However, this computation is exponential in terms of both space and time. Lemma 2 proves this claim. Proof of optimality of the FDSS selected is given in Lemma 1.

Lemma 1 *OPTF computes the optimal FDSS.*

Proof. We prove this claim through contradiction. Let $FDSS_{opt} = (s_o, c_o)$ be reported as the optimal solution by the algorithm. We assume that there exists a way of replacing segment S_{xy}, represented by the pair (s_k, c_k), such that $s_k > s_o$ and $c_k \leq B$. Clearly, (s_k, c_k) is the correct optimal solution. Since (s_k, c_k) is a feasible way of replacing S_{xy}, the definition of F_{xy} implies that $(s_k, c_k) \in F_{xy}$. This further implies that $(s_k, c_k) \in F_{1l}$. $\therefore$ by Equation 4, (s_k, c_k) would be computed as the optimal solution. This contradicts our assumption that (s_o, c_o) is reported as the optimal solution. □

Lemma 2 *For a path with l edges, OPTF takes $O(3^l)$ time and space to select the optimal FDSS.*

Proof. The number of distinct subproblems of size $i = l-i+1$. The total number of distinct subproblems $= \sum_{i=1}^{l}(l-i+1) = \theta(l^2)$. Let us now consider the time it takes to compute the set F for each subproblem. Each element of F can be computed in $O(1)$ time. Let s_i denote the maximum size possible of set F_{xy}, where $i = y-x+1$, i.e., s_i denotes the maximum possible number of feasible solutions for some subproblem of size i. For $i = 1$, we have $s_1 = 1$. For $i > 1$, we have:
$s_i = \sum_{k=1}^{i-1}(2 * s_k + 1) + 1$
$= (2 * s_{i-1} + 1) + (\sum_{k=1}^{i-2}(2 * s_k + 1) + 1)$
$= (2 * s_{i-1} + 1) + s_{i-1} = 3 * s_{i-1} + 1$
Thus, the size of F increases exponentially with the size of the subproblems and $|F_{1l}|$ is $O(3^l)$. □

3.3.2 ***SUBOPTF: Sub-Optimal strategy for FDSS selection***

Since our first strategy for segment selection (OPTF) consumes an exponential time and space, it is not practical to use it for real-time applications. Therefore, in our second strategy (SUBOPTF), we drop the constraint of optimality of the solution. First, we explain the method used for selection of FDSS by this algorithm. We select a set of segments in a more efficient way than the OPTF strategy. The FDSS selected may not be the optimal solution but the total score gain of the computed FDSS is guaranteed to be greater than or equal to that of the segment with the highest value of *sgain*.

Selection of FDSS: Here, the set of feasible solutions for a subproblem is replaced by a single, local optimal solution. The solutions of subproblems of size 1 are initialized as $f_{ii} = \{(S_{ii}^{'}.sgain, S_{ii}^{'}.cgain)\}$, and $f_{(i+1)i}$ is initialized as $\{(0,0)\}$. The subproblems are then solved in increasing order of size using Equation 13. The function *max* defined in Equation 14, computes the local optimal from a set of feasible solutions S. The final solution is then given as f_{1l}.

$$f_{ij}(i<j) = max\{\{(s_x, c_x) : s_x = S_{ik}^{'}.sgain + s_y \text{ and } c_x = S_{ik}^{'}.cgain + c_y \,\forall (s_y, c_y) \in f_{(k+1)j}, c_x \leq Budget\} \cup \{(S_{ik}^{'}.sgain, S_{ik}^{'}.cgain)\} \cup \{f_{(k+1)j}\}, \text{where } i \leq k \leq j\} \quad (13)$$

$$max(S) = (s_x, c_x) \text{ where } (s_x, c_x) \in S, s_x \geq s_y \text{ and } c_x \leq c_y \forall (s_y, c_y) \in S \quad (14)$$

Example: The values marked in red in Figure 6 show the local optimal solutions corresponding to SUBOPTF for the example given in Table 1. The final solution in this case too is (29,20).

Lemma 3 *For a path with l edges, SUBOPTF takes $O(l^3)$ time and consumes $\theta(l^2)$ space for FDSS selection.*

Proof. Since, the total number of distinct subproblems is $\theta(l^2)$, and each subproblem has a single $(sgain, cgain)$ pair as its solution, the algorithm consumes a total of $\theta(l^2)$ space. Let us now consider the time it takes to compute the solution for each subproblem. The maximum number of cuts possible for any subproblem is l. For each possible cut in a given subproblem S_{ij}, a maximum of three elements are added to the set S which is passed as input to the *max* function. For the value of cut variable as k, these three elements include S'_{ik}, f_{kj} and S'_{ij}. Thus, the size of input to *max* for a subproblem is $O(l)$, which can be computed in $O(l)$ time. This gives a total time of $O(l^3)$ for the SUBOPTF segment selection strategy. □

Lemma 4 *The total score gain of the FDSS selected by SUBOPTF is guaranteed to be no less than that of the segment with the highest sgain value.*

Proof. Let S_{xy} be the segment with the highest value of *sgain*. Consider the solution $f_{xy} = (s, c)$ computed by SUBOPTF corresponding to the segment S_{xy}. The two solutions will be different if either of the following two conditions holds true:
1. $s > S'_{xy}.sgain$
2. $s = S'_{xy}.sgain$ and $c \leq S'_{xy}.cgain$
In either case $s \geq S'_{xy}.sgain$. But since $S'_{xy}.sgain$ is the highest *sgain* value, we have that $s = S'_{xy}.sgain$. $\therefore f_{xy} = S'_{xy}$. This would imply that the solution f_{1l} has an *sgain* value no less than that of S'_{xy}. □

3.3.3 ***OPTD: Optimal strategy for DSS selection***

In our third segment selection strategy, we drop the feasibility constraint from the FDSS definition to define what we call as a Disjoint Set of Segments:
Disjoint Set of Segments (DSS): DSS of a given path is a set of segments such that no two segments in the set share an edge.

The reason for dropping the feasibility constraint becomes clear in Section 4 on proposed algorithms. The OPTD strategy for segment selection is a DP based algorithm that selects the optimal DSS. Input to this algorithm is the set of *sgain* values for all segments of the initial seed path, that are computed using WBS. We observe that the DSS problem exhibits the optimal substructure property. We exploit this property to design a DP based solution to compute the optimal DSS which takes $\theta(l^3)$ time, and consumes $\theta(l^2)$ space (for a seed path with l edges). The central idea in this DP formulation is to consider the DSS problem analogous to that of the rod cutting problem. Our initial seed path P becomes the "rod" being cut. We aim to "break up this rod" into pieces such that the total score gain obtained by replacing the pieces formed is maximum.

Algorithm 3 DP algorithm for computing optimal DSS

Input: *sgain* values for all segments of seed path, **Output:** Optimal DSS

```
1: for spsize = 2 to l do                                   ▷ Subproblem size
2:     for i = 2 to l − spsize + 1 do                       ▷ First edge of segment
3:         j ← i + spsize − 1                               ▷ Last edge of segment
4:         f_ij ← 0                                         ▷ Initializing optimal solution for S_ij
5:         for k = i to l do                                ▷ Edge after which first cut is placed
6:             if S_ik.sgain + f_(k+1)j > f_ij then
7:                 f_ij ← S_ik.sgain + f_(k+1)j
8:             end if
9:         end for
10:    end for
11: end for
```

Equation 15 represents the underlying recurrence equation for our DP based solution. Here, f_{ij} denotes the optimal solution for a sub-problem having edges numbered i through j. The optimal solution for the initial seed path containing l edges is given by f_{1l}. Variable k denotes the edge after which the first cut in the optimal solution is assumed to be placed. For each possible first cut (i.e. for each value of k), we check the sum of *sgain* values of S_{ik} (solution found by WBS for segment towards the left of the cut) and $f_{(k+1)j}$. $f_{(k+1)j}$ is the optimal break-up computed by this algorithm for the sub-problem having edges $(k+1)$ through j. The option for having no cut, is considered when $k = j$. The base conditions for this recurrence equation are: $f_{ii} = S_{ii}.sgain$ (subproblem of size 1) and $f_{(i+1)i} = 0$. Algorithm 3 presents a bottom up procedure for computing this recurrence equation.

$$f_{ij} = \max_{i \leq k \leq j} (S_{ik}.sgain + f_{(k+1)j}) \tag{15}$$

Example: Figure 7 shows the computation of the optimal DSS for the *sgain* values given in Table 1. The optimal DSS in this case is $\{< e_1, e_2 >, < e_3, e_4 >\}$, which has a total *sgain* value 30.

Lemma 5 *For a path with l edges, MSR(OPTD) takes $O(l^3)$ time and consumes $\theta(l^2)$ space to select the optimal DSS.*

i \ j	1	2	3	4
1	7	*max*{7+7,15}= *max*{14,15}= 15	*max*{7+16,15+5,15}= *max*{23,20,15}= 23	*max*{7+22,15+15,15+0,10 }= *max*{29,30,15,0}= 30
2		7	*max*{7+5,16}= *max*{12,16} = 16	*max*{7+15,16+0,0}= *max*{22,16,0}= 22
3			5	*max*{5+0,15}= *max*{5,15} = 15
4				0

Fig. 7: Illustration of example for OPTD strategy

3.4 **Replacing segments**

The final step in our framework entails replacement of the selected segments from the seed path. After a set of segments has been selected, we attempt to replace all segments in the FDSS or DSS returned. For this, we follow a greedy approach. We pick segments from the FDSS/DSS in decreasing order of their *sgain* values. This is done in order to maximize the score gain, since it may not be possible to replace all segments of the seed path. As the segments are picked in decreasing order of their *sgain* values, their replacements are computed using WBS, and the path is updated after each computation of replacement.

4 Proposed algorithms

In this section, we explain five algorithms that use solution techniques described for the modules of the proposed framework (Figure 4). Note that more algorithms can be designed using a different combination of the solution techniques explained for each module.

4.1 **MSR(OPTF): Multiple Segment Replacement algorithm using OPTF strategy for FDSS selection**

Algorithm 4 presents the steps in MSR(OPTF). After the seed path has been computed, the *sgain* and *cgain* values are computed for each segment of the seed. Next, we compute the optimal FDSS, and then replace the segments in the FDSS returned. Note that, in step 4 we compute the replacements again using WBS. This is done to ensure that the path remains simple when it is updated, since the replacements computed in step 2 may not be disjoint.

Algorithm 4 MSR(OPTF)

Input: Road network G, source s, destination d, Budget B **Output:** Path P from s to d

1: Compute the initial seed path P from s to d
2: Estimate the *sgain* and *cgain* values for each segment S of P by invoking WBS on S
3: Compute the optimal FDSS for P using the OPTF strategy
4: Replace the segments in the FDSS returned

Time complexity analysis: For the time complexity analysis of our algorithms, we do not consider the complexity of initial seed path computation.

The second step of MSR(OPTF) estimates score gain values for all segments of the seed path using WBS. For a path with l edges, this takes $O(l^2|E|)$ time since the total number of segments in the path is $\theta(l^2)$. The optimal FDSS is then computed in the third step using the OPTF strategy in $\theta(3^l)$ time (by Lemma 2). Step four involves updating the path by calling WBS on all segments in the optimal FDSS. Since the maximum number of segments in an FDSS can be l, this step requires $O(l|E|)$ time. This gives a time complexity of $O(l^2|E| + 3^l)$ for MSR(OPTF).

4.2 **MSR(SUBOPTF): Multiple Segment Replacement algorithm using SUBOPTF strategy for FDSS selection**

The steps in MSR(SUBOPTF) are the same as that of Algorithm 4 for MSR(OPTF) except the method used for selection of FDSS (Step 3). Here, we use the SUBOPTF strategy to select the FDSS.

Time complexity analysis: The time complexities of the second and fourth steps of MSR(SUBOPTF) are $O(l^2|E|)$ and $O(l|E|)$ respectively (same as that of MSR(OPTF)). The time taken by the third step of this algorithm, that entails selection of an FDSS, is $O(l^3)$ (by Lemma 3). Thus, the time complexity of MSR(SUBOPTF) is $O(l^2|E|)$.

As the segments in the selected FDSS are replaced in the fourth step, the feasibility constraint may no longer hold for the replacements computed earlier. This situation arises when the replacements computed earlier share edge(s), and thus the new replacement returned by WBS in the fourth step changes in order to ensure that the final path has no loops. It is for this reason that we drop the feasibility constraint for computing the set of segments to be replaced in our next algorithm that uses the OPTD strategy.

4.3 MSR(OPTD): Multiple Segment Replacement algorithm using OPTD strategy for DSS selection

The working of MSR(OPTD) differs from that of MSR(SUBOPTF) in its third step. The third step of MSR(OPTD) entails computation of the optimal DSS using the OPTD strategy. The other steps of the algorithm remain the same.

Time complexity analysis: The time complexities of all four steps of MSR(OPTD) are the same as that of MSR(SUBOPTF). This gives a complexity of $O(l^2|E|)$ for this algorithm.

4.4 VA-MSR(OPTD): Vicinity Aware Multiple Segment replacement algorithm using OPTD strategy for DSS selection

Recall that MSR(OPTD) computes the *sgain* values for all the possible segments of the initial seed path. In other words, given an initial seed path with l edges, MSR(OPTD) calls the WBS algorithm $\theta(l^2)$ times to get the score gain value of each of the possible segments. Following which, it determines the optimal set of segments (DSS) for replacement. Invoking $\theta(l^2)$ instances of WBS may not be computationally scalable. To this end, this algorithm considers the VP value of each segment. The VP values serve as a proxy to the *sgain* values of the segments in the initial seed path.

Algorithm 5 VA-MSR(OPTD)

Input: Road network G, source s, destination d, Budget B **Output:** Path P from s to d

1: Compute the initial seed path P from s to d
2: Estimate the *sgain* value for each segment S of P by computing the VP value of S
3: Compute the optimal DSS for P using the OPTD strategy
4: Replace the segments in the DSS returned

Algorithm 5 presents the pseudocode for VA-MSR(OPTD). Given the initial seed path, the second step of VA-MSR(OPTD) involves computing the VP values of all segments of the seed path. In the third step, the optimal DSS is computed based on the VP values of segments. Next, the WBS algorithm is invoked to determine the actual replacements for the segments in the optimal DSS.

Time complexity analysis: The time complexity of the second step of VA-MSR(OPTD) is $\theta(l^2)$, since VP values are computed for $\theta(l^2)$ segments, and each such computation takes $O(1)$ time. The complexity of the remaining steps is the same as the steps of MSR(OPTD). Thus, the time complexity of VA-MSR(OPTD) is $O(l|E| + l^3)$.

4.5 MSR(OPTD)-KSEEDS: Multiple Segment Replacement algorithm using Vicinity Based Seed for seed path selection and the OPTD strategy for DSS selection

The comparative performance of the four proposed algorithms described in Sections 4.1 through 4.4, against the performance of the related work, indicated to us the relevance of the seed selection step. Our results indicated that the seed selection step plays an important role in determining the solution quality. Thus, in the MSR(OPTD)-KSEEDS algorithm we select K seeds, and then apply the improvement algorithm of MSR(OPTD) on all the K seeds. These seeds are selected using the vicinity based seeds approach. Finally, we return the solution having the highest navigability score.

Time complexity analysis: The complexity of MSR(OPTD)-KSEEDS algorithm is K times that of the MSR(OPTD) algorithm, i.e., $O(Kl^2|E|)$.

5 Generalization of proposed algorithms for Turn based navigability

In this section, we show how the proposed algorithms can be modified to consider turn based notion of navigability. We consider the application where a user would like to maximize the number of easily identifiable sites on the road segments, and minimize the number of left and right turns. For this, the edges in the graph can be assigned scores based on the number and type of prominently visible sites on the corresponding road segments.

To model a turn, the node representing a road intersection in the graph can be expanded into multiple nodes. Edges can be added between these nodes to represent the types of turns that can be taken from each node, i.e. left, right and straight. Figure 8 depicts this scenario. Here, the node a has been expanded into nodes $a1$ through $a8$. A total of twelve edges have been added to model all the turns possible at the road intersection a. For example, the right turn from edge ca to edge ab, is now modelled as edge $a7a4$ in the expanded graph. Using this concept, we can now assign scores to the new edges in the expanded graph that represent the scores corresponding to the turns. The scores can be assigned based on some fixed order of preference, say 15, 10 and 5 for straight, right and left turns respectively.

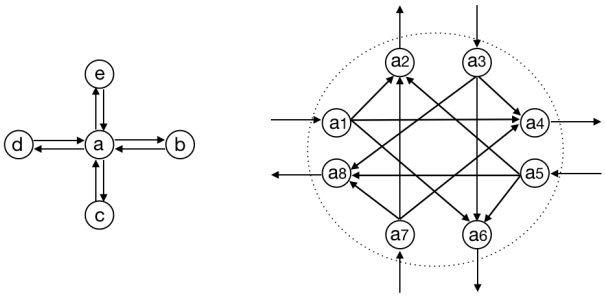


Fig. 8: Expansion of node a to model the turns at road intersection a

In our proposed system, turn based notion of navigability can be incorporated in the WBS algorithm (refer Section 3.2.1). This is done by adapting the expressions (Equation 1 and Equation 2) used for computing the forward and backward navigability potential of the neighbors in the WBS algorithm.

Equation 16 and Equation 17 detail the adapted versions of the Equation 1 and Equation 2 to consider the turn based notions of navigability. Here, F_{tail} and B_{tail} denote the current tail nodes of the forward and backward search frontiers respectively. $F^{'}_{tail}$ and $B^{'}_{tail}$ denote the tail nodes of the forward and backward search frontiers in the immediate previous iteration of WBS. u and v denote the outgoing neighbors of F_{tail} and B_{tail} respectively. The function $turnscore(x, y, z)$ gives the fixed score based on the type of turn traversed as one goes from edge (x, y) to edge (y, z). For the experiments we computed the type of turns on the fly, and compared the Γ values based on the modified definitions.

$$\Gamma^f(F^{'}_{tail}, F_{tail}, u, B_{tail}) = \frac{1 + score(F_{tail}, u) + turnscore(F^{'}_{tail}, F_{tail}, u)}{D_N(F_{tail}, u) + D_E(u, B_{tail})} \tag{16}$$

$$\Gamma^b(B^{'}_{tail}, B_{tail}, v, F_{tail}) = \frac{1 + score(v, B_{tail}) + turnscore(v, B_{tail}, B^{'}_{tail})}{D_N(v, B_{tail}) + D_E(v, F_{tail})} \tag{17}$$

6 Dominance zones of proposed algorithms

In this section, we describe the dominance zones of the MSR(OPTD) , MSR(OPTD)-KSEEDS and MSR(SUBOPTF) algorithms. To explain the comparative usage of the MSR(OPTD) and MSR(SUBOPTF) algorithms, we consider the example in Section 3.3 for the path $< e_1e_2e_3e_4 >$. The solution given by the SUBOPTF strategy for this example is $< e_1 >< e_2 >< e_3e_4 >$, which has a *cgain* value of 20 and *sgain* value of 29. The solution given by the OPTD strategy for this example is $< e_1e_2 >< e_3e_4 >$, with *cgain* and *sgain* values as 22 and 30 respectively. The *sgain* values suggests that the OPTD strategy outperforms the SUBOPTF strategy in this case. However, this may not hold. Let's consider two cases, i) Budget = 25, ii) Budget = 20. In the first case, OPTD clearly outperforms SUBOPTF. However, in the latter case SUBOPTF outperforms OPTD as both segments in the SUBOPTF solution cannot be replaced collectively because of the budget constraint. Therefore, in case of tight budget constraints the MSR(SUBOPTF) algorithm is more likely to return a better solution.

A case where SUBOPTF would outperform OPTD in terms of running time is when the replacements computed for the different segments do not overlap with each other. In this case, we can skip the fourth step of the algorithm that involves calling WBS for the segments selected for replacement. Here, the solutions computed in step two would make a feasible solution.

The MSR(OPTD)-KSEEDS algorithm computes K seed paths, applies the MSR(OPTD) algorithm to each seed path, and returns the best result. It is therefore more likely to return a more navigable path, but it would also take a higher computation time. Depending on the constraints on the budget and the running time, one can select the algorithm that has a higher chance of giving a better solution in that setting.

7 Experimental evaluation

Performance of the proposed algorithms was evaluated through experiments on four real-road networks of different sizes. We first describe our datasets, followed by a brief on the algorithms implemented and the experimental setup.

Datasets: All datasets constituted directed spatial graphs with vertices as road intersections and edges as road segments (shown in Table 2). Datasets 1 and 3 were obtained from OpenStreetMap[4]. Dataset 2 can be accessed from this url[5], and dataset 4 from this url[6]. Cost of each edge corresponded to the metric of distance. Navigability score values of edges were generated synthetically. The experiments were performed in two settings, using both random and normalized distributions to assign score values to the edges. For both settings, the navigable edges were distributed uniformly across the space. The edges were assigned scores in the range [0,15]. A fixed percentage of edges were assigned a non-zero score and the remaining edges were assigned a score value of zero.

Table 2: Description of datasets

Dataset	Place	Vertices	Edges
1	Delhi	11,399	24,943
2	California	21,048	39,976
3	Beijing	55,545	95,285
4	New York	2,64,346	7,33,846

Baseline algorithm: Given the score gain values of all segments of the initial seed path, a naive algorithm would be to replace the segment with the highest *sgain* value. We call this algorithm the Single Segment Replacement algorithm (SSR), and use it as the baseline algorithm.

Algorithms implemented: We implemented four of the five proposed algorithms for MNP: MSR(SUBOPTF), MSR(OPTD), VA-MSR(OPTD) and MSR(OPTD)-KSEEDS. MSR(OPTF) was not implemented since it explores an exponential search space for selecting the optimal FDSS. Further, two algorithms from the related work were implemented, modified versions of ILS(CEI)[19] and ARSC[16], hereafter denoted as ILS(CEI)* and ARSC*. The adaptation done in ILS(CEI)* is twofold. First, it yields a simple path as the solution. Second, unlike the original algorithm which performs the same iteration until some time threshold, our adaptation performs a single iteration of the original ILS(CEI) algorithm and then terminates, which is similar to the working of the proposed algorithms.

ARSC* adapts its original algorithm to include the budget constraint of MNP. The weights of the edges are considered as inverse of navigability score values. The objective function of the algorithm is: $Minimize \sum_{e_i \in path} \frac{1}{score_i}$, subject to $\sum_{e_i \in path} cost_i \leq budget$. The algorithm prunes the search space using lower bound estimates on the attribute score, computed by preprocessing the input graph to create a reference node embedding. To reduce the search space further, we used the euclidean distance to destination as a lower bound on cost, and ignored vertices for which this estimate exceeded the budget. Datasets 1, 2 and 3 were processed to create the reference node embeddings with 40 reference nodes, which were selected at random initially and later set as the medoids of their cluster. Given the size of Dataset 4, we did not create a reference node embedding for this dataset due to a long pre-computation time.

Experimental setup: All algorithms were implemented in Java language on a machine with a 2.6 GHz processor and a 32 GB RAM. The shortest paths were computed using the A* algorithm. For the VA-

[4] `https://www.openstreetmap.org`

[5] `https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm`

[6] `http://users.diag.uniroma1.it/challenge9/download.shtml`

MSR(OPTD) algorithm, the orders of the grid indices built for datasets 1, 2, 3 and 4 were 300 × 300, 500 × 500, 1000 × 1000 and 200 × 200 respectively. The idea was to create 1km × 1km grid cells in each navigability index. To study the effects of change in the available overhead, we set the overhead as a fraction of the length of the shortest path. An overhead of x% implies that the total length allowed for the resultant path is: shortest path length + x% of the shortest path length. The statistics reported for an overhead of 0% represent the values for the shortest path. We report the average statistics for 100 random query instances for all three datasets.

The rest of this section is organized as follows: In section 7.1, we analyze the comparative performance of the proposed algorithms. Section 7.2 compares the performance of the proposed algorithms against that of the related work. In section 7.3, we show the results for the turn based notions of navigability.

7.1 **Comparative analysis of proposed algorithms**

The experiments in this section were performed on datasets for which the score values on the edges were generated in the random distribution setting.

Effect of increase in overhead on the score of a path: This experiment was conducted to evaluate the change in the score value of a path as the overhead value is varied. Figures 9 and 10 show the results for different percentage distributions of navigable edges. The score value corresponding to overhead value of 0% marks the navigability score of the shortest path. The remaining points mark the score value of the navigable paths for different overhead values.

For most of the cases, MSR(OPTD) and MSR(SUBOPTF) perform the best. This is due to the fact that both the algorithms perform a multiple segment replacement, and also estimate the actual score gains in step 2 using the WBS procedure.

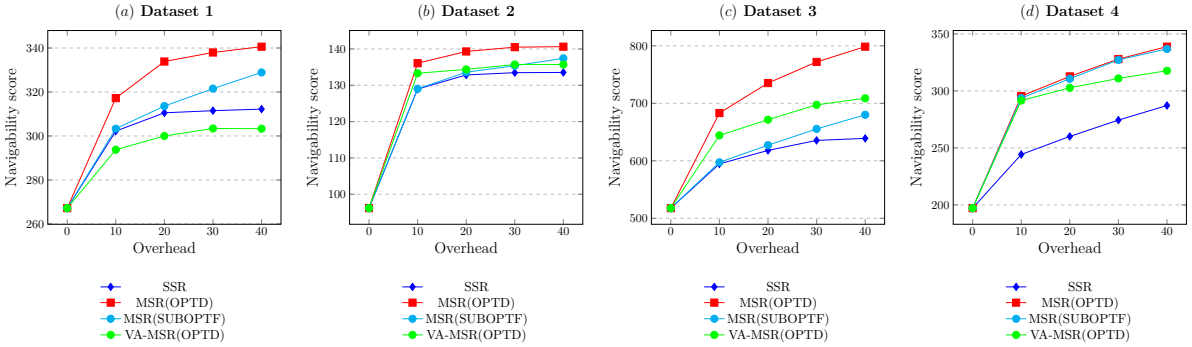


Fig. 9: Effect of overhead value on score of a path; Random distribution of scores with 60% navigable edges; Path length = 40 kms

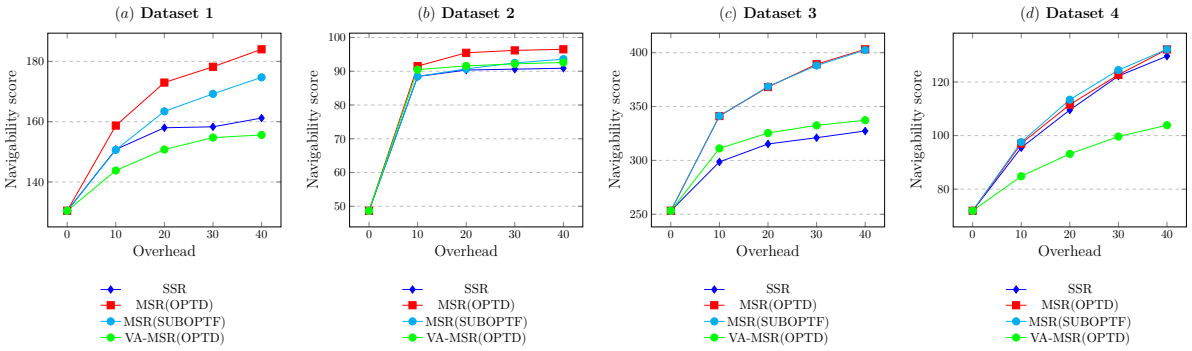


Fig. 10: Effect of overhead value on score of a path; Random distribution of scores with 30% navigable edges; Path length = 40 kms

Effect of increase in overhead on the running time: Through this experiment we studied the effect of variation in overhead value on the running time (absolute clock time) of our algorithms. We observe in Figure 11 that VA-MSR(OPTD) takes the least time of all algorithms, which is due to the indexing scheme used in the algorithm to estimate the VP values of all segments. SSR performs the second best as it replaces only a single segment, followed by MSR(OPTD) and MSR(SUBOPTF).

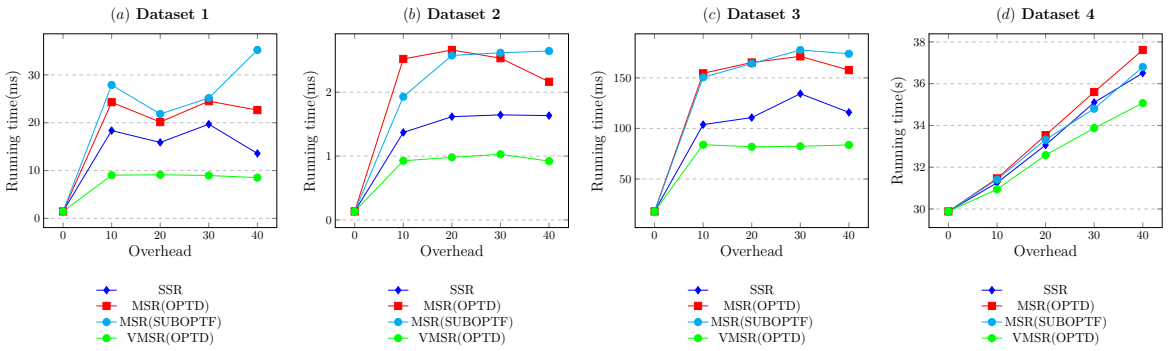


Fig. 11: Effect of overhead value on running time; Random distribution of scores with 60% navigable edges; Path length = 30 kms

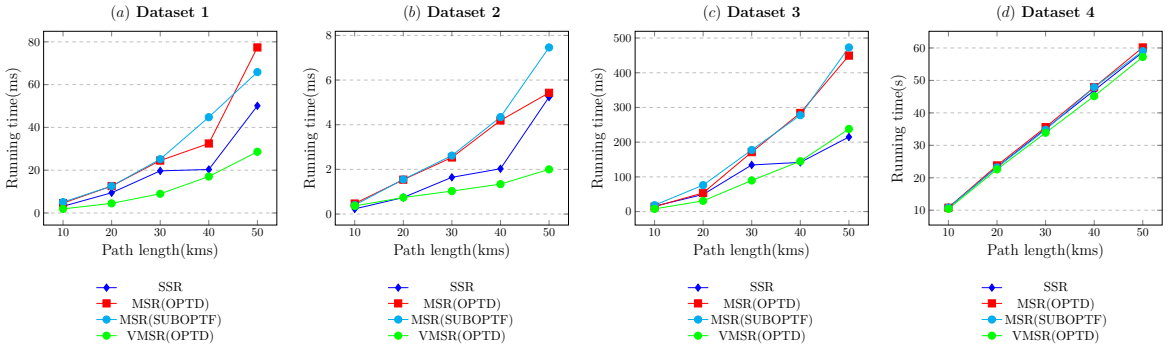


Fig. 12: Effect of path length on running time; Random distribution of scores with 60% navigable edges; Budget value = 30%

Effect of increase in path length on the running time: Figure 12 shows the results of the experiment conducted to measure the change in running time on varying the path length. For datasets 1, 2 and 3, it can be observed that VA-MSR(OPTD) performs the fastest, followed by SSR, MSR(OPTD) and MSR(SUBOPTF), in that order. Also, the rate of increase in running time for VA-MSR(OPTD) is steady as compared to that of the other algorithms. This is attributed to the indexing scheme used in VA-MSR(OPTD). However, for dataset 4 the running times for all algorithms is almost the same.

7.2 Comparative analysis of proposed algorithms and related work

Effect of the overhead value: Figures 13 and 14 depict the results of our experiments on datasets 1, 2, 3 and 4 for shortest path lengths of 20kms, 40kms, 30kms and 10kms respectively. Here, we also show the results for the MSR(OPTD)-KSEEDS algorithm.

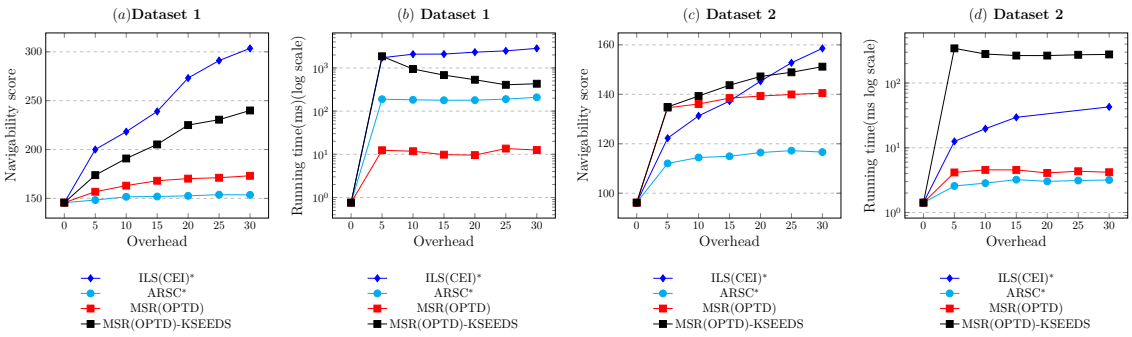


Fig. 13: Score and running time values with increasing overhead for **Datasets 1 and 2**; Random distribution of scores with 60% navigable edges; $K = 4$ for MSR(OPTD)-KSEEDS

For dataset 1, score values are best achieved by ILS(CEI)*, followed by MSR(OPTD)-KSEEDS, MSR(OPTD) and ARSC*, in that order (Figure 13(a)). However, there is a significant difference in the computation time of the four algorithms (Figure 13(b)). For instance, at 30% budget value, MSR(OPTD) takes only

0.01 seconds. Whereas, ARSC*, MSR(OPTD)-KSEEDS and ILS(CEI)* take 0.2, 0.4 and 2.84 seconds respectively.

For dataset 2, MSR(OPTD)-KSEEDS performs the best in terms of solution quality until the budget value of 20% (Figure 13(c)), and beyond this value ILS(CEI)* takes the lead. In terms of running time (Figure 13(d)), MSR(OPTD) and ARSC perform better than ILS(CEI)* and MSR(OPTD)-KSEEDS.

Figure 14 shows the results for datasets 3 and 4. In terms of the solution quality, MSR(OPTD)-KSEEDS performs the best for both datasets (Figures 14(a) and 14(c)). The comparative performance of ILS(CEI)* and MSR(OPTD) varies across these two datasets. ARSC performs the least for dataset 3. The results corresponding to ARSC were not computed for dataset 4 due to high values of pre-computation time. In terms of running time (Figures 14(b) and 14(d)), MSR(OPTD)-KSEEDS and MSR(OPTD) are orders of magnitude faster than ILS(CEI)* and ARSC. An important observation in this result is that the score value for ILS(CEI)* and ARSC* decreases with increase in overhead. As explained earlier, for an algorithm for MNP it is desired that the score value should not decrease with increase in the available overhead.

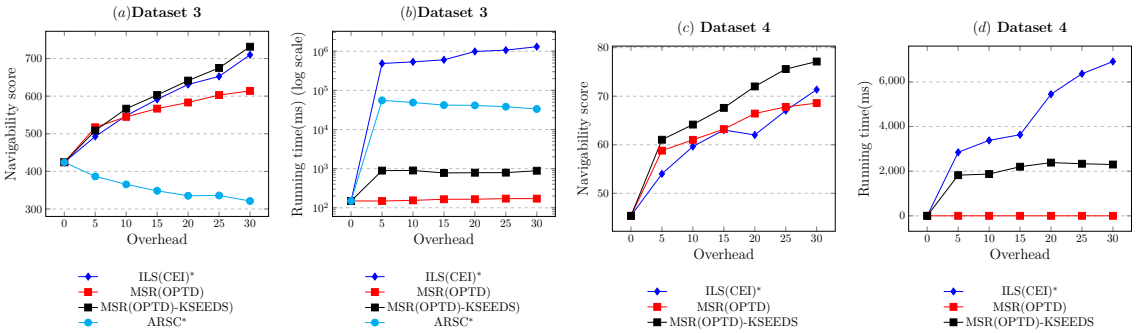


Fig. 14: Score and running time values with increasing overhead for **Datasets 3 and 4**; Random distribution of scores with 60% navigable edges; $K = 4$ for MSR(OPTD)-KSEEDS

Effect of the initial seed path: This experiment was conducted to study the effect of applying the improvement algorithms of MSR(OPTD) and ILS(CEI)* on the same initial seed. Figure 15 shows the results for dataset 4. We considered two seed paths, the shortest path (Figures 15(c) and 15(d)) and the seed path selected by ILS(CEI) (Figures 15(a) and 15(b)). Here, we apply the original algorithms except their method of selecting the initial seed. It can be seen from the results that MSR(OPTD) gives a better navigability score, and runs orders of magnitude faster than ILS(CEI)*. The difference in the results is more significant for the case where the shortest path has been selected as the seed. The significant difference in the running times can be attributed to the fact that ILS(CEI) performs multiple shortest path computations for computing the replacement for each segment.

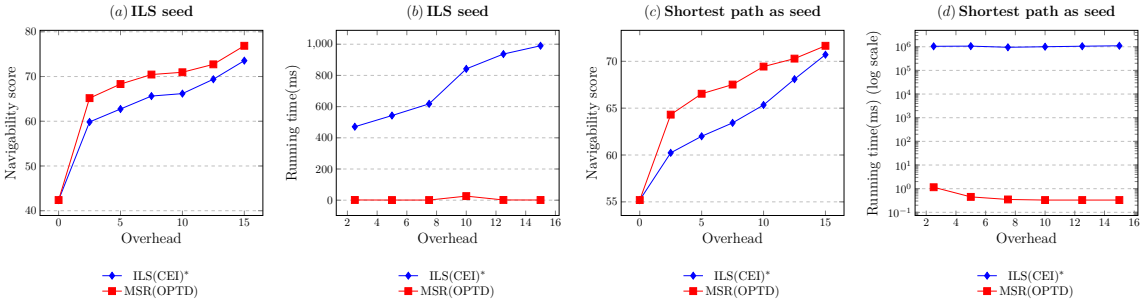


Fig. 15: Applying the improvement algorithms of MSR(OPTD) and ILS(CEI)* on the same seed; Normalised distribution of scores with 70% navigable edges, Path length = 10 kms

7.3 Results on generalized algorithms for Turn based navigability

The experiments in this section were conducted on the implementation based on turn based navigability (explained in Section 5). Here, we considered the datasets created in the random distribution setting with 60% navigable edges. In each experiment, we report the average statistics for hundred queries.

Effect of turn score values on the number of turns: In this experiment, we vary the fixed preference score value assigned to each type of turn (straight, left and right), and study its effect on the number of turns of each type in the path returned. The results of this experiment for dataset 2 are shown in Figure

16. **MNP I** denotes the solution corresponding to the the MSR(OPTD) algorithm. **MNP II** denotes the solution corresponding to the generalized version of the MSR(OPTD) algorithm based on turn based navigability. **SP** denotes the solution corresponding to the shortest path. Here, the label x_y_z for turn scores denotes that a score value of x was assigned for straight turns, y for left turns, and z for the right turns. We observe that the number of turns of each type changes in accordance with the difference between the fixed scores assigned to the different turns.

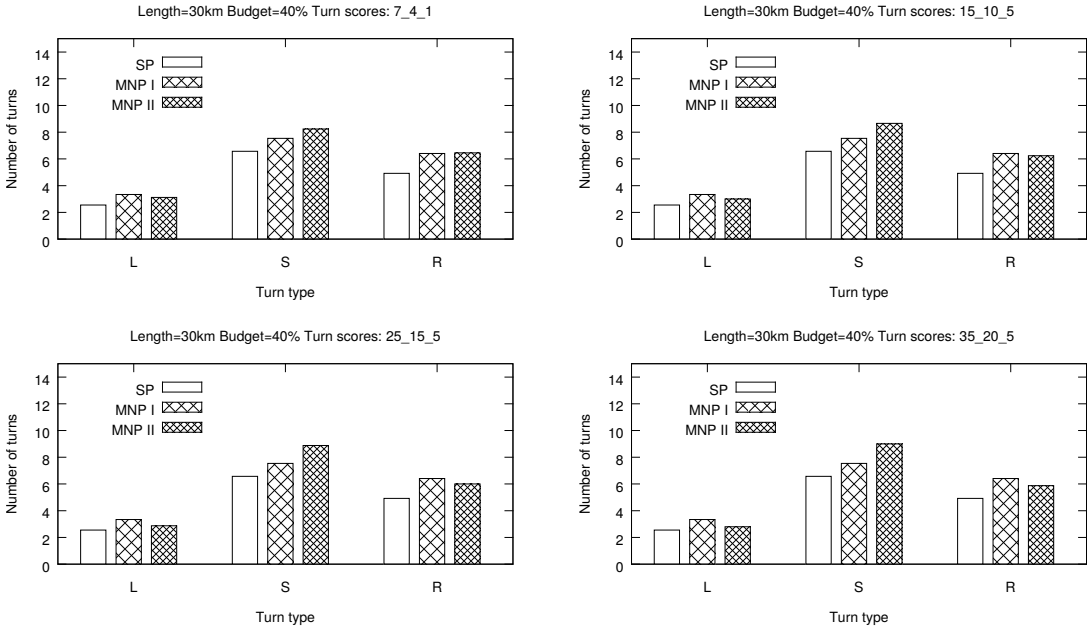


Fig. 16: Turns of each type for increasing values of turn scores; Path length = 30 kms; Budget value = 40%

As desired, with increase in the difference between the scores assigned to the straight, right and left turns, the number of straight turns in the solution for **MNP II** increases, and the number of left and right turns decreases. Figure 17 shows this change for each type of turn separately. Here, the first point on the x axis (S=0, R=0, L=0) denotes the values for **MNP I**.

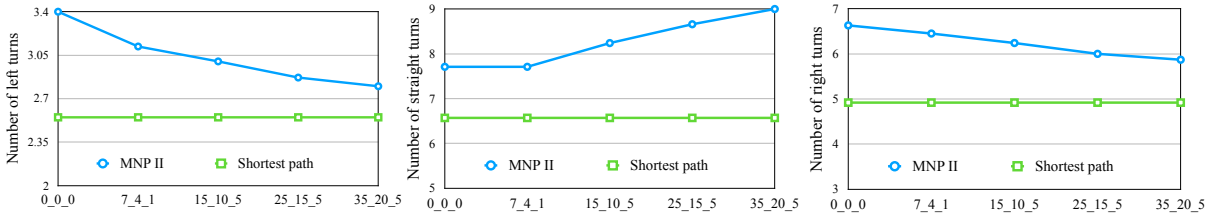


Fig. 17: Turns of each type for increasing values of turn scores; Path length = 30 kms; Budget value = 40%

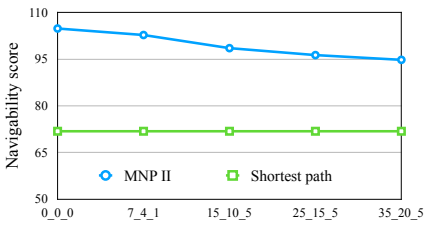


Fig. 18: Score as a function of fixed preference value

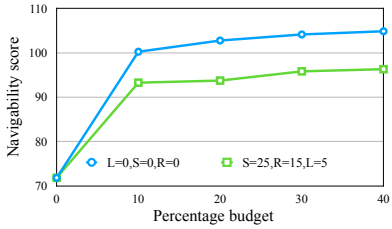


Fig. 19: Score vs. budget for MNP II and SP

Effect of turn score values on the score of path: Through this experiment we study the change in the navigability score of the solution as the difference between the fixed scores assigned to the straight, left and right turns is increased. Figure 18 shows the results of this experiment done on dataset 2 for queries with shortest path length of 30 kms, and a budget value of 40%. We observe that the score value decreases for the case of **MNP II**. In general, this is a trade-off between the score of solution and the

turns of each type. As the turn score values are increased, straight turns increase, left and right decrease, and the navigability score of the solution decreases. Thus, the score values for turns can be set as per user requirement.

Effect of budget value on the score of path: Figure 19 shows the score value for the shortest path and MNP II path (with fixed preference scores for the straight, left and right turns for MNP II as 25, 15 and 5 respectively). As can be seen, the score values are higher for the shortest path. These results correspond to dataset 2 for queries with shortest path length of 30 kms, and a budget value of 40%.

7.4 Case Study

In this section, we include a real-life case study. We consider a source and destination query, and compare the routes suggested by Google Maps to that produced by the MSR(OPTD) algorithm. Figure 20 shows the three routes suggested by Google Maps for traveling from IIIT-Delhi to Kandy's Pastry Parlour. The routes are ranked based on the travel time, Path 1 followed by Path 2 and then Path 3. For the disjoint portions of these routes, it can be seen that Path 1 and Path 2 cross the residential and market areas of Kalka Ji, which have narrow lane roads. These lanes are usually very busy during the evening hours. Further, all the three paths also have a significant number of turns. As shown in Figure 21, we consider the road segments and intersections of the road network relevant for this example to create a subgraph with 20 vertices and 44 edges (most of the edges shown in the figure are bi-directional).

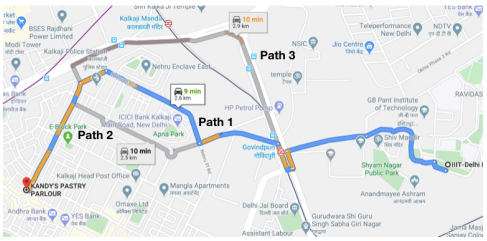


Fig. 20: Example query on Google Maps

Fig. 21: Subgraph with 20 nodes and 44 edges

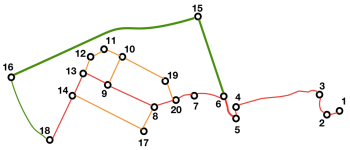


Fig. 22: Labelled Graph

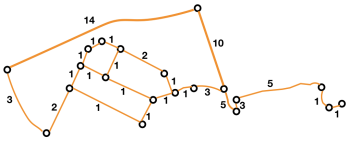


Fig. 23: Navigability Scores

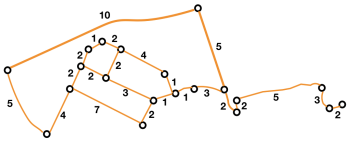


Fig. 24: Costs

Further, we annotate this graph and assign navigability scores and costs to its road segments, as shown in Figures 22, 23 and 24 respectively. The score values are assigned on the scale of 0 to 15, and consider the presence of some prominently visible site and width of the road segments. For example, the edge connecting vertices 15 and 16 is assigned a score value of 14, as it corresponds to a four lane road segment that crosses two metro stations, Kalkaji Mandir and Nehru Enclave. The cost values are assigned based on the travel time. We provide this graph as input to the MSR(OPTD) algorithm to see the results for the same source and destination query. The shortest path computed in this case is highlighted in red: 1 –> 2 –> 3–> –> 4 –> 5 –> 6 –> 7 –> 20 –> 8 –> 9 –> 13 –> 14 –> 18, which has score and cost values of 25 and 30 respectively. The most navigable path returned for a budget value of 20% is 1 –> 2 –> 3–> –> 4 –> 5 –> 6 –> 15 –> 16 –> 18 (includes the segment highlighted in green), which has score and cost values of 42 and 34 respectively. As can be seen, the path returned by MSR(OPTD) includes wide lane roads with prominently visible sites, and has a lesser number of turns.

8 Conclusions

In this paper, we introduced a novel problem of finding navigable paths that have the potential to suit the needs of travelers in developing nations. We formulated the Most Navigable Path (MNP) problem as a constrained maximization problem, and proposed five heuristic algorithms to solve it: MSR(OPTF), MSR(SUBOPTF), MSR(OPTD), VA-MSR(OPTD) and MSR(OPTD)-KSEEDS. We proposed an indexing structure for MNP that estimates the potential of a segment of path in giving a better alternative path. This estimation can be done in constant time, irrespective of the granularity of the index which

makes our algorithms far more efficient than the state-of-the-art algorithms. We demonstrated the effectiveness of our algorithms through experiments on four real road network datasets.

Acknowledgements This work was in part supported by the Infosys Centre for Artificial Intelligence at IIIT-Delhi, Visvesvaraya Ph.D. Scheme for Electronics and IT, and DST SERB(ECR/2016/001053).

References

1. Aly, A.M., Mahmood, A.R., Hassan, M.S., Aref, W.G., Ouzzani, M., Elmeleegy, H., Qadah, T.: Aqwa: Adaptive query workload aware partitioning of big spatial data. Proc. VLDB Endow. **8**(13), 2062–2073 (2015)
2. Archetti, C., Corberán, A., Plana, I., Sanchis, J.M., Speranza, M.G.: A branch-and-cut algorithm for the orienteering arc routing problem. Comput. Oper. Res. **66**(C), 95–104 (2016)
3. Bansal, N., Blum, A., Chawla, S., Meyerson, A.: Approximation algorithms for deadline-tsp and vehicle routing with time-windows. In: Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing, STOC '04, pp. 166–174 (2004)
4. Bolzoni, P., Helmer, S.: Hybrid best-first greedy search for orienteering with category constraints. In: Proceedings of SSTD 2017, pp. 24–42 (2017)
5. Bolzoni, P., Persia, F., Helmer, S.: Itinerary planning with category constraints using a probabilistic approach. In: Database and Expert Systems Applications, pp. 363–377. Springer International Publishing (2017)
6. Chekuri, C., Pal, M.: A recursive greedy algorithm for walks in directed graphs. In: Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science, FOCS '05, pp. 245–253 (2005)
7. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1) (1959). DOI 10.1007/BF01386390. URL `https://doi.org/10.1007/BF01386390`
8. Fischetti, M., Salazar González, J.J., Toth, P.: Solving the orienteering problem through branch-and-cut. INFORMS Journal on Computing **10**, 133–148 (1998)
9. Floyd, R.W.: Algorithm 97: Shortest path. Commun. ACM **5**(6), 345 (1962). DOI 10.1145/367766.368168. URL `https://doi.org/10.1145/367766.368168`
10. Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G.: A survey on algorithmic approaches for solving tourist trip design problems. Journal of Heuristics **20**(3), 291–328 (2014)
11. Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., Vathis, N.: Approximation algorithms for the arc orienteering problem. Information Processing Letters **115**(2), 313 – 315 (2015)
12. Golden, B.L., Levy, L., Vohra, R.: The orienteering problem. Naval Research Logistics (NRL) **34**(3), 307–318 (1987)
13. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
14. Hershberger, J., Maxel, M., Suri, S.: Finding the k shortest simple paths: A new algorithm and its implementation. ACM Trans. Algorithms **3**(4) (2007)
15. Kaur, R., Goyal, V., Gunturi, V.M.V.: Finding the most navigable path in road networks: A summary of results. In: Database and Expert Systems Applications, pp. 440–456 (2018)
16. Kriegel, H.P., Renz, M., Schubert, M.: Route skyline queries: A multi-preference path planning approach. 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010) pp. 261–272 (2010)
17. L. R. Ford, J.: Network flow theory. Technical Report P-923 (1956)
18. Laporte, G., Martello, S.: The selective travelling salesman problem. Discrete Applied Mathematics **26**(2), 193 – 207 (1990)
19. Lu, Y., Shahabi, C.: An arc orienteering algorithm to find the most scenic path on a large-scale road network. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '15, pp. 46:1–46:10 (2015)
20. Martins, E., Pascoal, M.: A new implementation of yen's ranking loopless paths algorithm. Quarterly Journal of the Belgian, French and Italian Operations Research Societies **1**(2), 121–133 (2003)
21. Nagarajan, V., Ravi, R.: Poly-logarithmic approximation algorithms for directed vehicle routing problems. In: Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX '07/RANDOM '07, pp. 257–270 (2007)
22. Nagarajan, V., Ravi, R.: The directed orienteering problem. Algorithmica **60**(4), 1017–1030 (2011)

23. Singh, A., Krause, A., Guestrin, C., Kaiser, W., Batalin, M.: Efficient planning of informative paths for multiple robots. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence, IJCAI'07, pp. 2204–2211 (2007)
24. Souffriau, W., Vansteenwegen, P., Berghe, G.V., Oudheusden, D.V.: The planning of cycle trips in the province of east flanders. Omega **39**(2), 209 – 213 (2011)
25. Tian, Y., Lee, K.C.K., Lee, W.C.: Finding skyline paths in road networks. In: Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '09, pp. 444–447. ACM, New York, NY, USA (2009). DOI 10.1145/1653771.1653840. URL http://doi.acm.org/10.1145/1653771.1653840
26. Vansteenwegen, P., Souffriau, W., Oudheusden, D.V.: The orienteering problem: A survey. European Journal of Operational Research **209**(1), 1 – 10 (2011). DOI https://doi.org/10.1016/j.ejor.2010.03.045
27. Verbeeck, C., Vansteenwegen, P., Aghezzaf, E.H.: An extension of the arc orienteering problem and its application to cycle trip planning. Transportation Research Part E: Logistics and Transportation Review **68**, 64 – 78 (2014)
28. Yen, J.Y.: Finding the k shortest loopless paths in a network. Management Science **17**(11), 712–716 (1971)