Investigating Computational Techniques for Micro-level and Macro-level Transportation Problems on Urban Road Networks

A Dissertation
Presented to
The Academic Faculty

By

Ramneek Kaur
Roll No. PhD16001

In partial fulfillment
of the requirements for the
Degree of Doctor of Philosophy

Under Supervision of

Prof. Vikram Goyal (IIIT-Delhi), and
Prof. Venkata M. V. Gunturi (IIT Ropar)

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

Department of Computer Science
Indraprastha Institute of Information Technology Delhi

June 2022

Investigating Computational Techniques for Micro-level and Macro-level Transportation Problems on Urban Road Networks

A Dissertation
Presented to
The Academic Faculty

By

Ramneek Kaur
Roll No. PhD16001

In partial fulfillment
of the requirements for the
Degree of Doctor of Philosophy

Under Supervision of

Prof. Vikram Goyal (IIIT-Delhi), and
Prof. Venkata M. V. Gunturi (IIT Ropar)

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY **DELHI**

Department of Computer Science
Indraprastha Institute of Information Technology Delhi

June 2022

CERTIFICATE

This is to certify that the thesis titled, "Investigating Computational Techniques for Micro-level and Macro-level Transportation Problems on Urban Road Networks", being submitted by Ramneek Kaur to the Department of Computer Science, Indraprastha Institute of Information Technology Delhi, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, is an authentic proof of work carried out by her under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The work in this thesis has not been submitted in any form for another degree or diploma at any university or another institute.

Date of Signature

07-oct-2022

Vikram Goyal

Professor, Department of CSE

IIIT-Delhi

23-Sept-2022

Venkata M. V. Gunturi

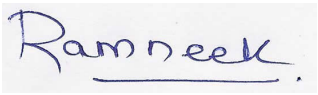
Assistant Professor, Department of CSE

IIT Ropar

DECLARATION

I hereby declare that the work presented in this thesis titled "Investigating Computational Techniques for Micro-level and Macro-level Transportation Problems on Urban Road Networks", submitted to the Department of Computer Science, Indraprastha Institute of Information Technology Delhi, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science, is an authentic proof of my own work carried out from June, 2016 to the present date under the supervision of Prof. Vikram Goyal and Prof. Venkata M. V. Gunturi.

The work in this thesis has not been submitted in any form for another degree or diploma at any university or another institute. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.



Ramneek Kaur

June, 2022

ACKNOWLEDGEMENTS

First and foremost, I extend a profound gratitude towards my Ph.D. advisors, Prof. Vikram Goyal and Prof. Venkata M. V. Gunturi. I would like to express a heartfelt thanks to them for their support throughout my Ph.D. journey. Specifically, I would like to thank them for always giving me the opportunity to openly discuss my ideas and for continually encouraging me to do better. I have learned a lot during my Ph.D. under their mentor-ship - from being a student volunteer to presenting my research papers at conferences, and from working with other students on research projects to delivering programming tutorials at workshops, my journey has been a great learning experience. I could always request a meeting or visit the office of my advisors for discussing my work, a privilege not so common for Ph.D. students.

Secondly, I would like to thank my collaborators. I am grateful to Prof. Cheng Long for his valuable time on the problem of task allocation in Navigation Systems for Spatial Crowdsourcing. I would also like to thank Ankita Mehta, Kaushal Sanadhya, Aakanksha Saini and Ritvik Gupta for collecting and analysing the data for our work on building a Navigation system for Safe Routing. A heartfelt thanks goes to Siftee Ratra and Divya Sharma for their contribution in the implementation work for my study.

I would also like to thank my thesis evaluation committee members, Prof. Arnab Bhattacharya, Prof. Xun Zhou, and Prof. KwangSoo Yang, for their valuable feedback on the thesis. Furthermore, I would like to acknowledge the Visvesaraya Ph.D. Scheme for Electronics and IT for funding my research.

Finally, I would like to thank my family for their unconditional support and motivation throughout my Ph.D. Particularly, I would like to acknowledge the emotional support of my beloved mother and my husband. Lastly, I would like to thank my friends at IIIT-Delhi and my lab mates for making this journey a memorable one.

ABSTRACT

Transportation is a fundamental task in modern-day civilization. Examples of transportation in our daily lives include going to the workplace, returning home after work, etc. In this thesis, we investigate computational techniques for **Micro-level and Macro-level transportation problems** on urban road networks. Micro-level transportation problems involve transportation of a single individual. Whereas, in the case of Macro-level transportation problems, multiple individuals need to be transported to their individual or common destination(s). In our work on Micro-level transportation problems, we consider **Constrained Path Optimization** for the use-cases of finding *Navigable Paths* and *Safe Paths* on road networks. The concept of Navigable paths has the potential to add value to the state-of-art navigation systems, so they can be easily used in developing nations. Likewise, the concept of Safe Routing has a high societal relevance, especially in the developing nations where the lack of infrastructure such as street lights, may contribute to higher crime rates. We devise algorithmic solutions that focus on the systems-oriented perspective, and also build a Navigation system for these application domains of Constrained Path Optimization. Our work on Macro-level transportation problems revolves around **Task Assignment in Spatial Crowdsourcing**. We consider the use-case of a taxi-hailing service, and propose algorithmic solutions for task assignment that focus on the systems-oriented perspective. Unlike most of the works in this domain, we consider the egalitarian version of the problem, meaning that we optimise the expectation of all entities of the Spatial Crowdsourcing platform.

Table of Contents

List of Tables

List of Figures

CHAPTER 1
INTRODUCTION

1.1 Background

Transportation has been a necessity for all civilizations in history. In the modern day civilization, examples of transportation include going to the workplace, returning home after work, etc. As per the terminology used in this thesis, transportation can be categorised into two types, '*Micro-level transportation*' and '*Macro-level transportation*'. Micro-level transportation refers to the movement of a single individual from her source location to her desired destination location. For example, an individual may want to travel from her home to a nearby shopping mall. In contrast, Macro-level transportation refers to the scenario in which multiple individuals are transported to their individual or common destination(s). Examples of Macro-level transportation include evacuation (or rescue) of citizens before (or after) some disaster, transportation of the users of some taxi-hailing service to their destinations, etc. In this thesis, we investigate computational techniques for both Micro-level and Macro-level transportation problems on urban road networks, from a systems-oriented perspective.

Micro-level transportation problems entail problems in which a person tries to optimise her individual preference metric. For example, an individual heading to a hospital might want to take the fastest route to her destination. In the present day scenario, several routing applications are available for such routing queries like finding the fastest route to some specific hospital. However, the state-of-the-art routing applications like Google Maps or Bing Maps optimise on a single preference metric (like time or distance).

In our work on Micro-level transportation problems, we aim to optimise a constrained preference metric. The problem involves the optimization of one metric, along with a constraint on some other metric. As an example, consider the following scenario. An individual might want to take a route to their destination that is easy to navigate, and so, she would like to maximise the 'navigability' of the route. However, the individual would not be willing to travel indefinitely for the sake of increasing the navigability. Therefore, she would also like to constrain the length of the route to lie within some threshold. We define the problem as a **Constrained Path Optimization** problem (CPO), where we would like to maximise the navigability of the route, while constraining its length to lie within some threshold. In Chapters 2, 3 and 4, we describe our work on CPO.

In our first work (Chapter 2), we devise algorithmic solutions for CPO. As a use-case, we consider the problem of finding the Most Navigable Path on road networks. The problem finds its application in navigation systems for developing nations, that typically have a compromised road network infrastructure. In our second work (Chapter 3), we propose advanced algorithms for CPO. Our third work (Chapter 4) implements a navigation system for CPO. As a use-case in this work, we consider the problem of 'Safe Routing', and build a navigation system that suggests Safe Routes to travelers.

Macro-level transportation problems entail problems in which multiple individuals are involved in the transportation, and a global preference metric has to be optimised. For example, the service provider of a taxi-haling service might want to reduce the sum of customer waiting times for all trips made. However, most of the works on Macro-level transportation problems are based on a utilitarian objective, i.e., the existing studies either optimise the expectation of the workers of the platform or optimise the expectation of the customers or maximise the profit of the platform owner.

In our work on Macro-level transportation problems, we consider an **egalitarian objective**, meaning that we consider optimising for all entities involved (for example, the drivers and the customers of a taxi-hailing service). In our fourth work (Chapter 5), we consider the problem of **Task Assignment in Spatial Crowdsourcing**. We consider a taxi-hailing service platform as a use-case. Specifically, we work on optimising the maximum waiting time for the customers, as well as the drivers, along with maximisation of the profit of the platform.

In the remainder of this chapter, we introduce each of the problems considered in our works.

1.2 Devising algorithms for Constrained Path Optimization

In this work, we devise algorithms for CPO on urban road networks. Traditionally, path optimization on urban road networks has been considered for single objective problems. For instance, the state-of-the-art routing application, Google Maps, recommends the shortest or the fastest route to the destination, which is essentially a minimization problem. On the contrary, we consider a constrained maximization objective, i.e., a setting in which one metric is maximised, along with a constraint on some other metric.

We consider the novel problem of finding *'navigable'* paths as a use-case for Constrained Path Optimization. The problem of finding the Most Navigable Path (MNP) 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 length (in terms of distance or travel-time) of the path is within the given budget. In other words, MNP is a

constrained maximization problem.

Figure 1.1: The Most Navigable Path Problem illustration

This problem finds its applications in navigation systems for developing nations. Quite often, streets 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 1.1. 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 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 1.1 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. Therefore, we model our problem as a constrained maximisation problem.

A note on the relationship between MNP and some classical routing problems: A well-known routing problem in the routing literature is the Travelling Salesman Problem (TSP). Given a set of vertices or nodes, and the distances between all pairs of vertices, the problem is to find a tour that visits each vertex only once, returns to the starting point, and minimises the total travel time/cost. Essentially, the TSP is a node routing problem wherein the focus is on including each node in the output tour such that the total cost is minimised. We dicuss the relationship between the node routing version of MNP, the Orienteering Problem (OP), and TSP. The objective of the OP is to maximize the total score collected from visited (selected) nodes. However, not all the available nodes may be possible to visit due to the limit on total cost of the solution, the budget. Therefore, the OP can be seen as a combination between two classical combinatorial problems, the Knapsack Problem and the TSP [1]. Another well-known problem in the routing literature is the Vehicle Routing Problem (VRP). Given a fleet of vehicles, a central depot, and a set of customers,

the VRP finds the set of least cost routes from the depot to the customers. There are several variants defined in the literature depending upon the constraints (capacity constraints and time windows). The VRP is essentially a constrained minimisation problem, and a generalisation of the TSP.

Our constrained maximisation problem is an NP-hard combinatorial optimization problem (for details refer Section 2.3 of Chapter 2). This makes the problem challenging to solve in real-life applications, that usually have a response-time requirement in milliseconds. The exact and approximation algorithms cannot be used in this case. We address these challenges in our heuristic algorithms presented in works [2] and [3]. We also propose an indexing structure for our problem which significantly reduces the running time of our algorithms. These studies are described in detail in Chapters 2 and 3 of the thesis.

1.3 Building a Navigation System for Constrained Path Optimisation

In this work, we build a navigation system for CPO in urban road networks. As a use-case for the study, we consider the application domain of Safe Routing. Safety in public places has become an essential aspect of route planning these days, especially for women travelers in developing nations. A recent international survey on street harassment shows that several women prefer taking longer routes to their destination that are safer than the otherwise shorter routes suggested by navigation applications like Google Maps and Bing Maps [4].

The problem of finding safe routes becomes even more relevant in developing nations where the lack of infrastructure such as street lights, may contribute to higher crime rates. The annual report by the National Crime Records Bureau (NCRB) of India records 3672 cases of women abduction in the capital city of Delhi in 2019. Figure 1.2 depicts a poorly lit street in Delhi, and Figure 1.3 shows a waterlogged street in Mumbai. Furthermore, safe navigation would also benefit tourists in planning

Figure 1.2: A poorly lit street in Delhi (Source: Twitter)

their travel itineraries, by helping them avoid travelling through areas with higher criminal activity.

Our goal in this work is to find routes that maximise safety, where the length of the routes does not exceed a given threshold. In other words, we model the problem as a constrained maximisation problem. The foremost challenge in this study is the conflicting requirements of increasing the 'safety' and constraining the length of the route. To address this challenge, we formulate the problem as a constrained maximisation problem, which makes our algorithms for finding the most navigable path, applicable for the Safe Routing problem.

Another challenge in this study is the availability of data that can be used as a reliable proxy for computing the street level safety. To compute the street level safety for the city of Delhi, we extracted data from various publicly available resources like government websites, news websites, and social media platforms. To transform

Figure 1.3: A waterlogged street in Mumbai (Source: MeraMumbai News)

the data into a usable form for safe routing, we applied processing techniques using tools from machine learning, natural language processing, and geocoding. Finally, we generate safety scores from the extracted data while ensuring that the safety scores are non-negative (a requirement of our route generation algorithms).

1.4 Proposing algorithms for Task Assignment in Macro-level transportation

The proliferation of smart phones and mobile internet has contributed to the gradual surge in the usage of **Spatial Crowdsourcing** (SC) services over the last decade. SC is a computing paradigm that involves human participation [5] [6] [7]. An SC platform releases tasks featured with spatio-temporal information, which can be taken up by the workers of the platform based on the underlying spatio-temporal constraints. Some use-cases of SC platforms include real-time taxi-hailing services (such as Uber [8] and Ola [9]), online food ordering services (such as Swiggy [10] and Zomato [11]),

etc. These platforms entail two kinds of entities: (i) the workers (of the platform owner or service provider), and (ii) the requesters or customers or users of the service. The primary goal of an SC platform is to assign the tasks released by the customers, to the workers such that the objective of the platform is met. Task assignment in SC may be done either in the offline setting, wherein the spatio-temporal information of the workers and customers is known apriori, or in the online setting, where the workers or the customers or both appear dynamically on the platform. The tasks are assigned to the workers such that some objective function is maximised or minimised. Examples of such functions include maximization of the number of performed tasks, maximization of some utility function (e.g., the total profit earned), minimization of the total cost or the bottleneck cost, and so on.

Egalitarian task assignment: Traditionally, task assignment in SC has been associated with a **utilitarian objective**, i.e., most of the works either optimise the customer expectation or the worker expectation. In contrast, we consider an egalitarian approach for task assignment in SC. An egalitarian approach considers the expectations of all entities involved, which is important for the success of any crowdsourcing platform. We aim to optimise both the customer and worker expectation, while maximising the profit earned by the platform.

Our study of the existing research in this domain shows that most of the works in SC are based on a utilitarian objective, such as maximising the profit of the platform [12–19] or optimising the customer expectation [20, 21] or optimising the worker expectation [22–24]. Only a few works are based on an egalitarian objective such as optimising the worker and customer expectation, while maximising the profit [25] [26].

We address an SC problem from an egalitarian perspective. For a better clarity on the problem, we consider examples from the use-case of a taxi-hailing service. We consider not only maximising the cardinality of the assignment, but also minimising

the maximum value of driver's idle time, and minimising the maximum value of customer's waiting time. Similar to the real-life scenario, we consider a fully-online setting where the drivers and customers appear dynamically on the platform. This setting implies that the spatio-temporal information of all the workers and customers cannot be known beforehand. Another practical scenario that we consider is the existence of deadlines of the drivers and customers. To the best of our knowledge, ours is the only study that aims to minimise the waiting time of both the drivers and the customers[1], while maximising the number of assignments in a fully-online setting along with the existence of deadlines of both entities (drivers and customers).

Our study is challenging due to various reasons. Firstly, the fully-online setting makes the problem challenging as the offline algorithms for finding the optimal bottleneck solution cannot be used. Another factor that adds to the complexity of the problem is the presence of dual objectives (reducing the waiting time and maximising the number of assignments), thereby creating a trade-off between the two. Furthermore, our problem considers another set of conflicting objectives, maximizing both the customer and worker expectation, thus making our problem more challenging. Lastly, the deadlines of both entities makes the existing online bipartite matching algorithms inapplicable in our setting. We plan to address these challenges by considering all these factors in our solution. We propose two heuristic algorithms to solve our problem: (i) the K Nearest Drivers based algorithm, and (ii) the Stable Matching based algorithm. Our algorithms incorporate both the driver's idle time and the customer's waiting time for the task assignment.

[1]For the workers of the platform, we use the terms idle time and waiting time interchangeably.

CHAPTER 2
ALGORITHMS FOR CONSTRAINED PATH OPTIMIZATION

2.1 Introduction

In this study, we propose algorithms for Constrained Path Optimization (CPO). As a use-case for CPO, we consider the novel concept of ***'Navigable paths'***, where *navigability* can be associated with multiple interpretations, depending on the user requirement. For example, navigability can be associated with the presence of prominently visible sites such as temples, historic monuments, etc. (that make the path description easier to memorise). Navigability can also be interpreted as paths with good quality road segments and/or fewer number of turns, etc. The goal is to find a path that maximises the navigability, such that the total length of the path lies within some given threshold (the terms route and path are used interchangeably in this thesis). We first introduced our problem in Chapter 1. In this chapter we formally define our problem, discuss the background and related work, present the algorithms we propose as solution to the problem, and finally discuss the performance of our algorithms.

2.2 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}.

2.2.1 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 thesis, 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*.

2.3 Background and Related Work

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 [27, 28]. The AOP problem can easily be reduced [29] to the Orienteering

Problem (OP) which is also NP-hard [30–32]. 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.

2.3.1 Computational Challenges

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. Therefore, one cannot trivially generalize algorithms developed for shortest path problems (e.g., [33–35]) to get an optimal solution for the MNP problem. For the same reason, algorithms developed for k shortest loop-less paths problem [36–38] and route skyline queries [39, 40] 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 [36–39], 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 [33–39] as minimization based approaches.

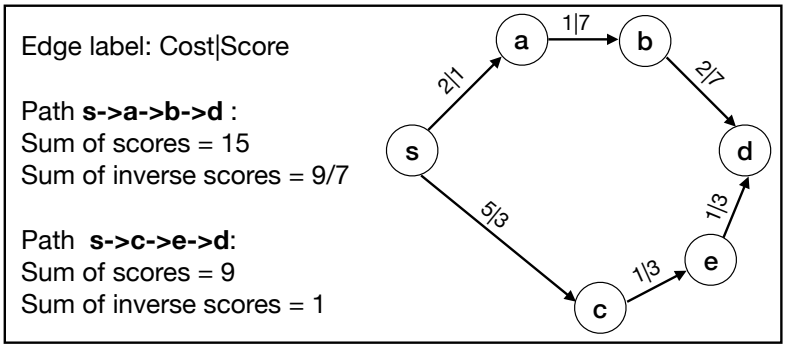


Figure 2.1: Example illustrating challenges of using inverses of navigability scores.

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 2.1 shows an example illustrating this fact. In the Figure, the path $s \rightarrow a \rightarrow b \rightarrow d$ 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 $s \rightarrow c \rightarrow e \rightarrow d$ as the solution, which is incorrect.

2.3.2 Limitations of related work

The existing algorithms for the AOP problem (or the OP as AOP can be reduced to OP [29]) can be divided into three categories: exact, heuristic and approximation algorithms. Exact algorithms have been proposed in [27, 41]. However, these algorithms cannot scale up to any real world road networks. For instance, algorithm proposed in [27] 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 [42, 43] and OP [44, 45] 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[46]). 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 [46]. 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.[28] 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 [47–51]. 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.

2.3.3 Practical considerations while using MNP

From a semantic perspective, the notion of navigability is a synergistic combination of both easily identifiable roads and easily identifiable turns. Ideally, a traveler may like an instruction such as: "keep going straight on this road; once you see a Shree Balaji temple (a popular landmark), take the immediate right turn, after 50 meters, you would see a Pharmacy shop, take a left turn..." In such a situation though the key aspects were the "right" and the "left" turns, the presence of prominent sites on the *edges* ("Shree Balaji temple" and "Pharmacy") made it easy to identify them. While using MNP in real life one would have to assign navigability scores to the road segments. Though the score values may be subjective, a rule of thumb could be followed. We could assign navigability values over a range (say 1-15) where higher values (e.g., 10-15) could be given to edges with unique/popular sites like a well known temple or a prominent building, and lower values (e.g., 1-5) to edges with sites like petrol pumps and ATMs. In case an edge has more than one identifiable site, one can either break the edge into two, or pick the most prominent one. One may also have a navigability score which is the sum of navigability scores of individual sites.

Our contributions: This work makes the following contributions:

1. Proposes the novel problem of finding the most navigable path. This problem has a potential to add value to the current navigation systems so that they can be easily used in developing nations.

2. Proposes a novel indexing structure (*navigability index*) for road networks which can estimate (in constant time) the potential of any given segment (a sequence of edges in a path) for navigability score improvement.

3. Presents two algorithms for the MNP problem which use a novel *Weighted Bi-directional Search (WBS)* procedure and the previously described *navigability index*.

4. Extensively evaluates the proposed algorithms on three real-road network datasets, and compares their performance against that of the current state-of-the-art algorithm, ILS(CEI) [46].

   Our experiments demonstrate that the proposed algorithms yield comparable or better solutions while being orders of magnitude faster than the state-of-the-art.

2.4 Proposed approach

Our proposed algorithms for the MNP problem primarily consists of the following steps (as shown in Figure 2.2). Firstly, we compute a shortest path from s to d which optimizes only on the cost attribute of the path. This path is referred to as the *initial seed path*. Next, we modify portions of this seed path with a goal of improving the navigability score of the solution. This is achieved in steps two, three, and four. While this is being done, we ensure that the total cost of the resulting path is within the budget value.

A key procedure used for improving the navigability of the seed path is our proposed *weighted bi-directional search*. We first describe the *weighted bi-directional search* procedure in Section 2.4.1. Following this, in Sections 2.4.2 and 3.3.4, we propose two novel algorithms for the MNP problem which use this search procedure on "segments" of the initial seed path to improve its navigability score.

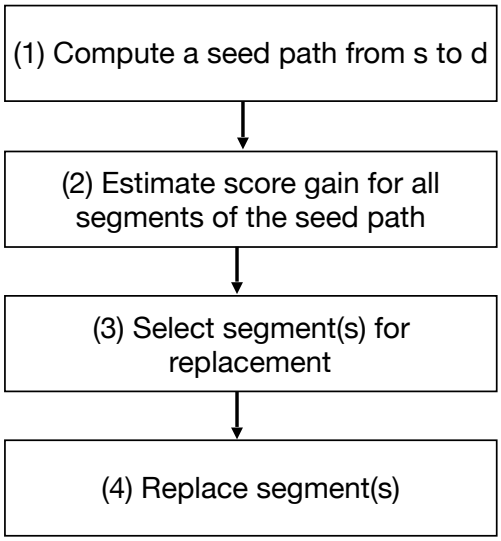


Figure 2.2: Proposed framework

2.4.1 Weighted Bidirectional Search (WBS)

Input to the WBS algorithm consists of the input road network, 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 the specified segment 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.

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 2.1) 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.

$$\Gamma^f(u, F_{tail}, B_{tail}) = \frac{1 + score(F_{tail}, u)}{cost(F_{tail}, u) + D_E(u, B_{tail})} \tag{2.1}$$

In Equation 2.1, 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}. The equation estimates the potential of a vertex u in being able to form an alternative path for replacement during the forward search. This formulation for the potential gives a higher value to a vertex that would lead to a higher score-gain and a lower cost-gain for the replacement. The numerator in the formulation is the score value of the outgoing edge that includes the vertex u in the replacement being constructed. Thus, higher the score value of this edge, higher would be the score-gain. The denominator in the formulation estimates the cost-gain that would be incurred on including the vertex u in the replacement. The cost-gain estimate is computed as follows: it is equal to the sum of the cost value of the outgoing edge that includes u in the replacement being formed, and the euclidean distance of u from the end point of the backward search, i.e., the tail vertex of the backward search (B_{tail}). The euclidean

[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.

distance serves as a lower bound on the network distance of u from the tail vertex of the backward search. Thus higher the denominator, higher would be the cost gain for the replacement. 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 Γ^f(u, F_tail, B_tail)
3: end for
4: Best-Successor of F_tail ← 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) can be computed using Equation 2.2. 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}. The formulation in Equation 2.2 follows the same rationale as Equation 2.1.

$$\Gamma^b(v, B_{tail}, F_{tail}) = \frac{1 + score(v, B_{tail})}{cost(v, B_{tail}) + D_E(v, F_{tail})} \tag{2.2}$$

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 2.1 (and F_{tail} in Equation 2.2) would change as the algorithm proceeds. To this end, the WBS algorithm employs a design decision to help in terminating quickly (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

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

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.

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). In the end, 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 concate-

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}

$F_{tail} \leftarrow S_{ij}.start$, $B_{tail} \leftarrow S_{ij}.end$
Mark F_{tail} as colored by forward search and B_{tail} as colored by backward search
while Forward and backward searches do not color a common node **do**
    if F_{tail} and B_{tail} are connected via a 1-hop or 2-hop path **then**
        $P_{cand} \leftarrow S_{ij}.start \rightsquigarrow F_{tail} \rightsquigarrow B_{tail} \rightsquigarrow S_{ij}.end$
        if $P_{cand}.cost \leq B^{'}$ **then**
            Save P_{cand} in the set (Ω) of candidate segments
        end if
    end if
    Compute Best-Successor of F_{tail} & its Gamma, Γ^{f}_{best} (using Algorithm 1)
    Compute Best-Predecessor of B_{tail} & its Gamma, Γ^{b}_{best}
    if $\Gamma^{f}_{best} > \Gamma^{b}_{best}$ **then**
        $F_{tail} \leftarrow$ Best-Successor of F_{tail} ▷ Move the forward search
        Compute Best-Predecessor of B_{tail}
        $B_{tail} \leftarrow$ Best-Predecessor of B_{tail} ▷ Move the backward search
        Mark F_{tail} and B_{tail} as colored by their respective searches
    else
        $B_{tail} \leftarrow$ Best-Predecessor of B_{tail} ▷ Move the backward search
        Compute Best-Successor of F_{tail} (using Algorithm 1)
        $F_{tail} \leftarrow$ Best-Successor of F_{tail} ▷ Move the forward search
        Mark F_{tail} and B_{tail} as colored by their respective searches
    end if
    if $(S_{ij}.start \rightsquigarrow F_{tail}).cost + (B_{tail} \rightsquigarrow S_{ij}.end).cost > B^{'}$ **then**
        Break
    end if
end while
if Forward and backward searches have colored a common node **then**
    $P_{cand} \leftarrow$ Reconstructed path between $S_{ij}.start$ and $S_{ij}.end$ by following the *Best-Successors/Best-Predecessors*
    Save P_{cand} in Ω
end if
Return the segment in Ω with highest navigability score

nating 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}.

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 $O(|E|)$.

2.4.2 MS(WBS): Multiple Segment replacement algorithm with WBS

As described earlier, the core idea of both our algorithms is to first compute an initial *seed path* (P) and then replace segments of this seed path to improve its navigability score. In this section, we describe an algorithm which efficiently chooses a set of segments of P, which when replaced by the output of WBS, lead to a high increase in the navigability score. Note that we may not be able to replace all the segments due to the budget constraint.

For sake of brevity we describe only the most crucial part of the algorithm, which is to determine the set of segments of the initial seed path for replacement. One can trivially put this along with the previously described WBS algorithm and the construction of the initial seed path to create a full pseudo-code.

In the MS(WBS) algorithm, we first execute an instance of WBS for each of the possible segments of the initial seed path. While calling an instance of WBS for a segment S_{xy} in the initial seed path, we pass a budget value $B' = B$ - cost of seed

Table 2.1: 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)

path $(P.cost) + S_{xy}.cost$. Here, B is the budget value given in the problem instance $(B = overhead + P.cost)$.

Following these calls to the WBS algorithm, we would have a pair *<score gain, cost gain>* for each of the possible segments S_{xy} in the initial seed path. Here, *score gain* of a segment S_{xy} is defined as the difference in the navigability score values of S_{xy} and its replacement S'_{xy}. $S_{xy}.sgain = S'_{xy}.score - S_{xy}.score$. *Cost gain* is also defined in an analogous way: $S'_{xy}.cgain = S'_{xy}.cost - S_{xy}.cost$

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) the budget constraint and, (c) replacements of the chosen segments may overlap.

As an instance of challenges (a) and (b), refer to Table 2.1. 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 in the table along with their sample sgain and cgain values. In the table, 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.

In addition, if the allowed overhead was 15 then, the combination $< e_2\, e_3 >$ and $< e_1 >$ can be collectively replaced. Whereas, segments $< e_1\, e_2 >$ and $< e_3 >$ cannot be collectively replaced, as their total cost gain 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 (initial seed 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, $< e_2\, e_3 >$ and $< e_1 >$ forms 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: (i) knapsack problem is known to be NP-hard, (ii) a path with n edges would have 2^{n-1} unique sets of disjoint segments. Given these reasons, we propose to drop the feasibility constraint of the FDSS definition. This gives us the concept of a Disjoint Set of Segments (DSS).

Disjoint Set of Segments (DSS): DSS of a given path (initial seed path P) is a set of segments such that no two segments in the set share an edge. Our algorithm attempts to compute a DSS which results in the highest increase in navigability score (optimal DSS) without considering the budget constraint.

Dynamic programming (DP) based solution for computing optimal DSS: 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" (i.e., initial seed path) into pieces (i.e., disjoint segments) such that the total score gain obtained by replacing these disjoint segments is maximum.

Understandably, the rod is assumed to be made up of edges and we are allowed to place cuts at the head or tail node of edges.

$$f_{ij} = \max_{i \leq k \leq j}(S_{ik}.sgain + f_{(k+1)j}) \tag{2.3}$$

Equation 3.5 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 P containing n edges (numbered 1 through n) is given by f_{1n}. 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 at all 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.

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 n do                                  ▷ Subproblem size
2:      for i = 2 to n − 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 n 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
```

Algorithmic details of MS(WBS): This algorithm has three primary steps. In the first step, sgain values are computed for all possible segments of the initial seed path using the WBS algorithm. After this, in the second step, we compute the optimal

DSS of the initial seed path using Algorithm 3. Next, WBS is invoked on all pieces in the optimal DSS in decreasing order of their sgain values, and the path is updated after each invocation of WBS. The reason for invoking WBS again is to ensure that the path remains free from loops as it is updated. The segments are extracted in decreasing order of their sgain values to maximize the improvement in the score as replacing all segments within the budget may not be feasible.

Time complexity analysis: For the time complexity analysis of MS(WBS), we do not consider the complexity of initial seed path computation. Given that we are using a shortest path from s to d as our seed path, this cost is upper bounded by the cost of Dijkstra's algorithm $O(|E| + |V| \log |V|)$.

The first step of MS(WBS) 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 $\theta(l^2)$. A DSS is then selected in the second step using our DP solution in $\theta(l^3)$ time. Step three involves updating the path by calling WBS on all segments in the selected DSS. Since the maximum number of segments in a DSS can be l, this step requires $O(l|E|)$ time. This gives a time complexity of $O(l^2|E|)$ for MS(WBS).

2.4.3 VAMS(WBS): Vicinity Aware Multiple Segment replacement algorithm with WBS

Recall that MS(WBS) computes the estimates of score gain values for all the possible segments of the initial seed path. In other words, given an initial seed path with l edges, MS(WBS) calls WBS algorithm $\theta(l^2)$ times to get the score gain 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 section proposes a novel metric called *vicinity potential* (VP) of a segment. The VP values serve as a proxy to the score gain values of the

$\theta(l^2)$ segments in the initial seed path. The algorithm (VAMS(WBS)) proposed in this section uses VP values instead of the score gain values. We now provide details on computing the VP value of a segment.

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 (depicted in Figure 2.3).

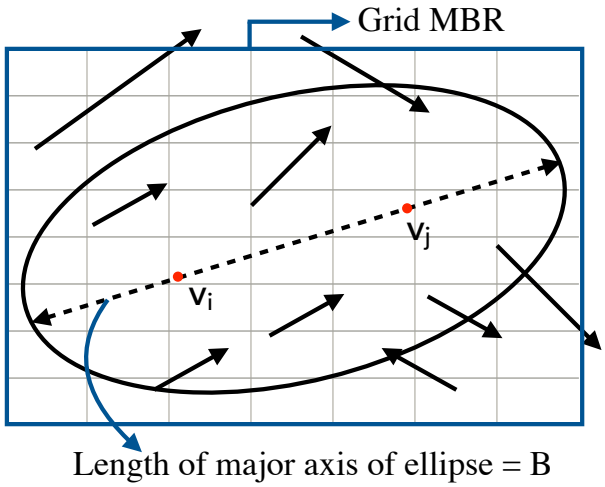


Figure 2.3: Ellipse property illustration

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 2.4 presents this formally[3]. Recall that only the edges having a navigability score value > 0 are referred to as navigable edges (denoted in Equation 2.4 as $'ne'$).

$$VP(S_{ij}, B') = \frac{\sum_{ne \in ellipse(S_{ij}.start, S_{ij}.end, B')} ne.score}{Count(ne \in ellipse(S_{ij}.start, S_{ij}.end, B'))} \tag{2.4}$$

[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.

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.

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 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 2.5 and 2.6. Here, $sum(x, y)$ and $count(x, y)$ respectively denote the sum and count values of cell (x, y).

$$Sum = sum(m, n) - sum(i-1, n) - sum(m, j-1) + sum(i-1, j-1) \quad (2.5)$$

$$Count = count(m, n) - count(i-1, n) - count(m, j-1) + count(i-1, j-1) \quad (2.6)$$

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 2.5 and 2.6. This makes the computation much faster. The idea of this index structure was inspired by the work

of Aly et al.[52].

Algorithmic details of VAMS(WBS): VAMS(WBS) is similar to MS(WBS) with exceptions in the first and the second steps. Given the initial seed path, the first step of VAMS(WBS) involves computing the VP values of all segments of the seed path. In the second step, the optimal DSS is computed based on the VP values of segments. Next, the VAMS(WBS) invokes the WBS algorithm to determine the actual replacements for the segments in the optimal DSS.

Time complexity analysis: The time complexity of the first step of VAMS(WBS) 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 MS(WBS). Thus, the time complexity of VAMS(WBS) is $O(l|E| + l^3)$.

2.5 Experimental evaluation

Performance of the proposed algorithms was evaluated through experiments on three real-road networks of different sizes (refer Table 2.2). Datasets 1 and 3 were obtained from OpenStreetMap (http://http://www.openstreetmap.org). Dataset 2 was obtained from Digital Chart of the World Server (https://www.cs.utah.edu/ lifeifei/ SpatialDataset.htm). All datasets constituted directed spatial graphs with vertices as road intersections and edges as road segments. Cost of each edge corresponded to the metric of distance. Navigability score values of edges were generated synthetically. We assumed that 40% of the total edges in each dataset have no prominently visible site, and assigned them a score value of zero. The remaining 60% of the edges were assumed to be navigable. These edges were assigned a non-zero score value. The navigable edges were distributed uniformly across space, and were assigned a random integral score value in the range 1-15.

Table 2.2: Description of datasets

Dataset	**Place**	**Vertices**	**Edges**	**Average degree**
1	Delhi	11,399	24,943	2.18
2	California	21,048	39,976	1.89
3	Beijing	55,545	95,285	1.71

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 with WBS (SS(WBS)), and use it as the baseline algorithm.

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 VAMS(WBS) algorithm, the order of the grid index built for datasets 1, 2 and 3 was 300 × 300, 500 × 500 and 1000 × 1000 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.

2.5.1 Comparative analysis of proposed algorithms

Effect of increase in overhead on the score of a path: Figure 2.4 illustrates the results of this experiment for queries where shortest path length was 40 kms. MS(WBS) gives paths with higher navigability score values than SS(WBS) and VAMS(WBS). This is because MS(WBS) actually computes the score gain values

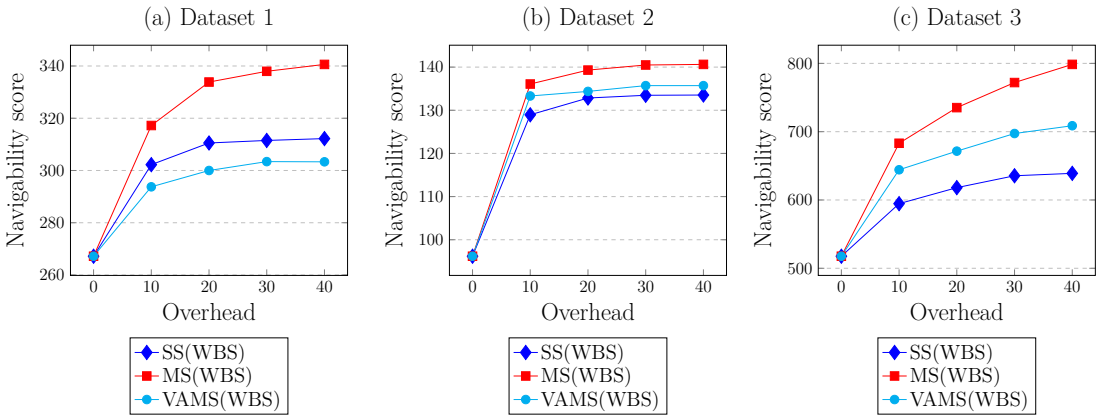


Figure 2.4: Effect of increase in overhead on score of a path

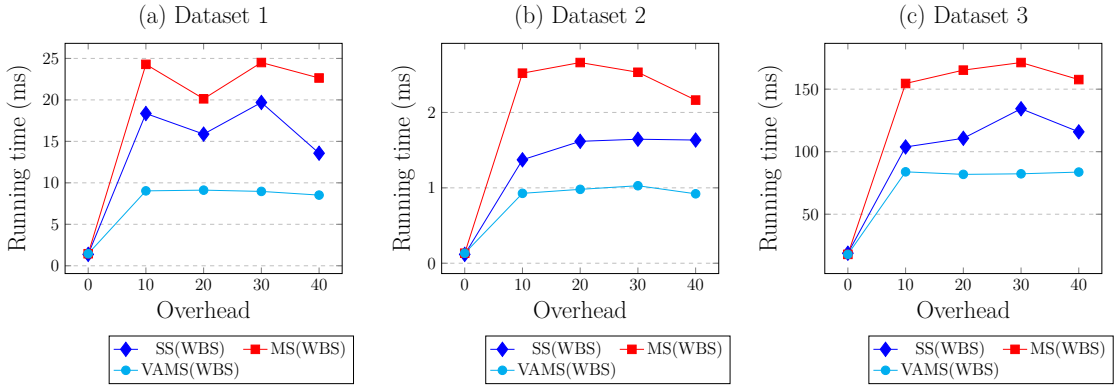


Figure 2.5: Effect of increase in overhead on running time

of all possible segments (unlike VAMS(WBS)). In addition, it computes the optimal DSS (unlike SS(WBS)) using these score gain values.

Effect of increase in overhead on the running time: Figure 2.5 shows the results of this experiment for queries where shortest path length was 30 kms. In general, VAMS(WBS) takes less time because it avoids the expensive repetitive invocation of WBS algorithm to compute the score gain value of each segment in the path. As the overhead is increased from 10% to 40% the increase in running time of all three algorithms is steady. The comparative performance of the algorithms is in accordance with their time complexities

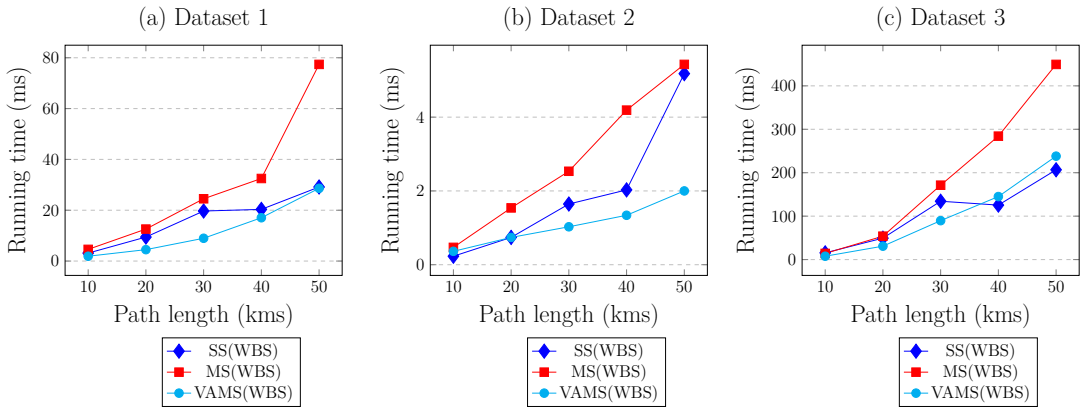


Figure 2.6: Effect of increase in path length on running time

Effect of increase in path length on the running time: Figure 2.6 shows the results of this experiment. The results are shown for an overhead of 30%. We observe that VAMS(WBS) performs the best, followed by SS(WBS) and MS(WBS), in that order. Also, the rate of increase in running time for VAMS(WBS) is steady as compared to the other two algorithms.

2.5.2 Comparative analysis of proposed algorithms and ILS(CEI)

Score value and running time as a function of overhead: Figure 2.7(a) shows the results of this experiment on Dataset 2 for queries with a shortest path length of 40 kms. For this experiment, we implemented an adaptation of ILS(CEI) [46] that yields a simple path. Further, the original ILS(CEI) repeats its iterations till the running time is less than some threshold value. We implemented a version which executes a single iteration of ILS(CEI) and terminates, similar to the working of our algorithms.

Our results show a superior performance of MS(WBS) over ILS(CEI) in terms of both solution quality and running time. However, the difference between the solution quality keeps decreasing with increase in overhead. For overhead values higher than 15%, the solution quality of ILS(CEI) exceeds that of MS(WBS). In contrast to this,

the difference in running times of both algorithms becomes increasingly significant as the overhead increases, making ILS(CEI) impractical for higher values of overhead.

(a) Score and time as a function of overhead (b) Score value per unit distance

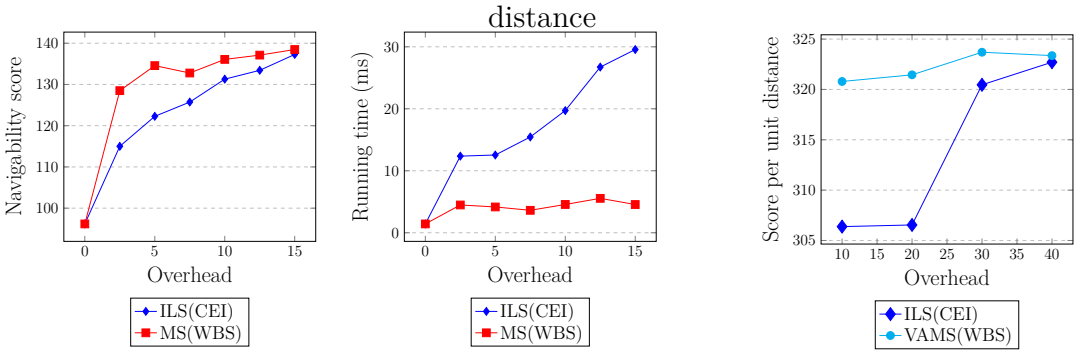


Figure 2.7: Comparative analysis of proposed algorithms and ILS(CEI)

Score value per unit distance as a function of overhead: In this experiment, we compared the score value per unit distance for the solutions given by VAMS(WBS) and ILS(CEI). Figure 2.7(b) shows the results of this experiment on Dataset 1 for queries with a shortest path length of 30 kms, with varying overhead values. VAMS(WBS) has higher score value per unit distance. The gap between the algorithms reduces as the overhead is increased.

2.6 Conclusions

In this study, we work on Constrained Path Optimization for urban road networks. We introduced the 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 two heuristic algorithms to solve the problem: MS(WBS) and VAMS(WBS). We proposed an indexing structure for MNP that estimates the potential of a segment of a baseline 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 three real road network datasets.

CHAPTER 3
ADVANCED ALGORITHMS FOR CONSTRAINED PATH OPTIMIZATION

3.1 Introduction

In this work, we propose advanced algorithms for Constrained Path Optimization (CPO) on urban road networks. Similar to the previous study, we consider the use-case of finding navigable paths. We also add new dimensions to the problem of finding navigable paths. We 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. We propose new algorithms based on a thorough analysis of the quality of results of the Vicinity-aware Multiple Segments Replacement Algorithm mentioned in the previous chapter. We also conduct a more extensive experimental evaluation, detailed as follows:

1. We conduct our experiments on a much larger road network (of New York, USA) which contains 264,346 nodes and 733,846 edges.

2. We implement all the newly proposed techniques and compare them against an adaptation of a route skyline algorithm [39], and the state-of-the-art work [46] most relevant to our problem-setting.

3. We introduce new experiments by considering different parameters.

    - We study the effect of score values (of edges) on the performance of algorithms. To this end, we assign the navigability scores in two ways: (a) random distribution (conference submission), (b) normalized distribution.

- We extensively study the effect of different parameters, on the solution quality and running time.
- 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.

3.2 Proposed framework

Our overall framework for solving the most navigable path problem primarily consists of four steps. 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.

Figure 3.1 shows the modules of the proposed framework along with the techniques proposed for each module. We explore a new technique for computing the seed path in Step-1, teh vicinity based seed path computation. We propose two new algorithms for the Step-3 called *OPTF* and *SUBOPTF*. We explain the new techniques in the rest of this chapter.

3.2.1 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.

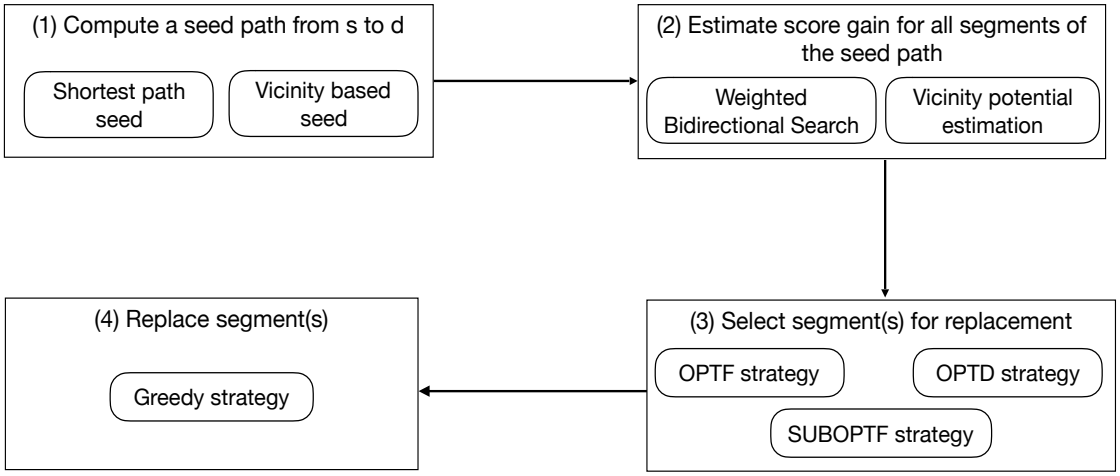


Figure 3.1: Proposed framework with details

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. 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. $\mathcal{P}$ 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.

3.2.2 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 3.2.2 and Section 3.2.2) 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 3.1), 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 **??**) 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 3.2.2, Section 3.2.2 and Section 3.2.2 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 3.1. 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

Table 3.1: 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)

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.

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.

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 3.1 gives the set of feasible solutions for subproblems of size > 1.

$$F_{ij}(i<j) = \{(s_x, c_x) : s_x = S'_{ik}.sgain + s_y \,\text{and}\, c_x = S'_{ik}.cgain + c_y \,\forall (s_y, c_y) \in F_{(k+1)j}, c_x \leq B\} \cup \{(S'_{ik}.sgain, S'_{ik}.cgain)\} \cup F_{(k+1)j}, \text{where } i \leq k \leq j \quad (3.1)$$

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 3.2.

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)

Figure 3.2: Illustration of example for OPTF and SUBOPTF strategies

$$FDSS_{opt} = \{(s_x, c_x) : (s_x, c_x) \in F_{1l}, c_x \leq c_i, s_x \geq s_i \, \forall (s_i, c_i) \in F_{1l}\} \tag{3.2}$$

Example: Consider the *sgain* and *cgain* values given in Table 3.1 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 3.2). 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).

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 3.2.2 proves this claim. Proof of optimality of the FDSS selected is given in Lemma 3.2.1.

Lemma 3.2.1. *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 3.2.2. *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 $O(3^l)$. □

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 3.3. The function max defined in Equation 3.4, computes the local optimal from a set of feasible solutions S. The final solution is then given as f_{1l}.

$$f_{ij}(i<j) = max\{\{(s_x,c_x) : s_x = S'_{ik}.sgain + s_y \,\text{and}\, c_x = S'_{ik}.cgain + c_y \,\forall (s_y,c_y) \in f_{(k+1)j}, c_x \leq Budget\} \cup \{(S'_{ik}.sgain, S'_{ik}.cgain)\} \cup \{f_{(k+1)j}\}, \text{where } i \leq k \leq j\} \tag{3.3}$$

$$max(S) = (s_x,c_x) \text{ where } (s_x,c_x) \in S, s_x \geq s_y \text{ and } c_x \leq c_y \forall (s_y,c_y) \in S \tag{3.4}$$

Example: The values marked in red in Figure 3.2 show the local optimal solutions corresponding to SUBOPTF for the example given in Table 3.1. The final solution in this case too is (29,20).

Lemma 3.2.3. *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 3.2.4. *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}. □

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 3.3 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 4 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 3.5 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 4 presents a bottom up procedure for computing this recurrence equation.

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

Figure 3.3: Illustration of example for OPTD strategy

$$f_{ij} = \max_{i \leq k \leq j} (S_{ik}.sgain + f_{(k+1)j}) \tag{3.5}$$

Example: Figure 3.3 shows the computation of the optimal DSS for the *sgain* values given in Table 3.1. The optimal DSS in this case is $\{< e_1, e_2 >, < e_3, e_4 >\}$, which has a total *sgain* value 30.

Lemma 3.2.5. *For a path with l edges, MSR(OPTD) takes $O(l^3)$ time and consumes $\theta(l^2)$ space to select the optimal DSS.*

3.2.3 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.

3.3 Proposed algorithms

In this section, we explain the following five algorithms that use solution techniques described for the modules of the proposed framework (Figure 3.1):

1. MSR(OPTF): Multiple Segment Replacement algorithm using OPTF strategy for FDSS selection
2. MSR(SUBOPTF): Multiple Segment Replacement algorithm using SUBOPTF strategy for FDSS selection
3. MSR(OPTD): Multiple Segment Replacement algorithm using OPTD strategy for DSS selection
4. VA-MSR(OPTD): Vicinity Aware Multiple Segment replacement algorithm using OPTD strategy for DSS selection
5. MSR(OPTD)-KSEEDS: Multiple Segment Replacement algorithm using Vicinity Based Seed for seed path selection and the OPTD strategy for DSS selection

Note that more algorithms can be designed using a different combination of the solution techniques explained for each module.

3.3.1 MSR(OPTF)

Algorithm 5 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.

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

Algorithm 5 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

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 3.2.2). 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).

3.3.2 MSR(SUBOPTF)

The steps in MSR(SUBOPTF) are the same as that of Algorithm 5 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 3.2.3). 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.

3.3.3 MSR(OPTD)

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.

3.3.4 VA-MSR(OPTD)

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 6 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 6 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 $O(l|E| + l^3)$.

3.3.5 MSR(OPTD)-KSEEDS

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., $O(Kl^2|E|)$.

3.4 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.

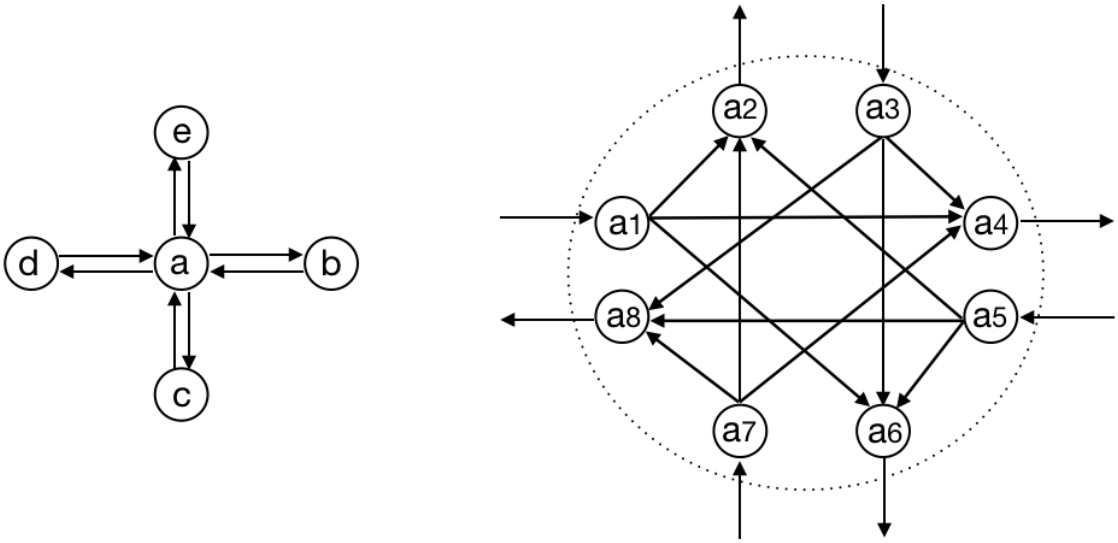


Figure 3.4: Expansion of node a to model the turns at road intersection a

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 3.4 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.

In our proposed system, turn based notion of navigability can be incorporated in the WBS algorithm using Equation 3.6 and Equation 3.7. 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.

$$\Gamma^f(F'_{tail}, F_{tail}, u, B_{tail}) = \frac{1 + score(F_{tail}, u) + turnscore(F'_{tail}, F_{tail}, u)}{D_N(F_{tail}, u) + D_E(u, B_{tail})} \tag{3.6}$$

$$\Gamma^b(B'_{tail}, B_{tail}, v, F_{tail}) = \frac{1 + score(v, B_{tail}) + turnscore(v, B_{tail}, B'_{tail})}{D_N(v, B_{tail}) + D_E(v, F_{tail})} \tag{3.7}$$

3.5 Experimental Analysis

Datasets: All datasets constituted directed spatial graphs with vertices as road intersections and edges as road segments (shown in Table 3.2). Datasets 1 and 3 were obtained from OpenStreetMap[1]. Dataset 2 can be accessed from this url[2], and dataset 4 from this url[3]. 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. In the random distribution setting, integral score values were assigned to 60% of the total edges in the range [1,15], such that each value in the range was equally likely to be assigned. The rest of the edges were assigned a score value of zero. In the normalized distribution setting, integral score values were assigned to 70% of the total edges in the range 1 to 15, and the remaining edges were assigned a score value of zero.

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.

[1] `https://www.openstreetmap.org`
[2] `https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm`
[3] `http://users.diag.uniroma1.it/challenge9/download.shtml`

Table 3.2: Description of datasets

Dataset	Place	Vertices	Edges	Average degree
1	Delhi	11,399	24,943	2.18
2	California	21,048	39,976	1.89
3	Beijing	55,545	95,285	1.71
4	New York	2,64,346	7,33,846	2.77

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)[46] and ARSC[39], 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-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.

3.5.1 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 3.5 and 3.6 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.

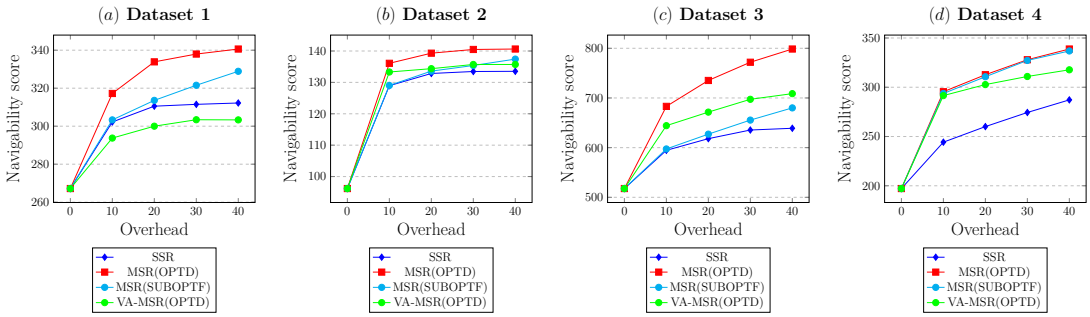


Figure 3.5: Effect of overhead value on score of a path; Random distribution of scores with 60% navigable edges; Path length = 40 kms

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.

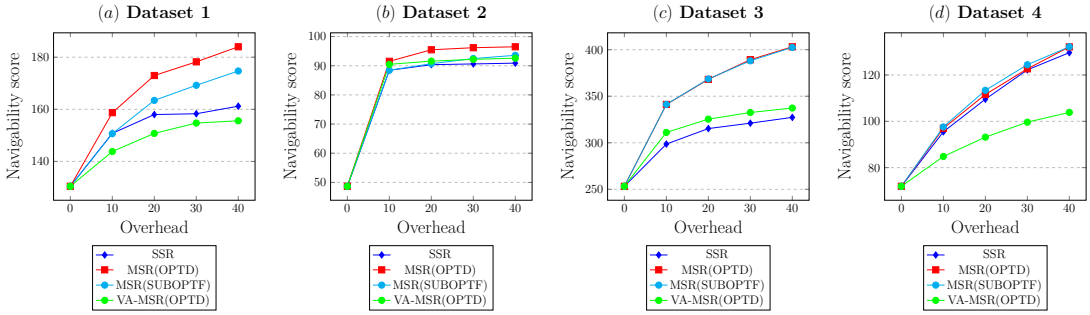


Figure 3.6: Effect of overhead value on score of a path; Random distribution of scores with 30% navigable edges; Path length = 40 kms

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 3.7 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).

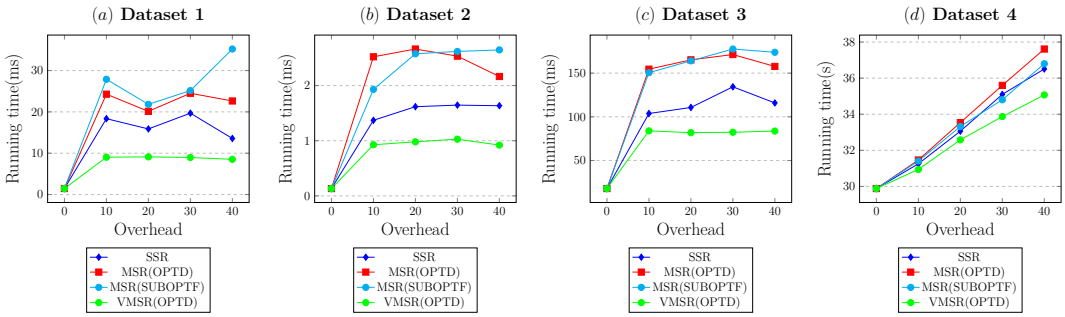


Figure 3.7: Effect of overhead value on running time; Random distribution of scores with 60% navigable edges; Path length = 30 kms

Effect of increase in path length on the running time: Figure 3.8 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.

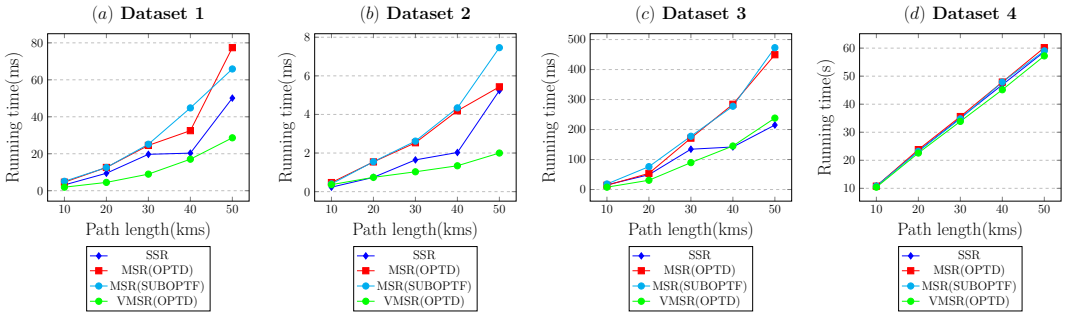


Figure 3.8: Effect of path length on running time; Random distribution of scores with 60% navigable edges; Budget value = 30%

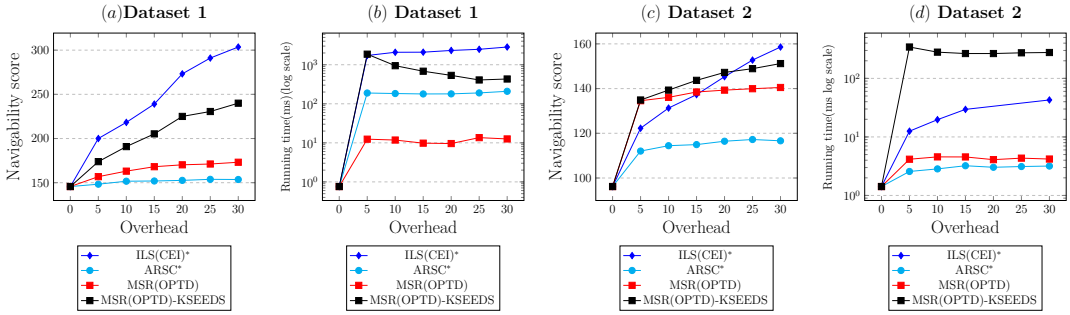


Figure 3.9: Score and running time values with increasing overhead for **Datasets 1 and 2**; Random distribution of scores with 60% navigable edges; $K = 4$ for MSR(OPTD)-KSEEDS

3.5.2 Comparative analysis of proposed algorithms and related work

Effect of the overhead value: Figures 3.9 and 3.10 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.

For dataset 1, score values are best achieved by ILS(CEI)*, followed by MSR(OPTD)-KSEEDS, MSR(OPTD) and ARSC*, in that order (Figure 3.9(a)). However, there is a significant difference in the computation time of the four algorithms (Figure 3.9(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 3.9(c)), and beyond this value ILS(CEI)* takes the lead. In terms of running time (Figure 3.9(d)), MSR(OPTD) and ARSC perform better than ILS(CEI)* and MSR(OPTD)-KSEEDS.

Figure 3.10 shows the results for datasets 3 and 4. In terms of the solution quality, MSR(OPTD)-KSEEDS performs the best for both datasets (Figures 3.10(a) and 3.10(c)). The comparative performance of ILS(CEI)* and MSR(OPTD) varies across

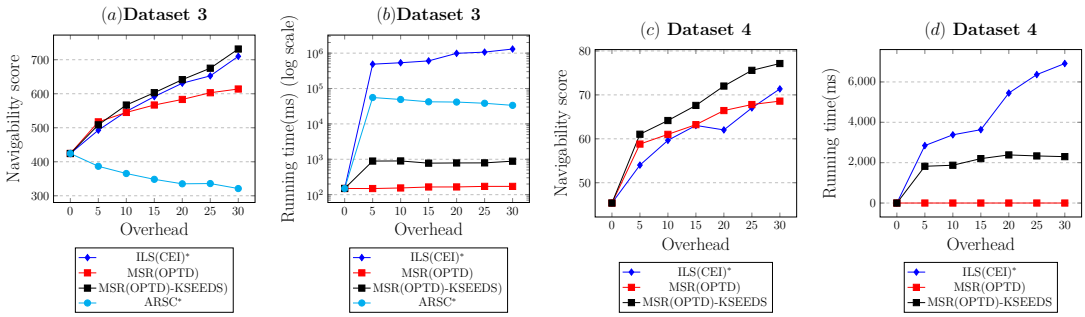


Figure 3.10: Score and running time values with increasing overhead for **Datasets 3 and 4**; Random distribution of scores with 60% navigable edges; $K = 4$ for MSR(OPTD)-KSEEDS

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 3.10(b) and 3.10(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.

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 3.11 shows the results for dataset 4. We considered two seed paths, the shortest path (Figures 3.11(c) and 3.11(d)) and the seed path selected by ILS(CEI) (Figures 3.11(a) and 3.11(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.

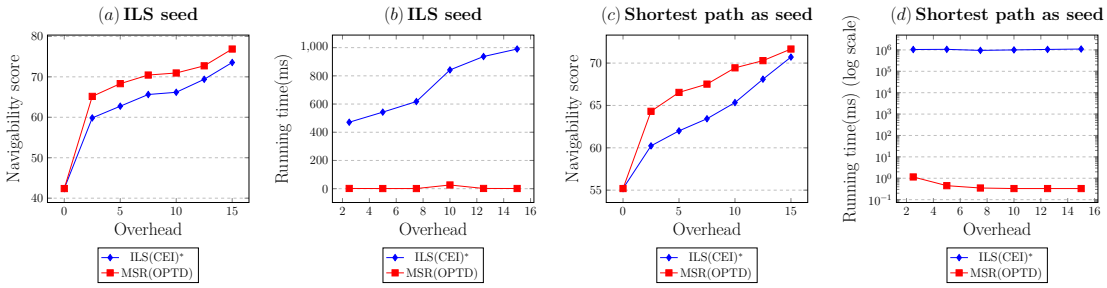


Figure 3.11: Applying the improvement algorithms of MSR(OPTD) and ILS(CEI)* on the same seed; Normalised distribution of scores with 70% navigable edges, Path length = 10 kms

3.5.3 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 3.4). 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 3.12. **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.

As desired, with increase in the difference between the scores assigned to the

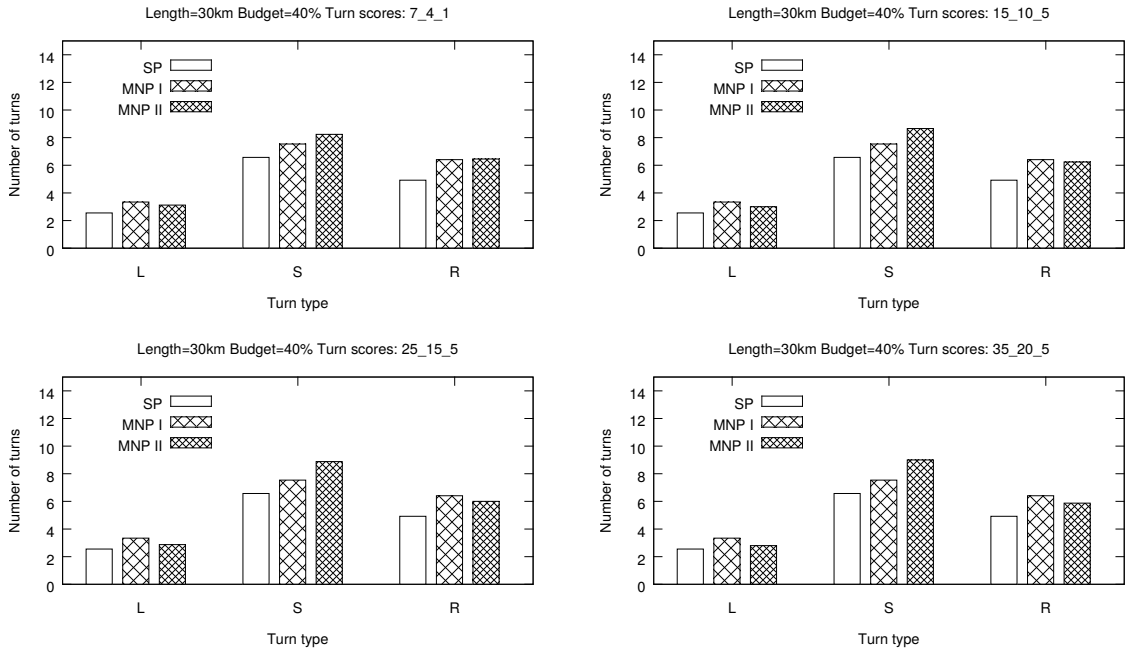


Figure 3.12: Turns of each type for increasing values of turn scores; Path length = 30 kms; Budget value = 40%

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 3.13 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**.

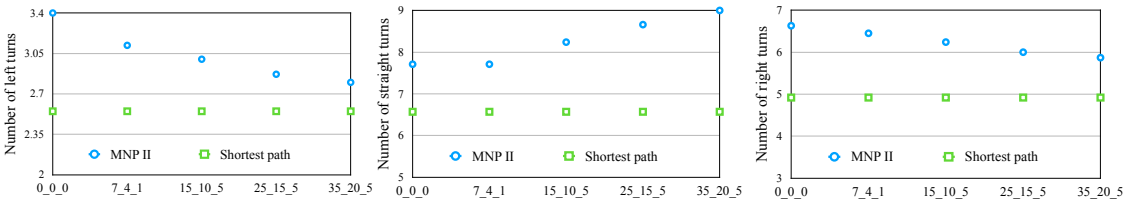


Figure 3.13: Turns of each type for increasing values of turn scores; Path length = 30 kms; Budget value = 40%

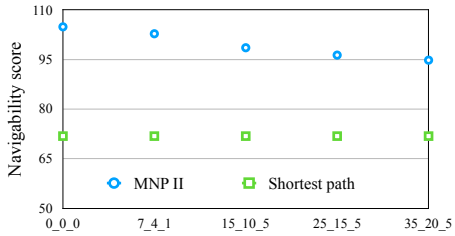


Figure 3.14: Score as a function of fixed preference value

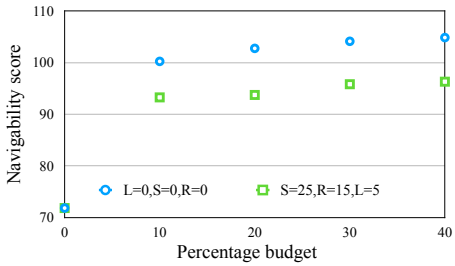


Figure 3.15: Score vs. budget for MNP II and SP

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 3.14 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 3.15 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%.

3.5.4 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 3.16 shows the three routes suggested by Google Maps for traveling from IIIT-Delhi to Kandy's Pastry Parlour. The routes are ranked

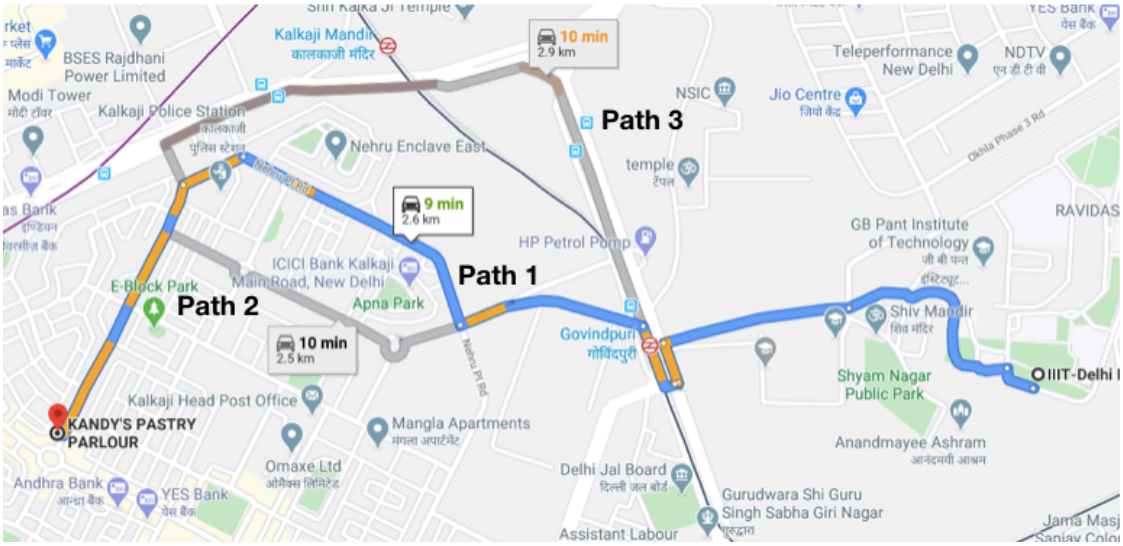


Figure 3.16: Example query on Google Maps

Figure 3.17: Subgraph with 20 nodes and 44 edges

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 3.17, 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).

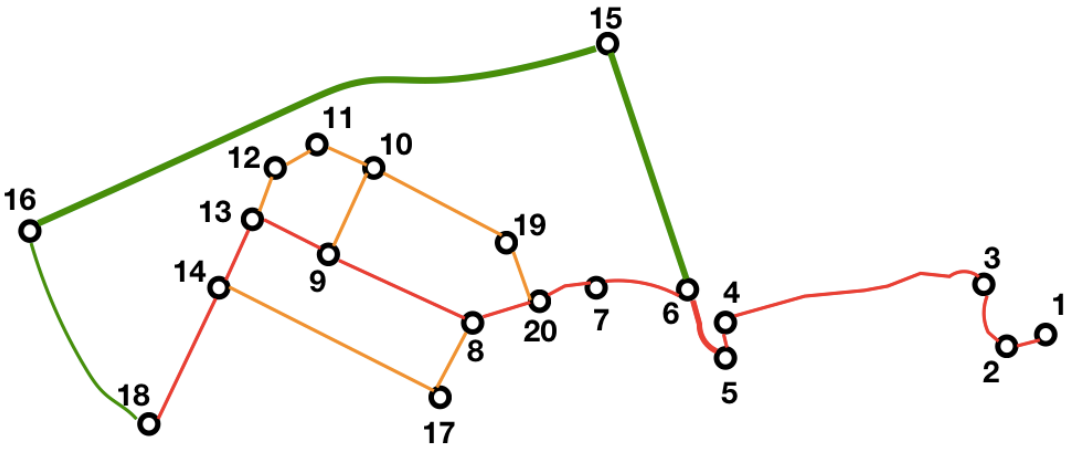


Figure 3.18: Labelled Graph

Further, we annotate this graph and assign navigability scores and costs to its road segments, as shown in Figures 3.18, 3.19 and 3.20 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.

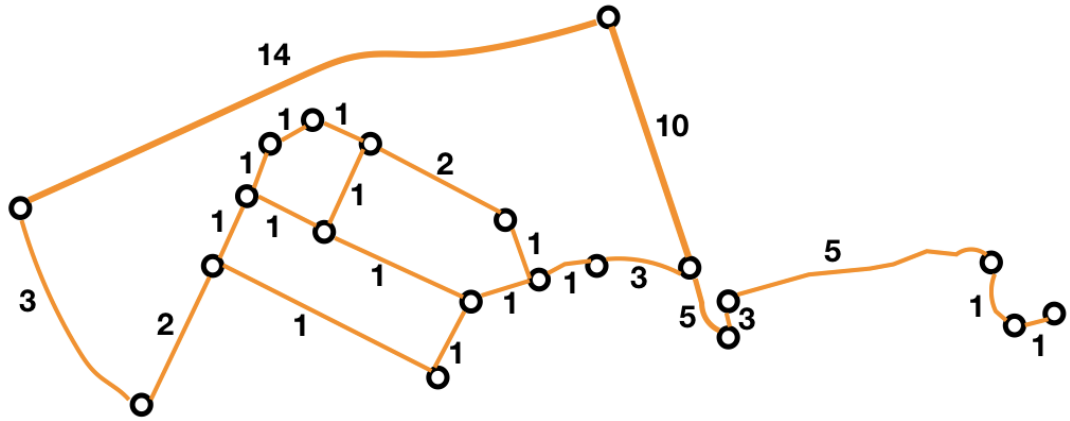


Figure 3.19: Navigability Scores

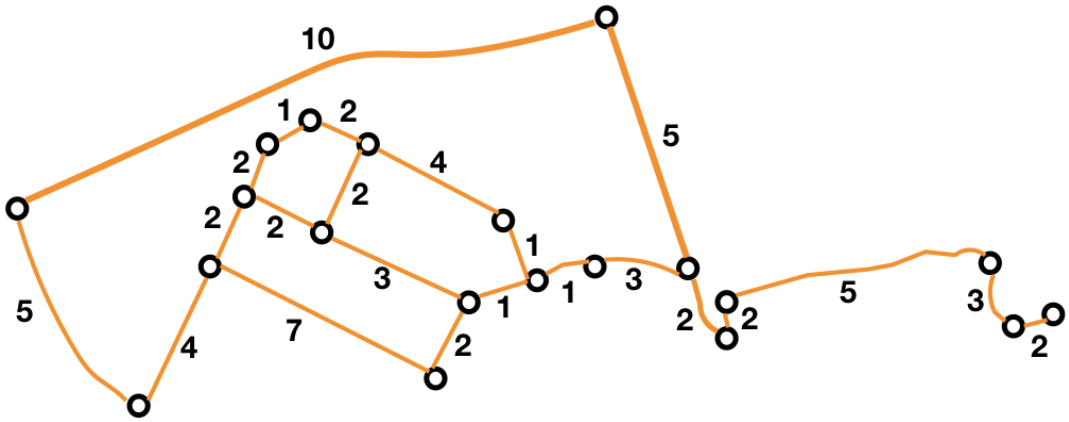


Figure 3.20: Costs

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.

3.6 Conclusions

In this study, we work on Constrained Path Optimization for urban road networks. We proposed advanced algorithms for the Most Navigable Path problem. We conduct more a more extensive experimental analysis, that includes a new and a much larger dataset, a different distribution for the navigability scores on the edges, a more extensive experimental evaluation, and a real-life case study for analysing the effectiveness of our algorithms.

CHAPTER 4
A NAVIGATION SYSTEM FOR CONSTRAINED PATH OPTIMIZATION

4.1 Introduction

In this work, we put together a system that suggests **"Safe Routes"** to travelers. Our objective is to find a route between the source and the destination which has a high "safety level". This is in stark contrast to the state-of-the-art navigation systems that are focused on computing the shortest or the fastest route to the destination. It is important to note that the goal of maximizing "safety" may lead to fetching of longer routes. However, this may not be desirable as a traveler would not be willing to compromise on the length of the route indefinitely. Therefore, we model our problem as a constrained maximization problem. A detailed introduction of the problem is presented in Chapter 1. In this chapter, we formally define our problem, present the related work, followed by the presentation of our navigation system.

4.2 Problem formulation

Problem definition: Given as input, (1) a graph representing the road network, where each edge is associated with a *cost* value and a *safety score*, (2) a source s and a destination d (3) a threshold on the total cost of the solution, the *budget* value; the goal is to find a route from s to d that maximizes the sum of *safety scores* of its edges, where the sum of *costs* of edges of the route is no more than the input *budget* value.

Challenges: Our problem can be reduced to an instance of the Arc Orienteering Problem, which is a well known NP-hard problem [53]. The problem is challenging

due to the conflicting requirements of maximizing the safety score while constraining the cost. As explained in [3], trivial modifications of minimization based strategies (e.g., Dijkstra's algorithm and its variants) would not be suitable for our problem.

There were challenges involved with the data collection, processing and visualization phases, that were resolved by making simple assumptions. This was done as our objective was to develop a prototype that checks the applicability of our algorithms proposed for the MNP, for the application of Safe Routing. The main challenges involved during the data collection and processing phase pertained to the quality of data, since the data collected was noisy (included missing and duplicate points) and imbalanced. For instance, for the image data, the aim was to collect the images for the streets in Delhi. However, the images we fetched using the Google Street View API represented a highly imbalanced dataset. Due to the restrictions imposed by the Indian government, the images fetched belonged to only a few colleges and historical monuments in Delhi, that could not be used as a representative of the entire region of Delhi. Therefore, we decided to use the Google Places API to fetch the images, wherein the images were uploaded mostly by the general public. This dataset included both indoor and outdoor images, and so, we used the Places 365 CNN to filter out the indoor images from the dataset. For the crime data collected from the news websites, a challenge we faced was in determining the correct location of the crime incident from a news article due to mention of several locations in each article. For this, we assumed that the location mentioned in the first sentence of an article corresponds to the location of the crime incident (confirmed manually for around one hundred articles).

Contributions: We build a navigation system for computing "Safe Routes", that is suited for vehicular and pedestrian navigation. For our study, we targeted the city of Delhi. An essential input to a "Safe Routing" system is data that can be reliably used as a proxy for public safety at the street level. To compute the street level safety,

we extracted data from various publicly available resources like government websites, news websites, and social media platforms. To transform the data into a usable form for safe routing, we applied processing techniques using tools from machine learning, natural language processing, and geocoding. For computing the “Safe Routes", we apply the algorithms proposed in an earlier work [3].

4.3 Related Work

Some researchers have started exploring the concept of safe routing. We mention some of these works in this section. It is important to note that none of these works model the problem as a constrained maximization problem, which is essential for our application domain.

A. M. de Souza et. al. [54] in their work consider a re-routing service based two factors, the vehicular traffic conditions and the public safety issues. For the safety level computation, the data was collected from official crime records from public safety departments, participatory sensing applications, social media and news websites. The work classifies the crime incidents into different crime types. Weights are assigned to each class of crime based on its complexity. For each road, a weighted mean is done for the number of crime incidents of each type on that road. The proposed algorithm computes a Pareto set of routes based on the two factors considered, traffic conditions and safety risk. The final route is selected using Boltzmann selection in a non-deterministic manner to avoid traffic overflow at certain roads. The experimental analysis was done for a small fragment of the Sao Paulo city of Brazil.

E. Galbrun et. al. [55] consider the problem of suggesting safe and short paths to travellers based on the civic datasets of criminal activity. To compute the risk model, they normalise the criminal activity in a region based on its population density (from GPS traces). The work proposes algorithms that compute a set of non-dominated paths based on varying trade-offs between the safety level and distance. The perfor-

mance of the algorithms is analysed for the cities of Chicago and Philadelphia.

J. Kim et. al. [56] propose a routing service that suggests safe routes to travellers, based on geotagged tweets from the social media platform Twitter. Their system consists of the following modules: data filtering, sentiment analysis, regional clustering, route finding, and visualization. For each tweet a sentiment value is assigned in the range -1 to 1 to its corresponding location. The clustering phase clusters the entire region into rectangular boxes based on the sentiment values. The routing algorithm computes a baseline route, and later modifies it repeatedly to remove regions with high value of negative sentiments from the baseline route. They did the experimental analysis for the city of Chicago.

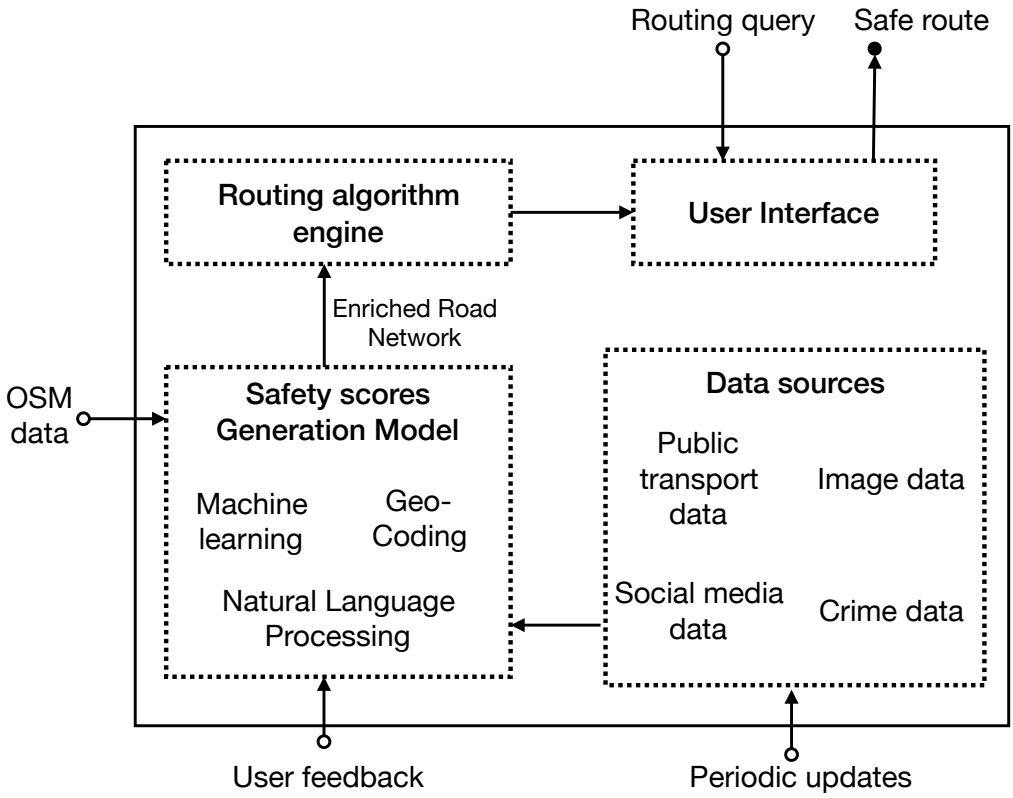


Figure 4.1: System Overview

4.4 System Overview

4.4.1 Data sources

The data sources we use can broadly be classified into four categories: crime data, public transport data, social media data, and image data (as shown in Figure 4.1). More specifically, we extract the following data: (1) **Crime data from news websites**: We scrape news articles for crime incidents for the period 2016 to 2020 for the region of Delhi from the Times Of India (TOI) website. (2) **Crime statistics data from the NCRB website**: We consider statistical data from the 2019 report for district-wise crime rates in Delhi. (3) **Public transport data**: We collect data from the Delhi Integrated Multi-Modal Transit System Ltd. website to get the live locations of the public transport buses in Delhi. This data is processed to find areas with a poor GPS connectivity (4) **Traffic data**: We scrape tweets from the Twitter handle of Delhi Traffic Police to get the dynamic congestion-related information on a daily basis. (5) **Image data**: We fetch images for streets across Delhi (uploaded mostly by the general public) using the Google Static Street View API. (6) **Road quality data**: This data was downloaded from the Public Works Department (PWD) website to gather the complaints registered by citizens regarding the quality of streets, which is updated every month. We extracted complaints related to the following safety-related issues: malfunctioning street lights, poor sanitation (open potholes and drain covers) and poor quality road segments (broken flyovers, roads and footpaths). (7) **Police stations coordinates**: We also consider the locations of Police stations across the city as one of the parameters for safety score computation.

4.4.2 Data processing

To gauge the level of safety on the streets, we process the extracted data using different techniques. The details of this phase are as follows: (1) **Image data**: The

street view images were first classified into two categories, indoor and outdoor images. The outdoor images were further processed to predict their scenic attributes, such as whether the site is a residential area, a construction site, a bus stop, a slum area, etc., from a fixed list of attributes. These tasks were done using the Places-365 CNN [57]. (2) **Road quality data**: We identified two patterns in the complaints registered on the PWD website, complaints that refer to a single location (e.g., the street light is not working at location X), and complaints that refer to more than one locations (example: street lights are not working from location X to location Y). To extract the location(s) from the text, we performed Named Entity Recognition using the SpanBERT model [58] trained on addresses of Delhi. The model generates a tag for each word of the sentence, and identifies the words representing spatial locations. Finally, we used HERE Geocoding Search API to find the longitude and latitude of the specified location. (3) **Public transport data**: We process the live feed coming from the public transport buses in Delhi to find areas with a poor GPS connectivity. Buses (belonging to Delhi Municipality) send signals (their current location) to a central server at periodic time intervals using a GPS system which is mounted on them by the Delhi Muncipality authorities. We track the areas where the buses do not send signals by evaluating the time lapse between successive signals. The areas that are reported by at least ten buses are considered as having a poor GPS connectivity. (4) **Traffic data**: Most of the tweets scraped from the Twitter handle of Delhi Traffic Police were not geotagged. We geotag these tweets using the SpanBERT model and the HERE Geocoding Search API, to compute the geo-coordinates of congested locations from the textual description. (5) **Crime data from TOI**: To process this data, we first lemmatise the text in the news articles using the spaCy library. We identify the location of the crime incident using the SpanBERT model and the HERE Geocoding Search API. We also extract the type of crime incident using the Smote classifier [59]. The crime incidents in Delhi were found to belong to fourteen

categories, viz. riot, rape, murder, kidnapping, trafficking, assault, scam, robbery, cybercrime, harassment, missing person, roadrage, gangwars and terrorism. In our model for safety scores generation, we do not use the information regarding the crime type. We only use the information regarding the presence or absence of a crime incident at a specific location.

4.4.3 Safety scores generation

The input to this module includes the road network data (from OpenStreetMap (OSM)) and the safety-related data after the processing phase described in Section 4.4.2. In this phase, we compute the impact of the previously mentioned data sources on the safety scores of the road segments. For e.g., proximity to a police station or being a part of a residential area positively affects the safety score of a road segment. In contrast, if the road segment passes through deserted regions or does not have proper lighting, its safety score decreases. The details of the computation of the safety scores of the road segments are described next.

First, each relevant data point from the previously mentioned data sources is map-matched to the road segments present in its spatial neighborhood. For example, in case of Police stations data source, we attach the locations of all the police stations (e.g., Pahar Ganj Police station located at 28.6456° N, 77.2089° E) to their nearby road segments. After all the relevant data points have been mapped to their corresponding road segments present in their spatial neighborhood, we compute the safety score of each road segment.

Table 4.1: Dataset Statistics

City	**Vertices**	**Edges**	**Safety score range**
Delhi	11,399	24,803	[0,15]

Initially, we assign a safety score of eight to all the road segments (median in the range 1 through 15). Following this, we consider the impacts of the data points

mapped to the road segment. For example, if a road segment has a police station nearby (mapped to it), its safety score value is incremented by value three. On the contrary, if the road segment passes through a deserted region (if indicated by one of the attributes of image data post the processing phase), its safety score value is decremented by value two. Likewise, we compute the impacts of all the data points on a road segment to compute its safety score. If the computed score is less than zero, we set its value to zero. This is done to ensure that the safety scores are non-negative (a requirement of our route generation algorithm). Note that we directly assign a score value of zero to all the road segments that have some crime incident data point mapped to them (least safety score value). In such a case, we don't consider the other data sources (e.g., police stations, etc.) for determining the safety score of the particular road segment. The range of the computed safety scores is [0,15]. Table 4.1 details the statistics of the dataset constructed.

4.4.4 Route generation

The road network enriched with the safety scores is passed to the Routing algorithm engine module. "Safe Routes" are computed using the **Multiple Segment Replacement** algorithm proposed in [3]. The algorithm employs a four-step framework: (i) finding a seed path from s to d, (ii) estimating the potential of segments of the seed path for safety score improvement, (iii) selecting a set of segments for replacement with the objective of improving the *safety score*, (iv) replacing the chosen segments in the seed path while checking the *budget* constraint. The query parameters (s , d, algorithm and *budget*) are specified by the user through the User Interface (UI). The UI then shows a visualization of the resulting routes, the Shortest Route and the "Safe Route".

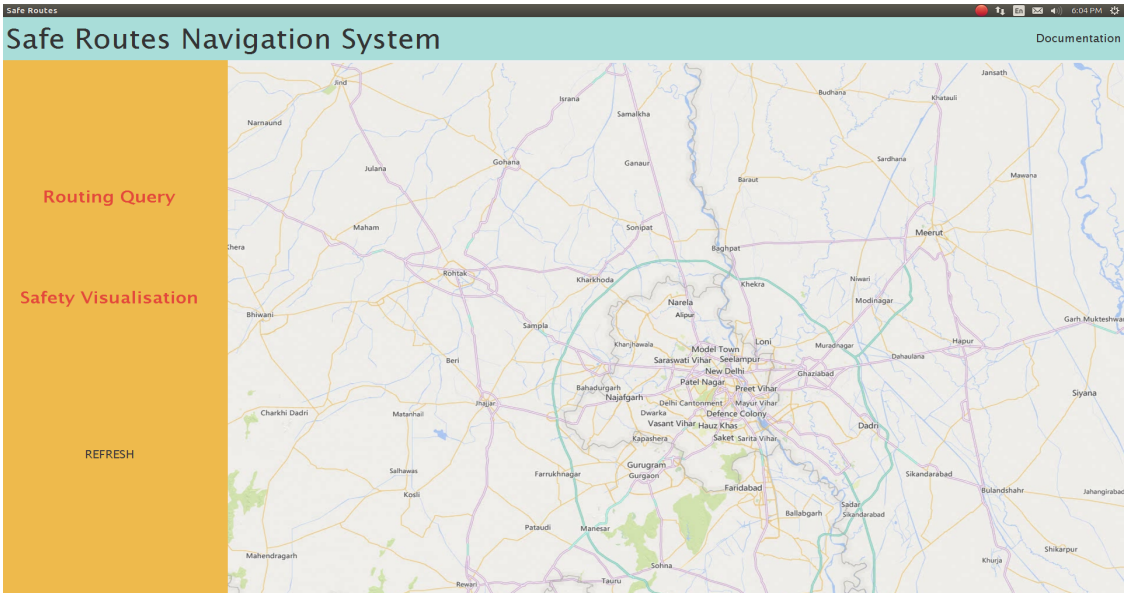


Figure 4.2: User Interface illustration

4.5 DEMONSTRATION OUTLINE

Figure 4.2 demonstrates the UI of the Safe Routes Navigation System, designed in Java. The features offered by the system include routing queries and safety visualization. As part of the **Routing Query** feature, the user can submit a routing query and view the shortest route and the “safe route" between the desired source and destination. The budget parameter can be varied as per the user requirement. We denote the budget parameter as a percentage of the length of the shortest route. A budget value of 10% implies the constraint that the length of the safe route should not exceed the length of the shortest route by a factor of 10%. The shortest and the safe routes for a sample query are shown in Figures 4.3 and 4.4 respectively.

The road segments in these figures are coloured to denote their level of safety, as per the colour coding scheme shown in the legend (top-left corner of the map region).

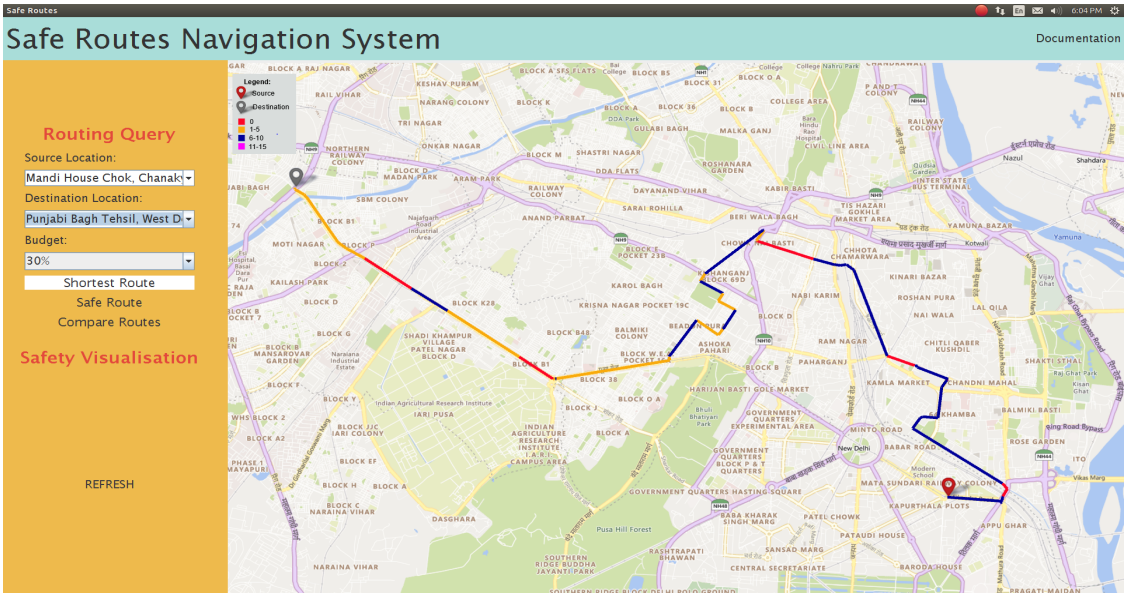


Figure 4.3: Shortest route illustration

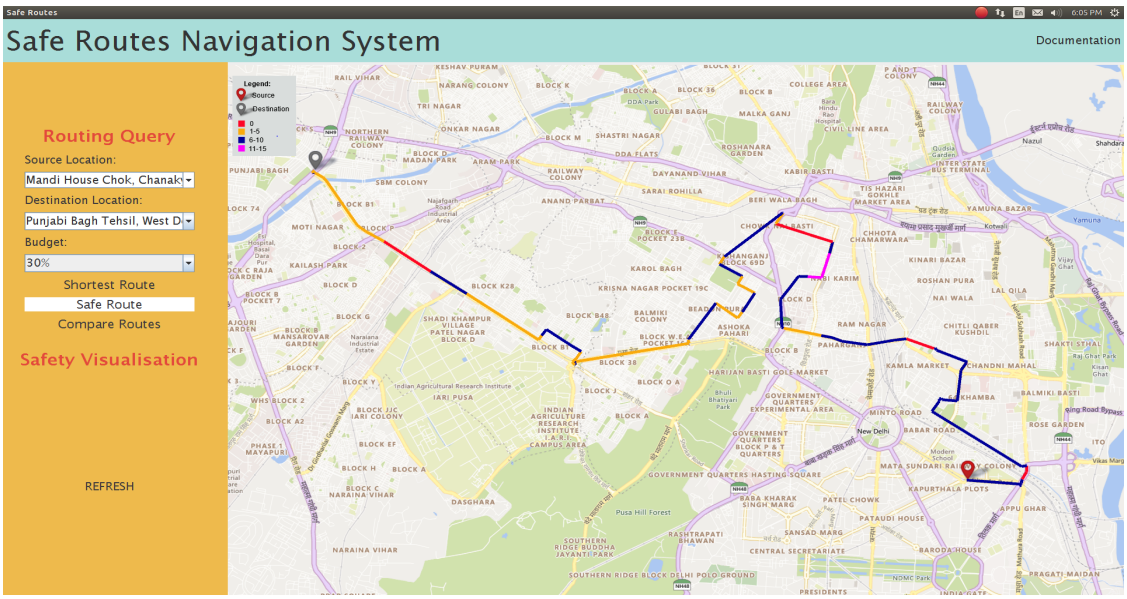


Figure 4.4: Safe route illustration

The user can also view a comparative analysis of the two routes (as depicted in Figure 4.5). This option allows the user to view the lengths of the two routes, their safety scores, and the computation times for computing the shortest and the safe routes.

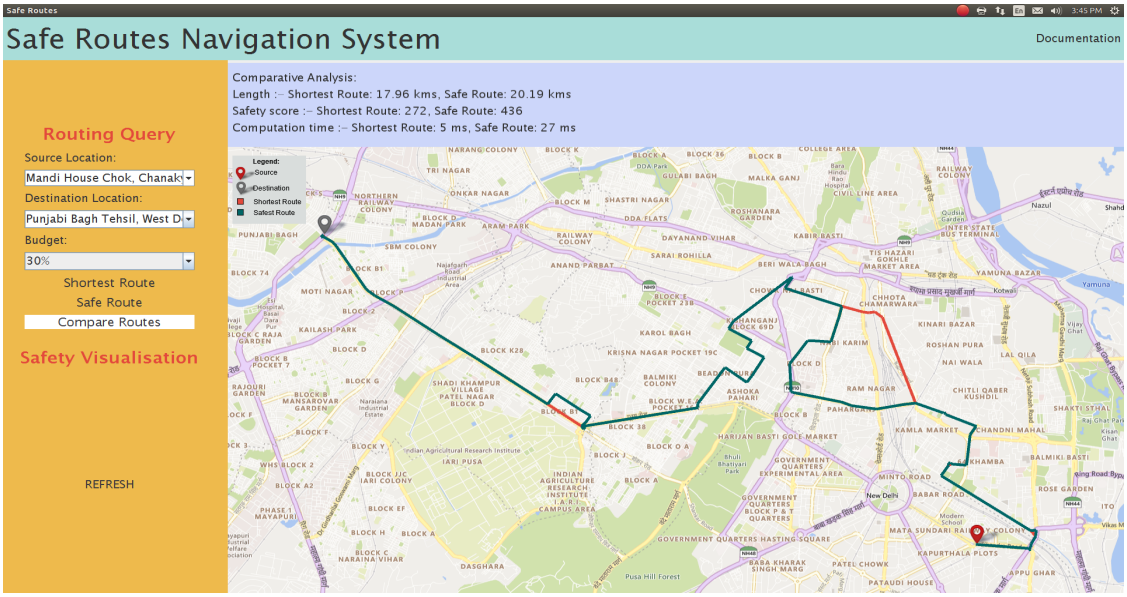


Figure 4.5: Comparing the shortest and safe routes

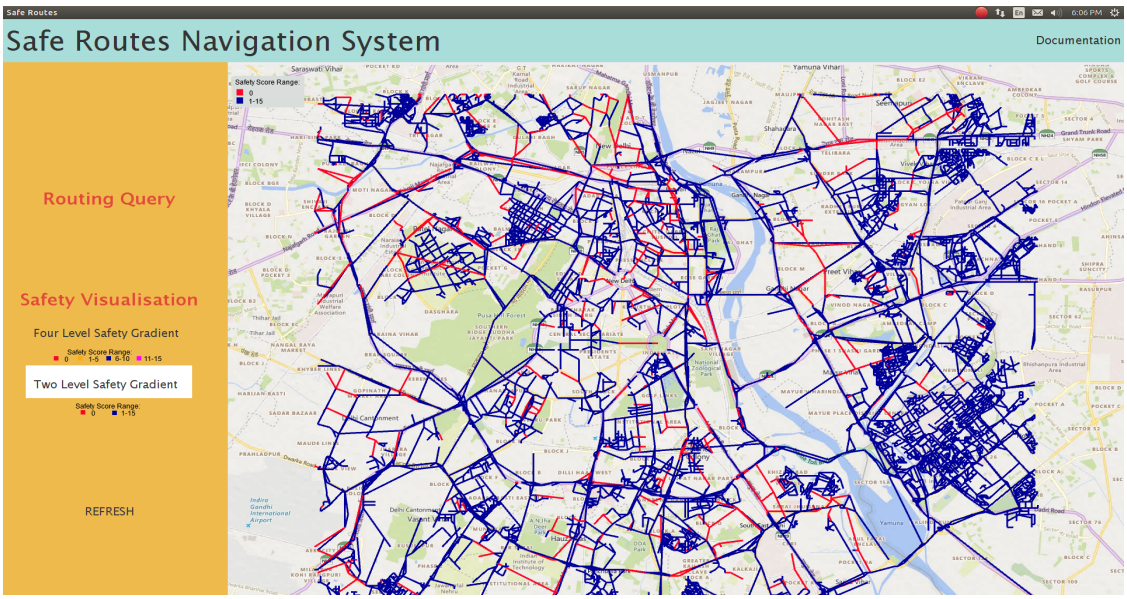


Figure 4.6: Two level safety gradient visualisation

Under the **Safety Visualization** feature, the user can view the distribution of safety levels across the road network. The two level safety gradient shows a two-color visualization with different colours for edges with zero and non-zero scores respectively. Similarly, the two and four level safety gradients illustrate a two-colour and

four-colour visualizations of the edges in the road network respectively (as depicted in Figures 4.6 and 4.7).

Here, the road segments are coloured to denote their level of safety, as per the colour coding scheme shown in the legend. Lastly, the user can **Refresh** the map to clear previous results, and also view the **Documentation** regarding application usage.

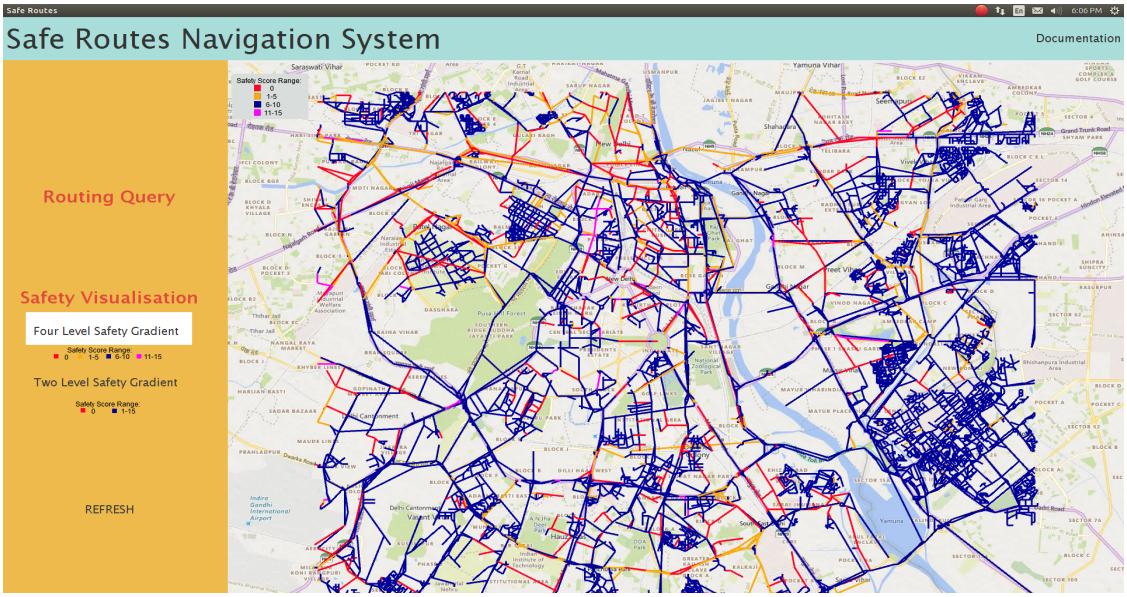


Figure 4.7: Four level safety gradient visualisation

4.6 Conclusions

In this work, we build a navigation system for Constrained Path Optimization in urban road networks. We propose a novel system that recommends "Safe Routes". We gather data from various publicly available resources to compute the street level safety for the city of Delhi. We process the gathered data and generate safety scores for all road segments in the road network. Routes recommended by our system balance the conflicting requirements of *increasing the safety* and *constraining the total length* of the path to be within a reasonable limit, as desired by the user.

CHAPTER 5
ALGORITHMS FOR TASK ASSIGNMENT IN MACRO-LEVEL TRANSPORTATION

5.1 Introduction

In this work, we propose algorithms for **Task Assignment in Macro-level transportation** that consider an egalitarian perspective. As a use-case for Macro-level transportation, we consider the application domain of **Spatial Crowdsourcing (SC)** (e.g., taxi-hailing services, online food ordering services, etc.). We attempt to minimise the maximum waiting time for any customer of the platform, the maximum idle time for any driver of the platform, along with maximisation of the profit of the platform. We consider a fully-online scenario, i.e., both the workers and customers appear dynamically on the platform. We also consider the existence of deadlines for both the entities.

A detailed introduction for this problem is presented in Chapter 1. In this chapter, we first present a formal problem in Section 5.2. Section 5.3 presents a brief summary of the related work. In Section 5.4, we present the baseline algorithms and the proposed algorithms. The experimental evaluation of the proposed algorithms is presented in Section 5.5. Finally, we conclude our findings in Section 5.6.

5.2 Basic concepts and problem definition

First, we define the key terms relevant to our SC problem from the context of a taxi-hailing service.

Definition 4. ***Customer:*** *A customer, denoted by* $c =< s_c, t_c, at_c, et_c >$*, has source location* s_c*, destination location* t_c*, arrives on the platform at timestamp* at_c*, and is*

available on the platform for time duration et_c. Post the timestamp $at_c + et_c$, the customer request expires.

Definition 5. ***Driver:*** *A driver, denoted by $d =< s_d, t_d, at_d, et_d >$, has source location s_d, destination location t_d, arrives on the platform at timestamp at_d, and is available on the platform for time duration et_d. Post the timestamp $at_d + et_d$, the driver is no longer available on the platform.*

We denote the number of drivers available on the platform by $|D|$, where D denotes the set of incoming drivers.

Definition 6. ***Driver's Idle Time:*** *The idle time of a driver d, denoted as $idle_d$, between two successive assignments is defined as the time difference between d's previous customer's drop time (or the start of service time in case of first assignment of the driver) and the intimation of assignment of his/her current customer's ride request.*

Definition 7. ***Customer's Waiting Time:*** *The waiting time of customer c, denoted as $waiting_c$, corresponding to his/her assignment to a driver, is the time difference between the customer's arrival time and the time he/she is picked up by some driver.*

Definition 8. ***Matching:*** *A Matching M, is defined as a set of pairs (c, d), where $c \in C$ and $d \in D$. The number of pairs in a matching M is denoted by $Cardinality(M)$. The cost of a matching M is defined in Equation 5.1. Here, the parameters α and β denote the weights corresponding to the driver's idle time and the customer's waiting time respectively, in Equation 5.1.*

$$Cost(M) = \alpha \arg\max_{(c,d)\in M} idle_d + \beta \arg\max_{(c,d)\in M} waiting_c \tag{5.1}$$

5.2.1 Problem Statement

Input consists of:

(1) A stream of incoming customers C

(2) A set of drivers D who also appear on the platform dynamically

Output A matching M

Objective function: Equation 5.2 defines the objective function for our problem. The parameter γ denotes the weight corresponding to the number of drivers in Equation 5.2.

$$\textbf{\textit{Maximize}}\, Cardinality(M) - (Cost(M) + \gamma * |D|) \tag{5.2}$$

Constraints:

1. **Deadline constraint**: A customer request, $c =< s_c, t_c, at_c, et_c >$, can be paired until timestamp $at_c + et_c$. Likewise, a driver, $d =< s_d, t_d, at_d, et_d >$, can be paired until timestamp $at_d + et_d$.

2. **Invariable constraint**: Once a matching pair is formed, it cannot be broken. For example, in the case of taxi-hailing service, once a driver is assigned to a customer by the task assignment algorithm, the specific customer-driver pair in the matching M remains unchanged thereafter.

The reason behind subtracting the number of drivers in Equation 5.2 is as follows. The objective function for egalitarian task assignment must also cater to the expectation of the platform owner. We ensure this by maximising the cardinality of the assignment, since each accepted request generates a revenue for the platform. However, in order to increase the value of the cardinality one may increase the number of drivers on the platform. Although a larger number of drivers on the platform might lead to a better cardinality, but the same also implies a greater workforce that

has to be paid, which would reduce the profit earned by the platform. Therefore, we introduce the term $|D|$ in the subtraction component in Equation 5.2.

Computational Challenges The challenges in solving our problem are as follows:

1. Fully online setting: In our problem setting, the spatial locations and arrival times of the workers and customers are not known beforehand. This makes the problem challenging as the offline algorithms for finding the optimal solution cannot be used.

2. Dual objectives: Another factor that adds to the complexity of the problem is the presence of dual objectives (reducing the waiting time and maximising the number of assignments), thereby creating a trade-off between the two.[1]

3. Trade-off between the customer and worker expectations: Our problem considers another set of conflicting objectives, maximizing both the customer and worker expectations, thus making our problem more challenging.

4. Deadlines of customers and workers: Lastly, the deadlines of both entities makes the existing online bipartite matching algorithms inapplicable in our setting.

Our Contributions: Our contributions in this work are as follows:

1. We introduce a novel problem of SC for egalitarian task assignment. Our problem aims to reduce the waiting time for both entities of the SC platform, while trying to maximise the profit earned by the platform, in a fully-online setting along with the existence of deadlines for both entities.

2. We propose two heuristic algorithms to solve our SC problem, and analyse the complexity of our proposed algorithms.

[1] A solution could refuse all the customers requests, which would make the maximum value of a customer's waiting time to be zero, but this would cause the maximum value of a worker's waiting time to reach its maximum possible value.

3. We implement a taxi-hailing framework that mimics the real-life mobility model of workers on an SC platform. We employ our framework to implement a baseline greedy algorithm, a state-of-the-art algorithm for the fully-online bottleneck matching problem with deadlines (the Local Isolated Point Greedy algorithm (LIPG) [60]), and the two proposed algorithms.

4. We extensively evaluate the performance of the proposed algorithms on a real taxi-trips records dataset. Our analysis shows that our Stable Matching based algorithm is well-suited for the problem considered, and performs better than the LIPG algorithm and the baseline greedy algorithm.

5.3 Related Work

Task assignment in SC in a widely studied problem. The existing studies in this domain may be divided into two categories based on the objective function of the problem being addressed:

(I) **General online matching in Spatial Crowdsourcing**: Most of the studies in this category are either based on the maximisation of the cardinality of the assignment [61] [62] or maximisation of the profit earned [63] [64] or minimisation of the cost incurred [65] or minimisation of the distance/time [66] [21][25]. However, none of these studies consider an egalitarian objective function in a fully online setting along with the deadlines of workers and customers. For instance, S. Ji et. al [21] work on reducing the waiting time for the users of a food delivery service. The authors model the problem as a task grouping problem, and propose heuristic algorithms in their solution. However, their work does not consider the waiting time for the delivery personnel on the platform, and it therefore classifies as a utilitarian objective.

Another study done by L. Li et. al. [25] does consider the minimisation problem from an egalitarian perspective. They address the minimisation of the waiting time of both workers and 'requesters' in a two-sided online setting of spatial crowdsourcing.

However, their work does not consider deadlines for the customers and workers, which is an important aspect in a real-time taxi-hailing service. A customer would not like to wait for too long to get a response from the platform, and the availability of drivers would also vary across the day. Therefore, it is important to associate a deadline with the customer requests, as well as with the availability of drivers.

(II) **Bipartite bottleneck matching in Spatial Crowdsourcing**: The studies in this category are based on minimisation of the bottleneck cost of the assignment [20][25][60]. Similar to the case of general online matching, none of the existing studies in this category consider an egalitarian objective function in a fully online setting along with the deadlines of workers and customers. For example, Z. Chen et. al. in their work [20] address an SC problem with a two-sided online setting with the objective to minimize the worst case waiting time for any user. The waiting time in the study, is defined as the sum of the assignment time and the travel time from the worker to the customer. In other words, they do not consider the waiting time for the workers of the platform.

A work that is closely related to our problem is that by L. Li et. al. [60]. Their work considers a fully-online bottleneck matching problem along with deadlines. However, their work considers the maximum distance between the driver and the customer of all matching pairs. Semantically, this definition considers only the waiting time for the customers on the platform, and not the waiting time (i.e., the idle time) for the drivers (and therefore, they do not consider an egalitarian perspective). On the contrary, our work considers the waiting time of the customers, as well as the idle time of the drivers. This makes our problem more challenging as we have to consider the trade-off between the waiting times of the two entities.

Note on Data-driven approaches for taxi demand prediction: It is important to acknowledge the work done in the area of passenger demand prediction for taxi drivers (e.g., [67–73]). Researchers in this area have addressed the problem from

multiple facets. For e.g., [68, 70, 72, 73] proposed techniques for recommending potential locations for picking-up passengers. Similarly, [69] proposed techniques which recommend route for idle taxi drivers where they are more likely to pick-up passengers. [74] and [75] propose a Markov Decison Process based model to recommend the direction in which an idle driver should move in order to maximise his/her total expected revenue over a given time period. The parameters of the model are learned from the historical trip-records data. [71] proposed techniques which can predict the waiting time for getting a taxi at different locations. A key aspect in all these works being that they mine historic data on passenger pick-up and drop-off locations and, taxi route data for recommending potential pick-up locations and route to taxi drivers. It is important to note that our work is complementary to the work done in the area of taxi demand prediction. Our work takes the locations of the taxis as an input to make task assignment in an egalitarian fashion. Taxi drivers may internally use the previously mentioned data driven approaches for determining ideal locations to wait for the customers.

5.4 Baseline and Proposed algorithms

In this section we describe the baseline algorithms and our proposed algorithms.

5.4.1 Baseline algorithms

We consider two algorithms as the baseline algorithms for analysing the performance of our proposed algorithms.

(A) **Greedy approach:** We consider the greedy assignment as our first baseline algorithm. The idea behind the processing of requests using a greedy strategy is simple: for each request in a batch, we find the driver nearest to that request, and make an assignment between them. The pseudocode in Algorithm 7 shows the steps of the Greedy strategy.

Algorithm 7 Greedy algorithm

Input: Set of customer requests C, Set of available drivers D
Output: Matching M

1: Delete the expired customer requests and drivers from C and D respectively
2: **for** Each customer $c \in C$ **do**
3: $\quad d \leftarrow$ The nearest available driver of c
4: $\quad M \leftarrow M \cup (c, d), D \leftarrow D \setminus d$
5: **end for**

Time Complexity Analysis: The time complexity of the first step of the algorithm involving the deletion of expired customer requests and drivers, takes $O(C) + O(D)$ time, where C and D represent the set of customer requests and drivers available at the time of processing the batch of accumulated requests. The for loop in lines 2 through 5, takes $O(|C||D|)$ time. Thus, the overall time complexity of the Greedy algorithm is $O(|C||D|)$, and when $|C| = |D| = n$, the time complexity is $O(n^2)$, which is polynomial in n.

(B) Local Isolated Point Greedy (LIPG) algorithm: We consider the LIPG algorithm [60] as our second baseline algorithm. The algorithm was proposed for a fully-online bottleneck matching problem along with deadlines, a problem from the related work that is most closely related to the problem we consider in this work.

The pseudocode in Algorithm 8 shows the steps in the LIPG algorithm. The algorithm first deletes the expiring customer requests and drivers. Next, it computes the average pairwise distance between all customers and drivers (denoted by δ). Further, the Thresholded Local Density (TLD) is computed for all customers and drivers. TLD of a customer c refers to the number of drivers within a specific distance from c (this distance is proportional to δ). The TLD of a driver is defined along similar lines. The for loop in lines 4 through 11 then makes the assignments for all expiring customers. If the TLD of a customer is zero, it is assigned to its nearest driver. Otherwise, it is assigned to the driver with the lowest TLD from the set of candidate drivers (the set of drivers within some specified distance proportional to δ). Similarly,

Algorithm 8 Local Isolated Point Greedy algorithm

Input: Set of customer requests C, Set of available drivers D
Output: Matching M

1: Delete the expired customer requests and drivers from C and D respectively
2: $\delta \leftarrow$ the average pairwise distance between all customers and drivers
3: Compute the TLD for all customers and drivers ▷ based on the value of δ
4: **for** Each customer $c \in C$ reaching its deadline **do**
5: **if** TLD of $c = 0$ **then**
6: $d \leftarrow$ The nearest available driver of c
7: **else**
8: $CW \leftarrow$ Candidate set of drivers of c
9: $d \leftarrow$ The driver in CW with lowest TLD
10: **end if**
11: $M \leftarrow M \cup (c, d), D \leftarrow D \setminus d$
12: **end for**
13: **for** Each driver $d \in D$ reaching its deadline **do**
14: **if** TLD of $d = 0$ **then**
15: c $\leftarrow$ The nearest available customer of d
16: **else**
17: $CT \leftarrow$ Candidate set of customers of d
18: $c \leftarrow$ The customer in CT with lowest TLD
19: **end if**
20: $M \leftarrow M \cup (c, d), C \leftarrow C \setminus c$
21: **end for**

the assignments for all expiring drivers are made in the for loop from line 13 until line 21.

Time complexity analysis: The time complexity of the LIPG algorithm is $O(|C||D|)$. When $|C| = |D| = n$, the time complexity is $O(n^2)$.

5.4.2 Proposed algorithms

(A) K Nearest Drivers based (KND) algorithm: A drawback of the Greedy and LIPG algorithms, from the context of our problem, is that they do not consider the idle time of the drivers of the platform. Therefore, in our first algorithm, we consider the idle time of the drivers along with the waiting time of the customers. The pseudocode in Algorithm 9 shows the steps in the algorithm. First, for each

customer request, its K nearest drivers are computed. The algorithm then pairs each customer with the driver who has been waiting for the longest time from the set of its K nearest drivers.

Time Complexity Analysis: The time complexity of the first step of the algorithm involving the deletion of expired customer requests and drivers, takes $O(C) + O(D)$ time, where C and D represent the set of customer requests and drivers available at the time of processing the batch of accumulated requests. The for loop in lines 2 through 7, takes $O(|C||D|)$ time. Thus, the overall time complexity of our KND algorithm is $O(|C||D|)$. When $|C| = |D| = n$, the time complexity is $O(n^2)$.

Algorithm 9 K Nearest Drivers based algorithm

Input: Set of customer requests C, Set of available drivers D, K
Output: Matching M

1: Delete the expired customer requests and drivers from C and D respectively
2: **for** Each customer $c \in C$ **do**
3: $N \leftarrow$ The K nearest drivers of c
4: Compute the idle time of the drivers in N
5: $d \leftarrow$ The driver with the highest idle time in N
6: $M \leftarrow M \cup (c, d)$, $D \leftarrow D \setminus d$
7: **end for**

(B) Stable Matching based algorithm: Our second algorithm is inspired by the algorithm for the stable matching problem. The motivation behind this algorithm is the existence of preference rankings in the algorithm for the Stable Matching problem proposed by Gale and Shapley [76]. We utilise the concept of preference rankings in the Stable Matching algorithm to incorporate the driver idle time and the customer waiting time in the preference function of the matching algorithm.

The traditional algorithm for the Stable Matching problem considers an equal number of both entities. However, in real-life applications such as a job application process or a taxi-hailing service, the cardinality of the two sets of entities may not be the same. Towards this end, we apply a trivial modification in the traditional

algorithm to make it applicable in our setting. First, we define the key concepts relevant to the working of our algorithm.

Preference function of drivers: Similar to the working of the traditional Stable Matching algorithm, our algorithm computes a preference ranking for both entities. The drivers rank the customers based on the preference function defined in Equation 5.3. The preference function for a driver d assigns a weight to each customer c based on two parameters: (i) the customer waiting time at timestamp t (denoted by $waiting_c^t$), that denotes the time duration for which the customer c has been waiting on the platform for a response on his/her ride request at timestamp t; (ii) the distance between the customer c and driver d, the function $dist_N(c, d)$ computes the normalised haversine distance between c and d.

$$Preference\,Weight_d(c, t) = \frac{waiting_c^t}{dist_N(c, d)} \tag{5.3}$$

Preference function of customers: Each customer c ranks the drivers available on the platform based on the preference function defined in Equation 5.4. The preference function for a customer c assigns a weight to each driver d based on two parameters: (i) the driver idle time at timestamp t (denoted by $idle_d^t$), that denotes the time duration for which the driver d has been sitting idle on the platform at timestamp t; (ii) the distance between the customer c and driver d, the function $dist_N(c, d)$ computes the normalised haversine distance between c and d.

$$Preference\,Weight_c(d, t) = \frac{idle_d^t}{dist_N(c, d)} \tag{5.4}$$

Computing the preference rankings (or preference lists): Once the preference weights have been computed for all the customers and drivers, the next task is compute the preference rankings. The preference list of each customer is the ordered list of all drivers sorted in decreasing order of their preference weights. Likewise, the

preference list of each driver is the ordered list of all customers sorted in decreasing order of their preference weights.

Algorithmic details: The pseudocode in Algorithm 10 lists the steps of our Stable Matching based algorithm. The algorithm begins with discarding the customer requests and drivers for which the deadline has expired. Next, the preference ranking of the remaining customers and drivers is computed. After the preference rankings have been computed, the procedure for the traditional Stable Matching algorithm may be called. However, the number of drivers and customers may not be the same. Towards this end, we make the following modification in the traditional algorithm: the entities in the set with lower cardinality make proposals to the entities in the set with higher cardinality (the if condition in Line 3). The rest of the algorithm proceeds in the traditional fashion. Finally, the failed requests are saved for the next batch of processing.

Time complexity analysis: The time complexity of the first step of the algorithm involving the deletion of expired customer requests and drivers, takes $O(C) + O(D)$ time, where C and D represent the set of customer requests and drivers available at the time of processing the batch of accumulated requests.

The second step of the algorithm entails computation of preference lists of the customers and drivers. This step takes $O(|C||D|)$ time. The matching procedure (Line 4 and Line 7) takes $O(|C||D|)$ time. Thus, the overall time complexity of our Stable Matching based algorithm is $O(|C||D|)$, and when $|C| = |D| = n$, the time complexity is $O(n^2)$, which is polynomial in n.

5.5 Experimental Evaluation

We evaluate the performance of the proposed algorithms on a real taxi trip records dataset. We first describe the dataset we used, followed by the details of the im-

Algorithm 10 Stable Matching based Algorithm

Input: Set of customer requests C, Set of available drivers D
Output: Matching M

Delete the expired customer requests and drivers from C and D respectively
Compute the preference ranking for all drivers $d \in D$ and for all customers $c \in C$
if $|C| <= |D|$ **then**
    Call procedure letCustomersPropose(C, D)
    $\forall c \in C, M \leftarrow M \cup (c,$ current partner of $c)$
else
    Call procedure letDriversPropose(D, C)
    $\forall d \in D, M \leftarrow M \cup$ (current partner of d, $d)$
end if

Procedure 1: letCustomersPropose(C, D)

Set the status of all $c \in C$ and $d \in D$ as free
while $\exists$ a free customer c **do**
    $d \leftarrow$ Highest ranked driver in the preference list of c to whom c has not proposed yet
    if d is free **then**
        (c, d) become engaged
    else
        $c^{'} \leftarrow$ current partner of d
        If d prefers $c^{'}$ to c, then c remains free, else (c, d) become engaged and $c^{'}$ becomes free
    end if
end while

Procedure 2: letDriversPropose(D, C)

Set the status of all $c \in C$ and $d \in D$ as free
while $\exists$ a free driver d **do**
    $c \leftarrow$ Highest ranked customer in the preference list of d to whom d has not proposed yet
    if c is free **then**
        (c, d) become engaged
    else
        $d^{'} \leftarrow$ current partner of c
        If c prefers $d^{'}$ to d, then d remains free, else (c, d) become engaged and $d^{'}$ becomes free
    end if
end while

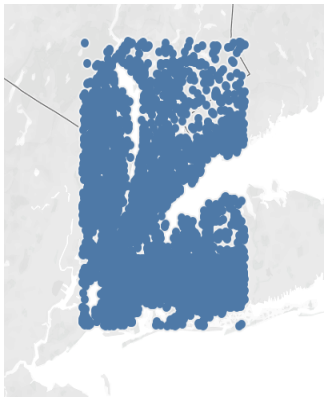


Figure 5.1: Vertices in NYC road network

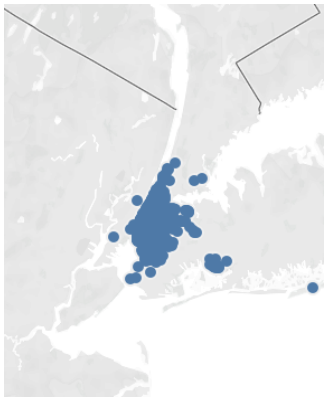


Figure 5.2: Sample customer distribution

plemented algorithms, the experimental setup, and finally the experimental analysis.

Dataset details: We evaluate our algorithms on the NYC Yellow Taxi dataset [77]. The dataset comprises trip records with various fields such as the customer pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, and so on. For evaluating the performance of the proposed algorithms, we consider the following fields in the trip records: the customer pick-up location, pick-up date and time, and the drop-off location. For the underlying road network dataset, we considered the New York city road network [78]. The dataset contains 2,64,346 vertices and 7,33,846 edges. The edges are associated with a cost value that represents the length of the corresponding road segment. Figure 5.1 visualizes the vertices in the NYC road network, and Figure 5.2 visualizes a sample customer requests distribution for a one hour duration.

Implemented algorithms: We implemented the Greedy algorithm as the baseline, and the two proposed algorithms: the KND algorithm and the Stable Matching based algorithm. From the related work, we implemented the LIPG algorithm. We implement our algorithms using the batch mode of processing, which means that the requests in the incoming stream of customer requests are collected in a periodic man-

ner (for a fixed time window), and then the processing algorithm is applied to the set of collected requests. The failed requests in each batch are passed on to the next batch until they expire.

Experimental setup: We implemented our taxi-hailing framework in Java language on a machine with a 1.6 GHz processor and an 8 GB RAM. Our framework is based on the batch mode of processing, meaning that the incoming customer requests are accumulated every t time interval, and then the processing algorithm is applied. We divide the timeline into fixed-sized time units, and measure the statistics in these time units. The rest of the details of our framework are as follows:

1. **Spatial distribution of drivers**: For the spatial distribution of drivers, we consider the drop-off locations of customers, as the location of the drivers. For their initial spatial locations, we consider a historical distribution [2].

2. **Temporal distribution of drivers**: We set the rate of appearance of drivers on the platform in proportion to the rate of appearance of customer requests on the platform. In other words:

$$Rate\,of\,appearance\,of\,drivers = \lambda * Rate\,of\,appearance\,of\,customers \quad (5.5)$$

3. **Driver mobility model**: We implement a driver mobility model similar to real-life mobility pattern of drivers. The locations of the idle drivers on the platform are moved to nearby location every t' time units.

4. **Customer and driver deadline**: We set the lifetime of the customer requests and the drivers in terms of our fixed-sized time units. When this time duration for an entity lapses, the specific entity no longer exists on the platform.

[2] A historical distribution also takes care of the disparity between the expanse of the considered road network and the limited region on the road network with customer requests. A uniform distribution on the other hand would not serve this purpose.

Since the definition of our cost function considers the maximum values of driver idle time and customer waiting time, we compare the worst case statistics for these two quantities. The parameters for our experiments were set to the following values: $\alpha = \beta = \gamma = 0.33$ (defined in Equations 5.1 and 5.2). The value of λ (defined in Equation 5.5) varies across the experiments, and is mentioned in the later sections.

5.5.1 Comparative analysis of the proposed algorithms

In this section, we compare the performance of the proposed algorithms and the baseline greedy strategy.

Effect of varying the Customer waiting time threshold: This experiment was done to study the effect of varying the threshold on the waiting time of the customer (for the corresponding assignment to be valid) on the different metrics. Figure 5.3 shows the effect of increase in the customer's waiting time threshold on the driver's idle time, the customer's waiting time and the cardinality of the assignment.

As can be seen in Figure 5.3(a), the driver's idle time value is the least for the Stable Matching algorithm, followed by that for the KND algorithm, and the Greedy algorithm, in that order. The idle time value decreases with increase in the customer's waiting time threshold for the Stable Matching algorithm. This happens due to the following reason: as the waiting time threshold for the customers in increased, more assignments become valid, and therefore more drivers can be assigned customer rides. This in effect decreases the idle time of the drivers on the platform. However, this is not the case for the KND algorithm and the Greedy algorithm. The idle time value does not change much with increase in the waiting time threshold for these two algorithms.

Figure 5.3(b) shows the effect of increase in the customer's waiting time threshold on the customer's waiting time. The waiting time value is the least for the Greedy strategy, followed by that for the KND algorithm and Stable Matching algorithm, in

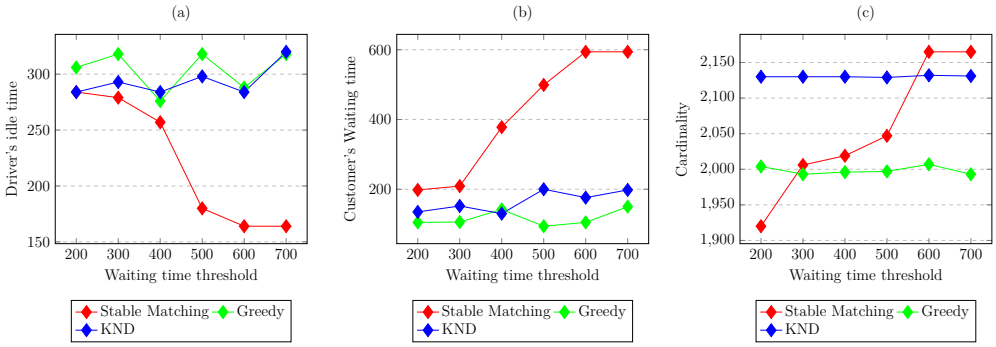


Figure 5.3: Effect of increase in Customer waiting time threshold on the driver's idle time, customer's waiting time and the cardinality of the assignment; Time duration of experiment = 36K time units; Number of customers = 2166; Number of drivers = 1865; $\lambda = 0.9$

that order. As one would expect, the waiting time value increases with increase in the waiting time threshold. However, the slope of the rise is the highest for the Stable Matching algorithm.

The trends in plot 5.3(b) follow a reverse pattern compared to the trends in plot 5.3(a). This shows that the Greedy strategy and the KND based strategy are inherently biased towards the customer's expectation. The performance of the Stable Matching algorithm on the contrary shows a trade-off between the driver's idle time and the customer's waiting time, and allows the platform owner to fine tune the settings as per the requirement.

The effect of increase in the waiting time threshold on the cardinality of the assignment is shown in Figure 5.3(c). Here, the slope of increase in the cardinality with increasing threshold, is the highest for the Stable Matching algorithm. The cardinality achieved by the KND algorithm is higher than that of the Greedy strategy. The highest cardinality is achieved by the Stable Matching based algorithm when the customer waiting time threshold is greater than 600 time units.

Effect of increase in the number of drivers: In this experiment, we study the effect of increase in the number of drivers, expressed as a percentage of the number

of customer requests. A value of x% implies that the number of drivers is set to x% of the total customer requests, and that the value of λ is $x/100$.

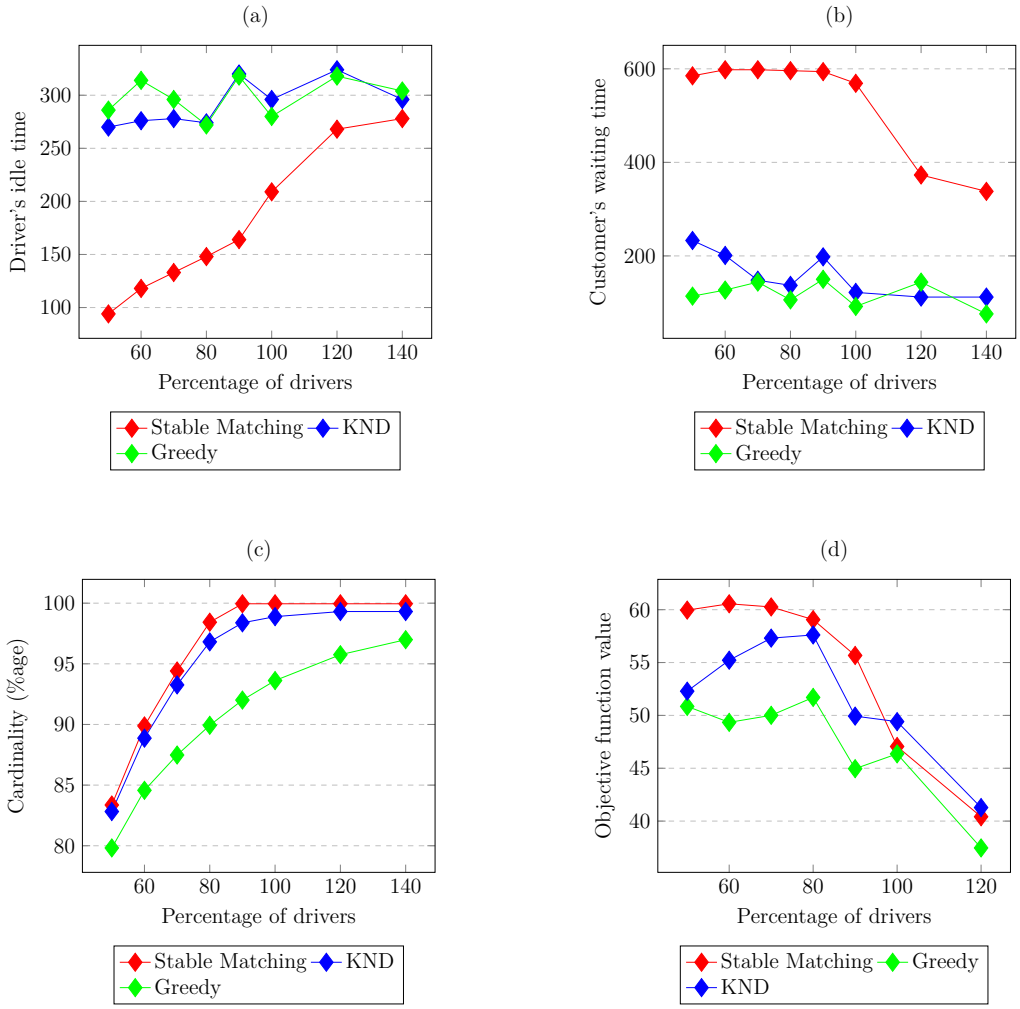


Figure 5.4: Effect of increase in the number of drivers on the driver's idle time, customer's waiting time, cardinality and the objective function value; Time duration of experiment = 36K time units; Number of customers = 2166; Customer's waiting time threshold = 700 time units

Figure 5.4(a) shows the effect of increase in the number of drivers on the driver's idle time value. The Stable Matching algorithms performs the best in this case, followed by KND algorithm and the Greedy algorithm, in that order. Figure 5.4(b) depicting the effect of increase in the number of drivers on the driver's idle time value, follows a reverse pattern. The trends in plots 5.4(a) and 5.4(b) are similar to the

trends in plots 5.3(a) and 5.3(b), respectively, and follow the same reasoning explained earlier. Similar to the previous experiment, the performance of the Stable Matching algorithm shows a trade-off between the driver's idle time and the customer's waiting time, and therefore allows the platform owner to fine tune the setting as per the requirement. On the contrary, the KND and Greedy algorithms are biased towards the customer's expectation.

Figure 5.4(c) shows the cardinality (number of assignments/number of customer requests, expressed as a percentage) versus the number of drivers plot. The Stable Matching algorithm performs the best in this case, followed by the KND and Greedy algorithms in that order.

The effect of increase in the number of drivers on the objective function value is depicted in Figure 5.4(d). The Stable Matching algorithm performs the best in this case, followed by the KND algorithm and the Greedy algorithm, in that order. The highest value for the objective function is achieved by the Stable Matching algorithm when the number of drivers is set to 60% of the number of customer requests. As the number of drivers increases, the objective function value decreases since the objective function considers maximisation of the profit earned. A larger number of drivers implies a greater workforce (that has to be paid), while the Stable Matching algorithm requires only 60% drivers to accept 90% of the customer requests (as depicted in Plot 5.4(c)).

5.5.2 Comparative analysis of the proposed algorithms and LIPG

In this section we compare the performance of the proposed algorithms against the LIPG algorithm. The customer waiting time threshold in this experiment for the Greedy, KND and Stable matching algorithms was set to 700 time units.

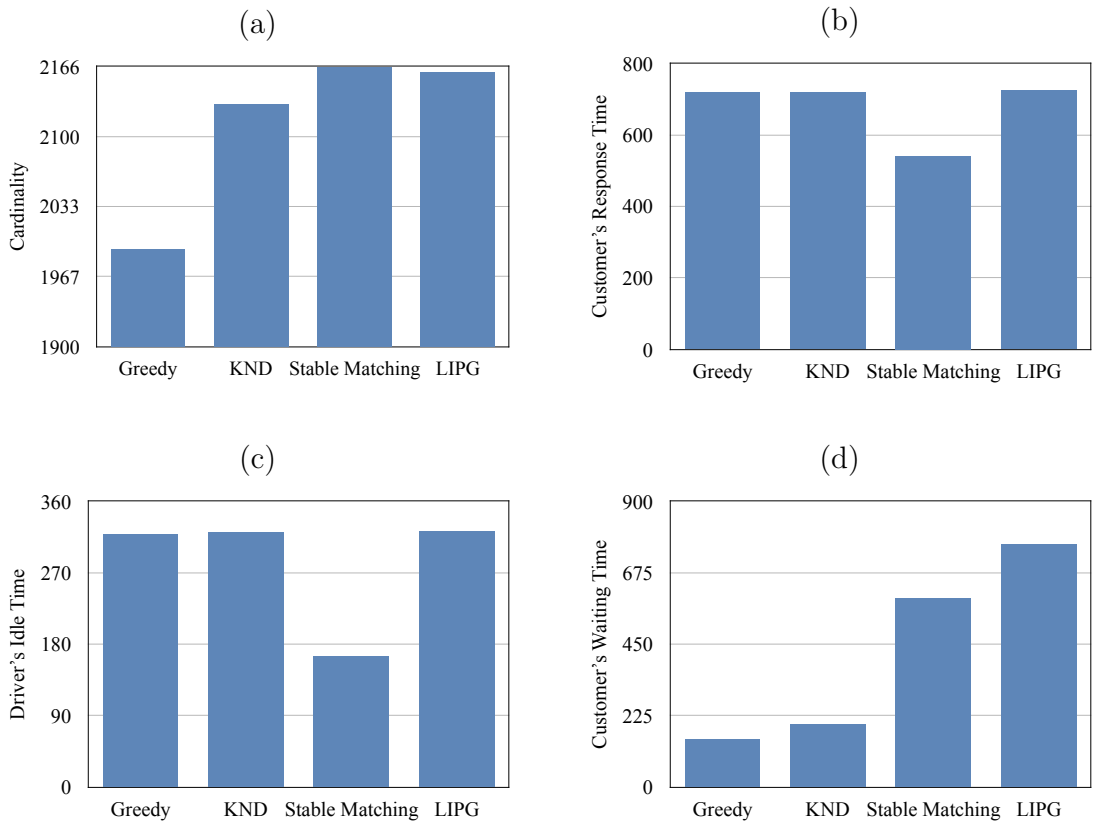


Figure 5.5: Comparing the Cardinality, Customer's response time, Driver's idle time and Customer's waiting time across all the algorithms; Time Duration = 36K time units; Number of Customers = 2166; Number of Drivers = 1865; $\lambda = 0.9$

Comparing the different metrics: This experiment was done to compare the cardinality of the assignment, the customer response time, the driver's idle time and the customer's waiting time for all the algorithms. The customer's response time is defined as the time difference between the arrival of the customer request and the intimation of acceptance of request to the customer.

As depicted in Figure 5.5(a), the cardinality is best achieved by the Stable Matching algorithm, followed by the LIPG algorithm, the KND algorithm and the Greedy algorithm, in that order. For the customer's response time and the driver's idle time, the Stable Matching algorithm performs the best. In the case of the customer's waiting time, the Greedy strategy performs the best, followed by the KND algorithm, the Stable Matching algorithm, and the LIPG algorithm, in that order.

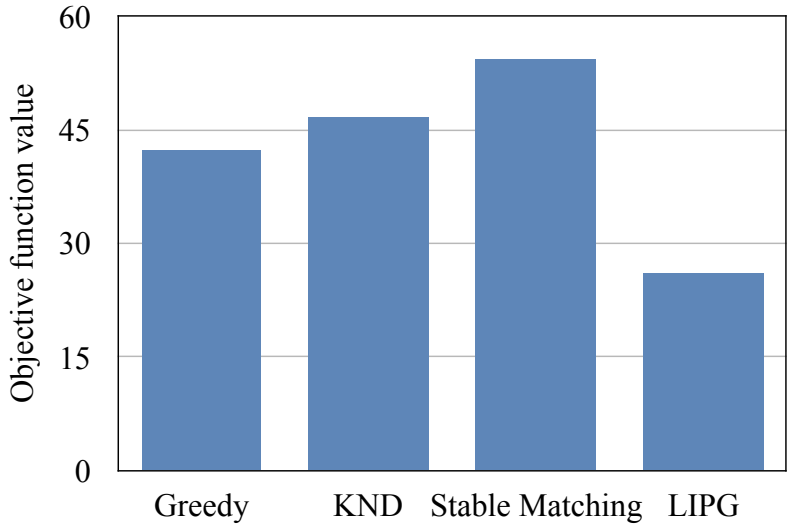


Figure 5.6: Comparing the objective function value across all algorithms; Time Duration = 36K units; Number of Drivers = 1865; Number of Customers = 2166; $\lambda = 0.9$

Figure 5.6 depicts the comparison of objective function value achieved by the different algorithms. The Stable Matching algorithm performs the best in this case, followed by the KND algorithm and the Greedy algorithm. This can be attributed to the reason that the objective function considers both the driver's idle time and the customer's waiting time. While the KND and Greedy algorithms yield low value for the customer's waiting time, their performance is poor in terms of the driver's idle time. On the contrary, the Stable Matching algorithm considers the trade-off between the two metrics. The LIPG algorithm achieves the lowest value for the objective function. Although the cardinality achieved by the LIPG algorithm is high, it fails to achieve low values for both the driver's idle time and the customer's waiting time.

5.6 Conclusions

In this work, we address a Spatial Crowdsourcing problem from an egalitarian perspective. We aim to minimise the waiting time for both the customers and the

drivers of the platform, while maximising the profit earned, in a fully-online setting with deadlines for both customers and drivers. We propose two heuristic algorithms to solve our problem: (i) the K Nearest Drivers based (KND) algorithm, and (ii) the Stable Matching based algorithm. We evaluate our algorithms on a real taxi trips records dataset, and compare their performance against the state-of-the-art algorithm for the fully-online bottleneck matching problem with deadlines. Our results show that the proposed algorithms exhibit a superior performance than the state-of-the-art algorithm in terms of both solution quality and response time. In future, we plan to extend the scope of our problem to include ride-sharing services, which would also entail route planning algorithms along with the matching algorithms.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

In this thesis, we investigate computational techniques for micro-level and macro-level transportation problems on urban road networks. In micro-level transportation problems, we work on constrained path optimisation. As use-cases of constrained path optimization, we consider finding navigable paths and safe paths. These use-cases are particularly relevant in developing nations where the road network infrastructure is compromised. We proposed heuristic algorithms for constrained path optimization, and evaluated the performance of our algorithms on real and synthetic datasets. Our analysis shows that our K seeds based algorithm performs better than the state-of-the-art algorithm for the Arc Orienteering Problem algorithm. We also put together a Navigation System to recommend 'Safe Routes' to travelers.

In macro-level transportation problems, we consider the use-case of task assignment in spatial crowdsourcing. As opposed to the traditional fashion of task assignment that considers a utilitarian objective, we consider an egalitarian objective, i.e., we optimise the expectation of all entities involved. We propose two heuristic algorithms for the problem considered: the K Nearest Drivers based algorithm and the Stable matching based algorithm. We evaluate the performance of our algorithms on a real dataset. Our analysis shows that our Stable Matching is well-suited for the problem considered, and it performs better than the state-of-the-art algorithm for the fully-online bottleneck matching problem.

Some future directions for this work are as follows: (i) **Personalised navigation**: The current system for Navigable/Safe Routes could be extended to include the option of personalised routing wherein the parameters of the model or the scores values changed based on the profile of the individual. For example, the notion of navigability

could be different for people from different age groups. Elderly citizens might prefer residential streets, whereas, the younger generation might prefer highways. (ii) **Time dependent routing**: The current system could also be extended to include the option of time dependent routing. Similar to the concept of personalised navigation, we could incorporate a model where the score values changed with the current time, based on how the safety/navigability of the roads changed with time. For example, one might prefer navigating on a route such that the driver is not facing the sun. (iii) **Validating the safety of solution**: In the study done on safe routing, we could not validate the quality of routes returned by the system vis-a-vis their real-life safety. A study cold also be undertaken to validate the same. (iv) **General improvements in the current work**: A more thorough evaluation of the proposed solutions could be done in terms of the characteristics of the graph for the road networks used in the study, in terms of how the solution changed with change in the characteristics of the underlying graphs. For this objective, synthetic graphs could also be generated using different graph generation models and more real-life graphs with known characteristics (such as the Atlanta road network dataset) could be selected for the study, and the proposed solutions could be evaluated on them.

PUBLICATIONS

[1] R. Kaur, V. Goyal, and V. M. V. Gunturi, “Finding the most navigable path in road networks: A summary of results,” in DEXA, 2018.

[2] R. Kaur, V. Goyal, and V. M. V. Gunturi, “Finding the most navigable path in roadnetworks,” GeoInformatica, Jan. 2021

[3] R. Kaur, V. Goyal, V. M. V. Gunturi, A. Saini, K. Sanadhya, R. Gupta, and S. Ratra, “A Navigation System for Safe Routing,” 22nd IEEE International Conference on Mobile Data Management, MDM 2021 (Demo paper)

[4] R. Kaur, V. Goyal, V. M. V. Gunturi, and C. Long, "A Matching Based Spatial Crowdsourcing Framework for Egalitarian Task Assignment," accepted at the 23rd IEEE International Conference on Mobile Data Management, MDM 2022 (Poster paper)

[5] R. Kaur, V. Goyal, V. M. V. Gunturi, and C. Long "Task Assignment in Spatial Crowdsourcing with Egalitarian Optimisation," submitted to the Big Data Research journal

REFERENCES

[1] A. Gunawan, H. C. Lau, and P. Vansteenwegen, "Orienteering problem: A survey of recent variants, solution approaches and applications", *European Journal of Operational Research*, vol. 255, no. 2, pp. 315 –332, 2016.

[2] K. Ramneek, V. Goyal, and V. M. V. Gunturi, "Finding the most navigable path in road networks: A summary of results", in *DEXA*, 2018.

[3] R. Kaur, V. Goyal, and V. M. V. Gunturi, "Finding the most navigable path in road networks", *GeoInformatica*, vol. 25, pp. 207–240, Jan. 2021.

[4] B. Livingston, M. Grillo, and R. Paluch. "Cornell international survey on street harassment". (2014).

[5] Y. Tong, L. Chen, and C. Shahabi, "Spatial crowdsourcing: Challenges, techniques, and applications", *Proc. VLDB Endow.*, vol. 10, no. 12, 1988–1991, Aug. 2017.

[6] Y. Tong and Z. Zhou, "Dynamic task assignment in spatial crowdsourcing", *SIGSPATIAL Special*, vol. 10, no. 2, 18–25, 2018.

[7] L. Kazemi and C. Shahabi, "Geocrowd: Enabling query answering with spatial crowdsourcing", ser. SIGSPATIAL '12, Redondo Beach, California, 2012, 189–198, ISBN: 9781450316910.

[8] Uber, https://www.uber.com.

[9] Ola, https://www.olacabs.com.

[10] Swiggy, https://www.swiggy.com.

[11] Zomato, https://www.zomato.com.

[12] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing", in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '15, Seattle, Washington: Association for Computing Machinery, 2015, ISBN: 9781450339674.

[13] Y. Tong *et al.*, "Flexible online task assignment in real-time spatial data", *Proc. VLDB Endow.*, vol. 10, no. 11, 1334–1345, Aug. 2017.

[14] U. U. Hassan and E. Curry, "A multi-armed bandit approach to online spatial task assignment", in *2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops*, 2014, pp. 212–219.

[15] Y. Li, J. Fang, Y. Zeng, B. Maag, Y. Tong, and L. Zhang, "Two-sided online bipartite matching in spatial data: Experiments and analysis", *GeoInformatica*, vol. 24, no. 1, Jan. 2020.

[16] Y. Zeng, Y. Tong, L. Chen, and Z. Zhou, "Latency-oriented task completion via spatial crowdsourcing", in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, 2018, pp. 317–328.

[17] Y. Tong, Y. Zeng, B. Ding, L. Wang, and L. Chen, "Two-sided online micro-task assignment in spatial crowdsourcing", *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 5, pp. 2295–2309, 2021.

[18] J. P. Dickerson, K. A. Sankararaman, A. Srinivasan, and P. Xu, "Assigning tasks to workers based on historical data: Online task assignment with two-sided arrivals", ser. AAMAS '18, Stockholm, Sweden, 2018, 318–326.

[19] Y. Tong, L. Wang, Z. Zhou, L. Chen, B. Du, and J. Ye, "Dynamic pricing in spatial crowdsourcing: A matching-based approach", ser. SIGMOD '18, Houston, TX, USA, 2018, 773–788, ISBN: 9781450347037.

[20] Z. Chen, P. Cheng, Y. Zeng, and L. Chen, "Minimizing maximum delay of task assignment in spatial crowdsourcing", in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 1454–1465.

[21] S. Ji, Y. Zheng, Z. Wang, and T. Li, "Alleviating users' pain of waiting: Effective task grouping for online-to-offline food delivery services", ser. WWW '19, San Francisco, CA, USA, 2019, 773–783, ISBN: 9781450366748.

[22] F. Basık and et al., "Fair task allocation in crowdsourced delivery", *IEEE TSC*, vol. 14, no. 4, pp. 1040–1053, 2021.

[23] Y. Zhao and et al., "Preference-aware task assignment in spatial crowdsourcing: From individuals to groups", *IEEE TKDE*, pp. 1–1, 2020.

[24] L. Zheng and L. Chen, "Maximizing acceptance in rejection-aware spatial crowdsourcing", *IEEE TKDE*, vol. 29, pp. 1943–1956, 2017.

[25] L. Li, J. Fang, B. Du, and W. Lv, "Spatial bottleneck minimum task assignment with time-delay", in *Database Systems for Advanced Applications*, G. Li, J.

Yang, J. Gama, J. Natwichai, and Y. Tong, Eds., Cham: Springer International Publishing, 2019, pp. 387–391, ISBN: 978-3-030-18590-9.

[26] M. Asghari, D. Deng, C. Shahabi, U. Demiryurek, and Y. Li, "Price-aware real-time ride-sharing at scale: An auction-based approach", in *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPACIAL '16, Burlingame, California: Association for Computing Machinery, 2016, ISBN: 9781450345897.

[27] C. Archetti, A. Corberán, I. Plana, J. M. Sanchis, and M. G. Speranza, "A branch-and-cut algorithm for the orienteering arc routing problem", *Comput. Oper. Res.*, vol. 66, no. C, pp. 95–104, Feb. 2016.

[28] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and N. Vathis, "Approximation algorithms for the arc orienteering problem", *Information Processing Letters*, vol. 115, no. 2, pp. 313 –315, 2015.

[29] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden, "The orienteering problem: A survey", *European Journal of Operational Research*, vol. 209, no. 1, pp. 1 –10, 2011.

[30] B. L. Golden, L. Levy, and R. Vohra, "The orienteering problem", *Naval Research Logistics (NRL)*, vol. 34, no. 3, pp. 307–318, 1987.

[31] G. Laporte and S. Martello, "The selective travelling salesman problem", *Discrete Applied Mathematics*, vol. 26, no. 2, pp. 193 –207, 1990.

[32] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "A survey on algorithmic approaches for solving tourist trip design problems", *Journal of Heuristics*, vol. 20, no. 3, pp. 291–328, 2014.

[33] E. Kanoulas, Y. Du, T. Xia, and D. Zhang, "Finding fastest paths on a road network with speed patterns", in *22nd International Conference on Data Engineering (ICDE'06)*, 2006, pp. 10–10.

[34] N. Jing, Y.-W. Huang, and E. A. Rundensteiner, "Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation", *IEEE Trans. Knowl. Data Eng.*, vol. 10, pp. 409–432, 1998.

[35] D. Delling, A. V. Goldberg, A. Nowatzyk, and R. F. Werneck, "Phast: Hardware-accelerated shortest path trees", *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 940 –952, 2013.

[36] E. Martins and M. Pascoal, "A new implementation of yen's ranking loopless paths algorithm", *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 121–133, 2003.

[37] J. Y. Yen, "Finding the k shortest loopless paths in a network", *Management Science*, vol. 17, no. 11, pp. 712–716, 1971.

[38] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation", *ACM Trans. Algorithms*, vol. 3, no. 4, Nov. 2007.

[39] H.-P. Kriegel, M. Renz, and M. Schubert, "Route skyline queries: A multi-preference path planning approach", *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pp. 261–272, 2010.

[40] Y. Tian, K. C. K. Lee, and W.-C. Lee, "Finding skyline paths in road networks", in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '09, Seattle, Washington: ACM, 2009, pp. 444–447, ISBN: 978-1-60558-649-6.

[41] M. Fischetti, J. J. Salazar González, and P. Toth, "Solving the orienteering problem through branch-and-cut", *INFORMS Journal on Computing*, vol. 10, pp. 133–148, May 1998.

[42] W. Souffriau, P. Vansteenwegen, G. V. Berghe, and D. V. Oudheusden, "The planning of cycle trips in the province of east flanders", *Omega*, vol. 39, no. 2, pp. 209 –213, 2011.

[43] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf, "An extension of the arc orienteering problem and its application to cycle trip planning", *Transportation Research Part E: Logistics and Transportation Review*, vol. 68, pp. 64 –78, 2014.

[44] P. Bolzoni, F. Persia, and S. Helmer, "Itinerary planning with category constraints using a probabilistic approach", in *Database and Expert Systems Applications*, Springer International Publishing, 2017, pp. 363–377.

[45] A. Singh, A. Krause, C. Guestrin, W. Kaiser, and M. Batalin, "Efficient planning of informative paths for multiple robots", in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07, 2007, pp. 2204–2211.

[46] Y. Lu and C. Shahabi, "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*, ser. SIGSPATIAL '15, 2015, 46:1–46:10.

[47] P. Bolzoni and S. Helmer, "Hybrid best-first greedy search for orienteering with category constraints", in *Proceedings of SSTD 2017*, 2017, pp. 24–42.

[48] C. Chekuri and M. Pal, "A recursive greedy algorithm for walks in directed graphs", in *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, ser. FOCS '05, 2005, pp. 245–253, ISBN: 0-7695-2468-0.

[49] V. Nagarajan and R. Ravi, "The directed orienteering problem", *Algorithmica*, vol. 60, no. 4, pp. 1017–1030, 2011.

[50] V. Nagarajan and R. Ravi, "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*, ser. APPROX '07/RANDOM '07, 2007, pp. 257–270.

[51] N. Bansal, A. Blum, S. Chawla, and A. Meyerson, "Approximation algorithms for deadline-tsp and vehicle routing with time-windows", in *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, ser. STOC '04, 2004, pp. 166–174.

[52] A. M. Aly *et al.*, "Aqwa: Adaptive query workload aware partitioning of big spatial data", *Proc. VLDB Endow.*, vol. 8, no. 13, pp. 2062–2073, Sep. 2015.

[53] D. Gavalas and et al., "Approximation algorithms for the arc orienteering problem", *Info. Processing Letters*, vol. 115, no. 2, pp. 313 –315, 2015.

[54] A. M. de Souza, T. Braun, L. C. Botega, R. Cabral, I. C. Garcia, and L. A. Villas, "Better safe than sorry: A vehicular traffic re-routing based on traffic conditions and public safety issues", *Journal of Internet Services and Applications*, vol. 10, 1 2019.

[55] E. Galbrun, K. Pelechrinis, and E. Terzi, "Urban navigation beyond shortest route: The case of safe paths", *Information Systems*, vol. 57, pp. 160 –171, 2016.

[56] J. Kim, M. Cha, and T. Sandholm, "Socroutes: Safe routes based on tweet sentiments", in *Proceedings of the 23rd International Conference on World Wide Web*, ser. WWW '14 Companion, Seoul, Korea: Association for Computing Machinery, 2014, 179–182, ISBN: 9781450327459.

[57] B. Zhou and et al., "Learning deep features for scene recognition using places database", in *Proc. of NIPS*, 2014, 487–495.

[58] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, *Spanbert: Improving pre-training by representing and predicting spans*, 2020. arXiv: 1907.10529 [cs.CL].

[59] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique", *J. Artif. Int. Res.*, vol. 16, no. 1, 321–357, Jun. 2002.

[60] L. Li and W. Lv, "Spatial two-sided online bottleneck matching with deadlines", *IEEE Access*, vol. 8, pp. 57 772–57 785, 2020.

[61] L. Kazemi, C. Shahabi, and L. Chen, "Geotrucrowd: Trustworthy query answering with spatial crowdsourcing", ser. SIGSPATIAL'13, Orlando, Florida, 2013, 314–323, ISBN: 9781450325219.

[62] D. Deng, C. Shahabi, and U. Demiryurek, "Maximizing the number of worker's self-selected tasks in spatial crowdsourcing", ser. SIGSPATIAL'13, Orlando, Florida, 2013, 324–333, ISBN: 9781450325219.

[63] Y. Tong, J. She, B. Ding, L. Wang, and L. Chen, "Online mobile micro-task allocation in spatial crowdsourcing", in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, 2016, pp. 49–60.

[64] Q. Tao, Y. Zeng, Z. Zhou, Y. Tong, L. Chen, and K. Xu, "Multi-worker-aware task planning in real-time spatial crowdsourcing", in *Database Systems for Advanced Applications*, J. Pei, Y. Manolopoulos, S. Sadiq, and J. Li, Eds., 2018, pp. 301–317, ISBN: 978-3-319-91458-9.

[65] Y. Tong, Y. Zeng, Z. Zhou, L. Chen, J. Ye, and K. Xu, "A unified approach to route planning for shared mobility", *Proc. VLDB Endow.*, vol. 11, no. 11, 1633–1646, 2018.

[66] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu, "Online minimum matching in real-time spatial data: Experiments and analysis", *Proc. VLDB Endow.*, vol. 9, no. 12, 1053–1064, 2016.

[67] X. Wan, H. Ghazzai, and Y. Massoud, "A generic data-driven recommendation system for large-scale regular and ride-hailing taxi services", *Electronics*, vol. 9, no. 4, 2020.

[68] J. W. Powell, Y. Huang, F. Bastani, and M. Ji, "Towards reducing taxicab cruising time using spatio-temporal profitability maps", in *Advances in Spatial and Temporal Databases*, Springer Berlin Heidelberg, 2011, pp. 242–260.

[69] N. Garg and S. Ranu, "Route recommendations for idle taxi drivers: Find me the shortest route to a customer!", in *Proc. of the 24th ACM SIGKDD Intl. Conference on Knowledge Discovery & Data Mining*, ACM, 2018, 1425–1434.

[70] J. Yuan and et al., "Where to find my next passenger", in *Proc. of the 13th Intl. Conference on Ubiquitous Computing*, ACM, 2011, 109–118.

[71] X. Xu and et al., "Taxi-rs: Taxi-hunting recommendation system based on taxi gps data", *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 1716–1727, 2015.

[72] S. H. Kang and et al., "Predicting taxi passenger demands based on the temporal and spatial information", in *Neural Information Processing*, 2017, pp. 267–274.

[73] N. J. Yuan and et al., "T-finder: A recommender system for finding passengers and vacant taxis", *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2390–2403, 2013.

[74] H. Rong, X. Zhou, C. Yang, Z. Shafiq, and A. Liu, "The rich and the poor: A markov decision process approach to optimizing taxi driver revenue efficiency", in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM '16, Indianapolis, Indiana, USA: Association for Computing Machinery, 2016, 2329–2334, ISBN: 9781450340731.

[75] X. Zhou *et al.*, "Optimizing taxi driver profit efficiency: A spatial network-based markov decision process approach", *IEEE Transactions on Big Data*, vol. 6, no. 1, pp. 145–158, 2020.

[76] D. Gale and L. S. Shapley, "College admissions and the stability of marriage", *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[77] TLC, *Trip record data*, https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[78] C. Demetrescu, *Dimacs*, http://users.diag.uniroma1.it/challenge9/download.shtml.