# Group Trip Planning Queries in Spatial Databases

by

Elham Ahmadi

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

University of Alberta

# Abstract

*Trip planning queries* are considered an important part of *Location Based Services*. As the first part of our research, we investigated Sequenced Group Trip PLanning Queries (SGTP) queries. Given a set of source locations and destinations for a group of $n$ users, and a sequence of Categories of Interests (COIs) that the group is interested to visit altogether, a SGTP query returns for each user, the route from his/her source location to his/her destination such that all users go through the same Points of Interests (POIs), while minimizing the group total travel distance.

As the second phase of our research, we assumed that users are interested to visit a POI belonging to the predefined COI altogether with the goal of minimizing the total detour distance towards group's preferred paths. In the third phase of this research, we investigated a combination of trip planning and path nearest neighbor queries, which we refere to as "Best-Compromise In-Route Nearest Neighbor". We investigated the problem where a user, traveling on his/her preferred path, needs to visit one (of many) POI while minimizing his/her total travel distance and also minimizing the detour distance incurred to reach the chosen POI.

Finally, we studied the k-CPQs in road networks. Given two sets of nodes $P$ and $Q$ on a road network, a $k$-Closest Pairs Query ($k$-CPQ) finds the pairs from $P \times Q$ which have the $k$ smallest network distances. Although this problem has been well studied in the Euclidean and metric spaces, this is the first time it is being investigated in the more realistic case of road networks.

Dedicated to my beloved parents Mohammad Sadegh and Ashraf. For their love, endless support, encouragement and sacrifices.

A bird sitting on a tree is never afraid of the branch breaking, because her trust is not on the branch but on her own wings.

– Unknown

# Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Mario A. Nascimento for the continuous support of my Ph.D study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Jörg Sander, Prof. Zachary Friggstad and Prof. Tony Qiu, for their encouragement and insightful comments.

Finally, I must express my very profound gratitude to my parents and to my sisters and brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author
Elham Ahmadi

# Contents


**1 Introduction** **1**

**2 Sequenced Group Trip Planning Queries** **4**
2.1 Introduction . . . 4
2.2 Related Work . . . 5
2.2.1 Single User Trip Planning Queries . . . 5
2.2.2 Group Trip Planning Queries . . . 8
2.3 Problem Definition . . . 11
2.4 Proposed Solutions . . . 12
2.4.1 Revised Iterative Approach (RIA) . . . 12
2.4.2 PGNE Approach . . . 17
2.4.2.1 Applied Pruning Strategies . . . 18
2.4.2.2 PGNE Algorithm . . . 21
2.4.3 Iterative Backward Search Approach . . . 23
2.4.4 Running example . . . 29
2.5 Experiments . . . 33
2.5.1 Effect of POI Density . . . 35
2.5.2 Effect of Query Area . . . 36
2.5.3 Effect of Group Size . . . 37
2.5.4 Effect of COI Density . . . 39
2.6 Conclusion . . . 40

**3 Optimal Meeting Points Minimizing Aggregate Detour Distances from Preferred Paths** **42**
3.1 Introduction . . . 42
3.2 Related Work . . . 44
3.3 Proposed Approaches . . . 46
3.3.1 Multiple Ellipse-based Pruning Approach (MEP) . . . 48
3.3.2 Single Ellipse-based Pruning Approach (SEP) . . . 52
3.4 Experimental Results . . . 55
3.4.1 Effect of POI Density . . . 57
3.4.2 Effect of Group Size . . . 60
3.4.3 Effect of Answer Size . . . 63
3.4.4 Summary of experimental results . . . 64
3.5 Conclusion . . . 65

**4 Best-Compromise In-Route Nearest Neighbor Queries** **66**
4.1 Introduction . . . 66
4.2 Related Work . . . 70
4.3 Preliminaries . . . 71
4.4 Our Proposed Approach . . . 77
4.4.1 Upper bounds for travel and detour distance . . . 77

4.4.2 Generating and pruning candidate paths . . . . . 79
4.4.3 Baseline approach . . . . . 83
4.5 Experimental Results . . . . . 84
4.5.1 Effect of Length of Preferred Path . . . . . 85
4.5.2 Effect of POI Density . . . . . 86
4.6 Conclusion . . . . . 87

**5 k-Closest Pairs Queries in Road Networks 89**
5.1 Introduction . . . . . 89
5.2 Related Work . . . . . 91
5.3 The G*-tree . . . . . 92
5.3.1 $G^*$-tree construction procedure . . . . . 93
5.3.2 G-tree vs. G*-tree . . . . . 96
5.3.3 Distance Computations in $G^*$-Tree . . . . . 97
5.4 Top-Down Traversal (TDT) . . . . . 98
5.5 Bottom-Up Sub-graph Joining (BUSJ) . . . . . 101
5.5.1 Pre-Computations in BUSJ Approach . . . . . 102
5.5.2 Algorithm Overview . . . . . 102
5.6 Experiments . . . . . 106
5.6.1 Effect of Number of Leaf-Nodes . . . . . 108
5.6.2 Effect of Number of Required CPs . . . . . 110
5.6.3 Effect of Dataset Size . . . . . 110
5.6.4 Effect of Imbalancing Factor . . . . . 111
5.7 Conclusion . . . . . 112

**6 Conclusion 113**

**Bibliography 116**

# List of Tables


2.1 Meaning of symbols used in SGTPQs . . . . . 12
2.2 Sample execution of IBS using the example of Figure 2.4. . . . . 32
2.3 Road networks used in the experiments. . . . . 33
2.4 Experimental parameters and their values (bold defines default values). . . . . 34

3.1 Aggregate detour distance for different POIs with respect to Figure 3.1. . . . . 43
3.2 Notation. . . . . 47
3.3 Summary of the dataset used in our experiments. . . . . 56
3.4 Experimental parameters and their values (**bold** defines default values). . . . . 56

4.1 Paths and their corresponding costs w.r.t. Figure 4.1. . . . . 68
4.2 Notation. . . . . 72
4.3 Experimental parameters and their values (**bold** defines default values). . . . . 84

5.1 TDT approach for 2-CPQ of example of Figure 5.1 . . . . . 101
5.2 Parameter values in the experiments . . . . . 107

# List of Figures


2.1 A sample road network with three types of POIs. . . . . 5
2.2 A sample road network with three types of POIs. . . . . 13
2.3 An example scenario . . . . . 15
2.4 A sample road network with three types of POIs. . . . . 30
2.5 Effect of POI density($D$). . . . . 36
2.6 Effect of query area ($A$). . . . . 37
2.7 Effect of group size ($n$). . . . . 38
2.8 Effect of Number of COIs ($m$). . . . . 39
2.9 COI density. . . . . 40

3.1 A sample road network. . . . . 43
3.2 Mapping of a k-$OMP^3$ query into a MALs problem. . . . . 45
3.3 The proposed pruning strategies. . . . . 48
3.4 Defining lower bound for detour distance. . . . . 50
3.5 Bus stops and POIs in Oslo. . . . . 56
3.6 Effect of $D_p$ on processing time for $sum$ aggregate function . . . . 58
3.7 Effect of $D_p$ on number of examined POIs for $sum$ aggregate function 58
3.8 Effect of $D_p$ on processing time for $max$ aggregate function . . . . 59
3.9 Effect of $D_p$ on number of examined POIs for $max$ aggregate function . . . . . 59
3.10 Effect of $n$ on processing time for $sum$ aggregate function . . . . . 60
3.11 Effect of $n$ on number of examined POIs for $sum$ aggregate function . . . . . 60
3.12 Effect of $n$ on processing time for $max$ aggregate function . . . . . 61
3.13 Effect of $n$ on number of examined POIs for $max$ aggregate function . . . . . 61
3.14 Effect of $k$ on processing time for $sum$ aggregate function . . . . . 62
3.15 Effect of $k$ on number of examined POIs for $sum$ aggregate function . . . . . 62
3.16 Effect of $k$ on processing time for $max$ aggregate function . . . . . 63
3.17 Effect of $k$ on number of examined POIs for $max$ aggregate function . . . . . 63

4.1 Alternatives paths on a simplified road network from $s$ to $d$ visiting one of POIs $o_1$ or $o_2$. Table 4.1 summarizes the travel and detour distance implied by each such path. . . . . 67
4.2 Conventional and linear skylines for the example shown in Figure 4.1 and summarized in Table 4.1. . . . . 69
4.3 Area linearly dominated by $\{\mathbf{p^1}, \mathbf{p^3}\}$. . . . . 75
4.4 Locations of restaurants and coffee shops in Amsterdam, Oslo and Berlin (overlaid on these cities' road network) . . . . . 85
4.5 Effect of path length $n$ on query processing time . . . . . 87
4.6 Effect of POI density $D_p$ on query processing time . . . . . 88

5.1 A sample road network . . . . . 90
5.2 Sample steps of the coarsening phase for the graph in Figure 5.1. . . 94
5.3 The hierarchical graph partitioning of example of Figure 5.1 . . . . 95
5.4 (a) The intra-distance matrix of sub-graph $G_4$, (b) the corresponding inter distance matrix for $G^*$-tree illustrated in Figure 5.3 . . . . 98
5.5 The LPQs for border node $v_7$ towards sub-graph $G_8$ . . . . . 102
5.6 The order of joining of the sub-graphs in Figure 5.3 . . . . . 104
5.7 Effect of parameter $\lambda$ . . . . . 109
5.8 Effect of parameter $k$. . . . . 110
5.9 Effect of Data Set Size . . . . . 110
5.10 Effect of Parameter $\gamma$ . . . . . 111

# Chapter 1

# Introduction

The term "*spatial database*" [22] is associated with a view of a database which provides effective and efficient retrieval and management of geometric objects such as points, lines and polygons. Spatial databases has been an active area of research in the last two decades [37], in which many important results in data modelling, spatial indexing, and query processing techniques have been obtained [15, 61, 45, 54, 59]. Nowadays, mobile devices equipped with a Global Positioning System (GPS), have become more and more popular in our daily life for allowing Location Bases Services (LBSs) that require spatial databases. LBSs can be defined as services that integrate the position of a mobile device with other information in order to provide added value to a user [48].

Trip planning queries are considered an important part of LBSs. Let us assume a road network where the vertices represent either Points of Interest (POIs) or road network branch points (i.e., road network junctions), and each POI belongs to exactly one Category of Interest (COI), e.g., a COI can be "Restaurants" and each POI in this COI is a specific instance of a restaurant. Given a source point, a destination, and a sequence of COIs that must be visited , the corresponding Trip Planing Query retrieves the shortest trip from source point to destination passing through one POI from each COI. An example of a trip planning application is the following: A user plans to travel from the University of Alberta to the Bonnie-Doon Shopping Center with the smallest travel distance and wants to stop at a supermarket, a bank, and a post office.

Nowadays, with the integration of social networks such as Facebook [1], Google+ [2],

and Loopt [3] with LBSs, more complex and advanced query types are needed to be supported. In this work, we studied three different types of trip planning queries: 1) Sequenced Group Trip Planning Queries (SGTPQs), 2) k-Optimal Meeting Points based on Preferred Paths (k-$OMP^3$) and 3) Best-Compromise In-Route Nearest Neighbor (BC-IRNN).

As the first part of our research, we investigated SGTPQs. Given a set of source locations and destinations for a group of $n$ users, and a sequence of COIs that the group is interested to visit altogether, a SGTPQ returns for each user, the route from his/her source location to his/her destination such that all users go through the same POIs, while minimizing the group total travel distance. An example of real world application of SGTPQs in road networks would be the following. Consider a group of friends, who are at different places (e.g., on a late afternoon all of them may still be at their offices and workplaces). Before going home in the evening, they are willing to plan a group trip to several COIs in a specific sequence such as dining at a restaurant, then seeing a movie, and finally having drinks at a pub.

Our main contributions for solving SGTPQs were two algorithms: Progressive Group Neighbor Exploration (PGNE) [6] and Iterative Backward Search (IBS) [4]. The former traverses the search space based on a mixed breadth-depth first search strategy. The latter explores the search space based on depth first search strategy, in which optimal group trips from different POIs are computed and reused for computing the optimal answer thus avoiding significant query processing overhead.

As the second phase of our research, we investigated a novel query type, the *k-Optimal Meeting Points based on Preferred Paths* (k-$OMP^3$). Given a set of preferred paths for a group of $n$ friends, as well as a COI that the group is interested to visit, then the k-$OMP^3$ query returns the $k$ unique POIs which incur the $k$ smallest group detour distances towards group members preferred paths. For processing k-$OMP^3$ queries, we proposed efficient solutions based on shrinking the search space that apply geometric properties of ellipses to prune the POIs that cannot be part of optimal answer set [8].

In the third phase of this research, we investigated a combination of trip planning and path nearest neighbor queries. Consider a user who is traveling on his/her

preferred path, needs to visit one POI while (1) minimizing his/her total travel distance and also (2) minimizing the detour distance incurred to reach the chosen POI. We call this new problem "Best-Compromise In-Route Nearest Neighbor" (BC-IRNN) query in order to emphasize that a route cannot typically optimize both criteria at the same time, but rather find a compromise between them. In fact, the competing nature of these two criteria resembles the notion of skyline queries. In that context, we proposed a solution based on using suitable upper-bounds to both cost criteria to prune uninteresting paths [5].

As the last phase of our research, we also investigated the k-Closest Pairs Queries (k-CPQs) in road networks. Given two sets of nodes $P$ and $Q$ on a road network, a $k$-CPQ finds the pairs from $P \times Q$ which have the $k$ smallest network distances. Although this problem has been well studied in the Euclidean and metric spaces, this is the first time it is being investigated in the more realistic case of road networks. As our first contribution, we present a new hierarchical graph partitioning structure, named $G^*$-tree, which is designed to support our proposed algorithms. Then, we propose, as our main contribution, two different approaches for processing $k$-CPQs. While the first approach applies a top-down traversal paradigm by applying a best-first search strategy, the second approach looks for the $k$-closest pairs by traversing the $G^*$-tree in a bottom-up manner. Both of the these approaches employ an effective pruning strategy for shrinking the search space based on the minimum network distance between sub-graphs, which is main driver for the $G^*$-tree's construction [7].

# Chapter 2

# Sequenced Group Trip Planning Queries

## 2.1 Introduction

A Sequenced Trip Planning Query (STPQ) [52] is defined over a road network as follows: a *single* user specifies a starting point $s$, a destination $d$ and a sequence of categories of interest (COIs) $C$ that must be visited. The goal is then to find the shortest trip that starts at $s$, passes through one point of interest (POI) from each COI in $C$ and ends at $d$.

In this section, we study an extension of STPQs, namely Sequenced Group Trip Planning Queries (SGTPQs). SGTPQs have three parameters: the *sets* $S$ and $D$ containing the source locations and destinations of $n$ users, as well as a *sequence* of COIs $C$.The goal is to find the trip plan that goes from all users' origins to all of their destinations, passing through the same sequence of POIs (one from each COI in the sequence $C$) and where the group total travel distance is minimized.

Figure 2.1 illustrates a typical application of a SGTPQ (the road network itself is omitted for simplicity). Let us assume that a group of two users currently at locations $S_1$ and $S_2$ are interested in visiting, together, first a restaurant, then a museum and finally a pub. The destinations of the group members after visiting the COIs are $D_1$ and $D_2$, respectively. In this context the highlighted sequence of POIs $(R_1, M_1, P_1)$ defines a feasible candidate answer for such SGTPQ.

Clearly, STPQs are trivial special cases of SGTPQs. Thus an algorithm that provides optimal solutions for SGTPQs can solve STPQs optimally as well. For

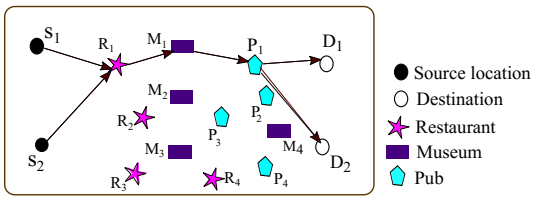



Figure 2.1: A sample road network with three types of POIs.

processing SGTP queries, we propose two approaches: Progressive Group Neighbor Exploration (PGNE) approach and Iterative Backward Search (IBS) approach. The PGNE approach explores the search space based on a mixed breadth-depth first search strategy. On the other hand, IBS explores the search space based on a depth-first search strategy, in which iteratively optimal group trips from retrieved POIs are computed and reused when the algorithm proceeds. Both PGNE and IBS apply efficient pruning strategies to shrink the search space.

## 2.2 Related Work

In Sections 2.2.1 and 2.2.2 we present a review on state-of-the-art techniques for processing single user and group trip planning queries in spatial data bases, respectively.

### 2.2.1 Single User Trip Planning Queries

In [37], Li et al. proposed solutions for Trip Planning Queries (TPQ). In a TPQ, the user specifies a set of COIs and asks for the optimal route (with minimum distance) from her starting location to a specified destination which passes through exactly one POI of each COI. TPQ can be considered as a generalization of the Traveling Salesman Problem (TSP) [10, 16], which is *NP-hard*. Any instance of *traveling Salesman problem* can be considered as a TPQ, in which each city can be considered as a COI containing only one POI. Consequently, finding an accurate solution to a TPQ becomes NP-hard as the size of candidate space significantly grows. For processing TPQs, Li et al. [37] proposed approximation algorithms to handle the exponential growth of the problem's search space.

Sharifzadeh et al. [52, 53] presented Optimal Sequenced Route (OSR) queries where the user asks for an optimal route from her starting location and passing through a number of COIs in a particular order (sequence). The R-LORD algorithm [52] performs range queries to filter out the points that cannot possibly be part of the optimal route by utilizing an R-tree index structure [23], and subsequently builds the optimal route in reverse sequence (i.e., from ending to the starting point). In [53] they proposed a pre-computation approach for processing OSR queries in both vector and metric spaces, in which firstly the Voronoi diagram [11] of solution space is constructed based on geometric properties of the solution space, and then the OSR route is obtained by recursively accessing the pre-constructed Voronoi diagram.

Unlike sequenced trip planning queries [52, 53, 32], where the query contains the total order of all COIs to be visited, Li et al. [39], Chen et al. [13] and Li et al. [37] studied processing the route queries with arbitrary and multi-partial order constraints. Chen et al. [13] proposed two heuristics for processing route queries with multi-partial order constraints. The first, named NNPSR, finds an approximate solution for the query by applying a greedy - approach; the second retrieves the nearest point of the query start position $q$ in every category, and then connects them to form a route. In addition, they also developed a simple combination of NNPSR and R-LORD [52], which answers a special case of the optimal route query with a total order of the categories to be visited. The hybrid solution first runs NNPSR to find a greedy route; then, it extracts the category of each point on the greedy route, and runs R-LORD with this category sequence as input. Similarly, Li et al. [37] investigated a variant of the optimal route query that specifies both a start point and an end position, but no order constraint between the COIs that are supposed to be visited.

For processing the optimal route queries with arbitrary order constraints, Li et al. [39] proposed two different techniques namely *Backward* search and *Forward* search algorithms. The former computes the optimal route from the last point to the first (similar to the R-LORD algorithm [52]), while the latter follows the first-to-last order of points. Although [39], [37] and [13] use the same query definition

as processing the route queries with arbitrary order constraints, the main difference between [39], [13] and [37] is that none of the solutions in [13] and [37] guarantees the optimality of the results, where [39] returns the optimal routes for route queries with arbitrary order constraints.

Kanza et al. [32] investigated interactive route search in the presence of order constraints that specifies that some types of entities should be visited before others. In an interactive route search, initially the user poses a route-search query; however, instead of providing to the user just one complete and unchanging route, the service provider creates the route gradually while interacting with the user. In each step, the system provides the next geographical entity on the route. The user goes to the entity and provides to the service provider feedback on whether the entity has satisfied the user. The feedback is used for computing the rest of the route.

Yan et al. [65] investigated Traffic-Aware Route search (TARS). In a TARS query, the user provides start and target locations and the COIs that must be visited, as well as time constraints for visiting the COIs. The goal of TARS query is to find the fastest route from the start location to the target via one POI from each COI by considering the traffic conditions of road network. The TARS query is also considered an interactive query, in which there is a possibility that some of the recommended/visited POIs will not satisfy the user. In this case, another POI from the same COI will be recommended. For processing TARS queries, Yan et al. [65] proposed three heuristic-based algorithms: a local greedy approach, a global greedy approach and an algorithm that computes a linear approximation to the travel speeds, formulates the problem as a Mixed Integer Linear Programming (MILP) problem and uses a solver to find the solution. In [71], Zhu et al. studied the trip search problem on categorical POI networks and proposed a spatial sketch-based approximate algorithm to maximize user satisfaction score within a given distance or travel time threshold.

The Multi-Request Route Planning (MRRP) problem was investigated in [41], proposing a framework to efficiently find a route where the user-specified requests can be served. Considering that in urban environments a POI may provide various kinds of services, in MRRP queries the goal is to plan a route for serving

multiple user-specified requests. The proposed framework in [41] for processing MRRP queries consists of two major modules: *planning module*, in which pruning and caching strategies are applied for planning a preliminary route, and *refinement module*, where refinement mechanisms were proposed for further enhancing the quality of the route. Shang et al. proposed the Path Nearby Cluster (PNC)[51] query to find regions of potential interest along the user-specified travel route. The authors considered the recommendation would be better if the best spatial distance and cluster density are taken into account.

Soma et al. [59] investigated trip planning queries with location privacy in spatial databases. The motivation behind this query is that users may not wish to disclose their exact locations to the location-based service provider (LBSP). The authors proposed a solution for processing TPQs without disclosing a user's actual source and destination locations to the LSP. The proposed technique protects the user's privacy by sending either a false location or a cloaked location of the user to the LSP but provides exact results of the TP queries. The key idea behind the proposed technique is that it refines the search space as an elliptical region using geometric properties.

### 2.2.2 Group Trip Planning Queries

Hashem et al. [26] proposed the Iterative Approach (IA) and Hierarchical Approach (HA) for processing SGTPQs in Euclidean spaces. Since the latter is not applicable to road networks, we focus on the former. IA iteratively examines different candidate trips, constructed based on a depth-first search strategy. This approach first generates an initial "greedy" group trip by repeatedly visiting the nearest POI belonging to an unvisited category (according to the predefined sequence of COIs). Then it iteratively generates and examines other alternative solutions by traversing the trip plan just obtained backwards and greedily replacing POIs (in the already examined group trips) with the next nearest ones.

The main drawback of IA [26] is that it does not keep a summary of previous discovered results, i.e., it is stateless, thus yielding high query processing cost for repeatedly retrieving and examining POIs already examined. Furthermore, IA can-

not provide an optimal answer. Therefore, we propose Revised Iterative Approach (RIA) as a modified version of IA[8], so that it can provide the optimal solutions

Samrose et al. [47] proposed the GOSR technique for processing SGTPQs in road networks. First an initial greedy route is constructed by iteratively applying nearest neighbor queries. Then, an ellipse for each user in the group is computed, where the focal points of each ellipse are located on the corresponding user's source and target locations, and the major axis of all ellipses equals to the initial group total travel distance computed based on the initial greedy solution. The intersection area of all computed ellipses is, provably, a refined search space, containing the POIs belonging to the optimal group route. The main shortcoming of this approach is that it becomes inefficient with the increase of group size particularly when the source and target locations of all group members located in far distances.

Hashem et al. [25] proposed the R-GTP and I-GTP approaches for processing SGTPQs, the problem that we address in this chapter. The R-GTP approach first finds a greedy solution based on iteratively extracting the nearest POI from the next unvisited COI in the given sequence of COIs, where the first POI is extracted as the nearest neighbor towards the centroid of source locations. Assuming $M_D$ as the length of the shortest group trip computed so far, the search space is refined by using the geometric properties of ellipses, in which all POIs with the total Euclidean distance towards the centroids of source and target locations greater than $M_D/n$ are discarded.

After shrinking the search space into a smaller refined area, R-GTP applies a dynamic programming-based approach in order to discover the optimal sequence of POIs among all POIs within the refined search space. The efficiency of this approach significantly deteriorates with the increase in the query area or when the cardinalities of COIs are not balanced. Assuming that all POIs of different COIs have been indexed using an $R^*$-tree, the I-GTP approach applies a best-first search strategy to incrementally retrieve and examine POIs in order of total distance to the centroids of source locations and destinations. For each dequeued POI, the algorithm checks whether it is possible to further minimize the shortest group trip discovered so far by computing new trips via the last dequeued POI. Furthermore,

the algorithm updates the refined POI search space with the incremental retrieval of POIs until the optimal group route has been identified. I-GTP's performance deteriorates with the increase of the query area, and similarly to R-GTP, when the cardinality of COIs is not balanced. Since all POIs from different COIs are indexed in one single $R^*$-tree, the POIs belonging to dense COIs have a higher chance for being extracted in comparison to those from sparse COIs. This leads to high query processing response time due to slow convergence towards the optimal answer.

Shang et al. [49] proposed Collective Trip Planning (CTP) queries in spatial databases. The goal of this query is to find the lowest-cost route connecting multiple sources and a destination, via at most $k$ meeting points. The motivation behind this query is that, when multiple travelers target the same destination, they may want to assemble at meeting points and then go together to the destination by public transport to reduce their global travel cost. For processing CTP queries efficiently, Shang et al. proposed two algorithms, including an exact algorithm and an approximation algorithm. The exact algorithm is capable of finding the optimal result for small values of $k$ (e.g., $k$=2) in interactive time, while the approximation algorithm, which has a 5-approximation ratio, is suitable for other situations.

Hashem et al. [27] proposed the Sub-Group Trip Planning queries in spatial databases. Given a group $G$ of $n$ users, the minimum subgroup size $n'$, a set of source locations $S$, a set of destination locations $D$, sets of $m$ types of data points, and an aggregate function $f$ the SGTP query returns for every subgroup size $n'' \in [n', n]$, a subgroup $G' \in G$ of $n''$ users and a set of data points that minimizes $f$.

Assuming that all POIs of different COIs have been indexed in separate $R^*$-trees Hashem et al. proposed a solution that evaluates the query answers for different subgroup sizes *concurrently*. This algorithm traverses the $R^*$-trees hierarchically top-down pruning the POIs that cannot be part of the SGTP answer using the smallest aggregate trip distance computed based on the already retrieved POIs from the database.

Jahan et.al [31] studied the problem of Group Trip Scheduling (GTS) in spatial databases. Given source and destination locations of group members, a GTS query enables a group of $n$ members to schedule $n$ individual trips such that all trips visit

together the required types of POIs and the total trip distance of $n$ group members is minimized. In the proposed solution for processing GTS queries, the authors proposed a dynamic-based programming technique that exploits the geometric properties of ellipses to refine the POI search space and prune POIs to reduce the number of possible combinations of trips among group members.

Tabassum et al. [60] introduced the concept of *dynamic groups* for Group Trip Planning queries and proposed the Dynamic Group Trip Planning (DGTP) queries. The traditional GTP query assumes that the group members remain static or fixed during the trip, whereas in the proposed DGTP queries, the group changes dynamically over the duration of a trip where members can leave or join the group at any POI such as a shopping center, a restaurant or a movie theater. The changes of members in a group can be either predetermined (i.e., group changes are known before the trip is planned) or in real-time (changes happen during the trip). The proposed solution in [60] for processing DGTP queries exploits the trip information of users to compute a pruning bound based on elliptical properties that allows it to search a small data space instead of the entire database.

## 2.3 Problem Definition

Before we present our proposed solutions for solving SGTPQs, we need to formally define the notions of group total travel distance and the SGTPQ itself. The symbols used in the remainder of this chapter are summarized in Table 2.1.

**Definition 2.3.1.** Given a set of source locations $S$ and destinations $D$, the *group total travel distance* for visiting a given sequence of POIs $O$=$(o_1, o_2, ..., o_m)$ is defined as:

$$TD(p) = d_n(o_1, S) + \ n \times \sum_{i=1}^{m-1} d_n(o_i, o_{i+1}) + d_n(o_m, D) \tag{2.1}$$

**Definition 2.3.2.** Given a set of source locations $S$ and destinations $D$, and a sequence of $m$ COIs $C$ the Sequenced Group Trip Planning Query, denoted as SGTPQ$(S, D, C)$, returns a sequence of POIs $O$ to be visited by all users together such that $TD(O)$ is minimal.

Table 2.1: Meaning of symbols used in SGTPQs

| | |
|---|---|
| $n$ | Number of users |
| $S = \{S_1, ..., S_n\}$ | Source locations of the users |
| $D = \{D_1, ..., D_n\}$ | Destinations of the users |
| $m$ | The number of COIs |
| $C = (C_1, ..., C_m)$ | An ordered sequence of COIs |
| $O = (o_1, o_2, ..., o_m)$ | A sequence of POIs |
| $o_i$ | A POI belonging to $C_i$ |
| $c_s, c_d$ | Centroid of $S$ and $D$ |
| $d_e(.,.), d_n(.,.)$ | Euclidean and Network distances |
| $d_n(a, S)$ | $d_n(a, S) = \sum_{i=1}^{n} d_n(a, S_i)$ |
| $d_n(a, D)$ | $d_n(p, D) = \sum_{i=1}^{n} d_n(a, D_i)$ |
| $d_e(a, S)$ | $d_e(a, S) = \sum_{i=1}^{n} d_e(a, S_i)$ |
| $d_e(a, D)$ | $d_e(a, D) = \sum_{i=1}^{n} d_e(a, D_i)$ |
| $M_D$ | The smallest group total travel distance computed so far |

## 2.4 Proposed Solutions

For processing SGTPQs, we proposed three approaches. The first approach, called Revised Iterative Approach (RIA) [6], is a modified version of Iterative Approach (IA) [26], so that it can provide the optimal solutions. Next we proposed the Progressive Group Neighbor Exploration (PGNE) approach [6] which traverses the search approach based on a mixed Breadth-Depth First search strategy. Our third proposed approach, Iterative Backward Search (IBS) approach [4], discovers the optimal sequenced group trip by traversing the search space based on a Depth-First search strategy. In the following, we discuss each of these proposed solutions with more details.

### 2.4.1 Revised Iterative Approach (RIA)

In this section, we discuss the IA and its drawbacks and our modifications on this approach. Although Hashem et al. [26] claimed that IA obtains the optimal solution (the shortest group trip), we show this approach is not able to produce the optimal solution, and we made some modifications on this approach to make it obtain the shortest possible group trip.

IA makes use of Group Nearest Neighbor (GNN) queries [42]. Given two sets

of points $P$ and $Q$, a GNN from $P$ with regards $Q$ returns a point from $P$, which has the smallest sum of distances to all points in $Q$. For example, let us suppose $n$ users are currently at locations $Q = \{q_1, ..., q_n\}$, and they are interested in choosing a restaurant to have dinner together among a set of restaurants at locations $P = \{p_1, ..., p_m\}$ in the city. The GNN query will return a restaurant with the smallest sum of distances to all users in the group. In this work, we denote by $o_i^k$=$GNN(k, \{a_1, ..., a_n\}, C_i)$ as the POI which belongs to category $C_i$ that has the $k^{th}$ smallest total Euclidean distance w.r.t. locations $\{a_1, ..., a_n\}$. In this work, we used the *single point method* [42] for processing GNN queries.

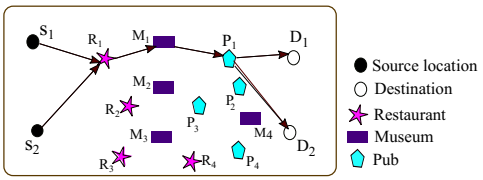



Figure 2.2: A sample road network with three types of POIs.

We describe IA by using the example of Figure 2.2, where $S = \{S_1, S_2\}$ and $D = \{D_1, D_2\}$ are the source locations and destinations of a group of two users, and $(R, M, P)$ is a sequence of three COIs that must be visited. The Iterative Approach works as follows: It firstly determines the first GNN from category $R$ w.r.t. $S$, $R_1$=$GNN(1, S, R)$. Then it finds the first nearest neighbor from category $M$ w.r.t. $R_1$, $M_1$=$NN(1, R_1, M)$[1]. Then it determines the first GNN from $P$ w.r.t. $M_1$ and $D$, $P_1$=$GNN(1, M_1 \cup D, P)$, where the first alternative trip $R_1 \rightarrow M_1 \rightarrow P_1$ will be formed.

After that, the Iterative Approach backtracks to $R_1$ and connects $R_1$ to its next NN from $M$, $M_2$=$NN(2, R_1, M)$, and continues with the prefix $R_1 \rightarrow M_2$. For that, IA finds the first GNN from $P$ w.r.t. $M_2$ and $D$, $P_2$=$GNN(1, M_2 \cup D, P)$, and checks whether the new group trip $R_1 \rightarrow M_2 \rightarrow P_2$ is shorter than the shortest group trip found so far as $R_1 \rightarrow M_2 \rightarrow P_2$. After that, IA backtracks again to $R_1$, and continues with the prefix $R_1$. When all of possible trips with prefix $R_1$ have

[1]We say point of interest $p_i^k$ belonging to category $C_i$ is the $k^{th}$-Nearest Neighbour (NN) w.r.t. location $a$, $NN(k, a, C_i)$, if it has the $k^{th}$-smallest Euclidean distance w.r.t. $a$ among all points belonging to $C_i$.

**Algorithm 1:** RIA for a sequence of three COIs

**Input:** $S = \{S_1, ..., S_n\}$, $D = \{D_1, ..., D_n\}$, $C = (C_1, C_2, C_3)$
**Output:** $R_{opt}$

$M_D = \infty$
$A = \{\}$
$l = 1$
**repeat**
  $o_1^l$= GNN$(l, S, C_1)$
  $v = 1$
  **repeat**
    $o_2^v$= NN$(v, o_1^l, C_2)$
    $j = 1$
    **repeat**
      $o_3^j$= GNN$(j, n \times o_2^v \cup D, C_3)$
      **if** $TD(o_1^l, o_2^v, o_3^j) < M_D$ **then**
        $R_{opt}$= $(o_1^l, o_2^v, o_3^j)$
        $M_D = TD(o_1^l, o_2^v, o_3^j)$
      $j = j + 1$
    **until** $d_n(o_1^l, S) + n \times d_n(o_1^l, o_2^v) + n \times d_e(o_2^v, o_3^j) + d_e(o_3^j, D) \leq M_D$
    $v = v + 1$
  **until** $d_n(o_1^l, S) + n \times d_e(o_1^l, o_2^v) \leq M_D$
  $l = l + 1$
**until** $d_e(o_1, S) \leq M_D$
**return** $R_{opt}$

been checked, then the Iterative Approach backtracks one step more and connects the source locations $S$ to the second GNN from category $R$, $R_2$=$GNN(2, S, R)$, and continues with prefix $R_2$. The process terminates when all feasible trips are examined, and the shortest route is reported as the query result.

Assuming that $C_m$ is the last COI that must be visited in sequence $(C_1, ..., C_{m-1}, C_m)$, and $o_{m-1} \in C_{m-1}$, the reason why the IA approach is not able to obtain the optimal solution is that the first GNN from $C_m$ w.r.t. $o_{m-1} \cup D$ does not necessarily produce the optimal sub-trip from $o_{m-1}$ w.r.t. the users' destinations. Although for computing the total travel distance, the distance from $o_{m-1} \in C_{m-1}$ to $o_m \in C_m$ is multiplied by the group size, but in the IA approach, $o_m$ is retrieved as a POI which has the minimum total distances w.r.t. $o_{m-1} \cup D$, regardless of the issue that the path between $o_m$ and $o_{m-1}$ is traversed by all group members.

This issue can be better explained by using the example illustrated in Figure 2.3, where $p_1$ belongs to $C_1$, and two POIs $p_2$ and $p'_2$ belonging to $C_2$, are respectively the $1^{st}$ and $2^{nd}$ GNNs towards $p_1 \cup D$, i.e, $p_2 = GNN(1, p_1 \cup D, C_2)$ and $p'_2 = GNN(2, p_1 \cup D, C_2)$. We observe that, although $p_2$ is the first GNN with respect to $p_1 \cup D$, but the group trip $(p_1, p'_2)$ with total group travel distance as 10 $(1 + 1 + 2 \times 1 + 3 + 3)$ will produce a shorter trip, in comparison to $(p_1, p_2)$ with total group travel distance as 11 $(1 + 1 + 2 \times 3 + 1 + 2)$.

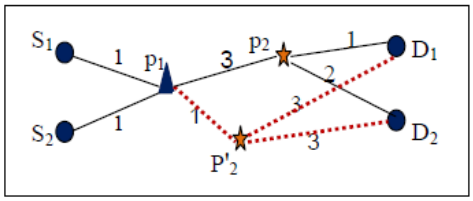



Figure 2.3: An example scenario

For solving this issue, we propose the *Revised* Iterative Approach (RIA) [6], where POIs from last category are retrieved w.r.t. a specific set containing $n$ times $o_{m-1}$ as well as set $D$. In other words, contrary to Iterative Approach, in RIA while retrieving POIs from last category, we will consider the location of previously retrieved POI (i.e., $o_{m-1}$) $n$ times (where $n$ is the group size). Algorithm 1 illustrates the Revised Iterative Approach for a sequence of three COIs. We stress that the apparently minor but crucial difference between RIA and IA is that for retrieving POIs from $C_3$, the RIA approach considers the location of $o_2$ for *all* users, i.e., $n$ times (Line 11).

In this section, we discuss the pruning strategy applied in the RIA approach. Assuming that in the given SGTPQ a sequence of $m$ COIs as $(C_1, C_2, ..., C_m)$ is to be visited, we divide the POIs belonging to these $m$ COIs into three different groups. The first group contains POIs belonging to $C_1$ , where POIs belonging to COIs $C_2, ..., C_{m-1}$ form the second group. The third group contains POIs belonging to $C_m$. The reason for this division is that the POIs belonging to the first group are supposed to be pruned based on the total Euclidean distance towards $S$. The POIs belonging to the second group are supposed to be appended to a pre-formed partial

group trip. Finally the POIs belonging to the third group are supposed to be pruned based on the last POI belonging to the previously formed partial trip and $D$. In the following, Lemma 2.4.1 specifically discusses the pruning of POIs belonging to the first group. Similarly, in Lemmas 2.4.2 and 2.4.6 2.4.3 we discuss the pruning strategy applied for POIs belonging to the second and third groups.

**Lemma 2.4.1.** Let $(C_1, C_2, ..., C_m)$ be an ordered sequence of COIs. Let $M_D$ be the smallest group total travel distance computed so far. Assuming $o_1^l$= GNN$(l, S, C_1)$, then there would be no other unexplored POI from $C_1$ belonging to optimal group trip, if the condition $d_e(o_1, S) > M_D$ becomes true.

*Proof.* Let $o_1^t$= GNN$(t, S, C_1)$, where $t > l$. Then we have $d_e(o_1^t, S) > d_e(o_1^l, S) > M_D$. This means that the total Euclidean distance of $o_1^t$ towards $S$ would be greater than $M_D$. So $o_1^t$ cannot be part of optimal group trip. □

**Lemma 2.4.2.** Let $(C_1, C_2, ..., C_m)$ be an ordered sequence of COIs. Let $M_D$ be the smallest group total travel distance computed so far. Furthermore, let $(o_1, o_2, ..., o_i)$ be a sequence of POIs, where $1 < i < m - 1$. Assuming $o_{i+1}^v$= NN$(v, o_i, C_{i+1})$, then there would be no other unexplored POI from $C_{i+1}$ belonging to optimal group trip, if the condition $d_n(o_1, S) + \sum_{j=1}^{i-1} d_n(o_j, o_{j+1}) + n \times d_e(o_{i+1}^v, o_i) > M_D$ becomes true.

*Proof.* Let $o_{i+1}^t$= NN$(t, o_i, C_{i+1})$, where $t > l$. Then we have $d_n(o_1, S) + \sum_{j=1}^{i-1} d_n(o_j, o_{j+1}) + n \times d_e(o_{i+1}^t, o_i) > d_n(o_1, S) + \sum_{j=1}^{i-1} d_n(o_j, o_{j+1}) + n \times d_e(o_{i+1}^v, o_i) > M_D$. This means that the the group trip for vising $(o_1, o_2, ..., o_i, o_{i+1}^t)$ would incur group total travel distance greater than $M_D$. So $o_{i+1}^t$ cannot be part of optimal group trip. □

**Lemma 2.4.3.** Let $(C_1, C_2, ..., C_m)$ be an ordered sequence of COIs. Let $M_D$ be the smallest group total travel distance computed so far. Furthermore, let $(o_1, o_2, ..., o_{m-1})$ be a sequence of POIs. Assuming $o_m^l$= GNN$(l, n \times o_{m-1} \cup D, C_m)$, then there would be no other unexplored POI from $C_{i+1}$ belonging to optimal group trip, if the condition $d_n(o_1, S) + \sum_{j=1}^{m-2} d_n(o_j, o_{j+1}) + n \times d_e(o_m^l, o_{m-1}) + d_e(o_m^l, D) > M_D$ becomes true.

*Proof.* Let $o_m^t$= GNN$(t, n \times o_{m-1} \cup D, C_m)$, where $t > l$. Then we have $n \times d_e(o_m^t, o_i) + d_e(o_m^t, D) > n \times d_e(o_m^l, o_i) + d_e(o_m^l, D)$, resulting in $d_n(o_1, S) + \sum_{j=1}^{m-2} d_n(o_j, o_{j+1}) + n \times d_e(o_m^t, o_{m-1}) + d_e(o_m^t, D) > M_D$. This means that the the group trip for vising $(o_1, o_2, ..., o_{m-1}, o_m^t)$ would incur group total travel distance greater than $M_D$. So $o_m^t$ cannot be part of optimal group trip. □

As mentioned before, in IA approach the POIs from the first and last categories are selected to be examined by applying GNN queries w.r.t. all users' source locations and destinations, respectively. The first drawback of IA is that with the increase of number of users in the group (i.e., $n$ become larger) selecting of POIs from the first and last categories will incur large computational overhead due to the processing of GNN queries w.r.t. $S$ and $D$. The second drawback of IA is that this approach does not consider the location of destinations $D = \{D_1, ..., D_n\}$, while iteratively retrieving POIs from $C_1$ to $C_{m-1}$. Motivated by these two drawbacks of IA [26] on SGTPQs, we propose the *Progressive Group Nearest Neighbour Exploration* (PGNE) approach, which traverses the search space based on a mixed Breadth-Depth First Search strategy.

### 2.4.2 PGNE Approach

The idea behind the PGNE approach [6] is to incrementally create a set of Partial Group Trips (PGTs) in the form of $PGT = (o_1, ..., o_h)$, $o_h \in C_h$, $1 \leq h \leq m$ and store them in a priority queue, ordered based on the *group total travel distance lower bound*. Each partial group trip $PGT = (o_1, ..., o_h)$ is stored in the queue in the form of $(PGT, len(PGT), TD_{LB}(PGT))$, where $TD_{LB}(PGT)$ defines a *lower bound* for the group total travel distance of PGT that is estimated as the sum of the length of the PGT with the total *Euclidean* distances from the last POI in the PGT (i.e., $o_h$) w.r.t. the users' destinations as the following:

$$TD_{LB}(PGT) = len(PGT) + \sum_{i=1}^{n} d_e(o_h, D_i) \tag{2.2}$$

where $d_e(.,.)$ indicates the Euclidean distance between two locations, and $len(PGT)$ represents the length of PGT as the total group travel distance from users' source

locations $S$ to the last POI in the PGT (i.e., $o_h$) through points $(o_1, o_2, ..., o_{h-1})$. Formally, $len(PGT)$ is defined as:

$$len(PGT) = \sum_{i=1}^{n} d_n(S_i, o_1) + n \sum_{i=1}^{h-1} d_n(o_i, o_{i+1}) \tag{2.3}$$

The PGNE approach is an iterative process, in which, firstly we determine the first GNN from $C_1$ w.r.t. $\{c_s \cup c_d\}$ (i.e., $o_1^1 = GNN(1, \{c_s, c_d\}, C_1)$) generating the first Partial Group Trip as $PGT = (o_1^1)$ with length $\sum_{i=1}^{n} d_n(S_i, o_1^1)$, where the $TD_{LB}(PGT)$ is estimated as $TD_{LB}(PGT) = \sum_{i=1}^{n} d_n(S_i, o_1^1) + \sum_{i=1}^{n} d_e(o_1^1, D_i)$. Then, we will place the new generated PGT in a priority queue, where the elements of this queue are ordered based on $TD_{LB}(PGT)$. In each subsequent iteration of PGNE, a PGT from the top of the priority, which has the smallest $TD_{LB}$, is fetched and we perform two operations on it: *progressive operation* and *replacement operation*. In the replacement operation, the last POI in the $PGT$ will be replaced with another POI from the same category. On the other hand, in the progressive operation, based on the predefined order of visiting COIs, a POI from next COI will be appended into PGT. In the following we present the pruning strategies applied in PGNE approach, and then we discuss the two basic applied operations; replacement and progressive operations, with more details.

#### 2.4.2.1 Applied Pruning Strategies

In this section, we discuss the pruning strategy applied in the PGNE approach. Assuming that in the given SGTPQ a sequence of $m$ COIs as $(C_1, C_2, ..., C_m)$ is to be visited, we divide the POIs belonging to these $m$ COIs into two different groups. The first group contains POIs belonging to $C_1$, where POIs belonging to COIs $C_2, ..., C_m$ form the second group. The reason for this division is that POIs belonging to the second group are supposed to be appended to a pre-formed partial group trip. In the following, Lemma 2.4.4 specifically discusses the pruning of POIs belonging to the first group. Similarly, in Lemmas 2.4.5 and 2.4.6 we discuss the pruning strategy applied for POIs belonging to the second group.

**Lemma 2.4.4.** Let $o_1^k$ be the $k^{th}$ GNN from $C_1$ with respect to $c_s$ and $c_d$, i.e, $o_1^k = GNN(k, \{c_s, c_d\}, C_1)$, and let $M_D$ be the smallest group total travel distances

computed so far. Then, $o_1^k$ cannot be part of the optimal answer, as well as, there can be no other unexplored point in $C_1$ that can further minimize $M_D$, if:

$$d_e(o_1^k, c_s) + d_e(o_1^k, c_d) > M_D/n \tag{2.4}$$

*Proof.* According to [25], for $o_1^k$, we have $\sum_{i=1}^{n} d_e(o_1^k, S_i) \geq n \times d_e(o_1^k, c_s)$ and $\sum_{i=1}^{n} d_e(o_1^k, D_i) \geq n \times d_e(o_1^k, c_d)$. These two inequalities result in

$$\sum_{i=1}^{n} d_e(o_1^k, S_i) + \sum_{i=1}^{n} d_e(o_1^k, D_i) \geq n \times (d_e(o_1^k, c_s) + d_e(o_1^k, c_d)) \tag{2.5}$$

Then, assuming that the Inequality $d_e(o_1^k, c_s)$+$d_e(o_1^k, c_d) > M_D/n$ is true, based on inequality 2.5 we can conclude

$$\sum_{i=1}^{n} d_n(o_1^k, S_i) + \sum_{i=1}^{n} d_n(o_1^k, D_i) \geq \sum_{i=1}^{n} d_e(o_1^k, S_i) + \sum_{i=1}^{n} d_e(o_1^k, D_i) > M_D \tag{2.6}$$

Inequality 2.6 means that, since the total distance of $o_1^k$ to $S$ and $D$ is greater than the smallest group total distance computed so far, $o_1^k$ cannot be part of the optimal answer. After that, for any $t^{th}$ GNN from $C_1$ with respect to $c_s$ and $c_d$ such as $o_1^t$, i.e., $o_1^t = GNN(t, \{c_s, c_d\}, C_1)$, where $t > k$, we have

$$d_e(o_1^t, c_s) + d_e(o_1^t, c_d) > d_e(o_1^k, c_s) + d_e(o_1^k, c_d) \tag{2.7}$$

Then by considering inequalities 2.4 and 2.7 we have $d_e(o_1^t, c_s) + d_e(o_1^t, c_d) \leq M_D/n$. This means that $o_1^t$ cannot be part of the optimal group trip, since the total distance of $o_1^t$ to $S$ and $D$ would be greater than the smallest group total distance computed so far. □

**Lemma 2.4.5.** Let $PGT = (o_1, o_2, ..., o_h)$, $o_h \in C_h$, $1 \leq h < m$, be a partial group trip. The following inequality holds for any point $o_{h+1} \in C_{h+1}$:

$$TD(o_1, o_2, ..., o_h, o_{h+1}) \geq len(PGT) + n \times d_e(o_h, o_{h+1}) + n \times d_e(o_{h+1}, c_d) \tag{2.8}$$

*Proof.* According to [25], for any point $o_{h+1} \in C_{h+1}$, we have $d_n(o_{h+1}, D) \geq d_e(o_{h+1}, D) \geq n \times d_e(o_{h+1}, c_d)$. Then by adding $len(PGT) + n \times d_n(o_h, o_{h+1})$ to both sides of Inequality $d_n(o_{h+1}, D) \geq n \times d_e(o_{h+1}, c_d)$, we have

$$len(PGT)+n\times d_n(o_h, o_{h+1})+d_n(o_{h+1}, D) \geq len(PGT)+n\times d_n(o_h, o_{h+1})+n\times d_e(o_{h+1}, c_d) \tag{2.9}$$

The left side of inequality 2.9 equals to $TD(o_1, o_2, ..., o_h, o_{h+1})$. On the other hand, since $d_n(o_h, o_{h+1}) \geq d_e(o_h, o_{h+1})$, we can replace $d_n(o_h, o_{h+1})$ with $d_e(o_h, o_{h+1})$ in the right side of inequality 2.9. This leads to inequality 2.8. □

**Lemma 2.4.6.** Let $PGT = (o_1, o_2, ..., o_h)$, $o_h \in C_h$, $1 \leq h < m$ be a partial group trip. Furthermore, let $o_{h+1}^k \in C_{h+1}$ be the $k^{th}$ GNN with respect to $o_h$ and $c_d$, i.e., $o_{h+1}^k = GNN(k, \{o_h, c_d\}, C_{h+1})$, and let $M_D$ be the smallest group total travel distance computed so far. Then, appending $o_{h+1}^k$ to partial group trip $PGT$ will result in forming a partial group trip with group total travel distance greater than $M_D$, and further, there can be no other unexplored point in $C_{h+1}$ such that appending it to $PGT$ can further minimize $M_D$, if we have

$$d_e(o_h, o_{h+1}^k) + d_e(o_{h+1}^k, c_d) > (M_D - len(PGT))/n \tag{2.10}$$

*Proof.* Inequality 2.10 can be restated as:

$$len(PGT) + n \times d_e(o_h, o_{h+1}^k) + n \times d_e(o_{h+1}^k, c_d) > M_D \tag{2.11}$$

where based on inequalities 2.8 and 2.11, we can conclude:

$$TD(o_1, ..., o_h, o_{h+1}^k) > M_D \tag{2.12}$$

Based on inequality 2.12, appending $o_{h+1}^k$ to $PGT$ will generate a partial group trip with group total travel distance greater than $M_D$. After that, for any $t^{th}$ GNN from $C_{h+1}$ with respect to $o_h$ and $c_d$, i.e., $o_{h+1}^t = GNN(t, \{o_h, c_d\}, C_{h+1})$, where $t > k$, we have

$$d_e(o_h, o_{h+1}^t) + d_e(o_{h+1}^t, c_d) \geq d_e(o_h, o_{h+1}^k) + d_e(o_{h+1}^k, c_d) \tag{2.13}$$

And then, based on inequalities 2.11 and 2.13, we have

$$len(PGT) + n \times d_e(o_h, o_{h+1}^t) + n \times d_e(o_{h+1}^t, c_d) > M_D \tag{2.14}$$

where inequalities 2.8 and 2.14 lead to $TD(o_1, ..., o_h, o_{h+1}^t) \leq M_D$. This means that appending $o_{h+1}^t$ to $PGT = (o_1, o_2, ..., o_h)$ generates a partial group trip with group total travel distance greater than $M_D$. □

#### 2.4.2.2 PGNE Algorithm

As aforementioned the PGNE approach applies a mixed breadth-depth first search strategy for exploring the search space. For that, PGNE performs replacement and progressive operations. While the former explores the search space in breadth-first search strategy, the latter examines other possible candidate solutions based on a depth-first search strategy. In the following we provide more details about the aforementioned operations.

**1) Replacement operation:** In this operation, we will replace the last POI in the $PGT = (o_1, ...., o_{h-1}, o_h^k)$, $o_i \in C_i$ with another POI from the same category as following:

- If $h = 1$ and $o_1^k$ is the $k^{th}$ GNN w.r.t. $\{c_s, c_d\}$, (i.e., $o_1^k = GNN(k, \{c_s, c_d\}, C_1)$), then we will find $o_1^{k+1}$ as the $(k+1)^{th}$ GNN from $C_1$ w.r.t. $c_s$ and $c_d$ (i.e, $o_1^{k+1} = GNN(k+1, \{c_s, c_d\}, C_1)$). The new generated partial group trip as $PGT' = (o_1^{k+1})$ will be inserted into $queue$, if the following inequality holds:

  $$d_e(o_1^{k+1}, c_s) + d_e(o_1^{k+1}, c_d) \leq (M_D)/n \tag{2.15}$$

  The reason is that if this inequality does not hold for $o_1^{k+1}$, then according to Lemma 2.4.4 the total network distance of $o_1^{k+1}$ from $S$ and $D$ would be greater than $M_D$.

- If $h > 1$ and $o_h^k$ is the $k^{th}$ GNN toward $\{o_{h-1}, c_d\}$, (i.e., $o_h^k = GNN(k, \{o_{h-1}, c_d\}, C_h)$), then first we extract $o_h^k$ from PGT and generate a new PGT as $PGT'' = (o_1, ..., o_{h-1})$. Then, the next GNN from $C_h$ w.r.t. $o_{h-1}$ and $c_d$ as $o_h^{k+1} = GNN(k+1, \{o_{h-1}, c_d\}, C_h)$ is found. Let us suppose $PGT'$=$(o_1, ..., o_{h-1}, o_h^{k+1})$ is the new generated partial group trip resulted from appending $o_h^{k+1}$ to $PGT''$, then $PGT'$ can be inserted into the queue, if the following inequality holds:

  $$d_e(o_{h-1}, o_h^{k+1}) + d_e(o_h^{k+1}, c_d) \leq (M_D - len(PGT''))/n \tag{2.16}$$

  If this inequality holds for $o_h^{k+1}$, then according to Lemma 2.4.6 the appending $o_h^{k+1}$ to $PGT'' = (o_1, ..., o_{h-1})$ will result the partial group trip

$PGT'$=$(o_1, ..., o_{h-1}, o_h^{k+1})$ with group total travel distance greater than $M_D$. In addition, there would be no other POI in $C_h$, that replacing $o_h^{k+1}$ with it in $PGT'$ can further minimize the $M_D$.

**2) Progressive operation:** This operation is done on $PGT = (o_1, ..., o_h)$, if the number of POIs in the PGT is less than $m$. In this operation, we will determine a POI $o_{h+1}^1 \in C_{h+1}$ to be appended to PGT, where $o_{h+1}^1$=$GNN(1, \{o_h, c_d\}, C_{h+1})$. Let us suppose $PGT'$=$(o_1, ..., o_h, o_{h+1}^1)$ is the new generated partial group trip resulted from appending $o_{h+1}^1$ to PGT. The new generated partial group trip $PGT'$ will be inserted into the queue, if the following inequality holds:

$$d_e(o_h, o_{h+1}^1) + d_e(o_{h+1}^1, c_d) \leq (M_D - len(PGT))/n \tag{2.17}$$

The reason is that if this inequality becomes false for $o_{h+1}^1$, then according to Lemma 2.4.6 the group total travel distance of $PGT'$ will become greater than $M_D$. Consequently, any further progressive operation on $PGT'$ will result a partial group trip with greater group total travel distance than $M_D$. Similarly, no further replacement operation on $o_{h+1}^1$ in $PGT'$ can further minimize the $M_D$.

When the algorithm proceeds, if the new generated partial group trip $PGT'$ visits all required COIs, and the group total travel distance of $PGT'$ is less than the smallest group total travel distance found so far as $M_D$, i.e., $TD(PGT') < M_D$, then $PGT'$ will be stored as the shortest group trip computed so far. PGNE terminates when the *queue* becomes empty.

Algorithm 2 illustrates the PGNE approach, where the replacement and progressive operations are presented in Algorithms 3 and 4, respectively. In Line 2 of the PGNE Algorithm, we find the first GNN from $C_1$ w.r.t. $\{c_s, c_d\}$, $o_1^1 = GNN(1, \{c_s, c_d\}, C_1)$. Subsequently, the first partial group trip is formed as $PGT = (o_1)$ in Line 3. In Line 4, the new generated partial group trip is stored in the $queue$. In Line 6, the partial group trip $PGT$, which has the smallest $TD_{LB}$, is extracted from top of $queue$. In Lines 7 and 9 replacement and progressive operations are called, respectively. Note that the progressive operation is done on $PGT$, only if the number of POIs in $PGT$ is less than $m$ (Lines 8-9). If the new generated partial group trip, resulted from progressive or replacement operation, visits all of the

**Algorithm 2:** PGNE Approach

**Input:** $S = \{S_1, ..., S_n\}$, $D = \{D_1, ..., D_n\}$, $C = (C_1, C_2, ..., C_m)$
**Output:** $(o_1, o_2, ...., o_m)$

1 $M_D = \infty$, $queue$=$\varnothing$
2 $o_1^1 = GNN(1, \{c_s, c_d\}, C_1)$
3 $PGT = (o_1^1)$
4 Insert $PGT$ into $queue$
5 **repeat**
6  Extract $PGT$ from top of $queue$
7  $PGT_1'$= $Replacement(PGT)$ // Alg. 3
8  **if** $|PGT| < m$ **then**
9  $PGT_2'$=$Progressive(PGT)$ // Alg. 4
10  **if** *(*$|PGT_1'| = m$ *and* $TD(PGT_1') < M_D$*)* **then**
11  update($R_{opt}$, $M_D$, $PGT_1'$, $TD(PGT_1')$)
12  **if** *(*$|PGT_2'| = m$ *and* $TD(PGT_2') < M_D$*)* **then**
13  update($R_{opt}$, $M_D$, $PGT_2'$, $TD(PGT_2')$)
14  **if** $|PGT_1'| == 1$ **then**
15  Insert $PGT_1'$ into $queue$ if Eq.2.15 holds for $PGT_1'$
16  **else**
17  Insert $PGT_1'$ into $queue$ if Eq.2.16 holds for $PGT_1'$
18  Insert $PGT_2'$ into $queue$ if Eq.2.17 holds for $PGT_2'$
19 **until** $queue$ *is not empty*

required COIs, and the corresponding group total travel distance is less than $M_D$, then the current shortest group trip as $R_{opt}$ will be updated (in Lines 10-13). In Lines 14-18, based on Lemmas 2.4.4 and 2.4.6, we explore the possibility that the new generated partial group trips as $PGT_1'$ and $PGT_2'$ can be inserted into $queue$. The algorithm terminates, when the $queue$ becomes empty (Line 19).

### 2.4.3 Iterative Backward Search Approach

The Iterative Backward Search (IBS), similarly to the IA approach, also first produces a greedy based trip (i.e., a local optimal solution) by repeatedly visiting the nearest POI belonging to the subsequent *unvisited* COI, and then *backtracks* step by step, iteratively connecting POIs in the local optimal solution to their next nearest neighbors and checking whether the new route is shorter than the current best one. However, IBS differs from IA in three important aspects:

**Algorithm 3:** Replacement operation in PGNE approach

$Replacement(PGT)$
**Input:** $PGT = (o_1, ..., o_{h-1}, o_h^k)$
**Output:** $PGT' = (o_1, ..., o_{h-1}, o_h^{k+1})$
**if** $|PGT| = 1$ **then**
  $o_h^{k+1} = GNN(k+1, \{c_s, c_d\}, C_1)$
  $PGT' = (o_h^{k+1})$
**else**
  $o_h^{k+1} = GNN(k+1, \{o_{h-1}, c_d\}, C_h)$
  $PGT' = (o_1, ..., o_{h-1}, o_h^{k+1})$
**return** $PGT'$

**Algorithm 4:** Progressive operation in PGNE approach

$Progressive(PGT)$
**Input:** $PGT = (o_1, ..., o_{h-1}, o_h)$
**Output:** $PGT' = (o_1, ..., o_{h-1}, o_h, o_{h+1}^1)$
$o_{h+1}^1 = GNN(1, \{o_h, c_d\}, C_{h+1})$
$PGT' = (o_1, ..., o_h, o_{h+1}^1)$
**return** $PGT'$

- IBS considers the centroids of the users source locations and destinations while retrieving POIs from different categories,
- IBS applies a strong and efficient pruning strategy for shrinking the POIs search pace, and, more importantly,
- IBS relies on the *Suffix Optimality Principle* (stated next) in order to avoid the main drawback of both IA and PGNE techniques, namely incurring the computational overhead due to retrieving and examining the same POIs multiple times during the processing of a single query.

**Lemma 2.4.7** (Suffix Optimality Principle)**.** Let $r^* = (o_1^*, o_2^*, ..., o_m^*)$ be the optimal answer for SGTPQ(S,D,C), and let $r = (o_h^*, o_{h+1}^*, ..., o_m^*)$ be any suffix of $r^*$. Then $r$ is the optimal solution for SGTPQ$(S', D, C')$, where: $S' = \{S_1', ..., S_n'\}$ where $\forall j\ S_j' = o_h^*$ and $C' = (C_{h+1}, ..., C_m)$.

*Proof.* Considering the notation in the lemma's statement, let us assume there is a better solution $\bar{r}$ than $r$ for ***SGTPQ***$(S', D, C')$, i.e., $TD(\bar{r}) < TD(r)$. Let us denote

by $\bar{r}^*$ the trip obtained by replacing the suffix $r$ in $r^*$ by $\bar{r}$. Then we must have $TD(\bar{r}^*) = TD(r^*) - TD(r) + TD(\bar{r})$, and since $\bar{r}^*$ is a better suffix than $r$ then we must have $TD(\bar{r}^*) < TD(r^*)$, which contradicts the optimality of $r^*$, i.e., $r$ must be the optimal solution for ***SGTPQ***$(S', D, C')$. □

The Suffix Optimality Principle (SOP) is simple but yet powerful as it will allow us to save a considerable amount of computation as we shall see in the experiments described in Section 2.5. Next we describe our main contribution, the IBS algorithm, highlighting where the SOP is used.

For the sake of simplicity, but without loss of generality, we illustrate the processing of the IBS algorithm (whose pseudo-code is shown in Algorithm 5, which in turn uses Algorithm 6 for exploring optimal partial trips from different POIs) by finding the optimal solution for $SGTPQ(S, D, C)$ assuming a small sequence $C = (C_1, C_2, C_3)$. We also make use of the following notation:

- $R^*[o_i]$: The optimal partial group trip that starts from $o_i$, i.e., the source location of all users is the same POI $o_i$, and the group visits the sequence $(C_{i+1}, ..., C_m)$.
- $len(o_1, ..., o_i)$: The length of the partial group trip $(o_1, ..., o_i)$, i.e., $d_n(o_1, S) + n \times \sum_{k=1}^{i-1} d_n(o_k, o_{k+1})$.

Finally, IBS makes use of several Lemmas in order to ensure correctness of its pruning criteria. In the following we discuss IBS with more details.

As the first step, IBS initializes the optimal group trip starting from all POIs belonging to categories $C_2$ and $C_3$ as *unknown* (Alg. 5, Lines 1-2). Next, IBS determines the $1^{st}$ group nearest-neighbour (GNN) from $C_1$ with respect to the centroid of source locations and to the centroid of destinations, i.e., $o_1^1 = GNN(1, \{c_s, c_d\}, C_1)$ (Alg. 5, Line 5). Then, using Algorithm 6, IBS starts discovering the optimal partial group trip starting from $o_1^1$. For this, first IBS computes the $1^{st}$ GNN from $C_2$ with respect to $\{o_1^1, c_d\}$ (Alg. 6, Line 6). After that IBS checks whether the optimal group trip starting from $o_2^1$ has been computed so far (Alg. 6, Line 10). Since $R^*[o_2^1]$ is still *unknown*, IBS starts computing $R^*[o_2^1]$ by iteratively retrieving next GNNs

**Algorithm 5:** IBS Algorithm

**Input:**
$S = \{S_1, ..., S_n\}$, //The group members source locations
$D = \{D_1, ..., D_n\}$, //The group members destinations
$C = (C_1, C_2, ..., C_m)$, // The sequence of COIs
**Output:** $R_{opt}$

**for** $o_i$ *in* $\{C_2, ..., C_m\}$ **do**
  Set $R^*[o_i]$ as *unknown*
**end**
$Cand_{seq} = \varnothing$, $M_D = \infty$, $Set_{C_1} = \varnothing$, $l = 1$
$o_1^l$= GNN$(l, \{c_s, c_d\}, C_1)$
**while** $d_e(c_s, o_1^l) + d_e(o_1^l, c_d) \leq \frac{M_D}{n}$ **do**
  **//According to Lemma 2.4.4**
  $Cand_{seq}.add(o_1^l)$
  $R^*[o_1^l]$= $Find_{opt}(o_1^l, Cand_{seq}, M_D)$ **//Alg. 2**
  **if** $len(o_1^l) + TD(R^*[o_1^l]) < M_D$ **then**
    $M_D = len(o_1^l) + TD(R^*[o_1^l])$
  **end**
  $Set_{C_1}.add(o_1^l)$
  $Cand_{seq}.remove(o_1^l)$
  $l = l + 1$
  $o_1^l$= GNN$(l, \{c_s, c_d\}, C_1)$
**end**
$o_1^* = \underset{\forall o_i \in Set_{C_1}}{\operatorname{argmin}} \; d_n(S, o_i) + TD(R^*[o_i])$
**return** $R^*[o_1^*]$ as $R_{opt}$

from $C_3$ with respect to $\{o_2^1, c_d\}$. To be more specific, first the $1^{st}$ GNN from $C_3$ with respect to $\{o_2^1, c_d\}$ is computed, i.e., $o_3^1 = GNN(1, \{o_2^1, c_d\}, C_3)$, where the first candidate trip is formed as $(o_1^1, o_2^1, o_3^1)$. Then, the algorithm determines the $2^{nd}$ GNN from $C_3$ with respect to $\{o_2^1, c_d\}$, i.e., $o_3^2 = GNN(2, \{o_2^1, c_d\}, C_3)$, and forms the second candidate solution as $(o_1^1, o_2^1, o_3^2)$. Note that in each step the value of $M_D$, the smallest group trip computed so far, is updated if we find a shorter trip than the current $M_D$ value.

The process of finding the next GNN from $C_3$ with respect to $\{o_2^1, c_d\}$ for appending to partial group trip $(o_1^1, o_2^1)$ will be stopped when for the $k^{th}$ GNN $o_3^k = GNN(k, \{o_2^1, c_d\}, C_3)$, the inequality $d_e(o_2^1, o_3^k) + d_e(o_3^k, c_d) > (M_D - len(o_1^1, o_2^1))/n$ is satisfied. The reason is that according to Lemma 2.4.6 once this condition be-

**Algorithm 6:** Optimal partial group trip discovering

1 $Find_{opt}(o_v^u, Cand_{seq}, M_D)$
**Input:**
$o_v^u$ // A POI belonging to $C_v$,
$Cand_{seq}$// A partial group trip,
$M_D$//The smallest group total travel distance so far,
**Output:**
$R^*[o_v^u]$ // The optimal group trip starting from $o_v^u$,
2 **if** $(v == |COIs|)$ **then**
3 $R^*[o_v^u] = o_v^u, TD(R^*[o_v^u]) = d_n(o_v^u, D)$
4 **else**
5 $Set_{C_{v+1}} = \varnothing$, $k = 1$
6 $o_{v+1}^k$= GNN$(k, \{o_v^u, c_d\}, C_{v+1})$
7 **while** $d_e(o_v^u, o_{v+1}^k) + d_e(o_{v+1}^k, c_d) \leq \frac{M_D - len(Cand_{seq})}{n}$ **do**
8 **//According to Lemma 2.4.6**
9 $Cand_{seq}.add(o_{v+1}^k)$
10 **if** $R^*[o_{v+1}^k]$ *is unknown* **then**
11 $R^*[o_{v+1}^k]$=$Find_{opt}(o_{v+1}^k, Cand_{seq}, M_D)$
12 **else**
13 $R^*[o_{v+1}^k]$ is retrieved **// use of SOP**
14 **if** $len(Cand_{seq}) + TD(R^*[o_{v+1}^k]) < M_D$ **then**
15 $M_D = len(Cand_{seq}) + TD(R^*[o_{v+1}^k])$
16 $Set_{C_{v+1}}.add(o_{v+1}^k)$
17 $Cand_{seq}.remove(o_{v+1}^k)$
18 $k = k + 1$
19 $o_{v+1}^k$= GNN$(k, \{o_v^u, c_d\}, C_{v+1})$
20 $o_v^* = \underset{\forall o_i \in Set_{C_{v+1}}}{\operatorname{argmin}} \; n \times d_n(o_v^u, o_i) + TD(R^*[o_i])$
21 $R^*[o_v^u] = (o_v^u, R^*[o_v^*])$
22 $TD(R^*[o_v^u]) = n \times d_n(o_v^u, o_v^*) + TD(R^*[o_v^*])$
23 Set $R^*[o_v^u]$ as *known*
24 Return $R^*[o_v^u]$

comes true, then there would be no other unexamined POI belonging to $C_3$ that appending it to $(o_1^1, o_2^1)$ would result in incurring smaller group travel distance in comparison to the smallest travel distance computed so far. Then, among all POIs belonging to $C_3$ which have been examined, we will select $o_3^* \in \{o_3^1, o_3^2, ..., o_3^k\}$ which incurs the smallest group total travel distance from $o_2^1$ w.r.t. group's destinations $D$ (Alg. 6. Line 20). Subsequently, we will update the optimal partial group trip

starting from $o_2^1$ as $R^*[o_2^1] = (o_2^1, o_3^*)$. Note that in Line 21, $R^*[o_v^u] = (o_v^u, R^*[o_v^*])$ indicates the optimal partial group trip which starts from $o_v^u$ and follows the route $R^*[o_v^*]$.

As the next step, IBS backtracks to $o_1^1$ and determines the $2^{th}$ GNN from $C_2$ with respect to $\{o_1^1, c_d\}$, i.e, $o_2^2 = GNN(2, \{o_1^1, c_d\}, C_2)$, (Alg. 6, Line 6), and consequently forms the partial group trip $(o_1^1, o_2^2)$. Since the optimal partial group trip starting from $o_2^2$ is still $unknown$, IBS repeats the same aforementioned procedure for $o_2^1$ in order to obtain the $R^*[o_2^2]$. To be more specific, IBS approach iteratively finds the next GNNs from $C_3$ with respect to $\{o_2^2, c_d\}$. Then, among all POIs belonging to $C_3$ which have been examined for appending to the prefix $(o_1^1, o_2^2)$, we select a POI such as $o_3^*$, which incurs the smallest group total travel distance from $o_2^2$ w.r.t. $D$ passing through $o_3^*$. Subsequently, $R^*[o_2^2]$ will be updated as $R^*[o_2^2] = (o_2^2, o_3^*)$. Then, IBS algorithm backtracks to point $o_1^1$, and finds the point $o_2^3$ as $o_2^3 = GNN(3, \{o_1^1, c_d\}, C_2)$, and forms the partial group trip $(o_1^1, o_2^3)$. Then IBS approach repeats the same aforementioned procedure for $o_2^1$ and $o_2^2$ to obtain $R^*[o_2^3]$.

Once again according to Lemma 2.4.6, the process of finding the next GNN from $C_2$ w.r.t. $\{o_1^1, c_d\}$ will be stopped, when for the $k^{th}$ GNN, $o_2^k$, the condition $d_e(o_1^1, o_2^k)$+$d_e(o_2^k, c_d) > (M_D - len(o_1^1))/n$ becomes true. Then, among all POIs belonging to $C_2$ which have been examined for appending to the prefix $(o_1^1)$, $Set_{c2} = \{o_2^1, o_2^2, ...o_2^k\}$, we select $o_2^*$ which minimizes the group total travel distance from $o_1^1$ w.r.t. the group's destinations (Alg. 6, Line 20). Based on selected $o_2^*$, then $R^*[o_1^1]$ will be updated as $R^*[o_1^1] = (o_1^1, R^*[o_2^*])$ (Alg. 6, Line 21), indicating a path which starts from $o_1^1$ and follows $R^*[o_2^*]$.

After exploring the optimal partial group trip starting from $o_1^1$, the IBS backtracks to $c_s$, and finds the $2^{nd}$ GNN from $C_1$ with respect to $\{c_s, c_d\}$ as $o_1^2$ (Alg. 5, Line 16). Then, IBS repeats the same procedure mentioned above for $o_1^1$, in order to obtain the optimal group trip starting from $o_1^2$ as $R^*[o_1^2]$. The process of finding the next GNN from $C_1$ with respect to $\{c_s, c_d\}$ will be stopped, when for the $l^{th}$ GNN $o_1^l$ the condition $d_e(o_1^l, c_s) + d_e(o_1^l, c_d) > M_D/n$ becomes true. The reason is that according to Lemma 2.4.4 (c.f. Appendix), once this condition becomes

true, then there would be no other unexamined POI from $C_1$ as part of a group trip incurring the smaller group travel distance in comparison to the shortest group trip computed so far. Then, among all POIs belonging to $C_1$ which have been examined as $Set_{c1} = \{o_1^1, o_1^2, ...o_1^l\}$, we will select $o_1^* \in Set_{C_1}$, which minimizes $d_n(S, o_1^*) + TD(R^*[o_1^*])$ (Alg. 5, Line 18). Finally, $R^*[o_1^*]$ will be returned as the final optimal answer.

One of the strong points of IBS is that, it avoids unnecessary repeated work by applying the *Suffix Optimality Principle* (Lemma 2.4.7). In order to illustrate the application of *Suffix Optimality*, let us suppose the current partial group trip $(o_1^2, o_2^k)$. Assuming that in the previous steps of IBS the optimal group trip starting from $o_2^k$ has already been computed as $R^*[o_2^k] = (o_2^k, o_3^*)$ , then there is no need for examining POIs from $C_3$. In this case, the pre-computed optimal suffix $R^*[o_2^k]$ will be appended to the current partial group trip $(o_1^2, o_2^k)$, forming the new candidate trip as $(o_1^2, o_2^k, o_3^*)$. That is, while iteratively retrieving POIs from different categories, IBS checks the availability of the optimal group trip starting from the new retrieved POI in order to avoid unnecessary processing.

### 2.4.4 Running example

We present IBS in more detail using the example of Figure 2.4. In this figure the road network has been illustrated as a network of equally sized connected squares, where the length of each edge of the small squares is 1 unit. Let us assume that a group of two users who are currently at locations $S_1$ and $S_2$ are interested in visiting a sequence of COIs $(A, B, C)$. The destinations of the group members after visiting the COIs are $D_1$ and $D_2$, respectively. Our goal is to find the optimal group trip minimizing the group total travel distance. Table 2.2 illustrates step by step the execution of IBS algorithm for this example scenario.

In the first step, we extract $a_2$ as the first GNN w.r.t. $\{c_s, c_d\}$ from A. In the second step, $b_2$ is retrieved as the first GNN w.r.t. $\{a_2, c_d\}$. In step 3, $c_2$ as the first GNN w.r.t. $\{b_2, c_d\}$ from $C$ will be examined, where the candidate group trips $(a_2, b_2, c_2)$ will be formed, and $M_d$ will be updated as 20. We also compute and store the optimal partial group trip starting from $c_2$ w.r.t. destinations (i.e., $R^*[c_2]$).

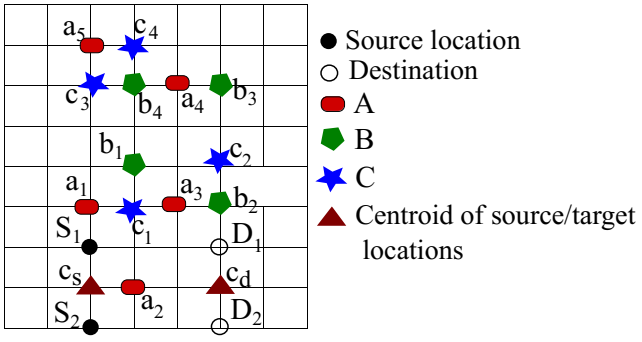



Figure 2.4: A sample road network with three types of POIs.

In step 4, $c_1$ as the second GNN w.r.t. $\{b_2, c_d\}$ will be examined. According to Lemma 2.4.6, examining different POIs from $C$ for appending to the partial group trip $PGT = (a_2, b_2)$ will be stopped in step 4, since for $c_1$ we have $d_e(b_2, c_1) + d_e(c_1, c_d) \simeq 4.8$, where $(M_D - len(a_2, b_2))/2 = (20 - 12)/2 = 4$. Subsequently, in step 5, $R^*[b_2]$ is formed.

After computing the optimal partial group starting from $b_2$, we aim at finding the optimal partial group trip starting from $b_1 = GNN(2, \{a_2, c_d\}, B)$. For that, through steps 7-9, we examine different POIs from $C$ for appending to the partial group trip $PGT = (a_2, b_1)$. According to Lemma 2.4.6, examining different POIs from $C$ w.r.t. partial group trip $(a_2, b_1)$, will be stopped at step 9, since we have $d_e(b_1, c_3) + d_e(c_3, c_d) \simeq 10.83$, where $(M_D - len(a_2, b_1))/2 = (20 - 10)/2 = 5$. Subsequently, in step 10, $R^*[b_1]$ is formed, where we select $c_1$ to be appended to partial group trip $PGT = (a_2, b_1)$, because it incurs the smallest group total travel distance among all examined POIs, i.e., $c_1$=argmin $2 \times d_n(b_1, o_i)$+$TD(R^*[o_i])$, where $o_i \in \{c_1, c_2\}$.

In step 11, $b_4$, the third GNN from $B$ w.r.t. $\{a_2, c_d\}$ is extracted. In this step, examining different POIs from $B$ for appending to $PGT = (a_2)$ will be stopped, since for $b_4$ we have $d_e(a_2, b_4) + d_e(b_4, c_d) \simeq 10.38$, where $(M_D - len(a_2))/2 = (20 - 4)/2 = 8$. Subsequently in step 12, $R^*[a_2]$ will be formed, where among all retrieved and examined POIs from $B$ for appending to partial $GTP = (a_2)$, we select a POI incurring the smallest group total travel distance, i.e., $b_2 =$ argmin $2 \times d_n(a_2, o_i) + TD(R^*[o_i])$, $\forall o_i \in \{b_2, b_1\}$. Recall that $R^*[b_2]$ and $R^*[b_1]$ have been

already computed and stored in steps 5 and 10, respectively. After computing the optimal partial group trip starting from $a_2$, then we start computing the optimal partial group trip starting from $a_3 = GNN(2, \{c_s, c_d\}, A)$. Meanwhile, the main point is that for each retrieved POI from $B$, we also need to check whether the optimal partial group trip starting from that POI has already been computed or not, in order to avoid unnecessary repeated work. To be more specific, considering that the optimal partial group trips starting from $b_2$ and $b_1$ have been already pre-computed in steps 5 and 10, respectively, according to the *suffix optimality principle* there is no need for extracting POIs from $C$ w.r.t. $b_1$ and $b_2$, and the previously computed optimal partial group trips can be used. In step 15, the shortest group trip computed so far will be updated as $(a_3, b_2, c_2)$ with $M_D = 18$.

A similar process will be repeated for $a_1 = GNN(3, \{c_s, c_d\}, A)$. In steps 22 and 24, the pre-computed optimal partial group trips for $b_2$ and $b_1$ will be re-used. Examining different POIs from $B$ for appending to the partial group $PGT = (a_1)$ will be stopped at step 25, since we have $d_e(a_1, b_4) + d_e(b_4, c_d) \simeq 8.54$, where $(M_D - len(a_1))/2 = (18 - 4)/2 = 7$. Then in step 26, $R^*[a_1]$ will be formed. In step 27, $a_4 = GNN(4, \{c_s, c_d\}, A)$ will be retrieved. According to Lemma 2.4.4, the examining different POIs from $A$ will be stopped in this step, since, we have $d_e(a_4, c_s)+d_e(a_4, c_d) \simeq 10.48$, where $(M_D)/2 = (18)/2 = 9$. Among all computed candidate trips starting from $a_1$, $a_2$ and $a_3$, the group trip $R^*[a_3] = (a_3, b_2, c_2)$, with group total travel distance 18, will be returned as the output of algorithm.

Table 2.2: Sample execution of IBS using the example of Figure 2.4.

| Step | Examined POIs from $A$ | Examined POIs from $B$ | Examined POIs from $C$ | $R^*$ |
|---|---|---|---|---|
| 1 | $a_2 = GNN(1, \{c_s, c_d\}, A)$ | | | |
| 2 | | $b_2 = GNN(1, \{a_2, c_d\}, B)$ | | |
| 3 | | | $c_2 = GNN(1, \{b_2, c_d\}, C)$ | $R^*[c_2] = c_2$ |
| 4 | | | $c_1 = GNN(2, \{b_2, c_d\}, C)$ halt! | |
| 5 | | | | $R^*[b_2] = (b_2, R^*[c_2])$ |
| 6 | | $b_1 = GNN(2, \{a_2, c_d\}, B)$ | | |
| 7 | | | $c_1 = GNN(1, \{b_1, c_d\}, C)$ | $R^*[c_1] = c_1$ |
| 8 | | | $c_2 = GNN(2, \{b_1, c_d\}, C)$ | |
| 9 | | | $c_3 = GNN(3, \{b_1, c_d\}, C)$ halt! | |
| 10 | | | | $R^*[b_1] = (b_1, R^*[c_1])$ |
| 11 | | $b_4 = GNN(3, \{a_2, c_d\}, B)$ halt! | | |
| 12 | | | | $R^*[a_2] = (a_2, R^*[b_2])$ |
| 13 | $a_3 = GNN(2, \{c_s, c_d\}, A)$ | | | |
| 14 | | $b_2 = GNN(1, \{a_3, c_d\}, B)$ | | |
| 15 | | | SOP is applied | |
| 16 | | $b_1 = GNN(1, \{a_3, c_d\}, B)$ | | |
| 17 | | | SOP is applied | |
| 18 | | $b_4 = GNN(3, \{a_2, c_d\}, B)$ halt! | | |
| 19 | | | | $R^*[a_3] = (a_3, R^*[b_2])$ |
| 20 | $a_1 = GNN(3, \{c_s, c_d\}, A)$ | | | |
| 21 | | $b_2 = GNN(1, \{a_1, c_d\}, B)$ | | |
| 22 | | | SOP is applied | |
| 23 | | $b_1 = GNN(2, \{a_1, c_d\}, B)$ | | |
| 24 | | | SOP is applied | |
| 25 | | $b_4 = GNN(3, \{a_1, c_d\}, B)$ halt! | | |
| 26 | | | | $R^*[a_1] = (a_1, R^*[b_2])$ |
| 27 | $a_4 = GNN(4, \{c_s, c_d\}, A)$ | Algorithm Terminates! | | |

## 2.5 Experiments

In this section, we compare the performance of our proposed approaches PGNE and IBS to R-GTP and I-GTP [25]. Furthermore, in order to highlight the benefit of using the *Suffix Optimality Principle* (SOP) discussed in Lemma 2.4.7, we also compare IBS to a version of the same but *without* applying SOP, denoted by IBS-no-SOP. Note that, IBS-no-SOP is a modified (stronger) version of RIA, where based on paper [25] we applied a stronger pruning strategies for shrinking the search space based on the distance of POIs towards the $c_s$ and $c_d$. We also tried to compare our approaches to GOSR [47] but based on our preliminary experiments it required at least two times more query processing time in comparison to other competitors for all cases, and for this reason we do not report the performance of that approach in the experiments presented here. Table 2.3 shows the metadata of the real road networks [2] used in our experiments.

Table 2.3: Road networks used in the experiments.

| Network (Notation) | Number of vertices / edges |
|---|---|
| Oldenburg (OL) | 6,105 / 7,034 |
| California (CA) | 21,047 / 21,692 |
| North America (NA) | 175,813 / 179,179 |

It is not realistic to plan a group trip considering a whole state or continent. The networks used should thus be considered abstractions of possibly very fine grained cities, where the concept of a SGTPQ is realistic. For instance, OL could represent a city where only bus stops are vertices and NA the same city, but where every corner is mapped onto a network vertex.

The CA network contains the actual location 105,752 POIs of 42 COIs, thus is considered a real data set *de facto*. Note that most POIs lie in an edge of the network, hence why there are much more POIs than vertices in the network. Even though the OL and NA networks are real, we do not have the POI locations for them, thus we consider them as synthetic data sets. In each set of synthetic data set experiments, based on a pre-specified number of POIs, we randomly select network

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

vertices and set them as POIs (i.e., in synthetic data set experiments, we assume that all POIs have been placed on road network vertices) and keeping all COIs equally represented. We use the $R^*$-tree [12] to index POIs and keep in-memory graph data structure to store the road network. Furthermore, we applied the traditional Dijkstra algorithm for computing the shortest network distance computations whenever needed.

To evaluate our proposed approach for processing SGTPQs in a wide range of settings we vary the following parameters:

- the POI density, $D$, as a percentage of the total number of vertices in the road network,
- the query area $A$, i.e., the minimum bounding rectangle covering the source locations and destinations as a percentage of the road network area,
- the group size (number of travellers) $n$,
- the number of COIs $m$, and finally,
- the POI distribution within the COIs as uniform (U) or Zipfian (Z).

The Zipfian distribution itself is divided into two groups as Zipfian-increasing (Z-inc) and Zipfian-decreasing (Z-dec). In the Z-inc distribution, the number of POIs in the given sequence of COIs follows an increasing order, where it gradually decreases in the Z-dec distribution type. Table 2.4 summarizes the parameters, their ranges, and default values (in bold) used in our experiments. For each set of experiments, we vary the value of one parameter, and fix other parameters to their default values.

Table 2.4: Experimental parameters and their values (bold defines default values).

| parameter | Range |
|---|---|
| POI density ($D$) | 0.75%, 1.5%, **3%**, 4.5%, 6% |
| Query area ($A$) | 0.25%, **0.5%**, 1%, 2% |
| Group size ($n$) | 1, **5**, 10, 30, 60 |
| Number of COIs ($m$) | 2, **3**, 4, 5 |
| COI density | Z-dec, Z-inc, **U** |

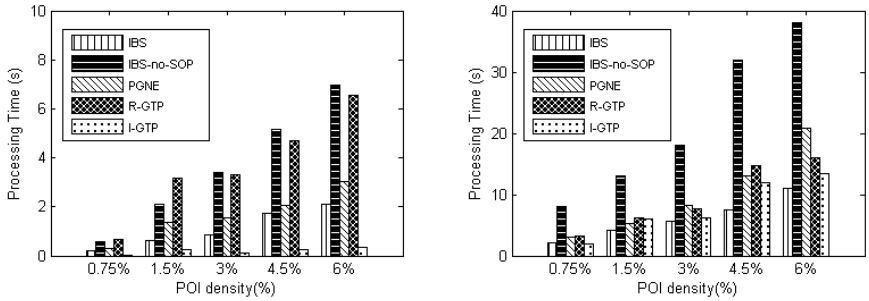
In the following, we compare the performance of different approaches regarding the number of retrieved POIs and overall query processing time. In each set of experiments, we run the experiment for 100 random SGTPQs and present the average result. We run all experiments using a computer with Intel Core i5 2.40 GHz CPU and 8GB RAM.

### 2.5.1 Effect of POI Density

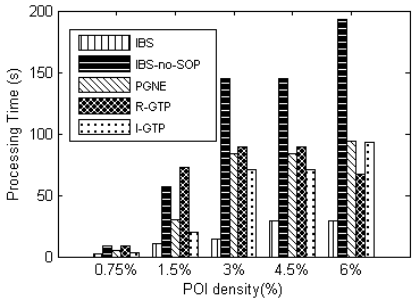
Figure 2.5 presents the performance comparison of different approaches in terms of processing time when varying the number of POIs, expressed as a percentage of number of vertices in the network. As expected all approaches incur higher processing time with the increase of data set size. The experimental results illustrate that in terms of response time, IBS requires on average 61%, 72% and 84% less processing time in comparison to IBS-no-SOP in OL, CA and NA road networks, respectively. That is, the larger the network the larger the savings yielded by using SOP, thus highlighting its importance.

I-GTP can be seen as the runner-up performer. In fact, IBS is outperformed by I-GTP on the smaller OL network. Nonetheless, for the medium CA and larger NA networks IBS was, on average, 33% and 50% faster than I-GTP. The main reason for that is, as discussed in Section 2.2, both I-GTP and R-GTP are based on a dynamic programming approach which requires non-trivial time overall for computing a potentially large number of shortest paths with the augmentation of the road network.

Our experimental results show that the PGNE approach incurs a smaller response time in comparison to IBS-no-SOP, even though both of them apply the same pruning strategy. The reason is that PGNE applies a mixed breadth-depth first search strategy, where the latter only focuses on traversing the search space based on a depth-first search strategy. On the other hand, IBS outperforms the PGNE technique by applying the SOP principle, in which it reuses the pre-computed optimal partial trips when the algorithm proceeds, resulting in reducing the query processing overhead.




(a) Oldenburg (b) California

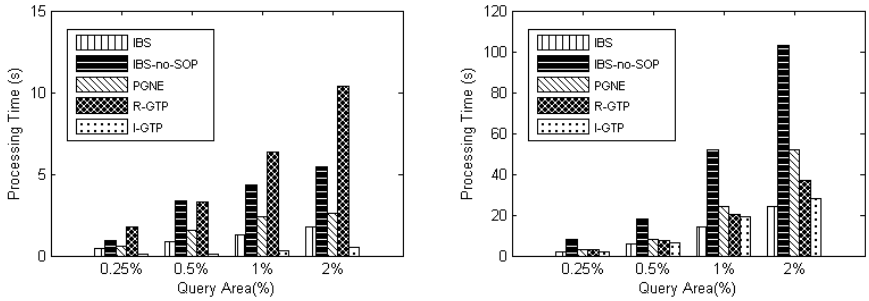



(c) North America

Figure 2.5: Effect of POI density($D$).

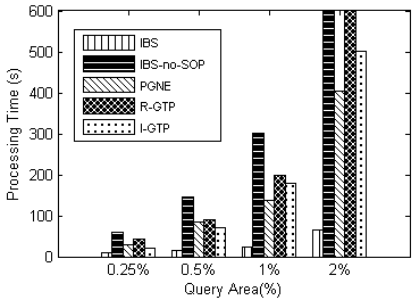
### 2.5.2 Effect of Query Area

Figure 2.6 shows the effect of increasing the query area on processing time. A simple inspection of scale of the Y-axis show that the query processing time of all approaches follows an increasing trend with the augmentation of query area. This is expected as for a larger query area more POIs need to be examined. Again, as Figure 2.6(a) illustrates, the I-GTP approach is superior to every other competitor the smaller OL road network, but for the CA network IBS is slightly faster and for the larger NA network (Figure 2.6(c)), IBS approach yields up to 5 times smaller in fact, response time in comparison to I-GTP method. Clearly, the gap between the performance of IBS and I-GTP approaches becomes larger with the increase of size of the road networks. In fact, while the query processing cost of all approaches increases exponentially with the increase of query area in the NA network, the response time of IBS increases with a much slower pace. The results of this set




(a) Oldenburg

(b) California




(c) North America

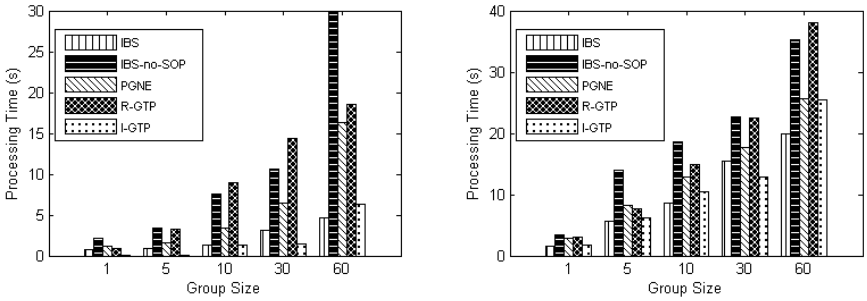
Figure 2.6: Effect of query area ($A$).

of experiments serves to demonstrates the superiority of IBS, and the advantage of using SOP within it, in terms of scalability with respect to query area in comparison to other techniques, in particular with the increase of the underlying network.

Our experimental results show that, even though both IBS and PGNE techniques apply the same pruning strategies for shrinking the search space, IBS takes advantage of applying SOP based on depth-first search strategy to reduce the query processing overhead by reusing the previously computed optimal partial group trips.

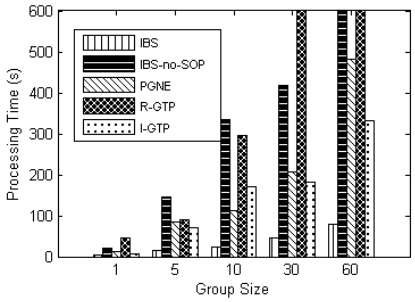
### 2.5.3 Effect of Group Size

Figure 2.7 shows that the processing time increases with the increase of group size due to incurring larger number of shortest path computations required. IBS is overall the best performer in particular for the large NA network where it is up to 3 times faster than the second best competitor; I-GTP is a better competitor for the




(a) Oldenburg

(b) California



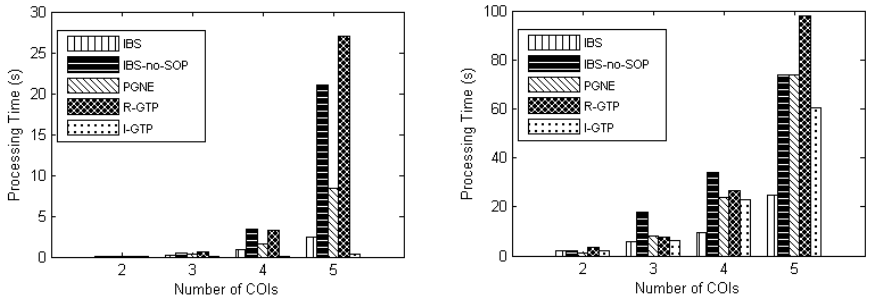

(c) North America

Figure 2.7: Effect of group size ($n$).

smaller and medium-sized networks, though. As in previous case the advantage of using SOP is clear, in particular for larger networks.

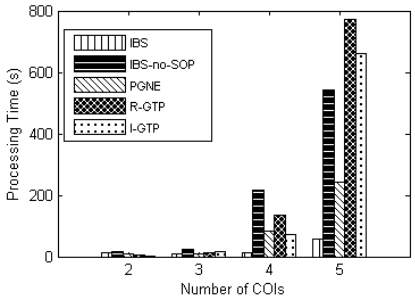
Finally, it is interesting to note that except for the case of the OL network, IBS is more efficient for processing the special case of a *single user* STPQ as well.

Figure 2.8 shows that, the query response time of all approaches follow an increasing trend with the increase of the number of COIs due to exponential augmentation of search space. I-GTP algorithm requires smaller response time for processing SGTPQs in OL road network in comparison to other competitors, but as Figures 2.8(b) and 2.8(c) illustrate, the IBS technique outperforms I-GTP approach for all settings of $m$ in larger road networks, CA and NA. The reason is that I-GTP incurs higher query processing overhead due to the higher number of shortest path computations needed in the underlying dynamic programming approach. Furthermore, the gap between the query processing cost of IBS and other competitors




(a) Oldenburg

(b) California




(c) North America

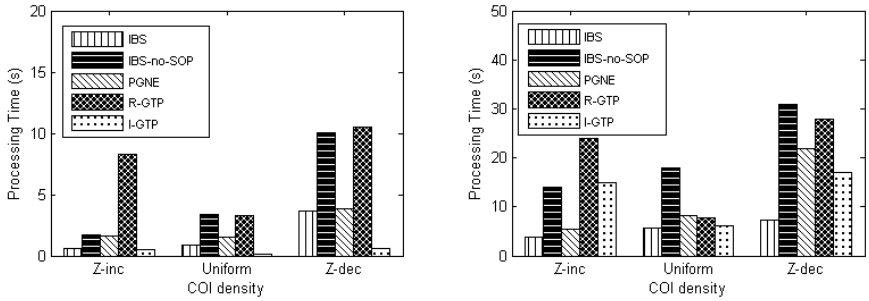
Figure 2.8: Effect of Number of COIs ($m$).

(particularly IBS-no-SOP) highlights yet again the advantage of using SOP.

### 2.5.4 Effect of COI Density

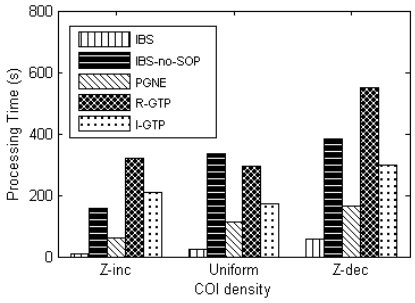
Our final set of experiments were aimed to compare the performance of different approaches when the densities of COIs are different. In this set of experiments, we varied the data set distribution between Uniform (U) and Zipfian (Z), the latter in two variants Z-inc and Z-dec. In the Uniform case the number of POIs for each COIs is nearly the same, whereas in the Z-inc, respectively Z-dec, cases, the COIs to be be visited first have a smaller, respectively larger, number of the POIs than the COIs that have be visited later in the COI sequence.

As Figures 2.9(a) and 2.9(c) show, the IBS, PGNE and IBS-no-SOP approaches with Z-inc data set distributions incur smaller query processing cost in comparison to U and Z-dec distributions. This is because the POIs belonging to COIs that




(a) Oldenburg

(b) California




(c) North America

Figure 2.9: COI density.

must be visited earlier have higher chance for appending to partial group trips with smaller length, hence they have higher chance for being examined, in comparison to POIs belonging to COIs that must be visited later. Consequently, with the increase of the cardinality of POIs in the initial positions in the given sequence of COIs, the search space growths with a faster pace in comparison to other data set distributions. As seen before, I-GTP is the best performer for the smaller OL network, but IBS, aided by SOP is the clear winner as the network gets larger.

## 2.6 Conclusion

In this chapter, we investigated Sequenced Group Trip Planning Queries (SGTPQs) in spatial databases. Firstly we revised the previous proposed approach IA in order to find the optimal answer. Additionally, we proposed PGNE and IBS approaches relying on mixed breadth-depth depth first and depth-first search strategies for ex-

ploring the search space, respectively. The basic idea of PGNE approach is that it iteratively generates and examines different potential partial group trips. As an iterative process it retrieves a partial group trip that incurs the smallest group travel distance and expands it by performing *progressive* and *replacement* operations. By applying those aforementioned operations, PGNE approach explores the search space in depth and breadth first search strategies, respectively. Furthermore we proposed IBS approach, a stateful algorithm, which applies the *suffix optimality principle* concept. As an iterative process, IBS computes the optimal partial trips starting from different POIs, in which the precomputed optimal partial trips are reused when the algorithm proceeds. Our experimental results show that while for smaller networks the previously proposed I-GTP approach may outperform our IBS approach, IBS is more scalable with respect to all parameters when the size of network increases, in which case it may outperform I-GTP by a rather large factor. This is important as crowdsourced maps become quickly finer grained, meaning that even if the mapped geographical area remains the same, the number of vertices and edges are bound to increase.

# Chapter 3

# Optimal Meeting Points Minimizing Aggregate Detour Distances from Preferred Paths

## 3.1 Introduction

With GPS-enabled devices, such as smartphones and the like being ubiquitous nowadays, location-based services and location-aware applications for social networks are common place. In this context, we investigate the *k-Optimal Meeting Points over Preferred Path* (k-$OMP^3$). A k-$OMP^3$ query takes as input a group of travellers, with their preferred path as well as a set of points of interest (POIs), and returns the $k$ POIs that yield the $k$ smallest aggregate detour distances between the preferred paths for all group members and the chosen POIs.

As an example let us consider a group of travellers, who are at different places (e.g., their workplaces). Before going home in the evening, using bus lines, the group would like to meet at a restaurant, but would like it to be the restaurant which would yield the smallest total group detour distance[1]. For visiting the meeting place, each group member will take a detour from the nearest bus stop in their preferred paths towards the meeting place. After meeting each group member will return back to their daily paths via the same bus stop and continue their journeys towards their destinations. The motivation behind this query is that the users would prefer

[1] We define the detour distance between a POI $o$ and a path $p$ as the shortest distance from a branching point in $p$ to $o$, and the total group detour distance is defined as the sum of all group members detour distances towards the meeting place.

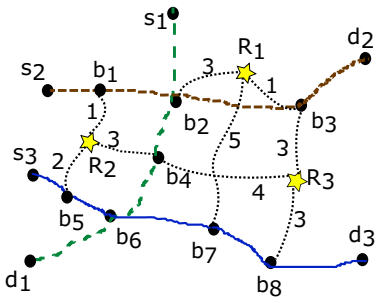



Figure 3.1: A sample road network.

to follow the path that they are familiar with.

In order to illustrate k-$OMP^3$, Figure 3.1 shows a sample road network, where three travellers ($t_1$, $t_2$ and $t_3$) are traveling from their initial locations ($s_1$, $s_2$ and $s_3$) towards their destinations ($d_1$, $d_2$ and $d_3$) through the preferred paths $p_1$=$(s_1, b_2, b_4, b_6, d_1)$, $p_2$= $(s_2, b_1, b_2, b_3, d_2)$ and $p_3$= $(s_3, b_5, b_6, b_7, b_8, d_3)$, where, each $b_i$ is a branching point[2] in the travelers' routes (e.g., a bus/subway stop). The stars in the figure illustrate three restaurants ($R_1$, $R_2$ and $R_3$). In this context, consider a k-OMP$^3$ query to find the two restaurants which incur the two smallest aggregate detour distances, based on preferred paths $p_1$, $p_2$ and $p_3$. Table 3.1 lists the aggregate detour distances for all POIs for aggregate functions *sum* and *max*, as well as the corresponding branch points with respect to each of the travellers' routes. For example, assuming $R_1$ is the meeting point, then $t_1$, $t_2$ and $t_3$ would take a detour from their routes towards the meeting point via branch points $b_2$, $b_3$ and $b_7$, respectively; yielding 9 and 5 aggregate detour distance units, for aggregate functions $sum$ and $max$, respectively. In this particular example, the answer sets $\{R_2, R_1\}$ and $\{R_2, R_3\}$ would be returned as the answers for aggregate functions *sum* and *max*, respectively.

Table 3.1: Aggregate detour distance for different POIs with respect to Figure 3.1.

| Traveller | Branch points | | | aggregate detour distance | |
|---|---|---|---|---|---|
| | $p_1$ | $p_2$ | $p_3$ | sum | max |
| $t_1$ | $b_2$ | $b_3$ | $b_7$ | 9 (3+1+5) | $max\{3,1,5\}$=5 |
| $t_2$ | $b_4$ | $b_1$ | $b_5$ | 6 (3+1+2) | $max\{3,1,2\}$=3 |
| $t_3$ | $b_4$ | $b_3$ | $b_8$ | 10 (4+3+3) | $max\{4,3,3\}$=4 |

As we shall discuss shortly (Section 3.2) previous approaches to solve related

[2]A branching point is a point in path where one can exit from/return to.

problems have at least one of the following shortcomings: they are either applicable only to a single user, they rely on the notion of individual shortest paths discovered during query processing time instead of using the notion of a (pre-defined) preferred path, or they assume the users meet and *stay together* at the selected POI or that they *move together* towards a *common* destination. As our main contribution we propose two pruning-based approaches to answer k-$OMP^3$ as described above for different aggregate functions, both are based on geometric properties of ellipses, and overcome the shortcomings mentioned above. The former firstly prunes the search space around each preferred path based on the total distance of POIs towards the corresponding source location and destination, and then returns the intersection area of all pruned areas as the final refined area. On the other hand, the latter prunes the POIs search space based on the total distance of POIs towards the centroids of source locations and destinations.

## 3.2 Related Work

To the best of our knowledge, only a restricted case of the k-$OMP^3$ query, namely when a *single* user is considered, has been studied: In-Route Nearest Neighbour (IRNN) [55], Path Nearest Neighbour (PNN) [14], and Best Point Detour (BPD) [50]. In IRNN the problem setting is essentially the same as ours, but it cannot be easily extended, if at all, to multiple users. PNN does not have the notion of a given preferred path. Instead the shortest path between origin and destination is used, and the solution does depend on that assumption, i.e., the meeting point is determined as the shortest path is computed. Finally, BPD allows a user to leave his/her preferred path at one branching point and return to it at a different branching point, which is more generic than our assumption. However the approach requires the user to determine a detour distance threshold in order to be practical; we, on the other hand, do not impose such a constraint.

A seemingly related, but in fact rather different task, is that of finding optimal meeting points, which has been addressed in [66] and [63]. In the former work the query returns a single point where all users meet *and stay*. In the latter research the

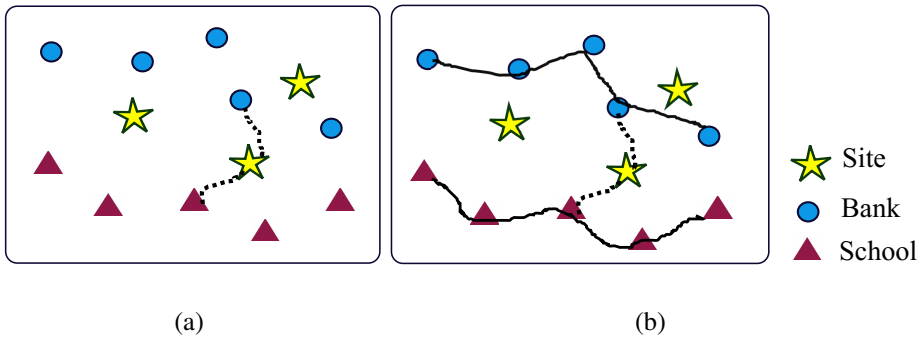



Figure 3.2: Mapping of a k-$OMP^3$ query into a MALs problem.

query returns a single point where all users meet *and continue together to a common* destination. In both cases the paths from the users origin to the optimal meeting point are determined as part of the answer at query time, whereas we assume that different users have different destinations and their preferred paths are given as part of the query's input.

In [40] Lin et al proposed *One-Tree Method* (OTM) for finding the most accessible locations (MALs) among a set of possible sites towards a set of amenities such as schools, supermarkets, and hospitals, etc. In the MALs problem, the goal is to find $k$ sites among a set of candidate sites incurring the $k$ smallest accessibility costs. The accessibility cost of a site is defined based on the total distance towards the nearest facility instance from different pre-specified category types. Figure 3.2(a) shows a sample MALs instance, where we are looking for a site which incurs the minimum accessibly cost towards two sets of amenities: bank and schools.

The MALs problem is particularly relevant in the context of our work as the k-$OMP^3$ query can be mapped into the MALs problem as follows: (1) each single preferred path in the k-$OMP^3$ query is mapped to an amenity type in a MALs problem, where each branching point is considered as a facility instance, (2) the set of POIs that the group is interested to visit in the k-$OMP^3$ query is mapped into the set of candidate sites in a MALs problem. Once such mapping is done, the OTM technqiue can also be used for solving k-$OMP^3$ queries. Figure 3.2(b) illustrates the mapping of a k-$OMP^3$ query into a MALs problem. Given its relevance and

the fact we compare our proposal to the use of the OTM to solve a k-$OMP^3$ query-mapped-into-MALs problems we discuss OTM in more detail next.

When solving the MALs problem the POIs of all amenities and sites are indexed in two different $R^*$-trees, $R_a$ and $R_s$, respectively. The basic idea of OTM is applying spatial join operation based on heap-algorithm [17] performed on $R_a$ and $R_s$. Each node $u$ in $R_s$ owns exactly one Local Priority Queue (LPQ), a min-heap that maps node $u$ to different nodes in $R_a$, where its elements are ordered by the lower bound of total accessibility cost of node $u$ towards different amenities. The OTM also uses a Global Priority Queue (GPQ), a min-heap that maintains the generated LPQs of all explored nodes in $R_s$. The elements of GPQ are ordered according to lower bound accessibility cost of all explored nodes in $R_s$. For more details about OTM, we refer the interested reader to paper [40]. We will use OTM as a baseline in our experiments.

## 3.3 Proposed Approaches

As our main contribution we present two approaches, namely *Multiple Ellipse-based Pruning* (MEP) and *Single Ellipse-based Pruning* (SEP), for processing k-$OMP^3$ queries. But before discussing these approaches in details let us first formally define the k-$OMP^3$ query.

We assume a road network modeled by an undirected graph $G(V, E, W)$, where $V$ is the set of road intersections and end-points, and $E$ is the set of edges containing all road segments. The set $W$ indicates the weight of edges in $G$, in our case the length of road network segments. The set of preferred paths of a group of $n$ users on a road network is denoted by $P = \{P_1, P_2, ..., P_n\}$, where each preferred path is a sequence of branch points from source location w.r.t. the destination location, i.e., $P_i$=$(v_i^1, v_i^2, ..., v_i^{|P_i|-1}, v_i^{|P_i|})$. The length of preferred path $P_i$, denoted by $l_i^*$, is defined as the accumulated weight of the sequence of edges traversed through the sequence of branching points belonging to $P_i$, i.e., $l_i^* = \sum_{j=1}^{|P_i|-1} d_n(v_i^j, v_i^{j+1})$. Table 3.2 defines the notation used in this chapter.

Before we formally state the k-$OMP^3$ queries we need to define the notions of

Table 3.2: Notation.

| *Notation* | *Meaning* |
|---|---|
| $P = \{P_1, ..., P_n\}$ | The set of $n$ preferred paths |
| $P_i = (v_i^1, ..., v_i^{\lvert P_i \rvert})$ | The preferred path of $i^{th}$ user |
| $v_i^j$ | The $j^{th}$ branch point in path $P_i$ |
| $s_i = v_i^1$, $d_i = v_i^{\lvert P_i \rvert}$ | the starting and ending points of path $P_i$ |
| $l_i^*$ | The length of path $P_i$ |
| $O$=$\{o_1, ..., o_{\lvert O \rvert}\}$ | The set of POIs |

detour distance and aggregate detour distances.

**Definition 3.3.1.** The detour distance for POI $o_j \in O$ with respect to path $P_i$, denoted as $dd(o_j, P_i)$, is the minimum network distance from $o_j$ to the nearest branch point in $P_i$, i.e, $dd(o_j, P_i) = \min_{v_i^k \in P_i} \{d_n(o_j, v_i^k)\}$.

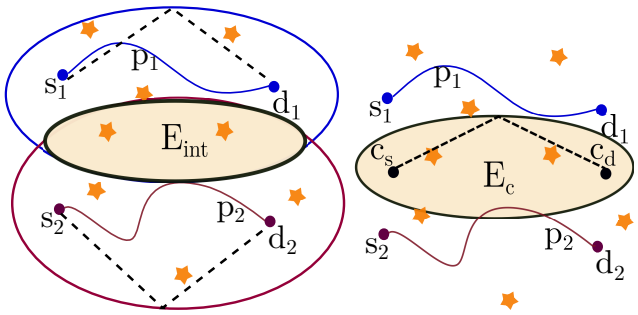
**Definition 3.3.2.** The aggregate detour distance of POI $o_j$ w.r.t. all paths in $P$, denoted as $Add(o_j, P)$, is defined as: $Add(o_j, P) = f_{i=1}^n dd(o_j, P_i)\ \forall P_i \in P$. Where $f$ is an aggregate function. We consider two cases for $f$ in this section: *sum* or *max*. For instance if $f$ is *sum*, then $f_{i=1}^n dd(o_j, P_i) = \sum_{i=1}^n dd(o_j, P_i)$.

We can now formally define the k-$OMP^3$ query:

**Definition 3.3.3.** Given a set of preferred paths $P$ and a set of POIs $O$, the k-$OMP^3$ query returns a set of $k$ ***unique*** POIs $A \subseteq O$ where $\forall o_i \in A$, $\nexists o' \in O \setminus A$ s.t. $Add(o') < Add(o_i)$

Now with the problem defined we can proceed to present MEP and SEP in detail. In both of these approaches we retrieve, using an heuristic, an initial answer set containing $k$ candidate POIs from the database and compute an initial upper bound for the $k^{th}$ smallest aggregate detour distance. The obtained upper bound is used for pruning POIs that cannot be part of answer set.

The main difference between these approaches is that MEP uses the intersection of several ellipses, whose focal points depend on *each* of the preferred paths (Figure 3.3a), whereas SEP uses a single ellipse with focal points on the centroid of origins and destinations of *all* preferred paths (Figure 3.3b). Both of these approaches use the geometric properties of ellipses that the total distance of each point




(a) MEP strategy (b) SEP strategy

Figure 3.3: The proposed pruning strategies.

located outside an ellipse, is greater than the major axis of that ellipse [25]. We also use this property to prune the non-promising POIs that cannot belong to the optimal answer set. For that we first retrieve an initial answer set containing $k$ POIs. Subsequently based on the retrieved initial answer set, we compute an upper bound for $Add^k$. This upper bound will be used as the major axis for ellipses that are applied for pruning the search space. The MEP approach firstly prunes the search space around each preferred path based on the total distance of the POIs w.r.t. the corresponding source location and destination, and then returns the intersection area of all pruned areas as the final refined area. On the other hand, SEP prunes the POIs search space based on the total distance of POIs w.r.t. the centroids of the source locations and destinations.

In the next sections we detail the main differences between both approaches and pose the criteria necessary to guarantee their correctness.

### 3.3.1 Multiple Ellipse-based Pruning Approach (MEP)

As mentioned above, first an initial answer set $A = \{o_1, o_2, ..., o_k\}$ is retrieved. In this work, we use group nearest neighbor queries [42] w.r.t. the centroids of source locations $c_s$ and destinations $c_d$ for extracting the $k$ nearest POIs from $C$ w.r.t. $c_s$ and $c_d$. After computing the aggregate detour distance for all POIs belonging to that initial answer set $A$, an initial upper bound for the $k^{th}$ smallest aggregate detour distance $Add^k$ is computed. In the next step, we compute one ellipse $E_i$ for

each path $P_i$. The focal points of $E_i$ are $\langle s_i, d_i \rangle$, where the major axis is defined as $2 \times Add^k + l_i^*$, in which $l_i^*$ is the length of path $P_i$ (c.f. Lemmas 3.3.1 and 3.3.2). After computing the set $E = \{E_1, E_2, ..., E_n\}$, the intersection region of all computed ellipses, $E_{int}$=$\cap_{i=1}^{n} E_i$, is returned as the final refined search space, based on Theorems 3.3.3 and 3.3.4. Finally, the $k$ optimal meeting points are obtained by comparing the corresponding aggregate detour distance of all POIs residing in $E_{int}$. Algorithm 7 illustrates the pseudo-code of MEP approach.

**Algorithm 7:** *MEP Approach for* $sum/max$

**Input:** $P = \{P_1, ..., P_n\}$, $C = \{o_1, ..., o_{|O|}\}$, and $k$.
**Output:** k-$OMP^3$.

$H_R \leftarrow \emptyset$, $i \leftarrow 1$, $Add^k \leftarrow \infty$
**while** $i \leq k$ **do**
    $o_i \leftarrow GNN(i, \{c_s \cup c_d\}, O)$
    $H_R.Enqueue(o_i, Add(o_i, P))$
update $Add^k$ based on $H_R$
$i \leftarrow 1$
**while** $i \leq n$ **do**
    $E_i \longleftarrow Ellipse(< s_i, d_i >, 2 \times Add^k + l_i^*)$
$E_{int} \longleftarrow \cap_{i=1}^{n} E_i$
**for** $\forall o_i \in E_{int}$ **do**
    $H_R.Enqueue(o_i, Add(o_i, P))$
return the top $k$ elements in $H_R$

Next we justify the use and formally state the Lemmas 3.3.1 and 3.3.2 and Theorems 3.3.3 and 3.3.4 that support MEP. Lemma 3.3.1 defines a lower bound for the detour distance of a POI w.r.t. a path. The obtained lower bound is used for pruning the POI search space around each preferred path $P_i$ according to Lemma 3.3.2. That is we compute an ellipse $E_i$ for each path $P_i$ in order to estimate the area containing the POIs belonging to the optimal answer set. In the next step, we further prune the search space by computing the intersection region of all pre-computed ellipses, $E_{int}$=$\cap_{i=1}^{n} E_i$, based on Theorem 3.3.3. This theorem indicates that the intersection region $E_{int}$ contains all the POIs belonging to the optimal answer set. Furthermore, due to Theorem 3.3.4 we know that the pruned search space $E_{int}$ for solving a k-$OMP^3$query contains at least $k$ POIs, assuming that $|O| \geq k$.

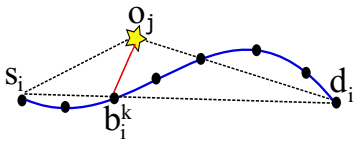



Figure 3.4: Defining lower bound for detour distance.

**Lemma 3.3.1.** Let $P_i = (v_i^1, v_i^2, ..., v_i^{|P_i|})$ be a preferred path from source location $s_i$ to destination $d_i$, with length $l_i^*$. The lower bound for the smallest possible network distance of $o_j$ from path $P_i$ (i.e., the lower bound for the detour distance of $o_j$ from path $P_i$), denoted by $dd_n^{lb}(o_j, P_i)$, is given by:

$$dd_n^{lb}(o_j, P_i) = \frac{d_e(o_j, s_i) + d_e(o_j, d_i) - l_i^*}{2} \tag{3.1}$$

*Proof.* Consider Figure 3.4 illustrating a preferred path $P_i$ from source location $s_i$ to destination $d_i$, where $v_i^k$ has the smallest network distance w.r.t. POI $o_j$ in comparison to all other branch points in $P_i$ (i.e., $v_i^k$ is the best detour branch point w.r.t. POI $o_j$). Furthermore, let us suppose that the smallest network distance of $v_i^k$ from $s_i$ and $d_i$ through the preferred path $P_i$ be $x$ and $l_i^* - x$, respectively. Then, based on the triangle inequality, we have:

$$d_e(o_j, s_i) \leq d_e(v_i^k, s_i) + d_e(v_i^k, o_j) \leq x + d_e(v_i^k, o_j) \tag{3.2}$$

$$d_e(o_j, d_i) \leq d_e(v_i^k, d_i) + d_e(v_i^k, o_j) \leq l_i^* - x + d_e(v_i^k, o_j) \tag{3.3}$$

By summing up the Equations 3.2 and 3.3, we have

$$d_e(o_j, s_i) + d_e(o_j, d_i) \leq l_i^* + 2 \times d_e(v_i^k, o_j) \tag{3.4}$$

Considering that $d_e(v_i^k, o_j) \leq d_n(v_i^k, o_j)$, then we can conclude that:

$$\frac{d_e(o_j, s_i) + d_e(o_j, d_i) - l_i^*}{2} \leq d_n(v_i^k, o_j) \tag{3.5}$$

□

**Lemma 3.3.2.** Given $Add^k$ as the $k^{th}$ smallest aggregate detour distance computed so far, let $E_i$ be an ellipse defined for path $P_i \in P$ with focal points $\{s_i, d_i\}$, and major axis as $2 \times Add^k + l_i^*$. Then, the POIs residing outside of $E_i$ cannot be part of optimal answer set.

*Proof.* Let $o_j$ is a POI residing outside of $E_i$, then we have $2 \times Add^k + l_i^* < d_e(o_j, s_i) + d_e(o_j, d_i)$ . On the other hand, based on the estimated lower bound of detour distance of $o_j$ w.r.t. $P_i$, defined in Lemma 3.3.1, we can conclude:

$$Add^k < \frac{d_e(o_j, s_i) + d_e(o_j, d_i) - l_i^*}{2} \leq dd_n^{lb}(o_j, P_i) \leq Add(o_j, P) \tag{3.6}$$

This means that $o_j$ cannot be part of optimal answer set, since it incurs aggregate detour distance greater than $Add^k$. □

**Theorem 3.3.3.** The intersection region $E_{int}$=$\cap_{i=1}^{n} E_i$ contains all the POIs belonging to the optimal answer set.

*Proof.* Let us assume $o_j$ as a POI residing outside of the intersection region $E_{int}$. Then it would be outside of at least one of the ellipses $\{E_1, E_2, ..., E_n\}$. Let $E_i$ be such an ellipse. Recalling that the major axis of $E_i$ equals to $2 \times Add^k$+$l_i^*$, then we can conclude $Add^k < (d_e(o_j, s_i) + d_e(o_j, d_i) - l_i^*)/2 \leq dd_n^{lb}(o_j, P_i) \leq Add(o_j, P)$. This means that the POI $o_j$ placing outside of intersection region $E_{int}$ would have aggregate detour distance greater than $Add^k$, thus it cannot be part of optimal answer set. □

**Theorem 3.3.4.** The intersection region $E_{int}$=$\cap_{i=1}^{n} E_i$ contains at least $k$ POIs.

*Proof.* Let us a assume $A = \{o_1, o_2, ..., o_k\}$ as the initial answer set[3], where the intersection region $E_{int}$ and the initial $k^{th}$ smallest aggregate detour distance $Add^k$ were defined based on $A$. Then $\forall o_j \in A$, $1 \leq j \leq k$ and $\forall P_i \in P$, $1 \leq i \leq n$, we have:

$$\frac{d_e(o_j, s_i) + d_e(o_j, d_i) - l_i^*}{2} \leq dd_n^{lb}(o_j, P_i) \leq Add(o_j, P) \leq Add^k \tag{3.7}$$

Then we can conclude that $d_e(o_j, s_i) + d_e(o_j, d_i) < 2 \times Add^k + l_i^*$. This means that all POIs in initial answer set $A$ are placed within all ellipses $\{E_1, E_2, ..., E_n\}$, consequently, residing in the intersection region $E_{int}$. Thus the intersection region $E_{int}$ contains at least $k$ POIs belonging to the initial answer set $A$. □

[3]Recall that the set $A$ is obtained via a heuristic approach such as applying group nearest neighbors queries w.r.t. $c_s$ and $c_d$.

**Algorithm 8:** *SEP Approach (where the aggregate $f$ is $sum$)*

**Input:** $P = \{P_1, ..., P_n\}$, $C = \{o_1, ..., o_{|O|}\}$, and $k$.
**Output:** k-$OMP^3$.

$H_R \leftarrow \emptyset$, $i \leftarrow 1$, $Add^k \leftarrow \infty$
**while** $i \leq k$ **do**
    $o_i \leftarrow GNN(i, \{c_s \cup c_d\}, O)$
    $H_R.Enqueue(o_i, Add(o_i, P))$
update $Add^k$ based on $H_R$
$E_c \longleftarrow Ellipse(< c_s, c_d >, \frac{2 \times Add^k + \sum_{i=1}^{n} l_i^*}{n})$
**for** $\forall o_j \in E_c$ **do**
    $H_R.Enqueue(o_j, Add(o_j, P))$
    update $Add^k$ based on $H_R$
return the top $k$ elements in $H_R$

### 3.3.2 Single Ellipse-based Pruning Approach (SEP)

The Single Ellipse-based Pruning Approach (SEP) approach shrinks the search space based on the total distance of POIs w.r.t. the centroids of source locations and destinations, $c_s$ and $c_d$, respectively. After obtaining the initial answer set using group nearest neighbor queries, as explained in Section 3.3.1, SEP computes the ellipse $E_c$, where the focal points are located at $c_s$ and $c_d$, and the major axis of this ellipse is defined based on Theorems 3.3.6 and 3.3.9 for *sum* and *max* aggregate functions, respectively (both theorems are presented next). Similar to the MEP approach, after shrinking the POI search space into the refined area $E_c$, then the top $k$ meeting points are obtained by comparing the corresponding aggregate detour distance of all POIs residing in $E_c$. In Theorems 3.3.7 and 3.3.10, we prove that $E_c$ contains at least $k$ POIs, assuming that the aggregate function is $sum$ and $max$, respectively. Furthermore, Algorithm 8 illustrates the pseudocode for the SEP approach, when the aggregate function is $sum$, and it can be easily changed, based on Theorem 3.3.9, to support the $max$ aggregate function.

**Lemma 3.3.5.** Let $P = \{p_1, p_2, ..., p_n\}$ be the set of preferred paths of $n$ users with source and destination $s_{set} = \{s_1, s_2, ..., s_n\}$ and $d_{set} = \{d_1, d_2, ..., d_n\}$, respectively, where $c_s$ and $c_d$ are the centroid of source locations and destinations. Assuming that the aggregate function is *sum*, then the lower bound for aggregate

detour distance of POI $o_j$ w.r.t $P$, denoted by $Add_{lb}^{sum}(o_j, P)$, is given by:

$$Add_{lb}^{sum}(o_j, P) = \frac{n \times d_e(o_j, c_s) + n \times d_e(o_j, c_d) - \sum_{i=1}^{n} l_i^*}{2} \tag{3.8}$$

*Proof.* Consider that the aggregate function is $sum$. Then by summing up the lower bound detour distance of POI $o_j$ w.r.t. the $n$ preferred paths in $P$, based on Lemma 3.3.1, we have:

$$\frac{\sum_{i=1}^{n}\{d_e(o_j, s_i) + d_e(o_j, d_i)\} - \sum_{i=1}^{n} l_i^*}{2} \leq Add(o_j, P) \tag{3.9}$$

According to work [25], we have $n \times d_e(o_j, c_s) \leq \sum_{i=1}^{n} d_e(o_j, s_i)$, and similarly, $n \times d_e(o_j, c_d) \leq \sum_{i=1}^{n} d_e(o_j, d_i)$. Consequently, we have:

$$\begin{aligned} &\frac{n \times d_e(o_j, c_s) + n \times d_e(o_j, c_d) - \sum_{i=1}^{n} l_i^*}{2} \\ &\qquad \leq \frac{\sum_{i=1}^{n}\{d_e(o_j, s_i) + d_e(o_j, d_i)\} - \sum_{i=1}^{n} l_i^*}{2} \\ &\qquad\qquad \leq Add(o_j, P) \end{aligned} \tag{3.10}$$

□

**Theorem 3.3.6.** Let $P = \{p_1, p_2, ..., p_n\}$ be the set of preferred paths of $n$ users with source and destination $s_{set} = \{s_1, s_2, ..., s_n\}$ and $d_{set} = \{d_1, d_2, ..., d_n\}$, respectively. Let $Add^k$ be the $k^{th}$ smallest aggregate detour distance computed so far, where the aggregate function is *sum*. Further let $E_c$ be an ellipse with focal points $\{c_s, c_d\}$, and with major axis equal to $(2 \times Add^k + \sum_{i=1}^{n} l_i^*)/n$. Then any POI located outside such ellipse $E_c$ cannot be part of optimal answer set.

*Proof.* Based on geometric properties of ellipses, the distance between two focal points via a point located outside the ellipse is greater than the length of the major axis of the ellipse. Thus for a POI $o_j$ located outside of $E_c$, we have

$$\frac{2 \times Add^k + \sum_{i=1}^{n} l_i^*}{n} < d_e(o_j, c_s) + d_e(o_j, c_d) \tag{3.11}$$

where, from Lemma 3.3.5 we can conclude:

$$Add^k \leq \frac{n \times d_e(o_j, c_s) + n \times d_e(o_j, c_d) - \sum_{i=1}^{n} l_i^*}{2} = Add_{lb}^{sum}(o_j, P) \tag{3.12}$$

This means that the POI $o_j$ located outside of $E_c$ cannot be part of optimal answer set, as it would yield an aggregate detour distance greater than $Add^k$. □

**Theorem 3.3.7.** Consider that the aggregate function is $sum$. The region $E_c$ contains at least $k$ POIs.

*Proof.* (By contradiction): Let $A = \{o_1, o_2, ..., o_k\}$ be the initial answer set, obtained via applying group nearest neighbors towards $c_s$ and $c_d$. Let us assume $o_i$, $1 \leq i \leq k$, be the $i^{th}$ group nearest neighbors towards $\{c_s \cup c_d\}$, placing outside of ellipse $E_c$. Then we have $(2 \times Add^k + \sum_{i=1}^{n} l_i^*)/n < d_e(o_i, c_s) + d_e(o_i, c_d)$. On the other hand, according to Lemma 3.3.5, we have:

$$Add^k \leq \frac{n \times d_e(o_i, c_s) + n \times d_e(o_i, c_d) - \sum_{i=1}^{n} l_i^*}{2} = Add_{lb}^{sum}(o_i, P) \tag{3.13}$$

This contradicts with our basic assumption that $Add^k$ is the $k^{th}$ smallest aggregate detour distance computed so far. Thus all $k$ elements of $A$ are within $E_c$, i.e. it contains at least $k$ POIs. □

**Lemma 3.3.8.** Consider the same notation used in Lemma 3.3.5. If the aggregate function is *max*, then the lower bound for aggregate detour distance of POI $o_j$ w.r.t $P$, denoted by $Add_{lb}^{max}(o_j, P)$, is given by:

$$Add_{lb}^{max}(o_j, P) = \frac{d_e(o_j, c_s) + d_e(o_j, c_d) - \max\limits_{1 \leq i \leq n}\{l_i^*\}}{2} \tag{3.14}$$

*Proof.* Considering the aggregate function as *max*, then by aggregating the lower bound detour distance of POI $o_j$ w.r.t. $n$ preferred paths in $P$, based on Lemma 3.3.1, we have:

$$\max_{1 \leq i \leq n}\{\frac{d_e(o_j, s_i) + d_e(o_j, d_i) - l_i^*}{2}\} \leq Add(o_j, P) \tag{3.15}$$

Considering that according to work [25], we have $d_e(o_j, c_s) \leq \max\limits_{1 \leq i \leq n}\{d_e(o_j, s_i)\}$ and $d_e(o_j, c_d) \leq \max\limits_{1 \leq i \leq n}\{d_e(o_j, d_i)\}$, thus, we can conclude:

$$\begin{aligned} &\frac{d_e(o_j, c_s) + d_e(o_j, c_d) - \max\limits_{1 \leq i \leq n}\{l_i^*\}}{2} \leq \\ &\qquad \max_{1 \leq i \leq n}\{\frac{d_e(o_j, s_i) + d_e(o_j, d_i) - l_i^*}{2}\} \\ &\qquad\qquad \leq Add(o_j, P) \end{aligned} \tag{3.16}$$

□

**Theorem 3.3.9.** Consider the same notation used in Theorem 3.3.6. If the aggregate function is *max*, then POIs located outside of an ellipse $E_c$ with focal points $\{c_s, c_d\}$, and with major axis as $2 \times Add^k + \max_{1 \leq i \leq n}\{l_i^*\}$ cannot be part of optimal answer set.

*Proof.* Due to geometric properties of ellipses, for a POI $o_j$ located outside of $E_c$, we have

$$2 \times Add^k + \max_{1 \leq i \leq n}\{l_i^*\} < d_e(o_j, c_s) + d_e(o_j, c_d) \tag{3.17}$$

Then based on Lemma 3.3.8, we can conclude that:

$$Add^k \leq \frac{d_e(o_j, c_s) + d_e(o_j, c_d) - \max_{1 \leq i \leq n}\{l_i^*\}}{2} = Add_{lb}^{max}(o_j, P) \tag{3.18}$$

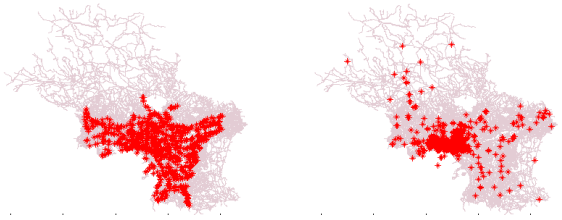
Therefore the POI $o_j$ located outside of $E_c$ cannot be part of the answer set, since the lower bound for aggregate detour distance of $o_j$ w.r.t. $P$ would be greater than $Add^k$. □

**Theorem 3.3.10.** Consider that the aggregate function is $max$. The intersection region $E_c$ contains at least $k$ POIs.

*Proof.* (By contradiction): Let us assume there are less than $k$ POIs inside the $E_c$. In other words, the POI $o_k$ incurring the $k^{th}$ smallest aggregate detour distance is placing outside of ellipse $E_c$. This leads to $2 \times Add^k + \max_{1 \leq i \leq n}\{l_i^*\} < d_e(o_k, c_s) + d_e(o_k, c_d)$. Then based on Lemma 3.3.8, we can conclude $Add^k < Add_{lb}^{max}(o_k, P)$. This contradicts with our basic assumption that $Add^k$ is the $k^{th}$ smallest aggregate detour distance computed so far. □

## 3.4 Experimental Results

We evaluate the performance of our proposed approaches, MEP and SEP, as well as our baseline, OTM, for processing k-$OMP^3$ queries using real datasets [9] with the public bus transportation networks and eateries (as POIs) in Amsterdam, Oslo and Berlin as of March/2007. Figures 5a and 5b illustrates some of the data from Oslo. Note that for all experiments we make the simplifying assumption that the *preferred paths* of the travelers are given by randomly selected bus routes. Such setting maintains our experiments realistic according to the motivation discussed earlier, e.g.,



(a) Bus-stops (b) Restaurants and coffee shops

Figure 3.5: Bus stops and POIs in Oslo.

colleagues traveling from their work to their homes and stopping somewhere to meet somewhere in between. Table 3.3 summarizes details of the data set used in our experiments.

Table 3.3: Summary of the dataset used in our experiments.

| | | Amsterdam | Oslo | Berlin |
|---|---|---|---|---|
| Road Network (size) | no. of vertices | 106,599 | 305,174 | 428,768 |
| | no. of edges | 130,090 | 330,632 | 504,228 |
| POIs | Eateries (restaurants and coffee shops) | 824 | 958 | 3,083 |
| Public Bus Network (size) | no. of routes | 82 | 107 | 236 |
| | no. of stops | 887 | 988 | 4,346 |

Table 3.4: Experimental parameters and their values (**bold** defines default values).

| Parameter | Range |
|---|---|
| Density of POIs ($D_p$) | 10%, 25%, 50%, **100%** |
| Group size ($n$) | 1,**2**,3,4 |
| Answer size ($k$) | **3**, 5, 10, 20 |

In what follows, we varied the following parameters: (1) the density of POIs ($D_p$), (2) the group size $n$ and (3) the number of requested POIs $k$. Table 3.4 summarizes the parameter names, their ranges, and default values. For each set of experiments, we vary the value of one parameter, and fix the other parameters to their default values. It is important to note that even though the qualitative behavior of all approaches is relatively similar across all the three datasets, quantitatively

there is a very clear difference due to the size of the datasets, i.e., all approaches becomes proportionally more expensive with the size of the cities.

For computing the aggregate detour distance for each retrieved POI, we use the well-known Dijkstra algorithm. For each set of experiments, we executed 50 k-$OMP^3$ queries and report average figures. All experiments were done using a computer with Intel Core i5 2.40 GHz CPU and 8GB RAM.

For the sake of clarity we need to explain how we computed the set of POIs within the intersection of ellipses of MEP or the single ellipse in SEP. For MEP we compute the corresponding MBR for each ellipse based on [38]. Subsequently, the intersection of all pre-computed MBRs (i.e., rectangles) as $E_{int}$ is computed. Finally, we perform a range-query to retrieve all POIs residing in the $E_{int}$. Similarly, for SEP approach we compute the corresponding MBR of $E_c$, also using [38]. Subsequently, we perform a range-query to retrieve all POIs within $E_c$.

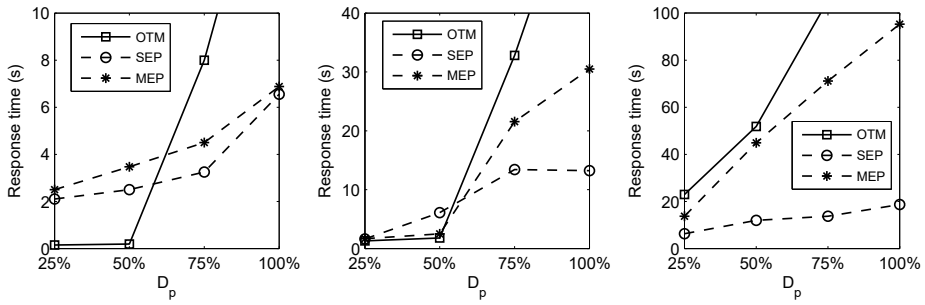
### 3.4.1 Effect of POI Density

In this set of experiments when $D_p < 100\%$ it means we randomly selected an accordingly smaller subset of the POIs from each dataset. We do this in order to investigate the scalability of the investigated approaches with respect to that parameter.

Our experimental result when using the *sum* aggregate, shown in Figure 3.6, shows that the response time of all approaches becomes larger with the increase of the density of POIs. We note when the density of POIs in Amsterdam and Oslo is smaller than 50%, OTM outperforms SEP and MEP. In all other cases the OTM approach requires higher processing time in comparison to other competitors. This is because the performance of OTM technique depends highly on the density of POIs. Assuming that all branch points of all users' preferred paths and POIs have been indexed by the $R^*$-trees $R_p$ and $R_c$, respectively, then the spatial join operation on on those trees incurs higher query processing time with the increase of density of POIs, due to the augmentation of height of corresponding $R^*$-trees. This explanation is in fact valid for all experiments we performed, i.e., the cost of this join is the determinant factor in the the relative poor performance of OTM and therefore we

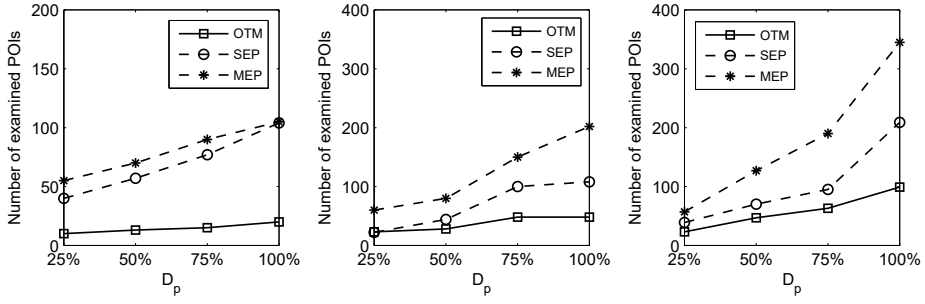
will not repeat it in the following.

Also our experimental results illustrate the superiority of SEP over MEP for all settings of $D_p$. The main reason is that, as illustrated in Figure 3.7, SEP technique provides a stronger pruning strategy in comparison to the MEP technique for pruning the non-promising POIs by considering the total distance of POIs towards the centroids of source locations and destinations. Particularly when the preferred paths of users are placed far from each other, then the pruned search space as the intersection region of all pruned areas around the preferred paths becomes larger than the pruned area around the centroid of source locations and destinations.




(a) Amsterdam (b) Oslo (c) Berlin

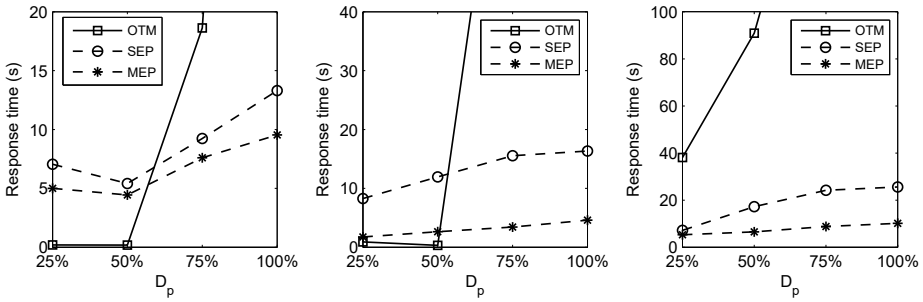
Figure 3.6: Effect of $D_p$ on processing time for $sum$ aggregate function




(a) Amsterdam (b) Oslo (c) Berlin

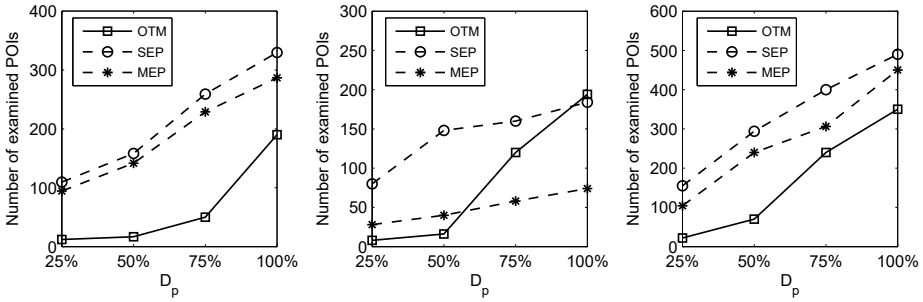
Figure 3.7: Effect of $D_p$ on number of examined POIs for $sum$ aggregate function

Figures 3.8 and 3.9 show the results of varying $D_p$ on the query processing time and number of examined POIs, respectively, when the aggregate function is *max*. As




(a) Amsterdam (b) Oslo (c) Berlin

Figure 3.8: Effect of $D_p$ on processing time for $max$ aggregate function

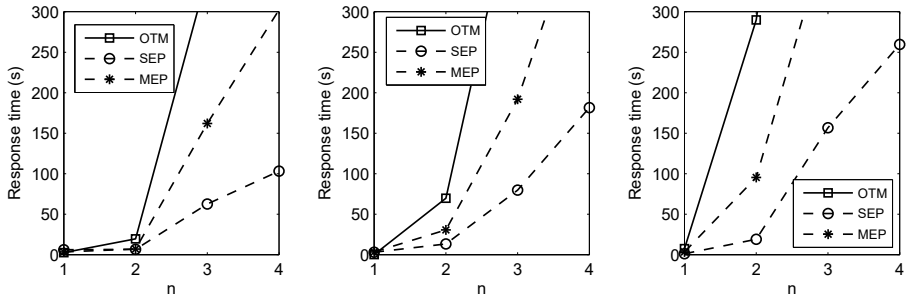



(a) Amsterdam (b) Oslo (c) Berlin

Figure 3.9: Effect of $D_p$ on number of examined POIs for $max$ aggregate function

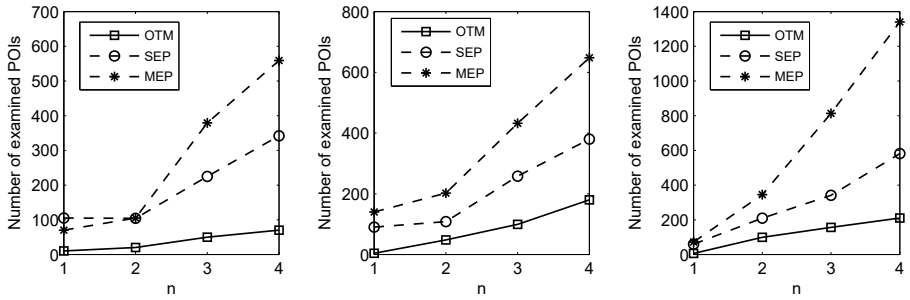
in the case above (*sum* aggregate), and for the same reason, the processing time of OTM technique increases with a faster pace than that of SEP and MEP approaches with the increase of $D_p$. Contrary to the *sum* aggregate, now MEP approach incurs smaller query processing cost by taking advantage of applying stronger pruning strategy and shrinking the search space into a smaller refined search space. In order to illustrate the performance comparison of SEP and MEP approaches, we compare the area of corresponding refined search space of these two techniques, $E_c$ and $E_{int}$. The MEP technique returns a refined search space as the intersection area of all pruned areas around the preferred paths based on corresponding length $l_i$. On the other hand, the SEP technique prunes the search space as an ellipse where the major axis is defined based on the maximum length of all users preferred paths, $2\times Add^k + \max_{1\leq i\leq n}\{l_i^*\}$. That is, as illustrated in Figure 3.9, the MEP approach exhibits

superiority over SEP due to shrinking the search space into a smaller refined area.




(a) Amsterdam (b) Oslo (c) Berlin

Figure 3.10: Effect of $n$ on processing time for $sum$ aggregate function



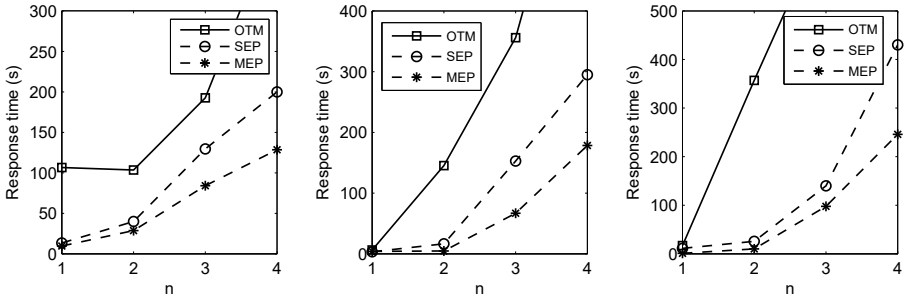

(a) Amsterdam (b) Oslo (c) Berlin

Figure 3.11: Effect of $n$ on number of examined POIs for $sum$ aggregate function

### 3.4.2 Effect of Group Size

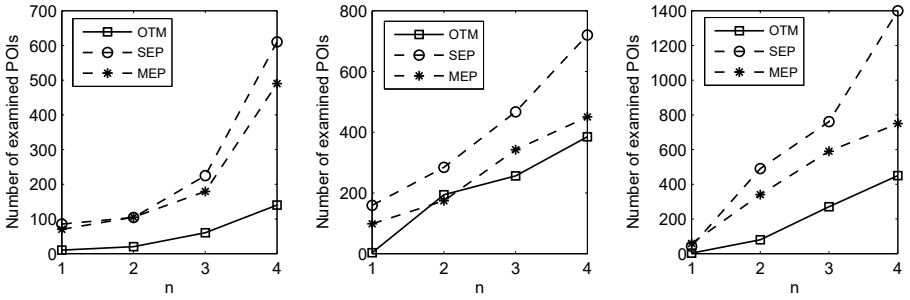
Figure 3.10 shows that when the aggregate is *sum*, the response time of MEP and OTM approaches increases sharply with the augmentation of group size, but the SEP approach scales better for a higher value of $n$ in comparison to its competitors. The main reason is that with the increase of number of users in the group, particularly when the preferred paths of users are placed far from each other, then the pruned search space around each preferred path and subsequently the intersection area of all pruned areas augments with a faster pace than that of SEP technique. That is SEP approach requires smaller response time for processing

k-$OMP^3$ queries, with the increase of number of users in the group, by pruning the search space around the centroid of all source locations and destinations. Figure 3.11 illustrates the performance comparison of MEP, SEP and OTM in terms of number of examined POIs, when the group size increases from 1 to 4, assuming that the aggregate function is $sum$.




(a) Amsterdam (b) Oslo (c) Berlin

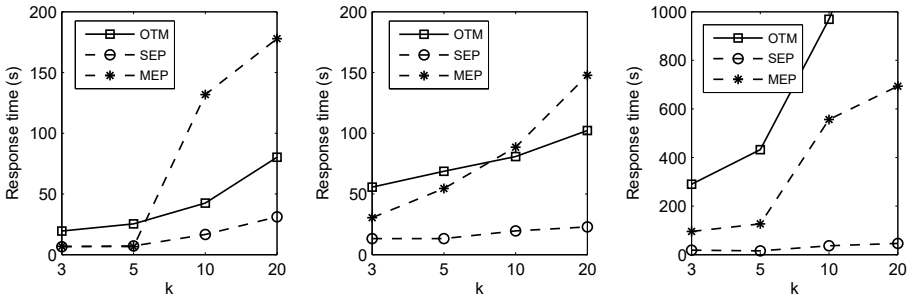
Figure 3.12: Effect of $n$ on processing time for $max$ aggregate function




(a) Amsterdam (b) Oslo (c) Berlin

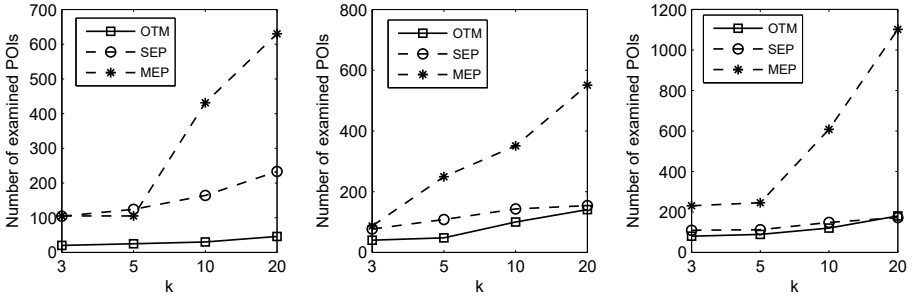
Figure 3.13: Effect of $n$ on number of examined POIs for $max$ aggregate function

When the aggregate is *max*, Figure 3.12 shows that all approaches incur higher response time with the increase of $n$, since as above, the larger group size the higher number of detour distance computations for each retrieved POI as well as higher number of POIs examinations. Again, unlike for the case of the *sum* aggregate MEP is the better performer. Based on our experimental results, MEP approach outperforms other competitors in terms of response time for all settings of $n$. Fur-




(a) Amsterdam (b) Oslo (c) Berlin

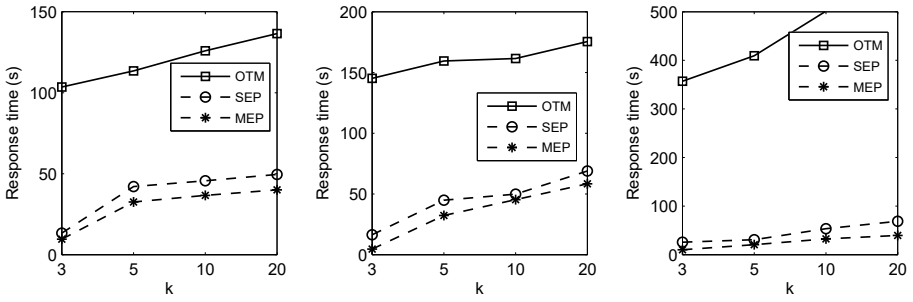
Figure 3.14: Effect of $k$ on processing time for $sum$ aggregate function




(a) Amsterdam (b) Oslo (c) Berlin

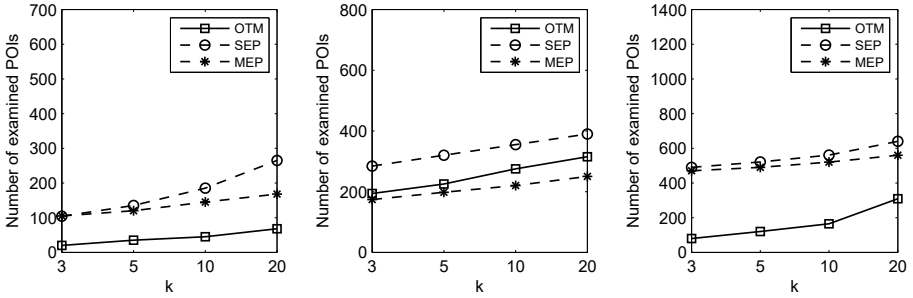
Figure 3.15: Effect of $k$ on number of examined POIs for $sum$ aggregate function

thermore, the gap between the response time of MEP and SEP augments with the further increase of $n$. The reason is that while MEP approach shrinks the search space as the intersection region of all pruned areas around all preferred paths based on corresponding length of preferred path, the SEP technique prunes the search space as an ellipse where the major axis is defined based on the maximum length of all users preferred paths, $2 \times Add^k + \max_{1 \leq i \leq n} \{l_i^*\}$. Because of that with the increase of $n$, the gap between the response time of these two techniques becomes larger. Figure 3.13 illustrates the performance comparison of SEP, MEP and OTM in terms of number of examined POIs, when the group size increases from 1 to 4, assuming that the aggregate function is $max$.




(a) Amsterdam (b) Oslo (c) Berlin

Figure 3.16: Effect of $k$ on processing time for $max$ aggregate function




(a) Amsterdam (b) Oslo (c) Berlin

Figure 3.17: Effect of $k$ on number of examined POIs for $max$ aggregate function

### 3.4.3 Effect of Answer Size

Figure 3.14 reflects the case that when increasing $k$ and the aggregate is *sum*, the search space becomes larger, which translates in larger query processing time for all approaches, and overall SEP displays the best performance, being actually quite robust. Figure 3.15 illustrates the performance comparison of MEP, SEP and OTM in terms of number of examined POIs, when $k$ increases from 3 to 20, assuming that the aggregate function is $sum$. Our experiemnetal results show that the SEP approach refines the search space into a smaller refined area in comparison to MEP approach, because it prunes the search space into a smaller refined area by considering the total distance of POIs towards the centroid of source locations and destinations.

When the aggregate function is $max$, Figures 3.16 and 3.17 indicate that with the increase of $k$, the query processing cost of all approaches increases, since for a larger $k$ they need to retrieve a higher number of POIs. Based on ourt results, the MEP approach requires smaller processing time in comparison to others.

### 3.4.4 Summary of experimental results

- Our proposed techniques, the MEP and SEP approaches, incur smaller response time for processing k-$OMP^3$ queries for both $sum$ and $max$ aggregate functions, in comparison to applying spatial join operation solution OTM approach. This is due to applying efficient pruning strategies for shrinking the search space into a smaller refined search space.

- The SEP approach is the most efficient and robust solution in terms of response time, for processing k-$OMP^3$ queries in the case that the aggregate function is $sum$, due to shrinking the search space into a smaller refined area by considering the total distance of POIs towards the centroids of source locations and destinations.

- In the case that the aggregate function is $max$, the MEP approach provides a stronger pruning strategy in comparison to other techniques, which results in incurring faster response time. The reason is that while MEP approach shrinks the search space as the intersection region of all pruned areas around all preferred paths, the SEP technique prunes the search space as an ellipse where the major axis is defined based on the maximum length of all users preferred paths, leading to a larger refined area than that of MEP technique.

- The performance of OTM technique highly depends on the density of POIs. The reason is that the spatial join operation on $R_c$ and $R_p$ based on top-down traversal of the aforementioned $R^*$-trees incurs higher query processing time particularly with the increase of density of POIs, due to augmentation of height of corresponding $R^*$-trees. That is due to the complexity of spatial join operation and top-down traversal of associated $R^*$-trees, the gap between the

response time of OTM approach and other competitors becomes larger with the increase in the density of POIs.

## 3.5 Conclusion

In this chapter, we proposed a new query type, namely the k-$OMP^3$ query. Given a set of preferred paths of a group of $n$ users and a set of POIs, the k-$OMP^3$ query finds $k$ POIs, incurring the $k$ smallest aggregate detour distances towards all group members preferred paths. For processing k-$OMP^3$ queries, we developed two efficient approaches: MEP and SEP. In order to reduce the query processing cost, the proposed approaches shrink the POIs search space into a smaller space based on geometric properties of ellipses. Our experimental results show that, in the case that the aggregate function is $max$, MEP approach provides a stronger pruning strategy in comparison to SEP approach, which results in incurring faster response time. On the other hand, SEP approach is the most efficient and robust solution in terms of response time for processing k-$OMP^3$ queries, in the case that the aggregate function is $sum$.

# Chapter 4

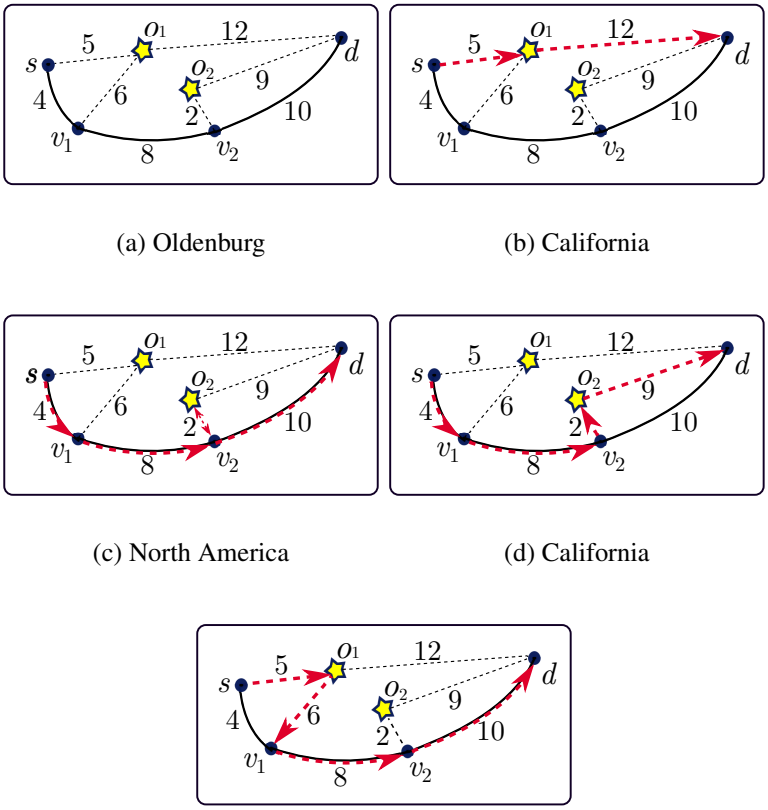
# Best-Compromise In-Route Nearest Neighbor Queries

## 4.1 Introduction

Consider a scenario where a user has his/her preferred path, say a particular bicycle path from work to home, and further assume that the user needs to find a certain type of point-of-interest (POI), e.g., an ATM, a supermarket or a water fountain, while traveling on this path. Assuming that the user wants to return to his/her preferred path after visiting the POI, it would make sense to search for the POI that yields the minimum detour from the preferred path. This type of problem has been previously defined as In-Route Nearest-Neighbor (IRNN) queries [55]. We extend the notion of IRNN queries to consider that it may better for the user to deviate more from his/her preferred path if that would lead to a shorter path overall.

For the sake of an example, let us consider a user's preferred path $P^* = \langle s, v_1, v_2, d \rangle$ from his work place $s$ to his home $d$ as illustrated in Figure 4.1, where $o_1$ and $o_2$ denote POIs, e.g., ATMs or restaurants. On the one hand, if the user wanted to minimize only the total travel distance while visiting a POI, path $P^1 = \langle s, o_1, d \rangle$ (with travel distance 17) would be the best solution. On the other hand, if the user wanted to visit a POI based on a path minimizing the detour distance[1] from $P^*$, i.e., the route found by issuing an IRNN query, the best path would be $P^2 = \langle s, v_1, v_2, o_2, v_2, d \rangle$ (with detour distance 4). While $P^2$ leads to a smaller detour than $P^1$ (4 vs 17), $P^1$ leads to a shorter path overall (17 vs 26). Which one would a user

[1] We defer the definition of detour distance to Section 4.3.




(a) Oldenburg

(b) California

(c) North America

(d) California

(e) North America

Figure 4.1: Alternatives paths on a simplified road network from $s$ to $d$ visiting one of POIs $o_1$ or $o_2$. Table 4.1 summarizes the travel and detour distance implied by each such path.

prefer in general? The answer is not obvious as the trade-off between minimizing the total travel distance and the detour distance may not be easy to assess. Let us consider other alternative paths. $P^3 = \langle s, v_1, v_2, o_2, d \rangle$ yields a total travel distance of 23 and a detour distance of 11, while $P^4 = \langle s, o_1, v_1, v_2, d \rangle$ yields a total travel distance of 29 but the same detour distance of 11.

Given all such alternatives, can we efficiently find a set that represents answers that are objectively better than others? Fortunately the answer is *yes*. Indeed, investigating this novel problem, which we name *Best-Compromise In-Route Nearest Neighbor* (BC-IRNN) query forms the main contribution of this chapter.

A simplistic way to address BC-IRNN queries is to combine both criteria (total travel distance and detour distance) and optimize the combined distance. However,

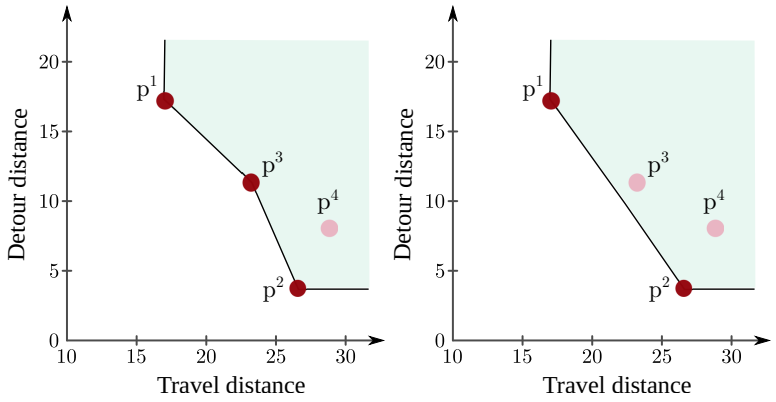
Table 4.1: Paths and their corresponding costs w.r.t. Figure 4.1.

| Path | Cost (distance) | |
|---|---|---|
| | Travel | Detour |
| $P^1$ | 17 | 17 |
| $P^2$ | 26 | 4 |
| $P^3$ | 23 | 11 |
| $P^4$ | 29 | 11 |

finding a single meaningful function means weighting the importance of each type of distance. This depends primarily on the user's preferences which can change often, e.g., on a rainy day one may prioritize the shortest path, whereas in a sunny day the preferred path, even if longer, may be more pleasant. Hence, finding an appropriate weighting would not only add an extra parameter to the problem but would make a solution found potentially short-lived and user-dependent.

Another alternative way to cope with BC-IRNN is to determine all results that are optimal under an arbitrary distance combination. This is the very notion of skyline queries [68]. We discuss those in details shortly in Section 4.3, but for the sake of motivation and in generic terms, the result set of a skyline query contains objects which are not dominated by any other one. An object $o_i$ is dominated by another object $o_j$ if for each cost criterion the cost of $o_i$ is equal or larger than the cost of $o_j$, and for at least one criterion the cost of $o_j$ is strictly smaller than the cost of $o_i$. If we consider the paths shown in Figure 4.1, we can say that $P^1$ (the one that yields minimum total travel distance), $P^2$ (which yields minimum detour distance) and $P^3$ are not dominated by any other path. $P^4$, however, is dominated by $P^2$ and $P^3$ and is therefore a non-interesting solution that can be discarded, and all others are equally interesting and should be offered as alternatives for the user. Figure 4.2(a) illustrates the concept of skyline queries. It shows the corresponding travel and detour distances of these four candidate paths. The linked dots denote the frontier of non-dominated paths and the shaded area contains the dominated ones.

However, it is well known that skyline queries may return a large number of results, potentially with similar costs, making it very hard to the user to choose a particular solution. In order to diminish the size of the solution to a more intuitive set, the authors in [54] proposed the notion of linear skyline queries. A linear sky-




(a) Conventional skyline (b) Linear skyline

Figure 4.2: Conventional and linear skylines for the example shown in Figure 4.1 and summarized in Table 4.1.

line returns a typically much smaller subset of the conventional skyline which is optimal under *any* linear combination of the competing criteria. Figure 4.2(b) illustrates the linear skyline set obtained for the example shown in Figure 4.1. Although $P^3$ is a non-dominated path from a conventional skyline perspective, it is linearly dominated by $\{P^1, P^2\}$. Note that $P^4$ that is already conventionally dominated is also linearly dominated by $\{P^1, P^2\}$. Therefore, for this example the answer set $\{P^1, P^2\}$ contains the (smaller and more diverse) linearly non-dominated paths which would be returned to the user.

Given the discussion above we propose an efficient approach based on the notion of linear skylines to process BC-IRNN queries. The first step of our proposal is to calculate upper bounds to the travel and detour distances of non-dominated paths. This allows us to prune paths that exceed the determined thresholds, i.e., dominated paths, during a network expansion from $s$ that explores the paths in increasing order of travel distance. Given the lack of any other research to solve BC-IRNN queries we compare our proposal to a straightforward approach that finds all shortest paths between all pair of vertices in $P^*$ that visit at least one POI. Our results indicate that our proposed approach can be orders of magnitude faster than such an alternative.

## 4.2 Related Work

Next we review previous research related to two particular topics which are closely related to BC-IRNNs and to the "backbone" of our proposed solution: path-finding based on pre-defined paths and skyline queries.

Shekhar and Yoo proposed the In-Route Nearest Neighbor (IRNN) query [55]. Their query aims at finding the POI through which the detour from the preferred path on the way to the destination is minimized. Differently from an IRNN query, which focus solely on minimizing the deviation from the original route, BC-IRNN queries provide the user with multiple alternative paths that yield different travel and detour distances, but that are still interesting in the sense that they are not linearly dominated by any other path.

In [5] we studied an extension of IRNN queries named k-Optimal Meeting Points based on Preferred paths ($k$-OMP$^3$) queries. Instead of a single user and his/her preferred path, a $k$-OMP$^3$ query takes as input a group of $n$ users and their corresponding preferred paths. It then finds the POI incurring the smallest possible total detour distance from the group's preferred path assuming that users leave from and return to their preferred paths through the same so-called branching point. In this work, we do not make such assumption. Similarly to IRNN queries, in $k$-OMP$^3$ queries there is no concern regarding the actual total path length, the only goal is to minimize the detour distance.

Chen et al. address a query similar to IRNN called $k$-Path Nearest Neighbor ($k$-PNN) query [14]. The main difference between the IRNN and $k$-PNN queries is that in the latter there is no notion of a preferred path. Instead the deviation is measured w.r.t. the shortest path between the user's origin and destination which is itself built "on-the-fly." Also, the returned path needs to visit $k$ POIs, whereas in our case one POI suffices.

Shang et al. [50] investigated the Best Point Detour (BPD) query. Given a preferred path, a BPD query discovers the detour point with the minimum detour *cost* subject to a used-defined threshold. The detour cost is defined as the extra network distance introduced by a detour w.r.t. the travel distance of the original path. In

this work we consider the notion of unconstrained detour *distance*, which is given by the sum of the length of the edges that do not belong to the preferred path. In addition, as opposed to the case of BC-IRNN queries, in BDP queries there is no concern about the total travel distance.

Differently from the works presented above which aim at minimizing a single distance, Huang et al. studied In-Route Skyline (IRS) queries [29]. In that work the authors consider the location of a user along their preferred path and return the set of *POIs*, w.r.t. that location, belonging to the skyline set based on criteria similar to the ones we use. We, on the other hand, do not consider the user's location, rather our returned skyline set is comprised of entire *paths* instead of POIs. In a sense IRN queries are for use during a trip whereas BC-IRNN are for use when planning a trip. Also, while the former return the conventional skyline set, the latter return the (smaller and more diverse) linear skyline set.

Also, in the context of path skyline queries, Kriegel et al. studied the problem of finding all conventionally non-dominated paths between two given vertices in a multicost road network [35]. The authors considered multiple cost criteria such as distance, driving time, the number of traffic lights, gas consumption. Shekelyan et al. investigated the problem of computing Linear Path Skyline in Bi-criteria Networks [54]. However, neither [35] or [54] consider the existence of a preferred path. Another fundamental difference between those papers and the problem addressed in this work is the fact that we aim at finding paths from a starting point $s$ to a destination $d$ that visit at least one POI, rather than POI-independent non-dominated paths from $s$ to $d$.

## 4.3 Preliminaries

We assume that a road network is modeled by an undirected graph $G(V, E, W)$, where $V$ is a set of vertices that represent the road intersections and end-points, $E$ is the set of edges containing all road segments and $W$ indicates the weight of edges in $E$. In our case, the weight of an edge connecting vertices $v_i$ and $v_j$ is given by the length of the road network segment that connects those vertices and is denoted

Table 4.2: Notation.

| *Notation* | *Meaning* |
|---|---|
| $P^i = \langle v_1^i, v_2^i, ..., v_n^i \rangle$ | A path $P^i$ ($P^*$ is the preferred one) |
| $v_j^i$ | The $j$-th vertex in $P^i$ |
| $s = v_1^i$, $d = v_n^i$ | The source and destination in $P^i$ |
| $w(v_i, v_j) = w(v_j, v_i)$ | Length of the edge connecting $v_i$ to $v_j$ |
| $O = \{o_1, ..., o_{\|O\|}\}$ | The set of POIs |
| $TD(P^i)$ | Travel distance of path $P^i$ |
| $DD(P^i, P^*)$ | Detour distance of path $P^i$ w.r.t. $P^*$ |
| $p^i = (p_1^i, p_2^i)$ | Cost vector of path $P^i$ |
| $P^i \prec P^j$ | $P^j$ is conventionally dominated by $P^i$ |
| $\{P^i, P^j\} \prec_L P^k$ | $P^k$ is linearly dominated by $P^i$ and $P^j$ |
| $P^{TD}$ | Shortest full path |
| $P^{UD}$ | Full path that yields the minimum detour distance |
| $DD^U$ | Upper bound for detour distance |
| $TD^U$ | Upper bound for travel distance |
| $OTD(P^i, d)$ | Optimistic travel distance of $P^i$ w.r.t. the destination $d = v_n^*$ |

by $w(v_i, v_j)$. Moreover, the set of POIs of the road network is denoted $O$.

We define a path $P^i = \langle v_1^i, v_2^i, ..., v_n^i \rangle$ in $G$ as a cycle-free sequence of vertices such that any two consecutive vertices $v_j^i$ and $v_{j+1}^i$, for $1 \leq j < n$, are directly connected by an edge $(v_j^i, v_{j+1}^i) \in E$. Particularly, a preferred path is denoted $P^* = \langle v_1^*, v_2^*, ..., v_n^* \rangle$, where $v_1^*$ represents the user's starting location $s$ and $v_n^*$ is the destination $d$. Additionally, a path $P^i = \langle s, ..., o_j, ..., d \rangle$ from $s$ to $d$ that visits at least one POI $o_j \in O$ is referred to as *full path*. Throughout this chapter we use the notation presented in Table 4.2.

In the following we provide the formal definitions for travel and detour distances. Subsequently, in order to distinguish between linear skylines and skylines in the ordinary sense, we first recall the definition of conventional skylines and the conventional dominance relation they are based on.

**Definition 4.3.1** (Travel Distance)**.** Given a path $P^i = \langle v_1^i, v_2^i, ..., v_n^i \rangle$ in $G$, its travel distance is given by the sum of the weights of the edges in it, i.e.,

$$TD(P^i) = \sum_{j=1}^{n-1} w((v_j^i, v_{j+1}^i)).$$

**Definition 4.3.2** (Detour Distance)**.** Given a path $P^i = \langle v_1^i, v_2^i, ..., v_n^i \rangle$ and the pre-

ferred path $P^*$, the detour distance of $P^i$ is defined as the sum of the length of the edges in $P^i$ that do not belong to $P^*$. That is:

$$DD(P^i, P^*) = \sum_{j=1}^{n-1} D(v_j^i, v_{j+1}^i, P^*),$$

where $D(v_j^i, v_{j+1}^i, P^*) = w((v_j^i, v_{j+1}^i))$ if $(v_j^i, v_{j+1}^i) \not\subset P^*$ and $D(v_j^i, v_{j+1}^i, P^*) = 0$, otherwise.

Consider for instance $P^3 = \langle s, v_1, v_2, o_2, d \rangle$ in Figure 4.1. Its travel distance is given by $TD(P^3) = w(s, v_1)+w(v_1, v_2)+w(v_2, o_2)+w(o_2, d) = 4+8+2+9 = 23$, whereas its detour distance, always with respect to the preferred path $P^*$, is given by $DD(P^3, P^*) = w(v_2, o_2) + w(o_2, d) = 2 + 9 = 11$.

Given a preferred path $P^*$ and a set of POIs $O$, our goal is to find the set $LS$ containing linearly non-dominated full paths w.r.t. detour and travel distance. The answer set found by a linear skyline is a subset of a conventional skyline result set. In order to distinguish between conventional and linear skyline operations, we first recall the definition of a conventional skyline. In the following definitions we denote the cost vector of a path $P^i$ as $p^i$, where $p_1^i$ and $p_2^i$ represent the two cost criteria that are to be minimized, e.g., travel and detour distances in our particular case of interest.

**Definition 4.3.3** (Conventional Dominance)**.** Let $\mathcal{P}$ be a set of paths in a two-dimensional cost space. A path $P^i \in \mathcal{P}$ conventionally dominates another path $P^j \in \mathcal{P}$, denoted as $P^i \prec P^j$, if

$$(p_1^i < p_1^j \ \wedge \ p_2^i \leq p_2^j) \vee (p_1^i \leq p_1^j \ \wedge \ p_2^i < p_2^j)$$

That is, $P^i$ is better in one criteria and at least as good as $P^j$ in the other one. The set of non-dominated paths, i.e. $\{P^i \in \mathcal{P} \mid \nexists P^j \in \mathcal{P} : p^i \prec p^j\}$, denotes the conventional skyline.

A linear skyline consists of the subset of the conventional skyline which is optimal under all linear combination functions. In our case the linear skyline set is composed of full paths that minimize $\mathcal{F} = \delta_1 \times p_1^i + \delta_2 \times p_2^i$ for all possible weight

vectors $\delta = (\delta_1, \delta_2)$. A vector $\delta$ represents the weights (importance) that a user could give to both criteria. It is worth emphasizing that although we find the optimal solution for all such $\delta$, we do not require any weight vector to be provided beforehand. In the following we present the definition of $\delta$-dominance which determines when a path linearly dominates another one for a particular $\delta$.

**Definition 4.3.4** (Linear Dominance)**.** A path $P^i$ $\delta$-dominates another path $P^j$, where $\delta$ is a vector such that $\delta \in \mathbb{R}^2_{\geq 0}$ and $\delta \neq (0, 0)$, if and only if $\delta^T p^i < \delta^T p^j$ [54].

In order to illustrate this concept let us consider Figure 4.1 again. The path $P^1$ $\delta$-dominates $P^2$ for $\delta = (1, 0)$, since $17 \times 1 + 17 \times 0 < 26 \times 1 + 4 \times 0$. However, this is not a sufficient condition for determining whether $P^2$ is a linearly dominated path in the general sense. Note that, for instance, $P^2$ $\delta$-dominates $P^1$ for $\delta = (0.5, 0.5)$. As formalized in the following definition, a path is only considered to be linearly dominated if it is either conventionally dominated or $\delta$-dominated for every possible $\delta$.

**Definition 4.3.5** (Linear Skyline)**.** Let $\mathcal{P}$ be a set of paths in a two-dimensional cost space. A subset $\mathcal{P}' \subseteq \mathcal{P}$ linearly dominates a path $P^j \in \mathcal{P}$, denoted as $\mathcal{P}' \prec_L P^j$, if and only if

$$(\exists P^i \in \mathcal{P}' \; s.t. \; P^i \prec P^j) \vee (\forall \delta \in \mathbb{R}^2_{\geq 0} \; \exists P^i \in \mathcal{P}' \; s.t. \; \delta^T p^i < \delta^T p^j)$$

The maximal set of *linearly non-dominated* paths is referred to as *linear skyline* [54].

In other words, a path $P^i$ is linearly non-dominated w.r.t. a set $\mathcal{P}'$ if it is not conventionally dominated by any path in $\mathcal{P}'$ *and* there is a vector $\delta$ and a path $P^j \in \mathcal{P}'$ such that $P^i$ $\delta$-dominates $P^j$. For the set of paths $\mathcal{P}' = \{P^1, P^2, P^3, P^4\}$ shown in Figure 4.1, $P^1$ and $P^2$ are considered linearly non-dominated since they are conventionally non-dominated and, as shown above, $P^1$ $\delta$-dominates $P^2$ for $\delta = (1, 0)$ and $P^2$ $\delta$-dominates $P^1$ for $\delta = (0.5, 0.5)$. Thus, both paths satisfy the conditions to be considered linearly non-dominated. On the other hand, although

$P^3$ is conventionally non-dominated, there is no weight vector $\delta$ for which $P^3$ $\delta$-dominates either $P^1$ or $P^2$. $P^4$ is a conventionally dominated path and thus it is also linearly dominated.

The problem addressed in this chapter can now be formally defined as follows.

**Definition 4.3.6** (BC-IRNN query)**.** *Given a user's preferred path* $P^* = \langle v_1^*, v_2^*, ..., v_n^* \rangle$ *and a set of POIs* $O$*, the BC-IRNN query aims at finding the set of all linearly non-dominated full paths, i.e., paths from* $s = v_1^*$ *to* $d = v_n^*$ *that visit at least one POI* $o_i \in O$*.*

Shekelyan et al. [54] showed that the notion of linear dominance has an intuitive graphical intuition, which does not require having to check all possible vectors $\delta$ in order to determine whether a path is linearly dominated. They argued that a path $P^i$ is linearly dominated by a set of paths $\mathcal{P}'$ if $p^i$ lies above the line between any two cost vectors of paths from $\mathcal{P}'$. This observation can be formalized as follows. Let $P^i$ and $P^j$ be two paths with $p_1^i < p_1^j$ and $p_2^j < p_2^i$. Let $n$ be the *normal vector* of the line between $P^i$ and $P^j$, such that $n^T p^i = n^T p^j$, and $u$ be the component-wise minimum of $P^i$ and $P^j$, such that $u_1 = \min(p_1^i, p_1^j)$ and $u_2 = \min(p_2^i, p_2^j)$. A path $P^k$ is linearly dominated by $\{p^i, p^j\}$, denoted as $\{p^i, p^j\} \prec_L p^k$, iff:

$$(u \prec p^k) \wedge (n^T p^k > n^T p^i = n^T p^j) \tag{4.1}$$

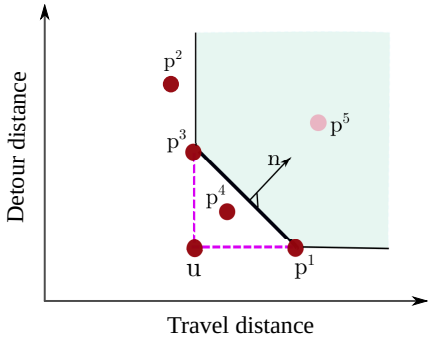



Figure 4.3: Area linearly dominated by $\{\mathbf{p^1}, \mathbf{p^3}\}$.

In order to illustrate this concept, let us consider Figure 4.3 which shows a set of paths $\{P^1, .., P^5\}$ mapped into a bi-criteria cost space. The colored area

shows the area linearly dominated by $\{p^1, p^3\}$, containing only $p^5$. In this example, $\{p^1, p^3\} \not\prec_L p^2$ because even though the condition $n^T p^2 > n^T p^1 = n^T p^3$ holds, $p^2$ is not conventionally dominated by $u = (p_1^3, p_2^1)$. Furthermore, although $p^4$ is conventionally dominated by $u$, it is not linearly dominated by $\{p^1, p^3\}$, because it is placed below the line connecting $p^1$ and $p^3$, i.e., $n^T p^4 < n^T p^1 = n^T p^3$.

In our proposed approach for processing BC-IRNN queries, the candidate full paths are retrieved in increasing order of travel distance. The linearly non-dominated full paths are stored in a list $LS = (P^1, ..., P^k)$ which is ordered in increasing order of travel distance. Let us assume that $P^i$ is a new full path found by our approach for which we want to determine whether it should be added to $LS$ or not. Note that $TD(P^i) > TD(P^k)$, since, as we shall show shortly, the paths are always found in increasing order of travel distance and thus $TD(P^i)$ is greater than the travel distances of all the other paths in $LS$. Therefore, $DD(P^i, P^*) < DD(P^k, P^*)$ must hold for $P^i$ to be considered a non-dominated path. As proved in [54], if this condition is satisfied, i.e., $P^i$ is a conventionally non-dominated path, then it is also linearly non-dominated in $LS$ and thus it can be added to that set.

After adding a new non-dominated path to $LS$, some paths may become linearly dominated. Therefore we need to check whether inserting $P^i$ into $LS$ leads to the removal of other paths from $LS$. Shekelyan et al. [54] showed that any path $P^j$ is only linearly dominated in $LS$ if it is dominated by both of its neighbors. Thus, in order to determine if $P^k$ is linearly dominated, it is sufficient to check whether $\{P^{k-1}, P^i\} \prec_L P^k$. Note that $P^i$ is the right neighbor of $P^k$ since it is inserted at position $k+1$. If this condition is satisfied, $P^k$ is removed from $LS$. After that, $P^{k-1}$ becomes the left neighbor of $P^i$. Subsequently, we need to examine whether $\{P^{k-2}, P^i\} \prec_L P^{k-1}$ holds. This process will be terminated when the first element of $LS$ has been examined or the current left neighbor of $P^i$ is a linearly non-dominated path.

## 4.4 Our Proposed Approach

Our proposed approach first finds the full path that yields the minimum detour distance and use its travel distance as an upper bound to prune paths that are too long to be considered non-dominated. Similarly, we also find the shortest full path and use its detour distance as an upper bound for pruning paths that deviate from $P^*$ more than the determined upper bound. Subsequently, the linearly non-dominated paths are retrieved by traversing the road network from $s$. During this traversal the previously determined upper bounds are applied for shrinking the search space.

### 4.4.1 Upper bounds for travel and detour distance

The first step of our proposed approach is to determine upper bounds for the detour and travel distances of non-dominated paths, denoted respectively by $DD^U$ and $TD^U$. The goal is to use those to prune paths that can be proven to be dominated.

In order to calculate these bounds, we need to find two paths: $P^{DD}$ which is the full path that yields the smallest detour distance from the preferred path $P^*$, and $P^{TD}$ which is the shortest full path. The corresponding detour and travel distances of $P^{TD}$ and $P^{DD}$, namely $DD^U$=$DD(P^{TD}, P^*)$ and $TD^U$=$TD(P^{DD})$ respectively, are the two bounds used for pruning the search space. In the following we discuss how $P^{DD}$ and $P^{TD}$ are computed.

**Computing $DD^U$: an upper bound for the detour distance**

Let $c_m$ be the centroid of $s$ and $d$ in Euclidean space, then for determining $P^{TD}$ and $DD^U = DD(P^{TD})$, according to Chapter 2, we start iteratively retrieving the (Euclidean distance-wise) nearest neighbors [69] from the set of POIs $O$ w.r.t. $c_m$. Let us denote as $o^j = NN(c_m, O, j)$ the $j$-th such nearest POI. For each retrieved POI $o_j$, we form a candidate trip $P^j = \langle s, ..., o_j, ..., d \rangle$, where the paths from $s$ towards $o_j$ and similarly from $o_j$ w.r.t. $d$ are the shortest ones. According to Lemma 4.4.1, the processes of forming candidate trips can stop when the condition $MTD < 2 \times d_e(o_j, c_m)$ is satisfied, where $MTD$ is the smallest travel distance computed so far and $d_e(.,.)$ indicates the Euclidean distance between two locations.

Finally among all generated candidate full paths, the one incurring the smallest travel distance will be returned as $P^{TD}$. In the example shown in Figure 4.4(a) $P^{TD}$= $P^1$=$\langle s, O_1, d\rangle$.

**Lemma 4.4.1.** Let $c_m$ be the centroid of $s$ and $d$, and $o_j$ the $j$-th retrieved nearest neighbor POI w.r.t. $c_m$. Furthermore let $MTD$=$TD(P^i)$ be the smallest travel distance computed so far. Then $P^i$ corresponds to $P^{TD}$, if $MTD < 2 \times d_e(o_j, c_m)$.

*Proof.* According to [38], we have $d_e(s, o_j) + d_e(o_j, d) > 2 \times d_e(o_j, c_m)$. Consequently, if $2 \times d_e(o_j, c_m) > MTD$ then we can conclude that $d_e(s, o_j) + d_e(o_j, d) > MTD$. This means that any full path that visits $o_j$ would incur travel distance greater than $MTD$. Furthermore $\forall o_z = NN(c_m, O, z)$, where $z > j$, we have $d_e(o_z, c_m)$¿$d_e(o_j, c_m)$, leading to $d_e(s, o_z) + d_e(o_z, d) > MTD$. We can then conclude that $TD(p^z) > MTD$, and there would be no other unexamined path incurring smaller travel distance than $MTD$. □

**Computing $TD^U$: an upper bound for the travel distance**

In order to compute the upper bound for the travel distance, $TD^U$, we need to find $P^{DD}$, the full path which incurs the smallest possible detour distance w.r.t $P^*$, i.e., $TD^U$=$TD(P^{DD})$. For that we need to find a POI incurring the smallest detour distance. In Chapter 3, we discussed how such POI can be determined. As we shall discuss in Section 4.4.2 all full (partial) paths which incur a travel distance greater than $TD^U$ are (will be) conventionally dominated by $P^{DD}$, and thus cannot be part of the answer set. Our strategy for determining $P^{DD}$ is to first find the POI $o_i$ closest to a vertex in $P^*$, then use it to determine $P^{DD}$. For determining $o_i$ we use a group nearest neighbor query [42], where one group is the set $O$ of POIs and the other group is the vertices in $P^*$. The answer of such a query is the POI that minimizes the total distance to all vertices in $P^*$, which serves as an approximation for $o_i$ sought[2].

Then, we compute the smallest possible network distance of $o_i$ w.r.t. $P^*$, denoted by $d_n^{min}(o_i, P^*)$=$\min_{\forall v_j^* \in P^*} d_n(o_i, v_j^*)$. Note that $d_n^{min}(o_i, P^*)$ indicates the

[2] Our preliminary results indicated that using only $\{s, d\}$ instead of all vertices in $P^*$ yielded virtually same results at a lower processing cost and thus we used only those vertices in our implementation.

half of the smallest possible detour distance for visiting $o_i$ from $P^*$, as one needs to leave $P^*$, visit $o_i$ and return to $P^*$. The distance $d_n^{min}(o_i, P^*)$ will be used for shrinking the search space. For that, we compute an ellipse $E$, where its focal points are placed at $s$ and $d$, and the major axis is defined as $2 \times d_n^{min}(o_i, P^*) + TD(P^*)$. According to Lemmas 3.3.1 and 3.3.2 (formally presented shortly), the POIs that are placed outside $E$ would incur a detour distance greater than that of $o_i$. Then all POIs within $E$ will be examined for determining the POI that incurs the smallest possible detour distance w.r.t. $P^*$. Assuming that POI $o_z \in O$ is that one POI and its reached via vertex $v_j^* \in P^*$, then the path $\langle s, ..., v_j^*, ..., o_z, ..., v_j^*, ..., d\rangle$ will be returned as $P^{DD}$, and $TD^U$ will be set as $TD(P^{DD})$. Revisiting the scenario in Figure 4.4(a), we have $P^{DD}$=$P^2 = \langle s, b_1, b_2, O_2, b_2, d\rangle$.

Next we discuss in details how the upper bounds $TD^U$ and $DD^U$ determined as detailed above are used for shrinking the search space while traversing the road network in order to produce the desired result set for the BC-IRNN query.

### 4.4.2 Generating and pruning candidate paths

After computing the upper bounds $TD^U$ and $DD^U$, we incrementally generate paths starting from $s$ through a network traversal based on an $A^*$ search [24]. An $A^*$ search expands paths based on an optimistic total travel distance. In this work, we assume that the optimistic distance from a vertex $v_i \in V$ to $d$ is given by their Euclidean distance, $d_e(v_i, d)$. It is easy to show that $d_e(v_i, d)$ is a lower bound to the optimal (minimum) network distance between $v_i$ and $d$, therefore it is an admissible heuristic and the $A^*$ search is guaranteed to return the optimal result. Let $P^i = \langle s, ..., v_j^i\rangle$ be a partial path ending at $v_j^i$. The optimistic travel distance for $P^i$ is then defined as $OTD(P^i) = TD(P^i) + d_e(v_j^i, d)$.

All paths from $s$ to any vertex $v_i \in V$ found during the network traversal are stored in a priority queue $Q$, where its elements are ordered based on their corresponding optimistic travel distance. At each iteration of our proposed algorithm, the path $P^i$ with minimum $OTD(P^i)$ value is popped from $Q$. If $P^i$ is a partial path and $DD(P^i, P^*) < DD^U$, then $P^i$ is expanded. Otherwise, if $DD(P^i, P^*) > DD^U$, then according to Lemma 4.4.2, presented next, $P^i$ cannot be part of a path belong-

ing to the linear skyline set and thus it can be safely pruned.

**Lemma 4.4.2.** Let $P^i$ be a partial path such that $DD(P^i, P^*) < DD^U$. Then $P^i$ leads to a dominated full path and thus it can be pruned.

*Proof.* Let $P^j$ be an extension of $P^i$ that ends at $d$ and contains at least one POI. Given that Once $P^{TD}$ is the shortest full path we have that $TD(P^j) > TD(P^{TD})$. Moreover, since by assumption $DD(P^i, P^*) > DD^U$=$DD(P^{TD}, P^*)$, then $DD(P^j, P^*) > DD(P^{TD}, P^*)$ also holds. Therefore, $P^j$ is dominated by $P^{TD}$. Thus $P^i$ leads to a dominated full path and thus it can be pruned. □

Moreover, when a non-dominated full path $P^i$ is dequeued from $Q$, the upper bound $DD^U$ can be updated to $DD(P^i)$ since all following non-dominated full paths must have a detour distance smaller than $DD(P^i)$, as proved in Lemma 4.4.3. This allows us to further shrink the search space by pruning paths whose detour distance from the preferred path $P^*$ is larger than the current non-dominated path incurring the smallest detour distance among all full paths computed during the network traversal.

**Lemma 4.4.3.** Whenever a new non-dominated full path $P^i$ is dequeued, $DD(P^i)$ is guaranteed to be an upper bound to the detour distance of non-dominated full paths that have not been examined yet and $DD^U$ can then be updated to $DD(P^i)$.

*Proof.* Let us suppose that there is an unexamined non-dominated full path $P^j$ such that $DD(P^j) > DD(P^i)$. Since the paths are found in increasing order of travel distance and $P^j$ has not been examined yet, then $TD(P^j) > TD(P^i)$. Therefore, $DD(P^j) < DD(P^i)$ must hold for $P^j$ to be considered a non-dominated path. However, by assumption $DD(P^j) > DD(P^i)$, which is a contradiction to the fact that $P^j$ is not dominated. Therefore all non-dominated full paths found after $P^i$ must have a smaller detour distance than $DD(P^i)$ and thus $DD^U$ can be updated to $DD(P^i)$. □

Furthermore, if $OTD(P^i) > TD^U$ the process of traversing the road network can be terminated since all linearly non-dominated paths will have already been examined, a result guaranteed by Lemma 4.4.4, stated next.

**Lemma 4.4.4.** The network traversal can be terminated once the first path $P^i$ such that $OTD(P^i) > TD^U$ is found.

*Proof.* Since $OTD(P^i) > TD^U = TD(P^{DD})$ and $OTD(P^i) \leq TD(P^i)$ it is also true that $TD(P^i) > TD(P^{DD})$. This means that $P^{DD}$ is better than $P^i$ in terms of travel distance. Thus one of the conditions that must be satisfied for $P^i$ to be considered a candidate to be explored is $DD(P^i) < DD(P^{DD})$. Let us analyze the two possible cases:

1. $P^i$ is a full path. Therefore $DD(P^i) < DD(P^{DD})$ does not hold since $P^{DD}$ is the full path that yields the minimum detour distance. Thus $P^i$ is a dominated path and can be pruned.

2. $P^i$ is a partial path. Let $P^j$ be an extension of $P^i$ that ends at $d$ and contains at least one POI. Since $TD(P^i) > TD(P^{DD})$, $TD(P^j) > TD(P^{DD})$ also holds. Moreover, by definition $DD(P^j) > DD(P^{DD})$. Therefore $P^j$ is dominated by $P^{DD}$ and thus $P^i$ does not lead to a non-dominated full path.

Therefore, no further path needs to examined beyond $P^i$, and consequently the network traversal can be safely stopped. □

When a full path $P^i$ is dequeued, we check whether $P^i$ is linearly non-dominated in the linear skyline set $LS$, which is done by simply checking whether $P^i$ is conventionally non-dominated, as explained in Section 4.3. If so, $P^i$ is inserted into linear skyline answer set $LS$.

Algorithm 9 describes our proposed technique for processing BC-IRNN queries in pseudo-code. In Lines 2-3, we determine the two upper bounds $DD^U$ and $TD^U$, as described in Section 4.4.1. In the next step, the first partial route containing only the vertex $s$ is generated and stored in the priority queue $Q$ (Lines 4-6). We recall that the elements in $Q$ are ordered according to their corresponding optimistic travel distance. Then as an iterative process, at each step the path $P$ with minimum $OTD(P)$ is dequeued (Line 8). For each path $P$ we first check whether $OTD(P) > TD^U$ (Lines 9-10). If this condition is satisfied, then according to Lemma 4.4.4, the network traversal can be terminated. Otherwise, we check

whether $P$ is a full path (Line 12). If so, $P$ is a candidate to be added to set $LS$.

**Algorithm 9:** Proposed Approach

**Input:** Starting point $s$, Destination $d$, Preferred path $P^* = \langle v_1^*, v_2^*, ..., v_n^* \rangle$ and $O = \{o_1, ..., o_{|O|}\}$

**Output:** Linear skyline $LS$ containing the paths that visit at least one POI from $O$

```
1  LS ← ∅
2  DD^U ← DD(P^TD, P*)
3  TD^U ← TD(P^DD)
4  P ← ⟨s⟩
5  P.previous ← s
6  Q.insert(P)
7  while Q ≠ ∅ do
8      P ← Q.pop()
9      if OTD(P) > TD^U then
10         return LS
11     v ← last vertex of P
12     if v = d & P contains at least one POI then
13         if P is not conventionally dominated then
14             k ← |LS|
15             Add P at position k+1 in LS
16             DD^U=DD(P)
17             while k > 2 & {P^(k-1), P} ≺_L P^k do
18                 Delete P^k from LS
19                 k=k-1
20     else if DD(P) < DD^U then
21         for all (v, u) ∈ E do
22             if (u ≠ P.previous) or (v is a POI) then
23                 P^u ← extend P with u
24                 P^u.previous ← v
25                 Q.insert(P^u)
26 return LS
```

For each such $P$, two verifications are necessary. Firstly, we check whether $P$ should be added to $LS$, i.e., if it is a linearly non-dominated path. After that, in the case that $P$ linearly non-dominated, we verify whether there are paths currently belonging to $LS$ that may have become linearly dominated and thus must be removed from $LS$. In the following we discuss how these verifications are performed.

In order to check whether a path $P$ should be added to the list $LS$, it is only necessary to verify if $P$ is a conventionally non-dominated path, as explained in Section 4.3. If this condition is satisfied, $P$ is added at the end of $LS$ (Lines 14-15), since it is the longest full path found so far and $LS$ is ordered in increasing order of travel distance. Also, according to Lemma 4.4.3, the upper bound $DD^U$ can be updated to $DD(P)$. Moreover, as aforementioned, we need to check whether the insertion of $P$ into $LS$ leads to the removal of other paths from $LS$ (Lines 17-19). For this, we traverse $LS$ in the left direction starting from $P$, as described in Section 4.3. Considering $P^k$ as the left neighbor of $P$, in the case that $P^{k-1}$ exists, we check whether $\{P, P^{k-1}\} \prec_L P^k$ holds. If this condition is satisfied, $P^k$ is removed from $LS$. After that, $P^{k-1}$ will be considered the left neighbor of $P$, where similarly the linearly non-dominancy of $P^{k-1}$ w.r.t. $\{P, P^{k-2}\}$ must be examined. This process will be terminated when the first element of $LS$ has been examined or the current left neighbor of $P$ is a linearly non-dominated path.

In case a dequeued path $P$ is partial, we need to verify if it is eligible to be expanded. For that, according to Lemma 4.4.2, we check whether the condition $DD(P) < DD^U$ holds (Line 20). If so, $P$ is expanded. A new path $P^u$ is created for each neighbor $u$ of the last vertex $v$ of $P$, i.e., $P$ is extended by adding $u$ to it. Note that in Line 22, we check if $u$ is the vertex that appears just before $v$ in $P$ and $v$ is not a POI, then the path $P^u$ is not created. By doing this we avoid returning to the previously visited vertex, unless it is after visiting a POI. All the paths $P^u$ created are inserted into $Q$. The steps above are repeated until the optmistic travel distance of the current shortest path exceeds the upper bound $TD^U$ or $Q$ is empty.

### 4.4.3 Baseline approach

Finally, let us now discuss a straightforward approach for solving BC-IRNN queries, which we use as the baseline reference in our experiments in Section 4.5. Given a preferred path $P^*$, the baseline is obtained by performing a trip planning query, using for instance the approach proposed in [4], for each pair of vertices $(v_i, v_j) \in P^*$; note that $v_i = v_j$ is a feasible pair as well. In other words, for each such pair we compute the shortest path starting from $v_i$ and ending at $v_j$ that visits at least one

POI. We assume that the traveler moves from $s$ towards $v_i$ along $P^*$, takes a detour for visiting a POI, returns back to $P^*$ through $v_j$ and continues his/her journey towards $d$ along $P^*$. Assuming that $P^*$ contains $n$ branch points, processing a BC-IRNN query based on this straightforward approach would require performing $O(n^2)$ trip planning queries in order to retrieve the $LS$ set containing all linearly non-dominated paths. Clearly, the performance of such a baseline deteriorates when $n$ increases, making it impractical for scenarios where $P^*$'s length is large. Nevertheless, given the lack of alternatives, we use it as a baseline for comparison purposes in the experiments discussed next.

## 4.5 Experimental Results

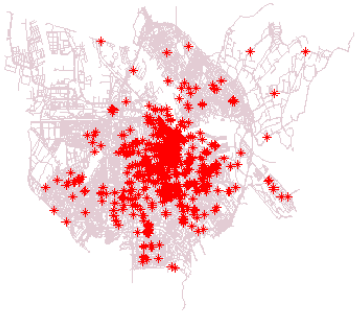
We evaluate the performance of our proposed approach, as well as the baseline approach (as discussed in Section 4.4) for processing BC-IRNN queries using real datasets [9]. The datasets reflect eateries (restaurants and coffee shops) serving as POIs in Amsterdam, Oslo and Berlin, depicted in Figure 3.17(c) as of March/2007. We used the real datasets of three cities Berlin, Oslo and Amsterdam road networks in our experiments, illustrated in Table 3.3. We used restaurants and coffee-shops as POIs in our experiments. Figure 4.4 illustrates Oslo, Berlin and Amsterdam road networks and corresponding POIs used in our experiments.

Table 4.3: Experimental parameters and their values (**bold** defines default values).

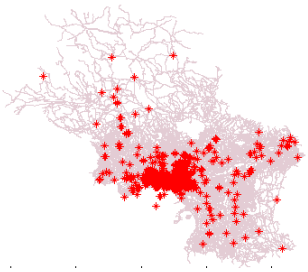
| Parameter | Range |
|---|---|
| Length of preferred path ($n$) | 10, **30**, 50, 70 |
| COI Density ($D_p$) | 10%, 25%, 50%, **100%** |

Our first observation was that, as expected, the cardinality of the answer set, i.e., the number of linearly non-dominated full paths was typically very small. In fact, none of our experiments yielded more than than 10 such paths and most were in the range of 5-6, which reinforces the usefulness of linear skylines.

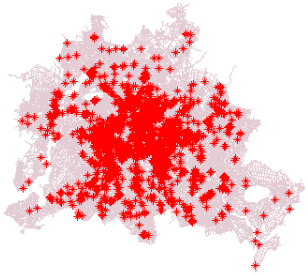
To establish the query processing efficiency, we varied the density of POIs $D_p$, and the length $n$ of the preferred path given as the number of vertices therein. In order to determine the latter, we randomly selected two vertices on the road net-



(a) Amsterdam road network



(b) Oslo road network



(c) Berlin road network

Figure 4.4: Locations of restaurants and coffee shops in Amsterdam, Oslo and Berlin (overlaid on these cities' road network)

work, computed the shortest path between them, and used the first $n$ vertices as the preferred path[3]. Table 4.3 summarizes the parameter ranges and default values. For each set of experiments, we vary the value of one parameter, and fix the other parameters to their default values. All experiments were done using a computer with Intel Core i5 2.40 GHz CPU and 8GB RAM.

### 4.5.1 Effect of Length of Preferred Path

Our experimental results shown in Figure 4.5 illustrates that the query response time of both approaches becomes larger with the increase in the length of the preferred path. The query response time of the baseline approach grows at a significantly faster pace than our approach's. This is because, as discussed in Section 4.4, the

[3]For the sake of reference two vertices in the road networks we use typically equate to the side of a city block.

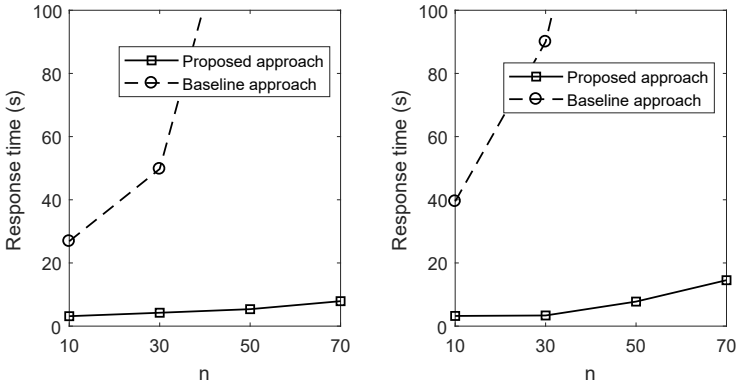
baseline approach performs a Trip Planning Query (TPQ) for each pair of vertices in the preferred path. Therefore, a greater $n$ requires the execution of more such queries.

Even though at a much more acceptable pace, the processing time of our approach also grows with $n$. The farther the starting location $s$ and the destination $d$ are from each other, the higher the number of possible paths between them. This implies that more paths will need to be explored during the network traversal, which in turn increases the query processing time. This also explains why the processing time is longer for larger road networks. Moreover, when the preferred path contains more vertices, the paths connecting $s$ and $d$ are more likely to contain more edges that do not belong to the preferred path. This potentially increases the value of the upper bound $DD^U$, which causes less paths to be pruned based on their detour distance.

### 4.5.2 Effect of POI Density

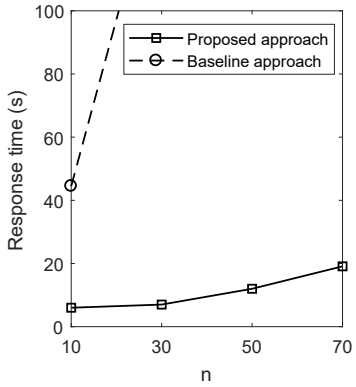
In this experiment we investigated how both solutions behave when the density of POIs $D_p$ is varied. To do so, we randomly selected a subset of the POIs from each dataset containing a certain percentage of the original set of POIs. Note that for $D_p = 100\%$, the whole set of POIs is used. Figure 4.6 shows that, contrary to our proposed approach for which the response time becomes smaller when $D_p$ increases, the baseline technique takes longer to processing a query. The baseline approach requires processing TPQ queries which aim at finding the POI that via which the travel distance from the $s$ to $d$ is minimized. The higher the density of POIs, the greater the number of candidate POIs that need to be examined in order to find the one that yields the smallest travel distance.

The main reason why our proposal is more efficient for larger values of $D_p$ is that $P^{DD}$, the path that yields the smallest detour distance, becomes shorter with the increase in the number of POIs, since it is easier to find a POI that is closer to the preferred path. Consequently, the upper bound $TD^U = TD(P^{DD})$ will also be smaller, which in turn allows the network traversal to be terminated earlier.




(a) Amsterdam

(b) Oslo

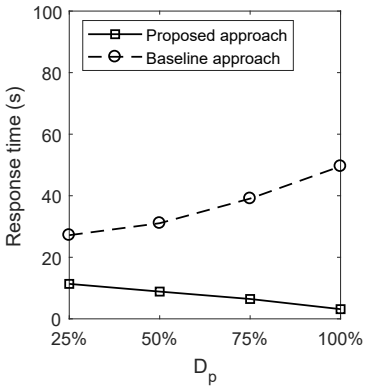



(c) Berlin

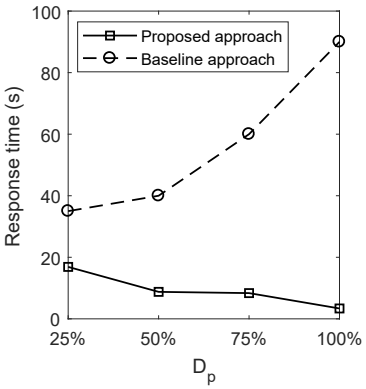
Figure 4.5: Effect of path length $n$ on query processing time

## 4.6 Conclusion

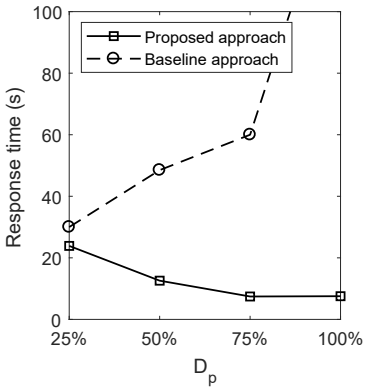
In this chapter, we proposed a new query type, namely the Best-Compromise In-Route Nearest Neighbor (BC-IRNN) query. Guided by the principles of skyline queries, we proposed a road network traversal-based approach, which applies a set of pruning strategies that shrink the search space and discard paths that are conventionally dominated and thus could not be part of the linearly non-dominated answer set. Our experimental results on real datasets show that our proposed approach incurs in significantly faster query processing time when compared to a straightforward baseline approach.




(a) Amsterdam




(b) Oslo




(c) Berlin

Figure 4.6: Effect of POI density $D_p$ on query processing time

# Chapter 5

# k-Closest Pairs Queries in Road Networks

## 5.1 Introduction

Given two spatial sets of points-of-interest (POIs) $P$ and $Q$ on a road network, a $k$-Closest Pairs Query ($k$-CPQ) returns the pairs of POIs from $P \times Q$, that have the $k$ smallest network distances between POIs in $P$ and POIs in $Q$.A typical application for $k$-CPQs is illustrated in Figure 5.1 where a road network is a graph, the set $P = \{v_2, v_6\}$ shown as dark grey vertices represent the locations of touristic sites, the set $Q = \{v_0, v_3, v_5, v_{11}\}$ shown as light grey vertices stand for hotels in the corresponding road network and white vertices are road intersections. The weights on the edges indicate the length of the corresponding road segments. A 2-CPQ will return the pairs of touristic sites and hotels that have the two *smallest network distances* among all possible pairs of sites and hotels. The answer set for this example is $\{(v_2, v_0)$,$(v_6, v_3)\}$, where the length of the shortest paths between $v_2$ and $v_0$ and between $v_3$ and $v_6$ are both equal to 3.

More formally, we consider a road network modeled as an weighted graph $G = \langle V, E\rangle$, in which the weight of edge $(v_i, v_j) \in E$, where $v_i, v_j \in V$, is denoted by $w(v_i, v_j)$. Each node in $V$ is either an intersection in the road network or a node belonging to either data set $P = \{p_1, ..., p_{|P|}\}$ or data set $Q = \{q_1, ..., q_{|Q|}\}$. Then a $k$-CPQ between $P$ and $Q$ over $G$ returns a set of pairs $A = \{\langle p, q\rangle\}$ such that: (1) $\langle p, q\rangle \in P \times Q$. (2) $\forall\langle p, q\rangle \in A, d_n(p, q) \leq d_n(r, s), \forall\langle r, s\rangle \in \{(P \times Q) \setminus A\}$. and (3) $|A| = k$; where $d_n(v_i, v_j)$, denotes the length of the shortest network path

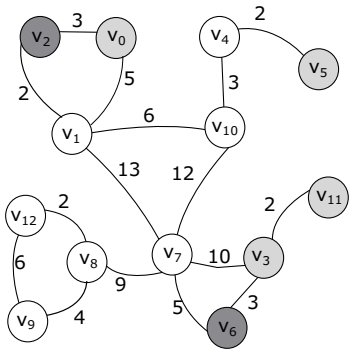



Figure 5.1: A sample road network

between $v_i$ and $v_j$ in graph $G$.

In this chapter, we first present a new hierarchical road network partitioning structure, called $G^*$-tree, designed to support efficient processing of k-CPQs on road networks. The $G^*$-tree can be seen as being similar to the $G$-tree [70], in the sense that both rely on the *multi-level graph partitioning* approach [33], which is constructed by recursively partitioning the road network into smaller sub-graphs. There is, however, a fundamental difference between the two. The optimal $G$-tree aims at generating sub-graphs of approximately same sizes with the goal of minimizing the number of border nodes, whereas the $G^*$-tree aims at maximizing the *minimum network distance* between sub-graphs in order to boost the pruning power of the tree for more efficient processing of $k$-CPQs.

Based on the $G^*$-tree hierarchical road network partitioning structure, we propose two different approaches for processing $k$-CPQs on road networks. The first one traverses the $G^*$-tree based on a *top-down* traversal method. The basic idea of this algorithm, is to exploit the hierarchical properties of $G^*$-tree and use a modified best-first search strategy to find the pairs of data points that have the $k$ smallest pair-wise network distances. The second approach looks for the $k$-closest pairs of POIS by applying a bottom-up traversal paradigm. First the leaf nodes of the $G^*$-tree are explored in order to obtain local optimal answer sets. Next, the $G^*$-tree is traversed bottom-up, where the final optimal list is gradually obtained by joining sibling sub-graphs. Both of proposed approaches employ aggressive pruning strategies based on *minimum network distances* between different sub-graphs, which is in fact the

main driving factor when building the $G^*$-tree.

## 5.2 Related Work

The existing approaches for processing of $k$-CPQs can be categorized into two groups. One studied the $k$-CPQs in Euclidean spaces [17, 18, 28, 46, 57, 62, 67, 56]. The other has investigated the $k$-CPQs in general metric spaces [20, 30, 43, 58, 36, 21]. To the best of our knowledge there has been no published work in the context of road networks, which is our main focus.

Assuming that the sets of POIs $P$ and $Q$ have been indexed in separate $R^*$-trees, [17, 18] exploit the hierarchical properties of the $R^*$-trees for boosting the pruning capability. Plane-sweep and access ordering techniques are also applied for further improving query efficiency. For processing of distance join queries [56, 57] propose the use of bidirectional node expansion and plane-sweep techniques for quickly pruning distant pairs, where the plane-sweep is further optimized by novel strategies for selecting a good sweeping axis and direction. Yang and Lin [67] proposed a new indexing structure, bRdnn-tree, for processing of $k$-CPQs in Euclidean spaces. The bRdnn-tree keeps track for each data point in $P$ its nearest neighbour in data set $Q$ for efficient processing the $k$-CPQs. The main shortcoming of this approach is its limited applicability, since for constructing the bRdnn-tree on $P$ as a sample data set, we need to discover the nearest neighbour of each POI in $P$ towards all data sets.

The authors of [36] proposed a divide-and-conquer based approach called *Adaptive Multi-Partitioning* approach. This approach uses the multi-ball partitioning strategy, where its basic idea is the iterative reduction of upper-bounds for the pair-wise distance of $k^{th}$ closest pair. Unfortunately, its efficiency degrades quickly with the size of the data sets. More recently, in [21] the authors proposed three approaches based on depth-first, best-first, and a combination thereof, traversal paradigms for processing of $k$-CPQs in general metric spaces using an $M$-tree [44].

CPQs are also highly connected to similarity joins queries. Similarity join algorithms find pairs of objects that lie within a certain distance $\epsilon$ of each other. Jacox

and Samet [30] presented a comprehensive overview of different approaches. The basic idea of the proposed solution in [43] for solving the similarity join problem is to index the two different data sets jointly in a single data structure as list of twin clusters, a metric index specially focused on the similarity join problem. The basic idea behind the proposed approach in [30], termed $Quickjoin$, is based on the Quicksort algorithm in that it recursively partitions the data into smaller partitions. This approach was further improved in [20]. In [19, 64] the similarity joins problem has been studied based on Map-Reduce approach, that partitions the data space based on the underlying data distribution. The similarity join solutions cannot handle $k$-CPQs problems efficiently, due to difficulty in choosing a proper value for $\epsilon$; a small value may lead to incomplete results and a large value may incur in too much overhead [21].

## 5.3 The G*-tree

In order to process $k$-CPQ queries efficiently we propose the $G^*$-tree, a new hierarchical road network partitioning structure particularly designed for this type of query. The $G^*$-tree is inspired on, but built on quite different premises than, the $G$-tree proposed recently in [70]. In the following, we briefly review the $G$-tree before presenting the details of the $G^*$-tree. (We discuss the differences between the two indices later in Section 5.3.2.)

The $G$-tree of a graph $G = \langle V, E \rangle$ is a balanced search tree, which is constructed based on the idea of *Multi-Level Graph Partitioning* approach [33]. In the $G$-tree, each node represents a sub-graph. The root node corresponds to the complete graph $G$, and each non-leaf node has $f \geq 2$ children and each leaf node contains $\tau \geq 2$ vertices. The idea is to recursively partition the sub-graph of a node into $f$ equally sized sub-graphs, each leading to one new child node, until such child nodes have no more than $\tau$ vertices. At the end this will lead a balanced search tree. For more details about the $G$-tree construction procedure, we refer the interested reader to [70].

### 5.3.1 $G^*$-tree construction procedure

For a given graph $G = \langle V, E \rangle$, the $G^*$-tree for $G$ is constructed by hierarchically partitioning $G$ into a set of *sub-graphs*, where each sub-graph corresponds to a node in $G^*$-tree.

The $G^*$-tree construction procedure contains two major phases: *coarsening* and *tree formation*. In both of these phases, two sub-graphs are iteratively selected and *collapsed*. Before discussing these two phases in details, we first discuss how such collapsing works.

Consider two sub-graphs $G_i = \langle V_i, E_i \rangle$ and $G_j = \langle V_j, E_j \rangle$. For the sake of discussion, we consider sub-graphs as special nodes in the network and we denote such nodes as $G_i$ and $G_j$, respectively. The collapsing operation on $G_i$ and $G_j$ yields a new sub-graph $G_z = \langle V_z, E_z \rangle$ of $G$, where $V_z = V_i \cup V_j$ and $E_z \subseteq E_i \cup E_j$. The edges incident on $G_z$ is set to the union of the edges incident to $G_i$ and $G_j$. For instance, considering the graph Figure 5.1, collapsing nodes $v_3$ and $v_{11}$ (note that a node itself is by definition a sub-graph) leads to a new sub-graph $G_0$ in the (partially) coarsened graph shown in Figure 5.2(a). Furthermore, if the new sub-graph $G_k$ is directly connected to the two sub-graphs $G_i$ and $G_j$, then the weight of the edge connecting $G_k$ and $G_z$ is set to the minimum of the weights of all edges connecting $G_k$ and $G_i$ as well as connecting $G_k$ and $G_j$, i.e., $w(G_z, G_k) = \min\{w(G_i, G_k), w(G_j, G_k)\}$. This is exemplified in Figure 5.2(b) which shows that sub-graph $G_1$ resulted from the collapsing of $v_1$ and $v_2$ in Figure 5.2(a). Since vertex $v_0$ is a direct neighbour of both vertices $v_2$ and $v_1$, we have $w(G_1, v_0) = \min\{w(v_0, v_2), w(v_0, v_1)\} = 3$.

**Definition 5.3.1.** Let us assume $G_i$ and $G_j$ are two neighbour sub-graphs of graph $G'$, which are selected to be collapsed. Then $w(G_i, G_j)$ in graph $G'$ is called the collapsing distance of $G_i$ and $G_j$.

With this in place, we can proceed to discuss the first phase of the $G^*$-tree construction, the *coarsening phase*. This phase produces the leaf-nodes of $G^*$-tree, where, step by step, two sub-graphs are selected and collapsed.

Choosing which sub-graphs to collapse and when to stop doing so, is an impor-

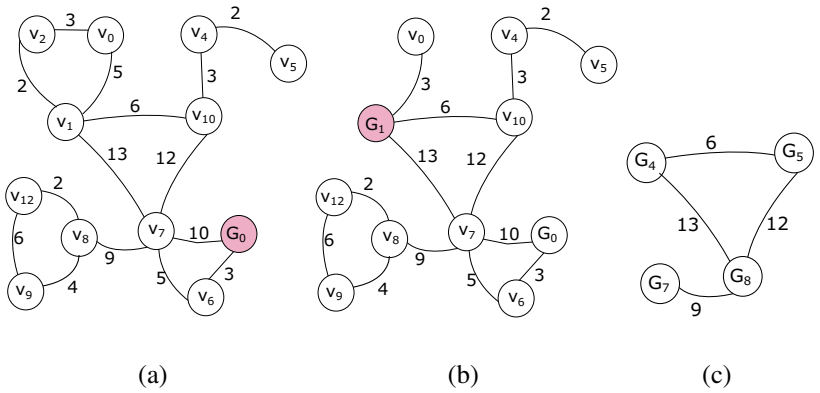



Figure 5.2: Sample steps of the coarsening phase for the graph in Figure 5.1.

tant issue. In this work, we use the idea of the *Light Edge Matching* (LEM) approach [34], in which two sub-graphs containing the smallest edge weight among all are selected to be collapsed. Therefore, by construction, the collapsing distance (Definition 5.3.1) minimizes the distance between two sub-graphs. In the coarsening phase the road network is gradually coarsened with the goal of maximizing the total edge weight of the coarsened graph for boosting the pruning capability based on the minimum network distance between sub-graphs. The coarsening phase will terminate when the number of sub-graphs in the coarsened graph is no more than $\lambda$, and at this point the remaining sub-graphs will become the *leaf nodes* of the $G^*$-tree associated with the original graph $G$. (In Section 5.6 we discuss the effect of this parameter on the query processing efficiency in terms of query response time and number of examined data points.) For instance Figures 5.2(a-b) illustrate the first two steps of coarsening phase, where, assuming $\lambda = 4$, Figure 5.2(c) shows the final result of coarsening phase for the example of road network in Figure 5.1.

In the second phase of the $G^*$-tree construction procedure, the *tree formation phase*, the $G^*$-tree as a hierarchical partitioning of the given road network is constructed, bottom-up, starting with the leaf-nodes produced in the coarsening phase. This phase is fundamentally similar to coarsening phase: in an iterative manner, and again based on the LEM approach, two sub-graphs are selected to be collapsed, where the new formed sub-graph will be the parent node of the collapsed sub-graphs. Assuming the coarsened graph in Figure 5.2(c), then in the first step of

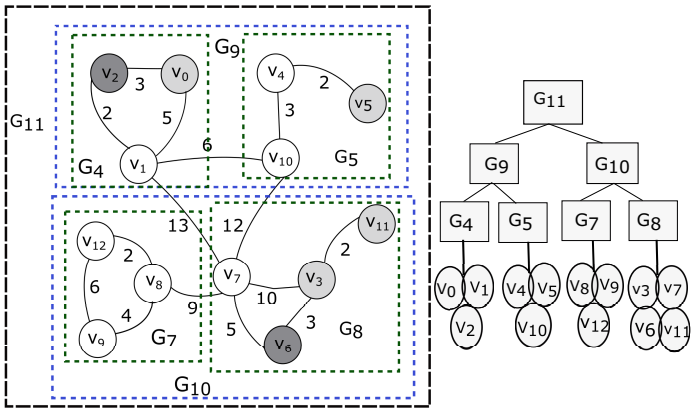



Figure 5.3: The hierarchical graph partitioning of example of Figure 5.1

*tree formation phase* the sub-graph $G_9$ is formed from collapsing sub-graphs $G_4$ and $G_5$ in the corresponding $G^*$-tree. The $G^*$-tree node corresponding to $G_9$ will be the parent of nodes $G_4$ and $G_5$. Then, in the second step of this phase, the sub-graphs $G_7$ and $G_8$ will be collapsed, where the new formed sub-graph $G_{10}$ will be considered the parent node of $G_7$ and $G_8$. As the final step, the sub-graph $G_{11}$ is formed by collapsing $G_9$ and $G_{10}$. Having no more sub-graphs to be collapsed, this phase is concluded and the single remaining node is the root node of the $G^*$. Figure 5.3 shows the corresponding $G^*$-tree and hierarchical road network partitioning for the given road network in Figure 5.1.

There are two important concepts pertaining to the $G^*$-tree that need to be defined as they are used throughout our proposed solutions.

**Definition 5.3.2.** Given a node $G_i = \langle V_i, E_i \rangle$ of $G = \langle V, E \rangle$, a vertex $u \in V_i$ is a border node if $\exists (u, v) \in E$ where $v \notin V_i$. We denote the set of border nodes of sub-graph $G_i$ as $\beta(G_i)$.

**Definition 5.3.3.** The occurrence point list of node $G_i$ with respect to data set $P$ ($Q$), denoted by $\Gamma^P_{point}(G_i)$ ($\Gamma^Q_{point}(G_i)$), contains the identifiers of all POIs in $P$ ($Q$) appearing in $G_i$ as one of its vertices.

For instance, the border nodes set of sub-graphs $G_4$ and $G_5$ are defined as $\beta(G_4) = \{v_1\}$, $\beta(G_5) = \{v_{10}\}$. Furthermore, for example the *occurrence point list* of $G_4$ with respect to sets $P = \{v_2, v_6\}$ and $Q = \{v_0, v_3, v_5, v_{11}\}$ are defined as

$\Gamma^{P}_{point}(G_4) = \{v_2\}, \Gamma^{Q}_{point}(G_4) = \{v_0\}$.

### 5.3.2 G-tree vs. G*-tree

Although both $G$ and $G^*$-trees represent a hierarchical partitioning of the given road network, these two structures are different in terms of: construction procedure, balanced/unbalanced characteristic, equal sized/unequal sized leaf nodes and indexing parameters. In the following we discuss each of these differences with more details:

**Construction procedure.** While the goal of the $G$-tree indexing structure is to represent a multi-level graph partitioning with the minimum number of border nodes (Definition 5.3.2), on the other hand, the $G^*$-tree is constructed with the goal of maximizing the minimum network distance between sub-graphs. This difference in goals can be specified by the difference in the corresponding construction procedure of these two structures. On the one hand, in each step of coarsening phase in the $G^*$-tree construction procedure, the two sub-graphs/vertices containing the minimum edge weight are selected to be collapsed so as to maximize the total edge weight of the coarsened graph. On the other hand, in the $G$-tree construction procedure, the road network graph is hierarchically partitioned into a set of smaller sub-graphs in a way that minimizes the total number of border nodes.

**Balanced/Unbalanced characteristic.** The $G$-tree indexing structure is considered a *balanced* search tree. This property comes from this fact that, in the partitioning phase of the $G$-tree construction procedure, each sub-graph $G_i$ is divided into approximately equal sized sub-graphs. On the other hand, there is no guarantee for balanced characteristic of the $G^*$-tree, since as aforementioned the corresponding tree structure is constructed only based on the LEM approach [34].

**Equal sized/Unequal sized leaf nodes.** As mentioned before, in the $G$-tree construction procedure each sub-graph $G_i$ is partitioned into equal-sized smaller sub-graphs, resulting approximately equal sized leaf-nodes. On the other hand, there is no guarantee for this characteristic in the $G^*$-tree. The reason is that, in the coars-

ening phase in the $G^*$-tree construction procedure, the collapsing operation takes place regardless of the size of sub-graphs. When the number of sub-graphs reaches to the predefined value $\lambda$, the remaining (possibly unequal sized) sub-graphs will be considered the leaf nodes.

**Indexing parameters.** While $G$-tree indexing structure requires two parameters, indicating the fanout of each non-leaf-node and the size of leaf-nodes respectively. The $G^*$-tree requires only one parameter as $\lambda$ indicating the number of leaf-nodes.

### 5.3.3 Distance Computations in $G^*$-Tree

Given the corresponding $G^*$-tree of a road network, two sets of distance information are pre-computed: (i) the intra-distance matrix for each leaf-node, containing the minimum network distance from each vertex $v$ to each border node of the leaf-node that contains $v$, and (ii) the inter-distance matrix, indicating the pairwise minimum network distance between each pair of border nodes of any two different leaf-nodes of $G^*$-tree.

Considering the $G^*$-tree in Figure 5.3, Figure 5.4(a) illustrates the corresponding intra-distance matrix of leaf-node $G_4$, that is the minimum network distance between all vertices in $G_4$ towards the only border node of $G_4$, i.e., $v_1$. Figure 5.4(b) illustrates the corresponding inter-distance matrix of the same $G^*$-tree, where the pairwise minimum network distance of all pairs of border nodes of different leaf-nodes is pre-computed. (Note that the inter-distance matrix is by definition symmetrical and with a null diagonal.)

Given the $G^*$-tree of a road network, for computing of *minimum network distance* between $v_i$ and $v_j$, we consider two cases. In the first case, both $v_i$ and $v_j$ are within the same leaf-node, where $d_n(v_i, v_j)$ is computed by applying the traditional Dijkstra algorithm. In the case where $v_i$ and $v_j$ belong to two different leaf-nodes $G_i$ and $G_j$, then $d_n(v_i, v_j)$ is computed based on a dynamic programming approach, according to [21], as follows:

$$\min_{\forall b_i \in \beta(G_i), b_j \in \beta(G_j)} \{d_n(v_i, b_i) + d_n(b_i, b_j) + d_n(b_j, v_j)\} \tag{5.1}$$

| $G_4$ | $v_0$ | $v_1$ | $v_2$ |
|---|---|---|---|
| $v_1$ | 5 | 0 | 2 |

(a)

| $G^*$ | $v_1$ | $v_7$ | $v_8$ | $v_{10}$ |
|---|---|---|---|---|
| $v_1$ | 0 | 13 | 22 | 6 |
| $v_7$ | 13 | 0 | 9 | 12 |
| $v_8$ | 22 | 9 | 0 | 21 |
| $v_{10}$ | 25 | 12 | 21 | 0 |

(b)

Figure 5.4: (a) The intra-distance matrix of sub-graph $G_4$, (b) the corresponding inter distance matrix for $G^*$-tree illustrated in Figure 5.3

Note that in Eq. 5.1, all three elements $d_n(v_i, b_i)$, $d_n(b_i, b_j)$ and $d_n(b_j, v_j)$ are considered pre-computed distances which are extracted from corresponding intra-distance and inter-distance matrices.

## 5.4 Top-Down Traversal (TDT)

The basic idea of the TDT approach is to exploit the hierarchical properties of the $G^*$-tree and use a modified *best-first search algorithm* for processing $k$-CPQs on road networks. One of these properties is the *minimum sub-graph distance* (Definition 5.4.1) between each pair of nodes (i.e., sub-graphs) of the $G^*$, which is used for pruning non-promising pairs of nodes of $G^*$-tree. The intuition behind this approach is that the pairs of nodes with small minimum sub-graph distance have a higher chance of containing pairs of data points belonging to the answer set. Alternatively, nodes that are far away from each other are less likely to contain closest pairs. In fact, this is exactly what allows one to do effective pruning of candidate nodes (Lemma 5.4.1).

**Definition 5.4.1.** The *minimum sub-graph distance* between sub-graphs $G_i$ and $G_j$, denoted by $d_{min}(G_i, G_j)$, is defined as:

$$d_{min}(G_i, G_j) = \min_{\forall b_i \in \beta(G_i), b_j \in \beta(G_j)} \{d_n(b_i, b_j)\} \tag{5.2}$$

**Lemma 5.4.1.** Let $mnd_k$ be the *minimum network distance* of the $k^{th}$-closest pair discovered so far. Then, any pair of sub-graphs $(G_i, G_j)$ in $G^*$ where $mnd_k <$

**Algorithm 10:** TDT Approach

**Input:** $G^*$: $G^*$-tree
**Output:** $k$-CPs on $P$ and $Q$

$H_T \leftarrow \varnothing$
$H_R \leftarrow \varnothing$
$mnd_k \leftarrow \infty$
$H_T$.push($\langle (G^*_{root}, G^*_{root}), d_{min}(G^*_{root}, G^*_{root}) \rangle$)
**while** $H_T$ *is not empty* **do**
    $\langle (G_i, G_j), d_{min}(G_i, G_j) \rangle \leftarrow H_T.pop()$
    **if** $d_{min}(G_i, G_j) > mnd_k$ **then**
        break  //Lemma 5.4.1
    **if** $G_i, G_j$ *are both leaf-nodes* **then**
        **for** $\forall p_i \in \Gamma^P_{point}(G_i)$ **do**
            **for** $\forall q_j \in \Gamma^Q_{point}(G_j)$ **do**
                $H_R.push\ (\langle (p_i, q_j), d_n(p_i, q_j) \rangle)$
        **for** $\forall q_i \in \Gamma^Q_{point}(G_i)$ **do**
            **for** $\forall p_j \in \Gamma^P_{point}(G_j)$ **do**
                $H_R.push\ (\langle (p_j, q_i), d_n(p_j, q_i) \rangle)$
        update the $mnd_k$
    **else**
        $W \leftarrow cross_{sets}(G_i, G_j)$
        **for** *each* $(G'_i, G'_j) \in W$ **do**
            **if** $(|\Gamma^P_{point}(G'_i)| > 0 \ \ \& \ |\Gamma^Q_{point}(G'_j)| > 0)$ *OR* $(|\Gamma^P_{point}(G'_j)| > 0 \ \ \& \ |\Gamma^Q_{point}(G'_i)| > 0)$ **then**
                $H_T$.push($\langle (G'_i, G'_j), d_{min}(G'_i, G'_j) \rangle$)
return the top $k$ pairs in $H_R$

$d_{min}(G_i, G_j)$, cannot result in forming a pair of vertices $(v_i, v_j)$, $v_i \in G_i$ and $v_j \in G_j$, such $d_n(v_i, v_j) \leq mnd_k$.

*Proof.* Any shortest path between $v_i$ and $v_j$ must contain two border nodes such as $b_i \in \beta(G_i)$ and $b_j \in \beta(G_j)$, where $d_{min}(G_i, G_j) \leq d_n(b_i, b_j) \leq d_n(v_i, v_j)$. Based on the assumption of the lemma, we have $mnd_k < d_{min}(G_i, G_j)$, resulting in $mnd_k < d_n(v_i, v_j)$. □

The pseudo-code for the TDT approach is presented on Algorithm 10. In the first step, TDT approach initializes two min-heaps, $H_T$ and $H_R$. The elements

of $H_T$, are pairs in the form of $\langle (G_i, G_j), d_{min}(G_i, G_j) \rangle$ and are ordered by their *minimum sub-graph distance* $d_{min}(.,.)$. On the other hand, the elements of $H_R$ are pairs in the form of $\langle (v_i, v_j), d_n(v_i, v_j) \rangle$ ordered according to their *minimum network distance* $d_n(.,.)$. In Line 4, the algorithm pushes the first element of $H_T$ as $\langle (G^*_{root}, G^*_{root}), 0 \rangle$, where $G^*_{root}$ is the root node of $G^*$-tree. In line 6, TDT iteratively pops the top element of $H_T$ as $\langle (G_i, G_j) \rangle$, and handles it according to whether elements of $G_i$ and $G_j$ are leaf or non-leaf nodes. At this point Lemma 5.4.1 can be used to prune non-promising nodes, i.e., nodes that cannot lead to a solution better than the ones already obtained. In the case that both $G_i$ and $G_j$ are leaf nodes, then all possible pairs of data points $(p_i, q_j)$, as illustrated in lines 10-15, are pushed into $H_R$. In the case that at least one of nodes $G_i$ and $G_j$ is a non-leaf node, then all possible combinations of $G_i$ and $G_j$'s children (determined by $cross_{sets}(G_i, G_j)$) are pushed into $H_T$, lines 18-21. Note that in Algorithm 10 before putting a pair of nodes into $H_T$, we should also check the corresponding *occurrence point lists* of nodes for pruning non-promising pairs of sub-graphs, line 20.

We illustrate the TDT approach functioning by using the network of Figure 5.1 in order to answer a 2-CPQ. Table 5.1 depicts the values stored in the heaps $H_T$ and $H_R$ in each step of the algorithm. In step 1, the first element $\langle (G_{11}, G_{11}), 0 \rangle$ is pushed into $H_T$. In step 2, $\langle (G_{11}, G_{11}), 0 \rangle$ is fetched from $H_T$, and then the different combinations of $G_{11}$'s children accompanied with their $d_{min}$ are pushed into $H_T$. In Step 3, the pair of sub-graphs $\langle (G_9, G_9), 0 \rangle$ will be popped from $H_T$ and all possible combinations of $G_9$'s children will be pushed into $H_T$, by considering the corresponding *occurrence point lists*. For example, since $\Gamma^P_{point}(G_5) = \varnothing$, the pair of sub-graphs $(G_5, G_5)$ will not be added into $H_T$. In step 4, the element $\langle (G_4, G_4), 0 \rangle$ will be popped from $H_T$. Since, both elements in this pair are leaf-nodes, then all possible pairs of data points from $P$ and $Q$ belonging to sub-graph $G_4$ will be pushed into $H_R$. In Step 5, the element $\langle (G_{10}, G_{10}), 0 \rangle$ will be popped, where among all possible combinations of $G_{10}$'s children the pair of sub-graphs $(G_8, G_8)$ will be pushed into $H_T$. In step 6, first, the pair of sub-graphs $(G_8, G_8)$ will be popped from $H_T$, and two elements $\langle (v_6, v_3), 3 \rangle$ and $\langle (v_6, v_{11}), 5 \rangle$ will be added into $H_R$, and $mnd_k$ will be updated as 3. In Step 7, the algorithm terminates,

Table 5.1: TDT approach for 2-CPQ of example of Figure 5.1

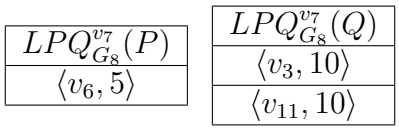
| **Step** | $H_T$ | $H_R$ |
|---|---|---|
| $Step1$ | $< (G_{11}, G_{11}), 0 >$ | |
| $Step2$ | $< (G_9, G_9), 0 >$,$< (G_{10}, G_{10}), 0 >$, $< (G_9, G_{10}), 12 >$ | |
| $Step3$ | $< (G_4, G_4), 0 >$,$< (G_{10}, G_{10}), 0 >$, $< (G_4, G_5), 6 >$,$< (G_9, G_{10}), 12 >$, | |
| $Step4$ | $< (G_{10}, G_{10}), 0 >$,$< (G_4, G_5), 6 >$, $< (G_9, G_{10}), 12 >$ | $< (v_2, v_0), 3 >$ |
| $Step5$ | $< (G_8, G_8), 0 >$,$< (G_4, G_5), 6 >$, $< (G_9, G_{10}), 12 >$ | $< (v_2, v_0), 3 >$ |
| $Step6$ | $< (G_4, G_5), 6 >$,$< (G_9, G_{10}), 12 >$, | $< (v_2, v_0), 3 >$, $< (v_6, v_3), 3 >$, $< (v_6, v_{11}), 5 >$ |
| $Step7$ | $< (G_4, G_5), 6 >$,$< (G_9, G_{10}), 12 >$ | $< (v_2, v_0), 3 >$, $< (v_6, v_3), 3 >$, $< (v_6, v_{11}), 5 >$ |

since the smallest *minimum sub-graph distance* in the $H_T$, belonging to the element $\langle (G_4, G_5), 6 \rangle$ is greater than $mnd_k$.

Note that TDT technique can also be applied on top of $G$-tree indexing, where Lemma 5.4.1 can be used for pruning the non-promising pairs of sub-graphs. Meanwhile, considering this point that the $G^*$ tree is constructed based on maximizing the minimum network distance between sub-graphs, implementing TDT on $G^*$-tree is inherently a more efficient alternative in comparison to the $G$-tree based on the applied pruning strategy (Lemma 5.4.1).

## 5.5 Bottom-Up Sub-graph Joining (BUSJ)

The basic idea of the BUSJ technique is to traverse the $G^*$-tree based on a *bottom-up* traversal paradigm for discovering the $k$-closest pairs. At the first, all leaf-nodes in the $G^*$-tree are explored for discovering the *local closest pairs*, in which the elements of each pair belong to same leaf node. Then, iteratively based on the order of collapsing of sub-graphs in the *tree formation phase* (in the construction procedure of $G^*$-tree), two sibling sub-graphs are selected to be explored together, aiming at finding possible closest pairs where the data points belong to different sibling sub-graphs.




(a) (b)

Figure 5.5: The LPQs for border node $v_7$ towards sub-graph $G_8$

### 5.5.1 Pre-Computations in BUSJ Approach

As mentioned before, the TDT approach requires distance pre-computations between each pair of border nodes. In comparison to the TDT approach, the BUSJ technique requires another extra group of pre-computations. It is assumed that each border node $b_i \in \beta(G_i)$ of sub-graph $G_i$ has one *Local Priority Queue* (LPQ) for each data set. In the following $LPQ_{G_i}^{b_i}(P)$ denotes the LPQ of border node $b_i$ towards sub-graph $G_i = \langle V_i, E_i\rangle$ and data set $P$. It is a min-heap with entries in the form $\langle v, d_n(v, b_i)\rangle$ where $v$ is a node in the same sub-graph as border node $b_i$. Finally, it is ordered by the minimum network distance between those two nodes. The $LPQ_{G_i}^{b_i}(Q)$ is defined in the similar way. The LPQs are used while investigating the possible closest pairs, in which elements of each pair belong to two different sibling sub-graphs. Figure 5.5 illustrates the corresponding LPQs for border vertex $v_7$ towards sub-graph $G_8$ and data sets $P = \{v_2, v_6\}$ and $Q = \{v_0, v_3, v_5, v_{11}\}$ for example of Figure 5.3.

### 5.5.2 Algorithm Overview

Algorithm 11 presents BUSJ's pseudocode. It contains two main steps as: (i) all leaf-nodes of $G^*$-tree are traversed for discovering the $k$-*Local Closest Pairs* (Lines 3-6), and (ii) the sibling sub-graphs in the $G^*$-tree are *joined* based on the order in which the sub-graphs were collapsed in the *tree-formation phase* (Lines 7-18). In the following, we discuss BUSJ technique with more details step by step.

Finding the $k$-*Local Closest Pairs* (k-LCPs) in a node based on traditional Dijkstra algorithm is a relatively simple task, and thus we omit algorithmic details. The goal is to search, within each leaf node (leaf sub-graph $G_i$) of the $G^*$-tree for the $k$

**Algorithm 11:** BUSJ Approach

**Input:** $G^*$: $G^*$-tree,
$Q_S$: a queue indicating the order of joining of sub-graphs
**Output:** $k$-CPs on $P$ and $Q$

```
1  H_R ← ∅
2  mnd_k ← ∞//the pairwise network distance of k^th-CP
3  for ∀G_i as a leaf-node of G* do
4      if |Γ^P_point(G_i)| > 0 & |Γ^Q_point(G_i)| > 0 then
5          H_R.enqueue(LCPs(k, G_i))
6          update mnd_k
7  while Q_S is not empty do
8      < (G_i, G_j), d_c(G_i, G_j) > ← Q_S.pop()
9      if d_c(G_i, G_j) > mnd_k then
10         break // Lemma 5.5.1
11     else
12         if |Γ^P_point(G_i)| > 0 & |Γ^Q_point(G_j)| > 0 then
13             H_R.push(G_i(P) ⋈^k_mnd_k G_j(Q)) // Alg. 12
14             update mnd_k based on H_R
15         if |Γ^P_point(G_j)| > 0 & |Γ^Q_point(G_i)| > 0 then
16             H_R.push(G_j(P) ⋈^k_mnd_k G_i(Q)) // Alg. 12
17             update mnd_k based on H_R
18     l ← l + 1
19 return the top k pairs in H_R
```

closest pairs of points $(p_i, q_i)$, $p_i \in \Gamma^P_{point}(G_i)$ and $q_i \in \Gamma^Q_{point}(G_i)$. Note that it is possible that there are $m < k$ such pairs within $G_i$; in such a case all $m$ are returned. When all leaf-nodes of $G^*$-tree are explored then the second phase is triggered.

Now, by applying a *bottom-up traversal paradigm*, we examine the candidate pairs in which elements of each pair belong to different leaf-nodes in $G^*$-tree. Such pairs are generated by *joining sibling sub-graphs* in $G^*$-tree, based on the order of collapsing of sub-graphs in the *tree-formation phase* of $G^*$-tree construction procedure. Let us assume $Q_S$ is a queue, in which different pairs of sibling sub-graphs accompanied with their corresponding *collapsing distance* (Definition 5.3.1) have been stored in the form of $\langle (G_i, G_j), d_c(G_i, G_j) \langle$. The intuition is that the sibling sub-graphs with smaller collapsing distance have higher chance for containing pairs of data points belonging to the answer set. As an example, the corresponding $Q_S$

for example of Figure 5.3 has been illustrated in Figure 5.6. Since in the tree formation phase, the sub-graphs $G_4$ and $G_5$ are the first pair that were collapsed with collapsing distance as $d_c(G_4, G_5)$=6, then the element $< (G_4, G_5), 6 >$ has been placed in the front of $Q_S$.

$\rightarrow$ | $<(G_9,G_{10}),12>$ | $<(G_7,G_8),9>$ | $<(G_4,G_5),6>$ | $\rightarrow$ front

Figure 5.6: The order of joining of the sub-graphs in Figure 5.3

In order to join two sub-graphs $G_i$ and $G_j$, denoted by $G_i \bowtie^k_{mnd_k} G_j$, we break down this problem into two smaller sub-problems $G_i(P) \bowtie^k_{mnd_k} G_j(Q)$ and $G_j(P) \bowtie^k_{mnd_k} G_i(Q)$. $G_i(P) \bowtie^k_{mnd_k} G_j(Q)$ looks for a set of at most $m$ pairs, which have the $m$ smallest minimum network distances among all pairs of points that can be formed by choosing points $p_i$ and $q_i$ from sub-graphs $G_i$ and $G_j$, respectively, where $m$ is defined as $m \leq \min\{k, |\Gamma^P_{point}(G_i)| \times |\Gamma^Q_{point}(G_j)|\}$, i.e., the minimum between the sought $k$ or the actual number of possible pairs. $G_j(P) \bowtie^k_{mnd_k} G_i(Q)$ is defined in the similar way.

Algorithm 12 represents the corresponding pseudocode of $G_i(P) \bowtie^k_{mnd_k} G_j(Q)$. For each pair of border vertices $(b_i, b_j)$, $b_i \in \beta(G_i), b_j \in \beta(G_j)$, we iteratively form candidate pairs of data points based on the corresponding LPQs of $b_i$ and $b_j$, i.e., $LPQ^{b_i}_{G_i}(P)$ and $LPQ^{b_j}_{G_j}(Q)$ respectively. In lines 6 and 11 of Algorithm 12, the data points $p_f$ and $q_h$ respectively indicate the data points from data sets $P$ and $Q$, that have the $f^{th}$ and $h^{th}$ smallest minimum network distance towards $b_i$ and $b_j$, respectively. We do not provide the algorithm of $G_j(P) \bowtie^k_{mnd_k} G_i(Q)$ due to space constraints and its similarity to $G_i(P) \bowtie^k_{mnd_k} G_j(Q)$.

Within BUSJ technique two sibling sub-graphs of $G^*$-tree are iteratively selected to be joined based on the order in which they were collapsed in tree formation phase in $G^*$-tree construction procedure. Fortunately we can apply effective pruning to reduce the search space and there the joining of sub-graphs. Lemma 5.5.1 states the condition under which the joining process of sub-graphs will be stopped, i.e., when the pair of sub-graphs in the front of $Q_S$ have their collapsing distance greater than $mnd_k$ (Algorithm 11, lines 9-10). Recall that, the collapsing distance between two sub-graphs $G_i$ and $G_j$ minimizes $w(G_i, G_j)$, i.e.,

**Algorithm 12:** $G_i(P) \bowtie^k_{mnd_k} G_j(Q)$

**Input:** $G_i, G_j, k, mnd_k$
**Output:** $H_R$

```
H_R ← ∅
for ∀b_i ∈ β(G_i) do
    for ∀b_j ∈ β(G_j) do
        f ← 1
        while f< min{k, Γ^P_point(G_i)} do
            p_f ← LPQ^{b_i}_{G_i}(P, f)
            if d_n(p_f, b_i) > mnd_k then
                break
            h ← 1
            while h< min{k, Γ^Q_point(G_j)} do
                q_h ← LPQ^{b_j}_{G_j}(Q, h)
                if d_n(b_j, q_h) > mnd_k then
                    break
                if d_n(p_f, b_i) + d_n(b_i, b_j) + d_n(b_j, q_h) < mnd_k then
                    H_R.push(< p_f, q_h >)
                h ← h + 1
            f ← f + 1
```

$d_c(G_i, G_j) = d_{min}(G_i, G_j)$.

**Lemma 5.5.1.** Let $mnd_k$ be the corresponding minimum network distance of $k^{th}$ closer pair discovered so far. And let us assume $(G_i, G_j)$ is a pair of sibling sub-graphs in the front of $Q_S$, where $d_c(G_i, G_j) > mnd_k$. Then, $G_i \bowtie^k_{mnd_k} G_j$ cannot result in forming a pairs of data points belonging to the optimal answer set, as well as, there would be no un-examined pairs of data points that can further minimize the $mnd_k$.

*Proof.* For any pair of data points such as $(p_i, q_j) \in G_i \bowtie^k_{mnd_k} G_j$ we have $d_c(G_i, G_j) = d_{min}(G_i, G_j) \leq d_n(p_i, q_j)$. On the other hand, assuming that $mnd_k < d_c(G_i, G_j)$, we can conclude that $mnd_k < d_n(p_i, q_j)$. This means that the pair of data points $(p_i, q_j)$ cannot be part of the answer set. Furthermore, considering that the pairs of sub-graphs in the $Q_S$ have been ordered with respect to their $d_c$, in which the pair of sub-graphs in the front of $Q_S$ as $(G_i, G_j)$ has the smallest $d_c$ among all

other elements of $Q_S$, then we can conclude that there would be no un-examined pairs of data points that can further minimize the $mnd_k$. □

Let us now illustrate the BUSJ approach through an example. Assuming the illustrated $G^*$-tree in Figure 5.3 for the network in Figure 5.1, then for discovering the 2-CPs on data sets $P$ and $Q$, in first step the leaf-nodes of $G^*$-tree as $G_4$, $G_5$, $G_7$ and $G_8$ will be explored separately returning a set of two local closest pairs as $\{(v_2, v_0)$,$(v_6, v_3)\}$, where $mnd_k = 3$. Then in the next step, according to Figure 5.6, initially the pair of sub-graphs in the front of $Q_S$ in the form of $< (G_4, G_5), 6 >$ will be extracted. Since this pair has the corresponding collapsing distance greater than $mnd_k$, then according to Lemma 5.5.1 the algorithm terminates, returning the answer set as $A = \{(v_2, v_0)$,$(v_6, v_3)\}$.

A final remark about the BUSJ is that while the discussion above assumed an underlying $G^*$-tree, it could be implemented on top of the G-tree as well, however, the pruning criteria (Lemma 5.5.1) could not be used. To be more specific, let us assume two sibling sub-graphs $G_i$ and $G_j$ with the parent node $G_k$, where $G_{k'}$ is the sibling of $G_k$ in the $G^*$-tree. Then according to $G^*$-tree construction procedure, we have $d_{min}(G_i, G_j) \leq d_{min}(G_k, G_{k'})$, in which Lemma 5.5.1 can be used as a pruning strategy while applying the BUSJ technique on top of $G^*$-tree. On the other hand, the $G$-tree indexing structure does not guarantee this condition. As a consequence, while implementing BUSJ technique on top of $G$-tree, the whole G-tree would have to be traversed, all the way to its root node. Clearly, the using $G^*$-tree instead is inherently a more efficient alternative.

## 5.6 Experiments

In this section we evaluate the performance of our proposed approaches, BUSJ and TDT techniques, through an extensive set of experiments. In order to show the superiority of the $G^*$-tree in comparison to $G$-tree in processing of $k$-CPQs, we compare the performance of the TDT approach based on both the $G$ and $G^*$ trees, where TDT on the $G$-tree is denoted by TDT-$G$. Following [70] we selected $\tau = 128$ and $f = 4$ as the fanout and the maximum number of vertices in leaf-

nodes for the $G$-tree, respectively. As discussed at the end of the previous section, there is no reason for one to believe that BUSJ would perform better on top of the G-tree than the $G^*$-tree and due to that we do not perform such a comparison. In the experiments, we used two different road networks; the California (CA) and Oldenburg (OL) road networks[1]. California's road network contains 21,047 vertices and 21,692 road segment edges, and Oldenburg's road networks has 6,104 vertices and 7,034 road segment edges. For each $k$-CPQ, the generated data sets $P$ and $Q$ are placed randomly on the road networks' vertices.

In the experiments, we study the effect of different parameters: (1) the sole and main parameter of $G^*$-tree, $\lambda$, indicating the number of leaf-nodes in the $G^*$-tree; (2) the number of requested closest pairs ($k$); (3) the cardinality of the data sets, and (4) the parameter $\gamma$ indicating the imbalancing factor as the cardinality of $P$ over $Q$, i.e., $\gamma = |P|/|Q|$. We measure the query processing time. Table 5.2 summarizes the values used for each parameter in our experiments and their default values. In the experiments, we change one of the parameters and set others as default values in order to verify the effect of each parameter. For each set of experiments, we execute 400 queries and show the average results.

Table 5.2: Parameter values in the experiments

| Parameter | Range | Default |
|---|---|---|
| Number of leaf-nodes ($\lambda$) | 2% – 32% | 14% |
| Cardinality of data sets | 1% – 12% | 4% |
| Number of required CPs ($k$) | 1 – 100 | 10 |
| The imbalancing factor ($\gamma$) | 1 – 5 | 1 |

In our preliminary experiments we used the Heap-Algorithm [17] as a loose baseline (since it was designed for the Euclidean space). Unfortunately it required at least two times more retrieved data points as well as response time in all experiments for a variety of settings. Therefore we do not plot the corresponding graphs of that algorithm in the figures that follow.

[1] `http://www.cs.fsu.edu/~lifeifei/SpatialDataset.htm`

### 5.6.1 Effect of Number of Leaf-Nodes

In the first set of experiments we investigate the effect of $\lambda$ on (1) the number of border nodes (Figure 5.7(a)), (2) the standard deviation of leaf-nodes sizes (Figure 5.7(b)), (3) the $G^*$-tree build time (Figure 5.7(c)) and (4) the response time (Figure 5.7(d)).

Note that the percentage of border nodes correlates with the amount of pre-computations overhead for calculating the distance between each pair of border nodes. A larger standard deviation of leaf-nodes sizes indicates that a larger number of data points are to be retrieved while examining the leaf-node sub-graphs. In this set of experiments, we make two observations. With the increase of $\lambda$, the percentage of border nodes increases linearly, and the variation of leaf-nodes sizes decreases exponentially. Although the ideal $\lambda$ is the one incurring a lower percentage of border nodes and also less variation of leaf-nodes sizes, our experimental results show that there is a trade-off between these two parameters that should be considered while selecting the parameter $\lambda$. On the one hand, when the parameter $\lambda$ increases, the higher percentage of border nodes incurs a higher load of distance pre-computations between each pair of border nodes. On the other hand, it requires a lower number of retrieved data points, because the deviation of leaf-nodes sizes from the average decreases exponentially with the increase of $\lambda$. Based on our experimental results in Figures 5.7(a) and 5.7(b), we selected $\lambda$ as 3000 and 800 (nearly 14% of the total number of vertices) the CA and OL road networks, respectively. This setting requires pairwise network distance pre-computations between nearly 28% of vertices as border nodes in both of road networks.

Figure 5.7(c) shows the effect of $\lambda$ on $G^*$-tree build time. This figure illustrates two groups of $G^*$-tree build time for each road network. Since BUSJ technique requires an extra group of pre-computations as construction of LPQs in comparison to TDT technique, we plot the $G^*$-tree build time for TDT and BUSJ techniques separately. As this figure shows, with the increase of $\lambda$ and subsequently with the increase of percentage of border nodes, there is a gap augmentation between $G^*$-tree build time for TDT and BUSJ techniques.

Figure 5.7(d) shows the impact of parameter $\lambda$ on the response time. As ex-

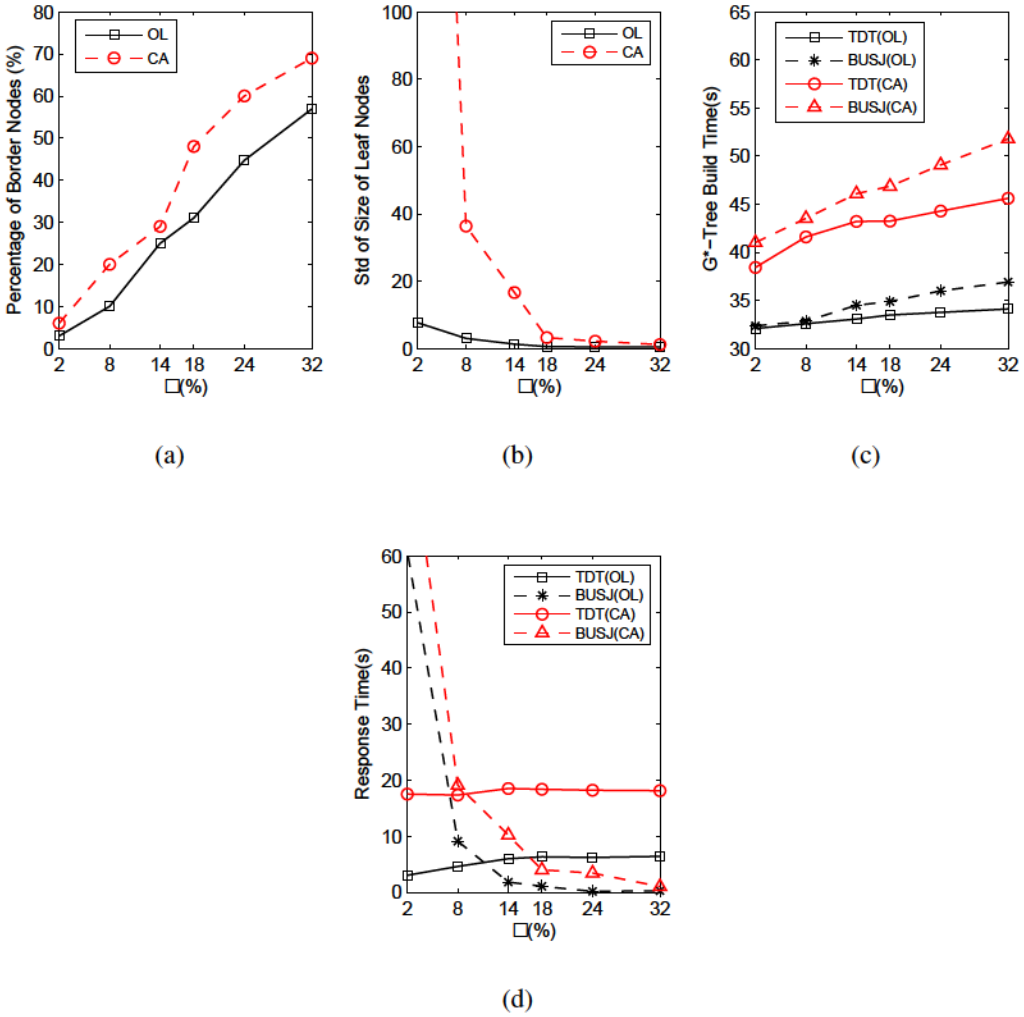



Figure 5.7: Effect of parameter $\lambda$

pected, when the parameter $\lambda$ increases, the BUSJ's response time drops exponentially due to reduction in leaf-nodes sizes. On the other hand, the TDT approach, in the OL road network, initially follows a steep rate when $\lambda$ increases from 2% up to 14%, but then, similar to CA road network the response time remains more stable with the further increase of $\lambda$. The main reason for the steady response time by the TDT approach with the change of $\lambda$ is the following. With the increase of $\lambda$ the size of leaf-nodes decreases, meaning that a lower number of points are supposed to be examined. On the other hand, it augments the height of the $G^*$-tree yielding a larger number of intermediate nodes in the tree. In other words, when $\lambda$ increases,

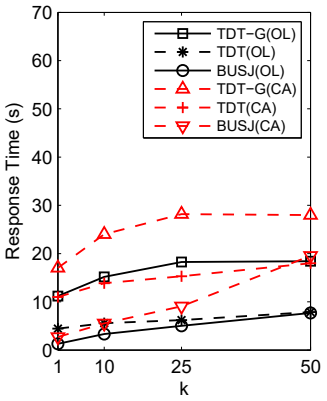



Figure 5.8: Effect of parameter $k$.

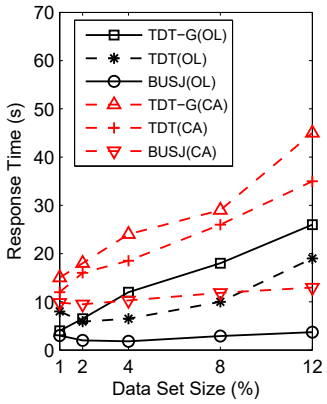



Figure 5.9: Effect of Data Set Size

the changes in the size of leaf-nodes and the height of tree balance the effect of each other in terms of response time.

### 5.6.2 Effect of Number of Required CPs

Figure 5.8 shows that the query response time in all approaches increases with the enlargement of $k$. The response time of the $G^*$-tree-based TDT grows at a slower pace than its competitors, where the fastest rate of increase of response time belongs to BUSJ technique. The results illustrate that for lower values of $k$, BUSJ shows a smaller response time in comparison to TDT technique, but with the increase of $k$ the response time of these two techniques converge towards each other. As well, the TDT approach based on $G^*$-tree outperforms the same technique based on $G$-tree in terms of response time for all settings of $k$. The main reason for the superiority of TDT approach on top of $G^*$-tree over the $G$-tree is that the $G^*$-tree provides a stronger capability for pruning non-promising pairs of sub-graphs and consequently shrinking the search space.

### 5.6.3 Effect of Dataset Size

Figure 5.9 plots the effect of varying the cardinality of data sets $P$ and $Q$ on the response time. The results indicate that although the response time of all approaches increases with the increase of data set size, the performance of the BUSJ approach increases at a slower pace in comparison to other competitors. The main reason for

why BUSJ outperforms the other two approaches is that on the one hand, with the increase of data sets sizes less number of vertices are traversed, since $mnd_k$ tends to be smaller when the cardinality of data set increases. On the other hand, TDT using both $G$ and $G^*$ requires examining a larger number of data points with the increase of data set size, since leaf-nodes become denser in both trees. Meanwhile, TDT technique outperforms TDT-$G$ in terms of query processing cost for all settings of data set sizes, due to $G^*$-tree provides a stronger pruning strategy in comparison to $G$-tree.

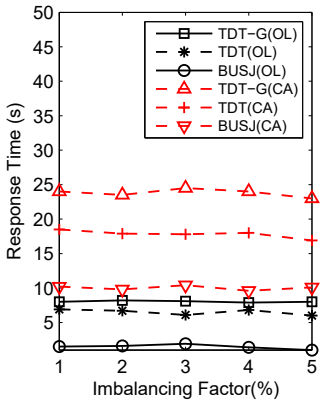



Figure 5.10: Effect of Parameter $\gamma$

### 5.6.4 Effect of Imbalancing Factor

In the last set of experiments we vary $\gamma$, i.e., the ratio of cardinality of $P$ over $Q$. Figure 5.10 illustrates our experimental results. Increasing $\gamma$ has two opposite effects on the performance of query processing cost of different approaches. On the one hand, when $\gamma$ increases, the search space becomes smaller, since the number of candidate pairs is reduced in the imbalanced data sets. On the other hand, the pairwise distances of the $k^{th}$ closest pair becomes larger, which decreases the pruning capability of different approaches and more nodes are examined. Because of these two opposite effects, as expected the response time of different approaches remains stable.

## 5.7 Conclusion

In this chapter, we investigated for the first time the $k$-CPQs problem in road networks. We proposed an efficient road network partitioning structure, $G^*$-tree, for efficient processing $k$-CPQs based on hierarchically partitioning road networks into smaller sub-graphs.

Using $G^*$-tree as a framework, we developed two approaches, named TDT and BUSJ, based on *top-down* and *bottom-up* traversal of $G^*$-tree, respectively. Both approaches make use of effective pruning, due to the way the underlying $G^*$-tree was constructed. Our experimental results, using real data sets and varying a number of parameters, show that BUSJ technique always incurs smaller query processing cost in comparison to TDT approach. This superiority is due to a set of pre-computations, specifically local priority queues for each border node in each sub-graph, which effectively orders the data points placing in the same sub-graph based on their distances from that border node. Comparing the performance of TDT over our proposed $G^*$-tree and the same algorithm using the state-of-the-art $G$-tree, shows that TDT over the $G^*$-tree is always more efficient. This shows the superiority of $G^*$-tree indexing structure by allowing the use of a stronger pruning strategy based on the maximizing the minimum network distance between sub-graphs, while hierarchically partitioning the road network into smaller sub-graphs.

# Chapter 6

# Conclusion

In this thesis, we studied an important class of *location based services* in spatial databases, namely *group trip planning queries*. For that, we investigate different trip planning queries: Sequenced Group Trip Planning Queries (SGTPQs), k-Optimal Meeting Points based on Preferred Paths (k-$OMP^3$), Best-Compromise In-Route Nearest Neighbor (BC-IRNN). We also investigated processing the k-Closest Pairs Queries (k-CPQs) in road networks.

For processing SGTPQs, we proposed the PGNE and IBS approaches. While the former applies a mixed breadth-depth first search strategies for exploring the search space, the latter traverses the POIs search space based on a depth-first search strategy, in which the optimal partial trips from different POIs are computed and reused when the algorithm proceeds. Both of aforementioned techniques apply efficient pruning strategies to shrink the search space. Our experimental results show that IBS is more scalable with respect to all parameters when the size of network increases.

As our second contribution in this thesis, we investigated (k-$OMP^3$) queries, finding the $k$ POIs which incur the $k$ smallest aggregate detour distances towards the given set of preferred path for a group users. We solved k-$OMP^3$queries for aggregate functions $sum$ and $max$. For processing k-$OMP^3$queries, we proposed efficient provably correct solutions as Multiple Ellipse Based Pruning (MEP) and Single Ellipse based Pruning (SEP). Both of these techniques shrink the search space based on geometrical properties of ellipses. The former first prunes the search space around each preferred path and then returns the intersection of all pruned

areas as the final refined area. On the other hand, SEP approach returns the final refined area as an ellipse where the focal points are placed on the centroid of source locations and destinations. Our experimental results show that, in the case that the aggregate function is $max$, MEP approach provides a stronger pruning strategy in comparison to SEP approach, which results in incurring faster response time. On the other hand, SEP approach is the most efficient and robust solution in terms of response time for processing k-$OMP^3$queries, in the case that the aggregate function is $sum$.

In the third phase of this research we investigated a combination of trip planning and path nearest neighbor queries as BC-IRNN queries. Given a preferred path for a user from source location towards a destination, and the COI that he/she is interested to visit, in BC-IRNN queries our goal is to find the linearly non-dominated paths from source location towards destination passing at least one POI based on two criteria, total traversed distance and detour distance from preferred path. For processing BC-IRNN queries we proposed a solution based on using suitable upper-bounds to both cost criteria to prune uninteresting paths. These upper-bounds are defined based on the travel distance of the path incurring the minimum detour distance and detour distance of the path which requires the minimum possible travel distance. Our experimental results on real datasets show that our proposed approach incurs in significantly faster query processing time when compared to a straightforward baseline approach.

In the last phase of our research, we worked on k-CPQs in road networks. Given two sets of nodes $P$ and $Q$ on a road network, a $k$-Closest Pairs Query ($k$-CPQ) finds the pairs from $P \times Q$ which have the $k$ smallest network distances. As our first contribution, we present a new hierarchical road network partitioning structure, named $G^*$-tree, which is designed to support our proposed algorithms. Then, we propose, as our main contribution, two different approaches, BUSJ and TDT, for processing $k$-CPQs. While TDT technique applies a top-down traversal paradigm by applying a best-first search strategy, BUSJ approach looks for the $k$-closest pairs by traversing the $G^*$-tree in a bottom-up manner. Both of the these approaches employ an effective pruning strategy for shrinking the search space based on the

minimum network distance between sub-graphs, which is main driver for the $G^*$-tree's construction. Our experimental results, using real data sets and varying a number of parameters, show that BUSJ technique always incurs smaller query processing cost in comparison to TDT approach. This superiority is due to a set of pre-computations, specifically local priority queues for each border node in each sub-graph, which effectively orders the data points placing in the same sub-graph based on their distances from that border node.

In real-life trip planning applications the travel time depends on the departure time and road network traffic, thus time-dependent group trip planning queries in road networks can be considered as a possible future work which has not been investigated so far. For this, all queries that have been studied in this thesis can be investigated in time-dependent road networks as possible future works.

# Bibliography


[1] Facebook. `http://www.facebook.com`.

[2] Google+. `http://plus.google.com`.

[3] Loopt. `http://www.loopt.com`.

[4] Elham Ahmadi and Mario A. Nascimento. IBS: An efficient stateful algorithm for optimal sequenced group trip planning queries in road networks. In *Proc. of the 18th Intl. Conf. on Mobile Data Mangement (MDM)*, pages 24–33, 2017.

[5] Elham Ahmadi, Camila F. Costa, and Mario A. Nascimento. Best-compromise in-route nearest neighbor queries. In *Proc. of the 25th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL)-Submitted*. ACM, 2017.

[6] Elham Ahmadi and Mario A. Nascimento. A mixed breadth-depth first search strategy for sequenced group trip planning queries. In *Proc. of the 16th Intl. Conf. on Mobile Data Management (MDM)*, 2015.

[7] Elham Ahmadi and Mario A. Nascimento. k-closest pairs queries in road networks. In *Proc. of the 17th Intl. Conf. on Mobile Data Mangement (MDM)*, pages 232–241, 2016.

[8] Elham Ahmadi and Mario A. Nascimento. k-optimal meeting points based on preferred paths. In *Proc. of the 24th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL)*, pages 47:1–47:4, 2016.

[9] Elham Ahmadi and Mario A. Nascimento. Datasets of roads, public transportation and points-of-interest in Amsterdam, Oslo and Berlin. In: `https://sites.google.com/ualberta.ca/nascimentodatasets/`, 2017.

[10] Sanjeev Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM (JACM)*, 45(5):753–782, 1998.

[11] Franz Aurenhammer and Herbert Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition Journal*, 17(2):251–257, 1984.

[12] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. R*-tree: an efficient and robust access method for points and rectangles. *Mathematical Programming Journal*, 19(2):322–331, 1990.

[13] Haiquan Chen, Wei-Shinn Ku, Min-Te Sun, and Roger Zimmermann. The multi-rule partial sequenced route query. In *Proc. of the 16th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL)*, pages 1–10, 2008.

[14] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, and Jeffrey Xu Yu. Monitoring path nearest neighbor in road networks. In *Proc. of the Intl. Conf. on Management of Data (ACM SIGMOD)*, pages 591–602, 2009.

[15] Reynold Cheng. Managing uncertainty of large spatial databases. *ACM SIGSPATIAL Special Journal*, 8(2):11–17, 2016.

[16] Thomas H Cormen. *Introduction to algorithms*. Massachusetts Institute of Technology (MIT) press, 2009.

[17] Antonio Corral, Yannis Manolopoulos, Yannis Theodoridis, and Michael Vassilakopoulos. Closest pair queries in spatial databases. *ACM Special Interest Group on Management of Data (SIGMOD) Record Journal*, 29(2):189–200, 2000.

[18] Antonio Corral, Yannis Manolopoulos, Yannis Theodoridis, and Michael Vassilakopoulos. Algorithms for processing k-closest-pair queries in spatial databases. *Data & Knowledge Engineering Journal*, 49(1):67–104, 2004.

[19] Akash Das Sarma, Yeye He, and Surajit Chaudhuri. Clusterjoin: a similarity joins framework using map-reduce. *Very Large Data Bases (VLDB) Journal*, 7(12):1059–1070, 2014.

[20] Kimmo Fredriksson and Billy Braithwaite. Quicker similarity joins in metric spaces. In *Intl. Conf. on Similarity Search and Applications*, pages 127–140. 2013.

[21] Yunjun Gao, Lu Chen, Xinhan Li, Bin Yao, and Gang Chen. Efficient k-closest pair queries in general metric spaces. *Very Large Data Bases (VLDB) Journal*, 24(3):415–439, 2015.

[22] Ralf Hartmut Güting. An introduction to spatial database systems. *The VLDB Journal—The International Journal on Very Large Data Bases*, 3(4):357–399, 1994.

[23] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *Mathematical Programming Journal*, 14(2):47–57, 1984.

[24] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics Journal*, 4(2):100–107, 1968.

[25] Tanzima Hashem, Sukarna Barua, Mohammed Eunus Ali, Lars Kulik, and Egemen Tanin. Efficient computation of trips with friends and families. In *Proc. of the 24th ACM Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 931–940, 2015.

[26] Tanzima Hashem, Tahrima Hashem, Mohammed Eunus Ali, and Lars Kulik. Group trip planning queries in spatial databases. In *Intl. Symp. on Spatial and Temporal Databases (SSTD)*, pages 259–276. 2013.

[27] Tanzima Hashem, Tahrima Hashem, Mohammed Eunus Ali, Lars Kulik, and Egemen Tanin. Trip planning queries for subgroups in spatial databases. In *Australasian Database Conference*, pages 110–122, 2016.

[28] Gísli R Hjaltason and Hanan Samet. Incremental distance join algorithms for spatial databases. *ACM Special Interest Group on Management of Data (SIGMOD) Record Journal*, 27(2):237–248, 1998.

[29] Xuegang Huang and Christian S Jensen. In-route skyline querying for location-based services. In *Proc. of the 4th Intl. Workshop on Web and Wireless Geographical Information Systems (W2GIS)*, pages 120–135, 2004.

[30] Edwin H Jacox and Hanan Samet. Metric space similarity joins. *ACM Transactions on Database Systems (ACM TODS) Journal*, 33(2):7, 2008.

[31] Roksana Jahan, Tanzima Hashem, and Sukarna Barua. Scheduling multiple trips for a group in spatial databases. In *Proc. of the 20th Intl. Conf. on Extending Database Technology (EDBT)*, pages 10–22, 2017.

[32] Yaron Kanza, Roy Levin, Eliyahu Safra, and Yehoshua Sagiv. Interactive route search in the presence of order constraints. *Very Large Data Bases (VLDB) Journal*, 3(1-2):117–128, 2010.

[33] George Karypis and Vipin Kumar. Analysis of multilevel graph partitioning. In *Proc. of the ACM/IEEE Conf. on Supercomputing*, page 29, 1995.

[34] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[35] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route skyline queries: A multi-preference path planning approach. In *Proc. of the 26th Intl. Conf. on Data Engineering (ICDE)*, pages 261–272, 2010.

[36] Hisashi Kurasawa, Atsuhiro Takasu, and Jun Adachi. Finding the k-closest pairs in metric spaces. In *Proc. of the 1st Workshop on New Trends in Similarity Search*, pages 8–13, 2011.

[37] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In *Proc. of the 9th Intl. Symp. on Spatial and Temporal Databases (SSTD)*, pages 273–290. 2005.

[38] Hongga Li, Hua Lu, Bo Huang, and Zhiyong Huang. Two ellipse-based pruning methods for group nearest neighbor queries. In *Proc. of the 13th annual ACM Intl. workshop on Geographic information systems (ACM GIS)*, pages 192–199, 2005.

[39] Jing Li, Yin David Yang, and Nikos Mamoulis. Optimal route queries with arbitrary order constraints. *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE) Journal*, 25(5):1097–1110, 2013.

[40] Qianlu Lin, Chuan Xiao, Muhammad Cheema, and Wei Wang. Finding the sites with best accessibilities to amenities. In *Intl. Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 58–72, 2011.

[41] Eric Hsueh-Chan Lu, Huan-Sheng Chen, and Vincent S Tseng. An efficient framework for multirequest route planning in urban environments. *IEEE Transactions on Intelligent Transportation Systems Journal*, 18(4):869–879, 2017.

[42] Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. Group nearest neighbor queries. In *Proc. of the 20th Intl. Conf. on Data Engineering (ICDE)*, pages 301–312, 2004.

[43] Rodrigo Paredes and Nora Reyes. Solving similarity joins and range queries in metric spaces with the list of twin clusters. *Journal of Discrete Algorithms*, 7(1):18–35, 2009.

[44] M Patella, P Ciaccia, and P Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases (VLDB)*, pages 1241–1253, 1997.

[45] George Roumelis, Antonio Corral, Michael Vassilakopoulos, and Yannis Manolopoulos. New plane-sweep algorithms for distance-based join queries in spatial databases. *GeoInformatica Journal*, 20(4):571–628, 2016.

[46] George Roumelis, Michael Vassilakopoulos, Antonio Corral, and Yannis Manolopoulos. A new plane-sweep algorithm for the k-closest-pairs query. In *Intl. Conf. on Current Trends in Theory and Practice of Informatics*, pages 478–490. 2014.

[47] Samiha Samrose, Tanzima Hashem, Sukarna Barua, Mohammed Eunus Ali, Mohammad Hafiz Uddin, and Md Iftekhar Mahmud. Efficient computation of group optimal sequenced routes in road networks. In *Proc. of the 16th Intl. Conf. on Mobile Data Management (MDM)*, pages 122–127, 2015.

[48] Jochen Schiller and Agnès Voisard. *Location-based services*. Elsevier, 2004.

[49] Shuo Shang, Lisi Chen, Zhewei Wei, Christian S Jensen, Ji-Rong Wen, and Panos Kalnis. Collective travel planning in spatial networks. *IEEE Transactions on Knowledge and Data Engineering Journal*, 28(5):1132–1146, 2016.

[50] Shuo Shang, Ke Deng, and Kexin Xie. Best point detour query in road networks. In *Proc. of the 18th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL)*, pages 71–80, 2010.

[51] Shuo Shang, Kai Zheng, Christian S Jensen, Bin Yang, Panos Kalnis, Guohe Li, and Ji-Rong Wen. Discovery of path nearby clusters in spatial networks. *IEEE Transactions on Knowledge and Data Engineering Journal*, 27(6):1505–1518, 2015.

[52] Mehdi Sharifzadeh, Mohammad Kolahdouzan, and Cyrus Shahabi. The optimal sequenced route query. *Very Large Data Bases (VLDB) Journal*, 17(4):765–787, 2008.

[53] Mehdi Sharifzadeh and Cyrus Shahabi. Processing optimal sequenced route queries using voronoi diagrams. *GeoInformatica Journal*, 12(4):411–433, 2008.

[54] Michael Shekelyan, Gregor Jossé, Matthias Schubert, and Hans-Peter Kriegel. Linear path skyline computation in bicriteria networks. In *Proc. of the 19th Intl. Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 173–187, 2014.

[55] Shashi Shekhar and Jin Soung Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In *Proc. of the 11th ACM Intl. Symp. on Advances in Geographic Information Systems (ACM GIS)*, pages 9–16, 2003.

[56] Hyoseop Shin, Bongki Moon, and Sukho Lee. Adaptive multi-stage distance join processing. *ACM Special Interest Group on Management of Data (SIGMOD) Record Journal*, 29(2):343–354, 2000.

[57] Hyoseop Shin, Bongki Moon, and Sukho Lee. Adaptive and incremental processing for distance join queries. *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE) Journal*, 15(6):1561–1578, 2003.

[58] Yasin N Silva, Spencer S Pearson, and Jason A Cheney. Database similarity join for metric spaces. In *Intl. Conf. on Similarity Search and Applications*, pages 266–279. 2013.

[59] Subarna Chowdhury Soma, Tanzima Hashem, Muhammad Aamir Cheema, and Samiha Samrose. Trip planning queries with location privacy in spatial databases. *World Wide Web Journal*, 20(2):205–236, 2017.

[60] Anika Tabassum, Sukarna Barua, Tanzima Hashem, and Tasmin Chowdhury. Dynamic group trip planning queries in spatial databases. In *Proc. of the 29th Intl. Conf. on Scientific and Statistical Database Management (SSDBM)*, page 38, 2017.

[61] David Taniar and Wenny Rahayu. A taxonomy for nearest neighbour queries in spatial databases. *Journal of Computer and System Sciences*, 79(7):1017–1039, 2013.

[62] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Transactions on. Database Systems (ACM TODS) Journal*, 35(3):1–20, 2010.

[63] G Tsatsanifos, P Petcovici, and Mario A. Nascimento. Meet-and-go: finding optimal single connecting points considering companionship preferences. In *Proc. of the 24th Intl. Conf. on Advances in Geographic Information Systems (ACM SIGSPATIAL)*, pages 87:1–87:4, 2015.

[64] Ye Wang, Ahmed Metwally, and Srinivasan Parthasarathy. Scalable all-pairs similarity search in metric spaces. In *Proc. of the 19th ACM Intl. Conf. on Knowledge Discovery and Data mining (ACM KDD)*, pages 829–837, 2013.

[65] Da Yan, Zhou Zhao, and Wilfred Ng. Efficient algorithms for finding optimal meeting point on road networks. *Very Large Data Bases (VLDB) Journal*, 4(11):1–11, 2011.

[66] Da Yan, Zhou Zhao, and Wilfred Ng. Efficient processing of optimal meeting point queries in euclidean space and road networks. *Knowledge and Information Systems (KAIS) Journal*, 42(2):1–33, 2013.

[67] Congjun Yang and King-Ip Lin. An index structure for improving closest pairs and related join queries in spatial databases. In *Database Engineering and Applications Symposium (IDEAS)*, pages 140–149, 2002.

[68] Man Lung Yiu, Nikos Mamoulis, and Dimitris Papadias. Aggregate nearest neighbor queries in road networks. *IEEE Transactions on Knowledge and Data Engineering (IEEE TKDE) Journal*, 17(6):820–833, 2005.

[69] Jun Zhang, Nikos Mamoulis, Dimitris Papadias, and Yufei Tao. All-nearest-neighbors queries in spatial databases. In *Proc. of the 16th Intl. Conf. on*

*Scientific and Statistical Database Management (SSDBM)*, pages 297–306, 2004.

[70] Ruicheng Zhong, Guoliang Li, Kian-Lee Tan, and Lizhu Zhou. G-tree: An efficient index for knn search on road networks. In *Proc. of the 22nd ACM Intl. Conf. on Information and Knowledge Management (CIKM)*, pages 39–48, 2013.

[71] Chenghao Zhu, Jiajie Xu, Chengfei Liu, Pengpeng Zhao, An Liu, and Lei Zhao. Efficient trip planning for maximizing user satisfaction. In *Proc. of the. 22nd Intl. Conf. on Database Systems for Advanced Applications (DASFAA)*, pages 260–276, 2015.