

Doctoral Dissertation

**A Study on Privacy-Preserving Route Planning
for Smart Mobility Applications**

Francis Jerome G. Tiausas

November 24, 2023

Graduate School of Science and Technology
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Science and Technology,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Francis Jerome G. Tiausas

Thesis Committee:

Professor Keiichi Yasumoto	(Supervisor)
Professor Kazutoshi Fujikawa	(Co-supervisor)
Professor Youki Kadobayashi	(Co-supervisor)
Associate Professor Hirohiko Suwa	(Co-supervisor)
Assistant Professor Yuki Matsuda	(Co-supervisor)

A Study on Privacy-Preserving Route Planning for Smart Mobility Applications *

Francis Jerome G. Tiausas

Abstract

Route Planning Services (RPS) are a core component of autonomous personal transport systems which facilitate safe and efficient navigation of dynamic urban environments. However, conventional RPS also require the disclosure of the user's origin and destination as input and the computed route as output which is a major privacy concern. Though a number of privacy-preserving RPS have been developed over the past decade, most are rendered impractical by the increased communication and processing overhead they entail. In this dissertation, the core challenge is to develop an RPS where: (1) route privacy is objectively quantified, (2) Utility, Performance, and Privacy objectives are adequately satisfied, and (3) the produced routes are valid and close-to-optimal.

The core idea is to use Private Information Retrieval (PIR) over partitions of a road network (distributed across multiple devices) to facilitate privacy-preserving route planning. To satisfy the different system objectives, this was then combined with Multi-Objective Genetic Algorithms (MOGA) to discover acceptable trade-offs between said objectives. However, this optimization step was found to be rather slow, and did not protect the intermediate route at all.

Thus, an improved approach called Hierarchical Privacy-Preserving Route Planning (HPRoP) was developed, combining Inertial Flow partitioning with a novel route planning heuristic which distributes route planning tasks across multiple levels to protect the entire route. Metrics were also formulated to quantify the privacy of the source/destination points (*endpoint location privacy*), and the route

*Doctoral Dissertation, Graduate School of Information Science,
Nara Institute of Science and Technology, November 24, 2023.

itself (*route privacy*). Evaluations on the road network of Osaka City showed that HPRoP reliably produced routes that deviate only by $\leq 20\%$ in length from optimal shortest paths, while being able to complete routes within ~ 25 seconds despite using PIR. Moreover, more than half of the produced routes achieved near-optimal endpoint location privacy (~ 1.0) and good route privacy (≥ 0.8).

Keywords:

Distributed route planning, Private Information Retrieval, Multi-objective Optimization, Edge computing

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem Statements	3
1.3	Organization of Dissertation	5
2	Related Literature	6
2.1	Smart Mobility and Privacy	6
2.2	Route Planning Systems (RPS)	6
2.3	Privacy-Preserving RPS (P2RPS)	7
2.3.1	Structured Encryption-based Techniques	8
2.3.2	PIR-based Techniques	8
2.3.3	Other Encryption-based Techniques	9
2.3.4	Other Privacy Techniques	10
2.4	Multi-objective Optimization	11
3	Assumptions and Key Ideas	12
3.1	Assumptions	12
3.1.1	Road Network Model	12
3.1.2	Deployment Environment	13
3.1.3	Network Architecture	14
3.1.4	Threat Model	15
3.1.5	Privacy-Preservation Mechanisms	15
3.1.6	Privacy-Preserving Route Planning Service (P2RPS)	15
3.1.7	Privacy Scope and Limitations	16
3.2	Key Idea	16

4	Practical PIR-based Route Planning via Multi-Objective Optimization (P2RoP-MO)	19
4.1	Assumptions	21
4.2	Key Idea	22
4.3	Mathematical Formulation	22
4.3.1	Processing Throughput	23
4.3.2	Privacy Protection Level	25
4.3.3	Travel Time Accuracy	25
4.3.4	Objective Functions	27
4.4	Multi-objective Optimization	28
4.5	Evaluation	29
4.5.1	Mobility and Vehicle Trip Data	29
4.5.2	NSGA-II Configuration	30
4.5.3	Experiment Setup	30
4.5.4	Experiment Results	33
4.6	Summary	33
5	HPRoP: Hierarchical Privacy- Preserving Route Planning	36
5.1	Models and Assumptions	36
5.1.1	Road Network Partitioning Model	36
5.1.2	Approximate Shortest Path Model	38
5.1.3	Assumptions	40
5.2	Key Idea	40
5.2.1	Exact Partial Region Dijkstra’s Algorithm (EPR-D)	41
5.2.2	Approximate Partial Region Dijkstra’s Algorithm (APR-D)	42
5.3	Privacy Metrics	43
5.3.1	Endpoint Location Privacy Model	43
5.3.2	Route Privacy Model	44
5.4	Hierarchical Privacy-Preserving Route Planning	46
5.4.1	Private Information Retrieval (PIR)	46
5.4.2	Inertial Flow Partitioning	47
5.4.3	Distributed Architecture	48
5.4.4	Heuristic Algorithm	49
5.4.5	Route Privacy Mechanism	53

5.4.6	Shortcut Connections	54
5.5	Evaluation	56
5.5.1	Environment	57
5.5.2	Methodology	57
5.5.3	Results	58
5.5.3.1	Effect of Shortcut Connections	58
5.5.3.2	Optimal Route Approximation	60
5.5.3.3	Endpoint Location Privacy	61
5.5.3.4	Route Privacy	61
5.5.3.5	Route Completion Time	64
5.5.3.6	Memory Usage	64
5.5.3.7	Pre-processing Time	65
5.5.4	Discussions	66
5.5.4.1	Optimal Route Approximation and Actual Route Lengths	66
5.5.4.2	Scalability	68
5.5.4.3	Improving Route Completion Times	69
5.6	Summary	70
6	Conclusion	71
6.1	Summary	71
6.2	Future Work	72
	Bibliography	75
	Publication List	82

List of Figures

4.1	(Left) Service Region divided into 4x4 <i>grids/partitions</i> . Source and Destination partitions can be a combination of multiple <i>base partitions</i> for better privacy. (Right) Each partition has at least one RSU (blue) and edge server (orange). Edge servers form a mesh network.	20
4.2	Practical route privacy preservation achieved through combining <i>privacy-cloaked</i> sections (orange lines) with non-privacy-cloaked or <i>intermediate</i> sections (red lines)	21
4.3	(Left) Mean travel time accuracy $A(q)$ and Selected Node Intersection Ratio r_q ; (Right) Processing Throughput $P(q)$ to actual processing time $\frac{1}{P(q)}$ (in seconds) with empirically-found values for $P(q)_{max} = 5.0$ (five queries every second) and $\lambda = -1.6$	26
4.4	Performance of NSGA-II, L-BFGS-B, and SLSQP in optimizing the individual objectives ($P(q)$, $V(q)$, $A(q)$) and individual utility ($U(q)$)	32
4.5	Comparison of execution times between different algorithms	32
5.1	Types of shortest paths stored by each partition	39
5.2	Queried Partitions using Dijkstra’s algorithm to calculate two routes with the same origin but different destinations (~ 1 km away from each other). The numbers indicate how many queries were handled by each partition.	45
5.3	Road network graph of Osaka City, Japan hierarchically-partitioned using the Inertial Flow algorithm	47
5.4	Cloud-based Architecture (Left) vs Distributed Architecture (Right)	48
5.5	Initialization: Find SP between lowest-level partitions containing s (blue circle) and d (green) separately.	50

5.6	Initialization: Use discovered path as basis route (purple line) for the next level	50
5.7	Source Subroute Conn.: Find and connect source subroute (blue line) to basis route (purple line)	50
5.8	Dest. Subroute Conn.: Find and connect dest. subroute (green line) to basis route (purple line)	50
5.9	Basis Route Merging: Merge routes and use as new basis route (purple line)	50
5.10	Demonstration of different shortcut connection strategies for improving the heuristic algorithm’s performance against the base case (Left), where the dark red shape represents the starting partition, and the lighter red shapes represent the partitions it connects to. The black outline represents the starting partition’s parent. (Middle) uses Same Parent Shortcuts, while (Right) uses 1-hop Neighbor Shortcuts.	56
5.11	Distribution of <i>Optimal Route Approximation</i> results for different Shortcut Connection methods	58
5.12	Comparison of total Route Errors for different Shortcut Connection methods	58
5.13	Distribution of Pre-Computation Times for different Shortcut Connection methods	59
5.14	Distribution of Optimal Route Approximation results using APR-D, EPR-D, and HPRoP	59
5.15	Distribution of Route Privacy $\Phi(Q^*)$ results for APR-D, EPR-D, and HPRoP	61
5.16	Visualization of Queried Partitions for APR-Dijkstra (Center), EPR-Dijkstra (Right) and HPRoP (Left) for the same route. Numbers indicate partitions that were queried multiple times.	62
5.17	Distribution of Optimal Route Approximation and Route Privacy results using HPRoP for 4,000 test routes. Note that both axes were reoriented to show the best results on the lower left.	63
5.18	Distribution of the required number of queries for route completion	63
5.19	Distribution of total PIR retrieval times in seconds	63

5.20	Distribution of Per-Partition Memory Usage (in MB) for the different approaches	65
5.21	Distribution of Per-Partition Pre-processing Time results (in seconds) for the different approaches	65
5.22	Visualization of differences in HPRoP routes at $1.19 < \alpha(r_{s,d}) < 1.2$: (Left) shows an example of minimal deviation (~ 1.0 km) from the actual SP; (Right) shows an example of maximal deviation (~ 5.63 km) from the actual SP	66
5.23	Visualization of differences in HPRoP routes with an Optimal Route Deviation between 0.95 km and 1.0 km; (Left) shows an example with good Optimal Route Approximation (~ 1.035); (Right) shows an example with bad Optimal Route Approximation (~ 1.162)	67
5.24	Relationship between Optimal Route Approximation and Optimal Route Deviation (difference between lengths of HPRoP's route and the actual shortest path)	67
5.25	<i>Maximum</i> values for Optimal Route Approximation given a selected percentage of all produced routes	68
5.26	<i>Maximum</i> values for Route Completion Time given a selected percentage of all produced routes	68

1 Introduction

1.1 Background

Route Planning Services (RPS) are web-based applications which can calculate routes or paths between two different locations in a transportation network according to some criteria (e.g. shortest distance, least time, least cost, etc.) given by the service user. They are therefore indispensable navigational aids for everyday commuters, vehicle operators, autonomous vehicles, and so forth. Nevertheless, use of (conventional) RPS is not without their risks — especially when it comes to privacy. Conventional RPS implicitly require the disclosure of two types of user data: (1) the user’s origin and destination, and (2) the computed route between those locations. That is, said RPS would not be able to compute the route without (1), and the user has no reason to use the RPS if it cannot produce (2). Herein lies the privacy risk. Analysis can be conducted on the aforementioned user data to reveal *points of interest (POIs)* [1] relevant to a particular user — such as their places of work or residence. Further, these POIs can also reveal highly-sensitive information about users’ political, sexual, or religious tendencies [2]. All of these expose users to massive personal risks such as targeted criminal acts, mass surveillance, discrimination, etc. when said data is not handled properly, or falls into the wrong hands — as evidenced by the rise in high-profile data breaches of the past decade. Consequently, this may also lead to the loss of public trust and the threat of legal action on the Service Providers (SP) themselves. It is therefore quite clear that protecting user location and route data is very important in any modern RPS.

With this in mind, many researchers have begun to focus on devising novel privacy-preservation mechanisms and incorporating them into RPS. For instance, privacy-preserving protocols for querying road traffic data have been developed

for Vehicle Ad-hoc Networks (VANETs). These either offload the computation cost of route planning onto the vehicle itself [3–5] or an external trusted entity [6,7] to preserve privacy. The latter are already done by most modern RPS, while the former are closer to being offline on-board navigation systems (ONS) than actual RPS. Some approaches use Structured Encryption (SE) schemes [8–10] to provide privacy-protection for both user queries and the road network data itself. These are relatively lightweight and efficient, but come at the cost of inherently leaking some query-related information which can be analyzed to circumvent privacy protections. Still other approaches use Private Information Retrieval (PIR) [11], a data exchange protocol with strong privacy guarantees but is often considered to be too computationally-heavy to be applied naively on large road networks (e.g. of a medium-sized city). As such, only a few recent examples of PIR-based RPS exist. The approach in [12] compresses road network graphs in a novel PIR-queryable manner but results in longer pre-processing and query response times. The approach in [13] partitions the road network into disjoint subgraphs which are individually retrieved via PIR and then used by a local routing algorithm on the user’s device. This requires making multiple PIR queries which consequently results in longer route completion times. Aside from these, other approaches [14, 15] utilize encryption techniques with stronger privacy guarantees but are also much less efficient and flexible. This, in turn, makes them less practical for a real-time RPS with a dynamic underlying road network.

A common issue across the aforementioned works seems to be that achieving a certain level of privacy always comes at the cost of some other aspect of the RPS — such as its Utility (e.g. whether its routes are accurate) or its Performance (e.g. how long it takes to calculate routes). This trade-off is typically fixed from the outset for most approaches, but PIR-based approaches are potentially a bit more flexible in this regard. That is, in PIR-based RPS, larger road networks mean having larger route databases and, consequently, slower route retrieval times. These large road networks, however, can presumably be subdivided into smaller, separate sub-networks, each with their own route database. This idea of making multiple PIR queries on these much smaller databases would be much faster and more memory-efficient than a single PIR query on one considerably larger database. This is even more advantageous if these smaller databases

can be distributed across several machines (either virtual instances or as physical edge servers) which can each do independent query processing. This is the most basic idea behind the Privacy-Preserving Route Planning Services — henceforth referred to as “P2RPS” for the sake of brevity — that will be presented in this dissertation. With this basic approach, a client application on the user’s device can then make PIR requests to these separate databases, decrypt the resulting PIR responses, and then construct the final route using the decrypted partial routes on the local device. On the side of the SP, this also provides a greater degree of control over the trade-off between Privacy, Utility, and Performance which would have been otherwise static — as is currently the case with existing approaches. This can potentially lead to better Quality-of-Service (QoS) which is a crucial step in making P2RPS practical to use in real-world applications.

In this dissertation, we discuss two PIR-based RPS approaches. The first approach, *Practical PIR-based Route Planning via Multi-objective Optimization (P2RoP-MO)*, utilizes a Multi-objective Genetic Algorithm (MOGA) to discover solutions that optimize the trade-offs between the three QoS objectives to satisfy user preferences. The optimization step itself, however, was found to be rather impractical since it slows down response times and holds queries back in order to batch them. More importantly, the approach only offers “partial” route privacy since the intermediate route is not protected to help reduce processing time but is a crucial vulnerability which can be used by adversaries to deduce the actual route. The second approach, *Hierarchical Privacy-Preserving Route Planning (HPRoP)*, is a direct improvement over P2RoP-MO and uses a novel Hierarchical Route Planning algorithm alongside other innovations to eliminate the vulnerabilities of the former while greatly improving the QoS across-the-board.

1.2 Problem Statements

Though the core idea seems rather simple and straightforward, developing a functional prototype P2RPS around this concept is not without its own set of unique and difficult challenges.

Challenge 1: How can we objectively quantify route privacy?

As existing works have an all-or-nothing approach to privacy, there has not

been any need to quantify the privacy of a route result received from the RPS. Since both of the approaches proposed here implicitly *reduce* Privacy to achieve better Utility and Performance, a way to quantify “partial” privacy should be considered. For the approach based on MOGA, the number of nodes (r_q) within the grid containing the partial routes as well as its geographical area (a_q) are used to describe privacy. For the approach based on HPRoP, a pair of novel privacy metrics — *endpoint location privacy* and *route privacy* — were instead formulated in order to describe route privacy in a more general manner.

Challenge 2: How can we control the trade-off between Privacy, Utility, and Performance in a P2RPS?

In existing approaches, the trade-off between these three QoS objectives remains relatively static throughout the operational lifetime of the P2RPS which may run counter to the SP’s ideal goal of providing providing good QoS to the service’s users. Our two approaches attempt to solve this by first performing a *Hierarchical Partitioning* of the road network such that multiple different-sized partitions can be used for any chosen location within it. P2RoP-MO uses a simple grid-based system for its partitioning during its pre-processing phase, and a Multi-Objective Genetic Algorithm (MOGA) during its routing phase to optimize the size and number of nodes in the origin and destination partitions. HPRoP, on the other hand, uses **Inertial Partitioning** to divide the road network into balanced partitions so that: (1) a base level of Endpoint Location Privacy is guaranteed, and (2) minimize PIR execution times during its pre-processing phase. It then uses a novel *Hierarchical Route Planning Algorithm* to minimize the number of PIR requests while keeping the accuracy of the route good, and the route privacy at an adequate level.

Challenge 3: How can we combine partial routes within and between subsections to arrive at a valid and close-to-optimal final route?

This challenge is exclusive to both of the P2RPS described here as it relies on dividing up the road network route data and distributing those to different partitions. Since the partitions are technically self-contained subgraphs of the larger road network, how can the “partial” routes from each sub-graph be combined to produce a valid and close-to-optimal final route? In P2RoP-MO, this is resolved by using a pre-trained predictive model called the Optimal Sequence

of Grids (OSG) in combination with allowing the intermediate route between the origin and destination partitions to be non-private. HPRoP solves this using the novel *Hierarchical Route Planning Algorithm* which creates a *merged route* by computing multiple basis routes between partitions at different levels of the hierarchy.

1.3 Organization of Dissertation

This dissertation is organized as follows. A comprehensive review of the recent related works is presented in Chapter 2. Preliminary assumptions about the deployment environment and the threat model, as well as key ideas about the P2RPS briefly described at the end of Sec. 1.1 are discussed in Chapter 3. In this dissertation, two PIR-based RPS approaches will be presented. The preliminary approach, P2RoP-MO, is discussed and evaluated in Chapter 4, while that of the improved approach, HPRoP, is done in Chapter 5. A brief summary of the aforementioned approaches as well as some suggestions and plans for future work are presented in Chapter 6.

2 Related Literature

This chapter covers recent works related to decentralized route planning systems and privacy-preserving mechanisms in route planning. Also, works providing important background information on private information retrieval (PIR) will be discussed.

2.1 Smart Mobility and Privacy

Smart Mobility is often considered as one of the key areas which characterize Smart Cities in practice as it has widespread and far-reaching effects on the quality-of-life of the city's inhabitants such as reduced congestion, increased safety, and improved transfer speed [16]. Since Smart Mobility applications typically make use of sensitive spatio-temporal data, ensuring that they can guarantee an adequate level of privacy is also of great importance. Examples of these can be found across different domains such as human mobility monitoring and prediction [17, 18], ride-sharing platforms [15], route planning [19, 20], etc.

2.2 Route Planning Systems (RPS)

Route planning systems use route planning algorithms to devise paths across a particular road network. Road networks are typically represented as a network graph $G = (V, E)$ where V are the vertices representing road intersections and E are the edges representing road segments. The objective is usually to find the shortest point-to-point path between an origin vertex $s \in V$ and a destination vertex $d \in V$. Classical algorithms like Dijkstra's and Bellman-Ford [21] calculate routes by repeatedly scanning connected vertices from some source point and assigning them weights until a path to the destination point is found. Modern al-

gorithms improve upon this by leveraging the unique properties of road networks. ALT [22] pre-computes distances to fixed landmarks and uses them as lower bounds to inform a bidirectional A* search [23]. *Contraction Hierarchies* [24] pre-processes the network graph to establish “shortcut edges,” facilitating faster route calculation between distant points. *Customizable Route Planning* [25] uses the network graph’s topology in their metric-independent hierarchical routing method. Recent approaches based on federated learning have also demonstrated how to construct a lightweight shared prediction models to determine shortest paths such as in [26, 27]. Using these models, the best route between any origin and destination point at any given time can be determined without the need for conventional route planning algorithms. These models are also kept updated by periodically sharing only model weights from the local system to decrease both data transfer costs and private data leakage.

2.3 Privacy-Preserving RPS (P2RPS)

As previously mentioned, deploying any of the conventional, non-P2RPS on a server inevitably means that the user’s origin and destination must be divulged so that the final route can be computed. A simple way to solve this is to deploy those same RPS on the client-side, but this creates its own problems. These include downloading large amounts of road network data, having to continuously update said data through additional downloads, and performing computationally-intensive route planning tasks on more resource-constrained machines. In short, deploying (conventional RPS) on the server compromises privacy while deploying on the client degrades functionality.

This highlights the need to focus on devising novel P2RPS as a lot of recent research works have done. The most recent techniques generally fall under three categories: (1) *Structured Encryption*-based, (2) *PIR*-based, and (3) *Other encryption-based* schemes. In addition, a number of schemes for privately querying road traffic information with applications to vehicle navigation systems also exist, but, since these either perform route planning on the vehicle itself [3–5] or an external trusted entity [6, 7], these works have been excluded here.

2.3.1 Structured Encryption-based Techniques

Structured Encryption [28] refers to techniques that allows data structures to be encrypted such that these can be queried later in a privacy-preserving manner. They are typically more efficient in terms of computation time and communication overhead at the cost of leaking a small amount of information about the data and the queries. The approaches in [9, 10] use structured encryption to find the shortest distance between vertex-pairs in encrypted graphs. These, however, are only able to find shortest distance values instead of the actual shortest paths, and are vulnerable to collusion between the storage and computing servers — which are essentially the same entity in the RPS context. In contrast, [8] is able to retrieve the actual shortest paths on a single-server setup which effectively eliminates the aforementioned issues. However, pre-computing the encrypted database for large sparse graphs (i.e. with $|V| \geq 10,000$ and $|E| \geq 30,000$) took upwards of 16.5 hours and produced very large files (around 4.4 GB), rendering it impractical for dynamic scenarios such as real-time route planning. Additionally, while all three have heavily-constrained leakage profiles, some of the information they inherently leak may be detrimental to route privacy. For instance, the number of potential query elements (i.e. the database size) can be used to determine the specific subgraph of the road network graph being used. Query repetitions and total queries for route completion can be jointly analyzed across multiple sessions to deduce the actual route. Different approaches may also leak other information in addition to the ones mentioned here.

2.3.2 PIR-based Techniques

PIR [11] is a technique that allows remote databases to be queried in such a way that the retrieved element would not be revealed to the service provider or any third-party entity. PIR-based schemes, therefore, have slightly stronger privacy guarantees in that repeated queries and underlying database sizes are not leaked, but have the disadvantage of much higher communication overhead. While most modern implementations [29–32] have become very communication-efficient (i.e. up to $O(\sqrt{N})$), they remain impractical for accessing a database of All-Pairs of Shortest Paths (APSP) in a large city. This is because PIR schemes need to go

through each individual element to avoid leaking information about the element being retrieved [33]. The approach in [12] describes a method for compressing road network graphs via sign-decomposition combined with Yao’s garbled circuits [34] and PIR to protect both user queries and said graph, giving it strong end-to-end privacy guarantees. However, it also has relatively long pre-processing and query response times since the protocol must operate on compressed and encrypted data at all times. The approach in [13] partitions the network graph into disjoint sections (i.e. each consisting of a separate subgraph) which can then be retrieved via PIR during local computation of a shortest path during the routing phase. While this requires minimal pre-processing time, the total route completion time remains rather long since the locally-run routing algorithm would need multiple PIR queries to complete a single route.

As this work aims to achieve strong privacy guarantees for users’ routes while also meeting utility and performance targets, PIR was chosen as the core privacy-preservation mechanism for the approaches presented in this dissertation. Unlike structured encryption, it does not leak information about query repetitions, which can potentially be analyzed by an adversary to distinguish between different route requests by the same user.

2.3.3 Other Encryption-based Techniques

Other encryption-based schemes with much stronger privacy guarantees also exist but they are also much less efficient than the previous two. For instance, [14] allows users to request routes between arbitrary source and destination partitions, as well as within said partitions in a privacy-preserving manner using 1-of-n Oblivious Transfer [35]. However, the scheme needs to compute APSP for the aforementioned partitions during the routing phase, drastically slowing down query response times. Similarly, the work in [15] uses Paillier’s Encryption to privately query outgoing edge weights from vertices in the road network graph which, in turn, is used to inform a route planning algorithm running locally on the user’s own device. The scheme, unfortunately, has a very high communication overhead since it has to make a separate query for every vertex that needs to be “scanned” by the routing algorithm.

2.3.4 Other Privacy Techniques

Aside from the above encryption-based privacy techniques, several other approaches of interest are worth mentioning here. Confidential Computing [36] is a data protection paradigm that aims to protect “data in use” as opposed to data at rest (i.e. in storage) and data in motion (i.e. being transmitted through the network). It makes use of technologies such as Trusted Execution Environments (TEE) [37] with Memory Encryption [38] to provide strong privacy and security guarantees (typically enforced at the hardware-level) for both data and programs running within its scope. These guarantees include protecting data in use from being viewed and/or modified, and protecting the code from being modified by unauthorized entities. In the context of RPS, it can potentially be much more efficient than any of the previously-mentioned encryption-based techniques since it does not require multiple rounds of communication and can perform more frequent operations on symmetric key encrypted memory (much lighter than public-key encrypted memory) within the TEE. However, TEE seem to be particularly vulnerable to side-channel attacks [39, 40] since they inevitably have to interact with other parts of the system (e.g. shared caches, etc.) which may not be as secure. Moreover, while TEE architectures have recently started appearing in modern processors, such capabilities may not be present in the existing edge-server infrastructure of Smart Cities and will likely require costly infrastructure upgrades to deploy. Nevertheless, it remains a potentially promising approach for the route planning use case.

Other privacy techniques are based around *Differential Privacy* or *DP* [41] which states that a mechanism is *differentially private* if its output remains the same even when one record from a dataset is removed or altered. These typically work by adding “noise” to the records (or the results) to improve privacy. In a P2RPS, the DP has to be framed from the point-of-view of an adversary that wants to rediscover the association between a specific route and the route privately retrieved by the user — and that the chosen DP-based technique should make this difficult. No approaches currently use these techniques in the context of route privacy as of the time of this writing, but it could still be useful in (1) devising metrics for evaluating route privacy, and (2) adding noise to the selection of origin and destination points to improve privacy.

2.4 Multi-objective Optimization

Finally, it should also be considered that privacy is only one among several objectives for a route planning service. Thus, there is a need to balance privacy with other objectives such as execution time and result accuracy. A Multi-Objective Optimization (MOO) scheme is therefore needed to simultaneously and adaptively balance these objectives. As having multiple equally-important objectives makes it difficult to compare two solutions with each, the Pareto optimal dominance must be considered instead. Approaches to solving MOO problems are generally classified into two types: *a priori* and *a posteriori*. In the *a priori* approach, the multi-objective problem is converted to a single-objective one through aggregation and the use of weights. *A posteriori* methods, on the other hand, aim to produce all the Pareto optimal solutions. Evolutionary algorithms such as the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [42] attempt to find a Pareto optimal solution in a single run and have become standard approaches to solving MOO problems. In particular, NSGA-II sorts solutions based on their dominance level where a solution that is non-dominated is ranked 1 while dominated ones are ranked 2 and so on. Lower ranked solutions have a higher chance to be selected to generate a new population.

The preliminary approach presented in Chapter 4 is based on NSGA-II which is used to find close-to-optimal solutions to the Privacy, Utility, and Performance objectives by varying the size and number of nodes within the partitions containing the exact origin and destination locations. The second approach presented in Chapter 5 does not implement an optimization step and therefore does not need to use any MOO algorithms.

3 Assumptions and Key Ideas

This chapter will first discuss the core assumptions about the road network model, deployment environment, and network architecture, before moving on to discuss key ideas behind the P2RPS in this dissertation. All information in this chapter apply to the two approaches presented in this dissertation.

Table 3.1: Summary of Symbols used in Chapter 3

Symbol	Description
$G = (V, E)$	Road network graph with road segments E and intersections V
$l_G(u, v)$	Weight of a road segment having endpoints (u, v)
s, d	Source (s) and Destination (d) vertices
$r_G(s, d)$	Path/route between s and d
$L_G(r_G(s, d))$	Total weight of path/route $r_G(s, d)$
$R_G(s, d)$	All possible paths between s and d
$\rho_G(s, d)$	<i>Shortest path</i> between s and d
R_G^*	Set of all possible SPs between any two vertices in G
L_G^{max}	Weight of the longest SP in the set, R_G^*
C_{pir}	A constant representing the change in PIR data retrieval times with respect to database size
\mathcal{N}_p	Average vertex count for some partition subgraph, p

3.1 Assumptions

3.1.1 Road Network Model

The road network is assumed to be modelled as a directed graph $G = (V, E)$ where the edges E represent road segments, and the vertices V represent ei-

ther road intersections or terminals. Each road segment $e \in E$ is a directed edge between two vertices $u, v \in V$ such that $e = (u, v)$, with parallel opposing lanes being represented by two directed edges going in opposite directions. The traversal cost for e is given by the edge weight, $l_G(u, v)$, which is a numerically-quantifiable property of that particular road segment. It can represent properties such as the physical length of that road segment, the time it takes to travel through it, the monetary cost to use it (for example, in the case of toll roads), etc. It is additionally assumed that edge weights can have dynamic values rather than having a single fixed value at the outset — for instance, when representing travel times through that particular road under different traffic conditions. In the practical case, travel times are a better metric for edge weight in the route planning context, but, due to the lack of available data, road lengths are used as a basis instead. A *route* or *path* between two vertices $s, d \in V$ is defined as a sequence of vertices $r_G(s, d) = (v_1^{sd}, v_2^{sd}, \dots, v_{n-1}^{sd}, v_n^{sd})$ where the first vertex $v_1^{sd} = s$ and the last vertex $v_n^{sd} = d$. The total weight of $r_G(s, d)$ is given by $L_G(r_G(s, d)) = \sum_{i=1}^{|r_G(s, d)|-1} l_G(v_i^{sd}, v_{i+1}^{sd})$. Denoting *all* possible paths between s, d as $R_G(s, d)$, the *Shortest Path* is then:

$$\rho_G(s, d) = \arg \min_{r \in R_G(s, d)} L_G(r) \quad (3.1)$$

3.1.2 Deployment Environment

The P2RPS is assumed to be deployed in a Smart City environment equipped with road-side units (RSUs) and edge servers throughout its network (i.e., integrated into traffic lights and lamp posts along city roads and highways). *Road-Side Units (RSUs)* are resource-constrained low-power computational node devices built up along city roads and highways to continuously collect data on the current traffic condition [27]. It is also assumed that the traffic data gathered by these RSUs and then forwarded to the edge servers which then automatically adjust the edge weights for the underlying road network during operation. *Edge servers* are computational nodes which are considerably more powerful than the resource-constrained RSUs, and are better suited for heavier data processing tasks. This Smart City infrastructure is assumed to provide real-time traffic monitoring with the aim of optimizing resource usage (i.e. roads) and maximizing service utility

for its users (citizens). More importantly, it is also assumed that the P2RPS is operated over previously-deployed infrastructure. Said infrastructure is also assumed to be still able to perform additional functions (i.e. such as distributed route planning) alongside its normal functions.

The primary users of the P2RPS are assumed to be vehicle operators such as personal car owners, taxi drivers, autonomous vehicles, etc. as they are the ones most able to utilize the routes calculated over the urban road network. Pedestrians can, of course, still use the P2RPS to calculate walking routes, but it is typically better for them to rely on public transport for longer distances. Note that both approaches described in this dissertation can readily be adapted to work with public transportation networks — as long as they can be represented by network graphs and combined with the existing road network graph. However, this application is currently outside the scope of this dissertation and reserved for future work instead.

The deployment environment is assumed to be limited to a *service region* defined by a predetermined geographical area of arbitrary shape and size that has fixed bounds. This area is assumed to have comprehensive road network data available such that an RPS can be used to calculate routes within it. This road network is assumed to be represented as a graph that can be partitioned in some arbitrary manner to produce multiple subgraphs henceforth called *partitions*.

3.1.3 Network Architecture

Each partition is assumed to have both RSUs and local edge servers deployed over existing road network infrastructure. Each RSU is able to communicate with vehicles and other RSUs in the same partition, but not with RSUs of other partitions. Edge servers have better processing and data storage capabilities, and work in collaboration with RSUs to process and respond to all queries within a reasonable time. They handle local user queries, divide user queries into routing tasks, distribute local routing tasks to local RSUs, redirect out-of-scope tasks to other partitions, and execute PIR requests. Unlike RSUs, each edge server is also capable of communicating with the edge servers of other partitions.

Partitions are assumed to be connected via mesh network and only the transfer of system-relevant data is considered in this dissertation. Wired fiber links are

assumed to exist between any two devices (e.g., RSUs) within each partition to ensure that data transfer is consistent, and that only transfer rate and transferred data size are relevant. Additionally, the same type of links are assumed to exist between any two edge servers across the entire service region. Data transfer rates are also assumed to be static throughout the duration of operation.

3.1.4 Threat Model

It is assumed that all entities other than the user are potential threats — henceforth called *adversaries* — and that they have some kind of interest in gaining access to the user’s unencrypted route information. Note that no distinction is made between the service providers themselves and malicious third-parties in this model. The kinds of information that can be leaked include the user’s: (1) exact origin, (2) exact destination, and (3) the entirety of the calculated “route” between these two locations.

3.1.5 Privacy-Preservation Mechanisms

We also define *route privacy-preservation mechanisms* to be methods or protocols that grant some level of privacy in terms of the routes recommended to the users of the system. Simply put, it is assumed that the adversary can know that a certain user accessed the system and through which device (i.e. RSU or edge server) the user accessed it, but should not be able to infer any route-related information (as described in Sec. 3.1.4) from it.

3.1.6 Privacy-Preserving Route Planning Service (P2RPS)

A basic RPS is assumed to take *route planning requests* from *users* as its **input**, calculates routes for each request, and sends these *routes* back to the users as its **output**. These requests are assumed to contain information about the origin and destination of the requested route in some form. The approaches described in this dissertation follow the same basic pattern but incorporate privacy-preserving changes. For instance, input route planning requests are broken down into en-

encrypted and unencrypted components, sent at different stages, to reduce the privacy leakage risk. Likewise, route calculation is changed to work with these kinds of requests and produce output routes consisting of encrypted and unencrypted components as well.

3.1.7 Privacy Scope and Limitations

In this work, the primary focus will be solely on preventing user route information from being found out by an adversary based on the user’s interactions with the RPS — more specifically, through the *route planning requests* described in Sec. 3.1.6. Protecting the identities of the users themselves is outside the scope of this work. However, we also acknowledge that allowing user identities to be known can indirectly allow adversaries to discover users mobility patterns — that is, the routes *actually taken* by the user. For example, the SP can likely leverage its city-wide network of RSUs and Edge Servers to detect and track the presence of a certain user connected to the Smart City WAN depending on which endpoints are able to receive broadcast packets from a similar device at which times. In this case, route information is gleaned not from the user’s queries but from other attributes which are associated with the user’s identity (e.g. the MAC addresses of their devices). The approaches in this work are unable to prevent these kinds of attack, so we must consider it as an unfortunate limitation which has to be resolved through additional privacy and security mechanisms.

3.2 Key Idea

The two P2RPS described in this dissertation use PIR to provide strong privacy guarantees by protecting the database representation of the road network graph used to calculate routes. Let us consider a case where a user wants to find a route between two points within one partition, and where said partition is a subgraph of the entire road network graph. During a *pre-processing phase* before the RPS goes live, or during an *update phase* repeated at regularly-scheduled intervals (e.g. every 10 minutes), the edge server for the partition calculates the All-Pairs Shortest Paths (APSP) and stores them in a route database where each route is indexed by its pairs of origin and destination vertices, (s, d) . During the *routing*

phase, the client device encodes the desired origin and destination pair in a PIR request and then send this to the server which has access to the aforementioned route database. The server uses this PIR request to blindly retrieve an encrypted record containing the desired route and send it back to the client device as a PIR response. Finally, the client device can decrypt the PIR response and obtain the desired route. Through PIR, the user is therefore able to retrieve any route between any two locations *within the partition* using a single query (i.e. $O(1)$) with very high privacy.

This simple idea works well for a single partition, but applying the same for an entire road network is infeasible in practice for two reasons: (1) it requires a prohibitively large amount of storage space, and (2) pre-computing APSP information for large road network graphs takes a very long time. Assuming a graph with 10,000 vertices, a longest path length of 20 vertices, and a 1-byte representation for vertex data, the total PIR database size is already be around 2 GB. This is even larger for city-sized road networks such as Osaka City’s which has $\sim 99,000$ vertices (~ 200.8 GB keeping all other parameters same). In short, letting $R_G^* = \{r_G^*(u, v) | u, v \in V\}$ and $L_G^{max} = L_G(\arg \max_{r \in R_G^*} |r|)$, the space complexity for such an approach is $O(|V|^2 \cdot L_G^{max})$. This issue is made even worse by PIR since individual record retrieval times become slower with larger database sizes. This has been verified through preliminary experiments where data retrieval times were observed to scale linearly with database sizes by a constant factor, \mathcal{C}_{pir} , such that accessing a single route would have a time complexity of $O(\mathcal{C}_{pir} \cdot |V|^2) \approx O(|V|^2)$ instead of the expected $O(1)$.

Since time complexity is heavily dependent on space-complexity for PIR-based approaches, existing works [12, 13] have focused on tackling the space complexity problem since pre-processing is assumed to be done offline only once. This is not the case for real-world road networks, however, where traffic conditions can change very quickly. One way to solve this is by distributing route data among several edge servers such that each one only manages vertices for a *distinct* partition. This reduces the time complexity to $O(|V| \cdot \mathcal{N}_p)$ where $\mathcal{N}_p = 1/|P| \cdot \sum_{p \in P} |V_p|$ is the average vertex count per partition, while the per-partition space complexity becomes $O(\mathcal{N}_p \cdot |V| \cdot L_G^{max})$. These databases still need to be kept updated, but it is much easier to do so in a distributed manner. Dividing and distributing the

data, however, does weaken privacy somewhat since the set of route data stored on the *accessed* edge server is assumed known to the adversary. Other measures must therefore be taken to ensure that route data is preserved — which is the primary motivation for developing P2RPS in the first place.

4 Practical PIR-based Route Planning via Multi-Objective Optimization (P2RoP-MO)

In this preliminary approach, we consider a privacy-preserving RPS that has to meet the three following service objectives:

- *Travel Time Accuracy*,
- *Processing Throughput*, and
- *Route Privacy Protection Level*.

Each of these represent the more general objectives of Utility, Performance, and Privacy respectively. *Travel Time Accuracy* shows the ability of the RPS to produce valid and close-to-optimal routes. Near-optimal is defined in terms of the generated route's total travel time compared to that of the optimal route. *Processing Throughput* is measured by the system's ability to process as many user requests within a given time frame. Finally, *Route Privacy Protection Level* ensures that a large part of the user's route is concealed from both external entities and the service provider itself.

Trade-offs exist between these three objectives such that maximizing one comes at the cost of the other two. For instance, increasing the processing throughput means compromising on either travel time accuracy or privacy protection, and vice versa. In this chapter, the goal of optimization is to find the Pareto-optimal sets for the Accuracy-Throughput-Privacy trade-off problem.

Optimization is done for the entire set of route requests within the given time frame instead of individually. This is because processing throughput is considered

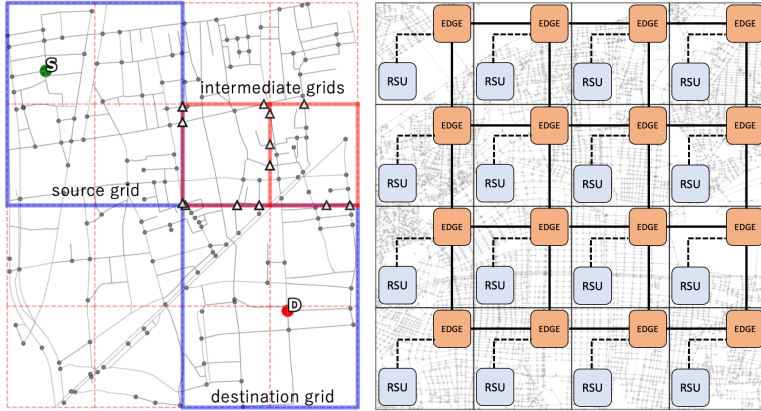


Figure 4.1: (Left) Service Region divided into 4x4 *grids/partitions*. Source and Destination partitions can be a combination of multiple *base partitions* for better privacy. (Right) Each partition has at least one RSU (blue) and edge server (orange). Edge servers form a mesh network.

from the system’s perspective where the processing time of each query can potentially affect all other queries. Assuming a system with finite capacity, routing tasks meant for the same partition are continuously added on to the same queues and executed sequentially. Tasks that take too long to execute may bottleneck other tasks when the partition is operating near maximum capacity. Thus, it is better for the system to maximize the overall processing throughput to make all queries complete within a reasonable time, instead of prioritizing certain queries.

Finally, the constraints of optimization can be relaxed by considering that different users will have different preferences with regards to the objectives they care the most about. Some users might prefer to receive route results faster at the expense of accuracy, while others might prefer more accurate routes with minimal privacy. It, therefore, makes sense to dedicate less computational power on privacy-preservation on users that do not care much about it so that the users that do care about it have more resources to work with. In other words, such user preferences have to be considered by the RPS as well.

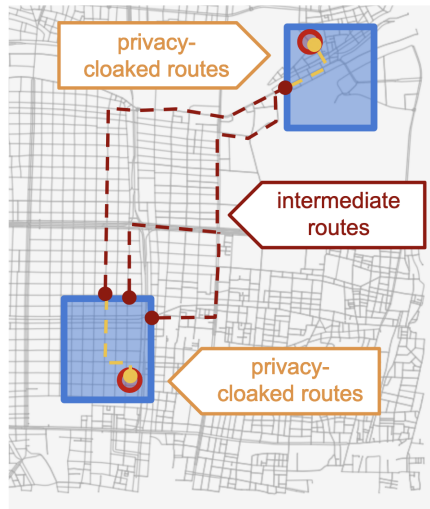


Figure 4.2: Practical route privacy preservation achieved through combining *privacy-cloaked* sections (orange lines) with non-privacy-cloaked or *intermediate* sections (red lines)

4.1 Assumptions

In this section, we consider a service region that has been partitioned into multiple grids of fixed size as shown in Fig. 4.1. These grids will represent the partitions for the sake of the basic PIR-based route planning described in Sec. 3.2. Given an origin and destination location (s, d) , we can consider the partitions, p_o and p_d , as the partitions where the highest-level of privacy is afforded and hence call them the *privacy-cloaked sections* of the route. These are represented by the orange lines in Fig. 4.2. Depending on the size of p_o and p_d , a large section of the route can exist outside this protected area. This section will be called the *intermediate section* of the route, represented by the red lines. While this section has functionally no privacy guarantees, some techniques for increasing the privacy for this area will be proposed in subsequent sections.

It is also assumed that the route planning mechanism will at least need to have two *privacy-cloaked route requests* (one for the origin, and one for the destination), and one *intermediate route request*. Privacy-cloaked route requests, in turn, will need information about specific partitions within the service region. Thus, *partition information requests* (for both origin and destination partitions)

will also be needed. These requests contain information on which set of nodes are belong to the partition for the purpose of finding the correct indices to be used in PIR-based route planning. Finally, *region information requests* are also needed to discover which partitions are available within the service region in the first place. This request contains information on which partitions are available and what their boundaries are as well

4.2 Key Idea

The key idea then is to adaptively select the optimization parameters for each query such that system-level processing throughput, route privacy protection level, and travel time accuracy objectives are achieved. The methods we considered are given by varying two parameters: (1) the number of retained intersection nodes/vertices (r_q), and (2) the size of the privacy-cloaked areas (a_q). The number of nodes can be controlled by selecting a subset of the partition’s vertices to be retained. Less vertices means faster route calculation times since it reduces the database size that the PIR is dealing with. At the same time, however, the produced routes might have longer than optimal travel times and be less private due to the reduction in number of possible routes as well. Increasing the size of the privacy-cloaked area, in contrast, guarantees privacy for a larger section of the route, but greatly impacts route calculation times. Careful selection of the parameters for each route request is therefore crucial to achieving system objectives and satisfying user-level query preferences. To realize this, we formulated a MOO problem and develop a MOGA-based heuristic algorithm to solve it.

4.3 Mathematical Formulation

Let $Q_t = \{q_1, q_2, q_3, \dots\}$ be the input representing the set of all queries received in time window t , while $P = \{p_1, p_2, p_3, \dots\}$ is a system parameter representing all partitions in the service region. For each q , vector \mathbf{p}'_q represents the *candidate partitions* derived from optimal sequence of grids (OSG) [26] where tasks *will* be assigned. Vector \mathbf{r}_q is an input parameter representing the ratio of selected node intersections for the partitions in \mathbf{p}'_q . Vectors \mathbf{v}_q , and \mathbf{e}_q are based on the

subgraphs of said partitions. They represent per-partition base counts of node intersections and edge/road segments. For brevity, the following sections will use the symbol “ \odot ” to denote element-wise multiplication of vectors (and matrices).

4.3.1 Processing Throughput

Processing throughput $\mathcal{P}(q)$ can be defined as a function of the total calculation time of all requests processed by each RSU as mentioned in Sec. 4.1. These include: (1) the region info calculation time, (2) origin and destination partitions’ info calculation times, (3) initial route calculation time, (4) origin and destination partitions’ PIR route calculation times, and (5) the overall communication overhead for the query. To simplify calculations, it is assumed that the level of system load (i.e., simultaneous active queries per RSU) is normal and that none of the RSUs are overloaded. This is formalized and explained further later on alongside the discussion on PIR route calculation in Eqn. 4.3. This assumption must be made because the effect of system load on the processing time result cannot be directly controlled by the optimization parameters: number of retained node intersections r_q , and size of the privacy-cloaked area a_q . This implicitly assumes that there are enough RSUs to properly handle the requests that might be generated from each routing task. Proper handling for this requires load balancing mechanisms which is outside the scope of the optimization discussed in this chapter.

Given normal system load, the calculation time for the *region information request* can be considered a constant as it is not expected to change drastically from one query to another, and can be pre-calculated to further reduce its effect on the processing time. The constant C_{ri} is defined to represent the typical region info calculation time based on preliminary evaluation results.

The *partition information request* calculation time can also be assumed to be constant, and can be pre-calculated by the RSUs independently of routing tasks and should simply be a matter of relaying already existing data. Thus, we also represent this with a constant, C_{pi} . The *total information calculation time* can then be defined as a single constant, $C_{it} = C_{ri} + 2 \cdot C_{pi}$.

The *initial route calculation time*, $R(\mathbf{p}'_q)$, estimates the route calculation time between the query q ’s origin and destination partition pair, \mathbf{p}'_q . Typically, route

calculation time would be proportional to the size and structure of the graph. However, as the use of a pre-trained model based on the optimal sequence of grids (OSG) is used by the proposed system, the time needed to make the initial route decision is only equivalent to the time needed to calculate $\hat{E}(p_s, p_d, \tau_{p_s})$ once. Thus, given normal system load, this is assumed to be equivalent to a simple constant time lookup C_{rc} corresponding to the typical initial route calculation time obtained during preliminary evaluations. This is therefore defined as:

$$R(\mathbf{p}'_{\mathbf{q}}) = ExecTime(\hat{E}(p_s, p_d, \tau_{p_s})) \approx C_{rc} \quad (4.1)$$

The route calculation defined by $R(\mathbf{p}'_{\mathbf{q}})$ and calculated via OSG represents the preliminary route calculation *prior to* adding the privacy-protected sections. Privacy-protected route calculation time is represented by $I(\mathbf{p}'_{\mathbf{q}})$ or $I(\mathbf{v}'_{\mathbf{q}}, \mathbf{x}_{\mathbf{q}})$, as defined in Eq. 4.2, which is the time taken by private route retrieval using PIR.

$$I(\mathbf{p}'_{\mathbf{q}}) \leftrightarrow I(\mathbf{v}'_{\mathbf{q}}, \mathbf{x}_{\mathbf{q}}) = (C_{search} + C_{load}) \cdot (\mathbf{x}_{\mathbf{q}} \cdot \mathbf{v}'_{\mathbf{q}})^2 \quad (4.2)$$

where $\mathbf{v}'_{\mathbf{q}} = (\mathbf{v}_{\mathbf{q}} \odot \mathbf{r}_{\mathbf{q}})$ is the *actual* selected node count for each partition, $\mathbf{x}_{\mathbf{q}}$ is an indicator vector with a value of 1 if the partition uses PIR, while C_{search} and C_{load} are the PIR search time scaling factor and PIR database load time scaling factor respectively. The vector $\mathbf{x}_{\mathbf{q}}$ ensures that PIR is done only on the all-pairs shortest path databases of the cloaked areas (i.e., the origin and destination partitions), and keeps $I(\mathbf{p}'_{\mathbf{q}})$ low by minimizing resource-intensive PIR operations.

Since PIR requires a lot of computation power, we also constrain RSUs in each partition to have a capacity in terms of the number of PIR requests to be processed at each time slot t . This constraint is represented as

$$\forall p \in P, |\{q \mid q \in Q_t, src(q) = p \vee dst(q) = p\}| \leq Cap(p) \quad (4.3)$$

where $Cap(p)$ is the computation capacity of partition p , and $src(q)$ and $dst(q)$ are source and destination partitions in q . The overall communication delay for a query is then:

$$C(\mathbf{p}'_{\mathbf{q}}) = F(\mathbf{l}_{\mathbf{q}}) \odot \zeta(\mathbf{p}'_{\mathbf{q}}) \quad (4.4)$$

where $\zeta(\mathbf{p}'_{\mathbf{q}})$ estimates the output size in bytes, and $F(\mathbf{l}_{\mathbf{q}})$ estimates the per-link transfer time per byte (i.e., inverse of bandwidth) where vector $\mathbf{l}_{\mathbf{q}}$ is represented

by an array of tuples corresponding to each partition in $\mathbf{p}'_{\mathbf{q}}$ and the next partition after it. The processing throughput for the candidate partitions $\mathbf{p}'_{\mathbf{q}}$ associated with a query q is defined as:

$$\mathcal{P}(q) = \frac{1}{\mathbf{1}^{\mathbf{T}} \cdot [R(\mathbf{p}'_{\mathbf{q}}) + I(\mathbf{p}'_{\mathbf{q}}) + C(\mathbf{p}'_{\mathbf{q}})] + C_{it}} \quad (4.5)$$

which represents the processing throughput (number of queries per unit time) before normalization. Fig. 4.3 (Right) shows the effect of node intersection ratio r_q on the normalized $P(q)$.

4.3.2 Privacy Protection Level

To preserve location privacy, PIR is applied on critical sections of the route which, to minimize computation costs, are in the vicinity of the origin and destination partitions only. Location privacy then depends on the chosen size of the cloaked area around said sections, and the selected node intersection ratio for the partitions within them. This decreases re-identifiability of exact locations by retaining more intersections and increasing the coverage of the privacy-protected sections. Privacy protection level for a query is formally defined as

$$\mathcal{V}(q) = \mathbf{1}^{\mathbf{T}} \cdot [(\mathbf{a}_{\mathbf{q}}^{\gamma} \odot \mathbf{v}'_{\mathbf{q}}) \cdot C_{lp}] \quad (4.6)$$

where $\mathbf{a}_{\mathbf{q}}$ is the estimated area of each candidate partition, γ is a system parameter controlling the effect of $\mathbf{a}_{\mathbf{q}}$, $\mathbf{v}'_{\mathbf{q}}$ is the actual selected node count for each candidate partition, and C_{lp} is the location privacy normalization factor. C_{lp} is obtained from the inverse of the mean number of intersections for all base partitions in the service region, while γ is determined empirically by choosing a value that prevents $\mathcal{V}(q)$ from easily outweighing other objectives when the per-partition node count $\mathbf{v}_{\mathbf{q}}$ is naturally large (e.g., in cities).

4.3.3 Travel Time Accuracy

To assess the quality of any section of the route, travel time accuracy is defined as a metric. It is formally defined as the ratio between the estimated travel time for a candidate solution and that of an optimal solution as shown below:

$$\mathcal{A}(q) = \mathbf{1}^{\mathbf{T}} \cdot M(\mathbf{v}_{\mathbf{q}}, \mathbf{r}_{\mathbf{q}}) \quad (4.7)$$

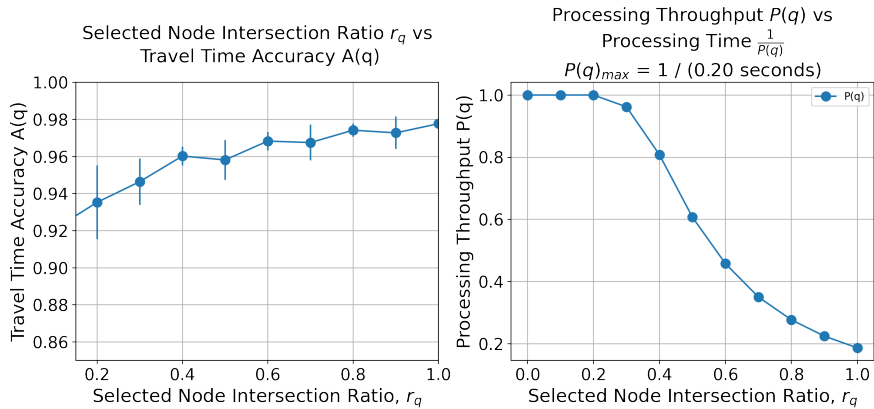


Figure 4.3: (Left) Mean travel time accuracy $A(q)$ and Selected Node Intersection Ratio r_q ; (Right) Processing Throughput $P(q)$ to actual processing time $\frac{1}{P(q)}$ (in seconds) with empirically-found values for $P(q)_{max} = 5.0$ (five queries every second) and $\lambda = -1.6$.

where \mathbf{v}_q is the base node count per partition, and \mathbf{r}_q is the selected node intersection ratio per partition. $M(\mathbf{v}_q, \mathbf{r}_q)$ estimates the difference from the optimal travel time through the candidate partitions given \mathbf{v}_q modified by \mathbf{r}_q . In this paper, $M(\mathbf{v}_q, \mathbf{r}_q)$ is modelled based on preliminary analysis of route travel times in the service region. The travel times of 500 optimal routes are compared against 500 candidate routes after randomly removing a certain percentage of candidate node intersections within each partition, simulating varying levels of \mathbf{r}_q until nearly all candidates are removed. The results are shown in Fig. 4.3 (Left). Based on this, the function can be modeled as:

$$M(\mathbf{v}_q, \mathbf{r}_q) \approx M(\mathbf{r}_q) = \frac{r_q}{e^{\lambda \cdot (1-r_q)}} \quad (4.8)$$

where λ is the empirically-determined steepness of the curve based on the preliminary results obtained from Fig. 4.3 (Left).

Table 4.1: Individual Objective Score Explanations

	P(q) Measure: queries per second	V(q) Measure: candidate nodes in area	A(q) Measure: convergence to optimal travel time
0.1	1 queries/sec	12.2 nodes	10%
0.5	2.5 queries/sec	61 nodes	50%
1.0	5 queries/sec	122 nodes	90%

4.3.4 Objective Functions

The individual objective scores for a set of queries, Q_t , are computed using the following functions

$$P(Q_t) = \frac{\sum_{q \in Q_t} \mathcal{P}(q) \cdot H_P(q)}{|Q_t|} \quad (4.9)$$

$$V(Q_t) = \frac{\sum_{q \in Q_t} \mathcal{V}(q) \cdot H_V(q)}{|Q_t|} \quad (4.10)$$

$$A(Q_t) = \frac{\sum_{q \in Q_t} \mathcal{A}(q) \cdot H_A(q)}{|Q_t|} \quad (4.11)$$

where $H_*(q)$ computes the numerical equivalents of the user’s preferences. Individual objective results are normalized against the empirically-determined values in Table 4.1. The goal is to maximize $P(Q_t)$, $V(Q_t)$, and $A(Q_t)$, subject to the constraint in Eq. (4.3) about the maximum number of PIR requests processed in each partition. Thus, the objective function is

$$\text{Maximize } (P(Q_t), V(Q_t), A(Q_t)) \text{ s.t. (4.3)} \quad (4.12)$$

The above optimization problem belongs to the NP-hard class. The disjunctively constrained knapsack problem [43], which is known to be NP-hard, is a special case of our problem. This can be shown by supposing that there is only one partition g with computation capacity $Cap(g)$ corresponding to a knapsack capacity, and the tuples of possible choices of \mathbf{a}_q and \mathbf{r}_q and the tuple of individual objective values $P(q)$, $V(q)$, and $A(q)$ are regarded as items, where only one choice among many possible choices of \mathbf{a}_q and \mathbf{r}_q can be made for each q and is represented as the disjunctive constraints.

4.4 Multi-objective Optimization

Algorithm 1: NSGA-II Based Optimization

Input: Set of queries Q_t
Output: Set of solutions Y_t

```

1 begin
   /* 1: Generate parent population */
2    $\mathcal{R}_0 \leftarrow \text{GenerateParents}(Q_t);$ 
   /* 2: Generate Offspring */
3    $\mathcal{F}_t \leftarrow \text{NonDominatedSort}(\mathcal{R}_0);$ 
4    $\mathcal{S}_0 \leftarrow \text{GenerateOffspring}(\mathcal{F}_0, \mathcal{R}_0);$ 
   /* 3: Evolve solutions */
5    $k \leftarrow 0;$ 
6   while  $k < \text{MAX\_ITERATIONS}$  do
7      $\mathcal{R}_{k+1} \leftarrow \text{NSGAII Sort}(\mathcal{R}_k, \mathcal{S}_k);$ 
8      $\mathcal{C}_{k+1} \leftarrow \text{Selection}(\mathcal{R}_{k+1});$ 
9      $\mathcal{M}_{k+1} \leftarrow \text{Crossover}(\mathcal{C}_{k+1});$ 
10     $\mathcal{S}_{k+1} \leftarrow \text{Mutation}(\mathcal{M}_{k+1});$ 
11     $k \leftarrow k + 1;$ 
   /* 4: Extract solutions */
12   $Y_t \leftarrow \text{ExtractSolutions}(\mathcal{S}_k);$ 
13  return  $Y_t$ 

```

A multi-objective approach was used to find diverse sets of non-dominated candidate solutions for the aforementioned problem, using NSGA-II as the base algorithm, with the full method being described in Algorithm 1. An initial parent population \mathcal{R}_0 is generated based on Q_t with initially-random values for two sets of \mathbf{r}_q and \mathbf{a}_q (for the origin and destination partitions each). \mathcal{R}_0 is then sorted in a non-dominated manner as per NSGA-II and used to generate an initial offspring population, \mathcal{S}_0 . Solutions are then evolved for several iterations. Every iteration, regular NSGA-II non-dominated sorting is first done followed by selection, crossover, and mutation. The final sets of solutions are extracted from the final offspring population, \mathcal{S}_N .

To further increase solution diversity, a modified fitness proportionate selection algorithm is developed. Four selection fitness criteria were used: balanced, processing throughput based, privacy protection level based, and travel time ac-

Algorithm 2: Modified Fitness Proportionate Selection

Input: Query set Q_t , Population R_k , Pairs to select N_{select}

Output: Pairs of individuals for Crossover C_k

```
1 begin
2   Initialize  $b_{select}, p_{select}, v_{select}, a_{select} \leftarrow []$ ,  $C_k \leftarrow \{\}$ ;
3   foreach  $r \in R_k$  do
4      $b_{select}[r] \leftarrow Mean(P(r.Q_t) + V(r.Q_t) + A(r.Q_t))$ ;
5      $p_{select}[r] \leftarrow P(r.Q_t)$ ;
6      $v_{select}[r] \leftarrow V(r.Q_t)$ ;
7      $a_{select}[r] \leftarrow A(r.Q_t)$ ;
8   for  $i \in [0, \dots, N_{select}]$  do
9      $pool \leftarrow Random(\{b_{select}, p_{select}, v_{select}, a_{select}\})$ ;
10     $c_0, c_1 \leftarrow pool[Random(R_k)], pool[Random(R_k)]$ ;
11     $C_k \leftarrow (c_0, c_1)$ ;
12  return  $C_k$ 
```

curacy based. The aim is to increase the chance of solutions with the best results for each of these objectives being retained each iteration. This is described in Algorithm 2.

4.5 Evaluation

In this section, the proposed multi-objective optimization method is validated via a case study based on simulating the road network of the central region of Osaka, Japan. The resulting parameters and its effect on the overall system are then discussed.

4.5.1 Mobility and Vehicle Trip Data

In this experiment, the road network of the Chuo and Kita wards — the most populous areas in Osaka, Japan [44] — are used to simulate the effectiveness of the optimization method. Mobility data is generated via simulations using VISSIM and VISUM traffic simulation software [45] over a three-hour time period from 9AM to 12PM. The simulated road network was then divided into 64 grids, where each grid is assumed to be an independent partition having at least one RSU. To

simulate vehicle routing, it is given an Origin-Destination (OD) matrix [46] with vehicles dispatch frequencies from various zones in the city. All experiments were initiated with the same OD matrix. Four hours of traffic data were generated consisting of 50,894 vehicles across an area $20km^2$ of Osaka city. The mobility data consists of vehicle flow speed per road segment gathered at a time resolution of every 10 seconds. Several OD pairs were selected from this matrix and used to generate the set of queries, Q_t , used as inputs to the optimization algorithm.

4.5.2 NSGA-II Configuration

For the NSGA-II algorithm combined with the modified selection method described in Algo. 2, a Simulated binary *crossover* (SBX) with a crossover probability of $p_c = 0.5$ and a distribution index of $\eta_c = 15$ was used. As is typical for real-coded NSGA-II, the mutation method used was polynomial *mutation* with a mutation probability of $p_m = 0.5$ and a distribution index of $\eta_m = 30$. The algorithm was run for 10 generations with 85 offspring for a total population size of 90 for each generation.

4.5.3 Experiment Setup

The performance of the proposed optimization algorithm is compared against two other well-known bounded optimization algorithms: SLSQP [47] and L-BFGS-B [48]. As both are single-objective algorithms, an overall utility score, $U(Q_t)$, based on the averaged individual objective scores was used as their objective function.

A set of queries is then generated as mentioned in the previous section and then assigned a set of preference values $(H_P(q), H_V(q), H_A(q))$. For simplicity, only one of the three objectives can be designated as HIGH preference while the two others are automatically set to LOW preference. The generalized preference function is described as

$$H_*(q) = \begin{cases} 0.8 & \text{if HIGH preference for } * \\ 0.1 & \text{otherwise} \end{cases} \quad (4.13)$$

Up to 100 queries are generated per query set with the assumption that approx. 100 queries arrive per 5-minute time frame. Note that this is only to accommodate

the execution times of the slowest optimization algorithm and, in practice, should be lower. Up to 100 different query sets are randomly-generated with different seeds and transformed into usable inputs for each algorithm, which are then executed for a total of 100 runs. The final solutions are evaluated in terms of their overall performance accounting for per-query individual objective scores ($\mathcal{P}(q)$, $\mathcal{V}(q)$, and $\mathcal{A}(q)$), and execution times.

Table 4.2: Other Parameter Values for the Experiment

Parameter	Value	Description
$ Q_t $	100	Number of Queries
γ	0.5	Partition area control parameter
λ	-1.6	Accuracy control parameter
C_{ri}	1.64 secs	Region info calc time
C_{pi}	16.25 secs	Partition info calc time
C_{it}	34.14 secs	Total info calc time ($C_{it} = C_{ri} + 2C_{gi}$)
C_{rl}	6.0 secs	Per partition OSG route lookup time
C_{search}	1.041×10^{-2}	PIR search time factor
C_{load}	9.369×10^{-3}	PIR load time factor
C_{comp}	1×10^{-12}	Computation power factor
C_{lp}	122	Location Privacy factor

For the other parameters besides \mathbf{a}_q and \mathbf{r}_q , the predetermined constant values are summarized in table 4.2. C_{search} , C_{load} , and C_{comp} were determined by conducting element retrieval and database loading tests over SealPIR [29] v3.2.0 using the default Brakerski/Fan-Vercauteren (BFV) homomorphic encryption scheme, with a database upper bound of $N = 2^{16}$ elements, a BFV plaintext modulus of $\log(t) = 12$, and the SealPIR-specific dimensionality factor, $d = 2$. λ is based on preliminary experiments described in Sec. 4.3.3. C_{lp} and γ are empirically determined as described in Sec. 4.3.2. All experiments were performed on a Macbook Pro with a 2.9 GHz Intel Core i7 processor and 16 GB DDR3 memory.

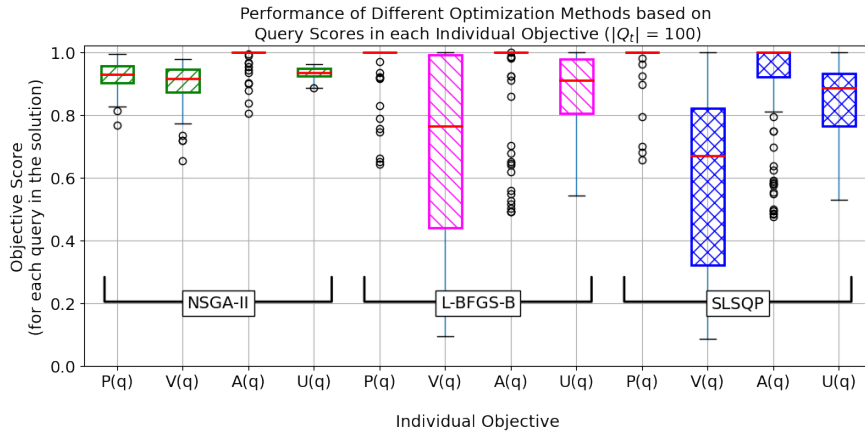


Figure 4.4: Performance of NSGA-II, L-BFGS-B, and SLSQP in optimizing the individual objectives ($P(q)$, $V(q)$, $A(q)$) and individual utility ($U(q)$)

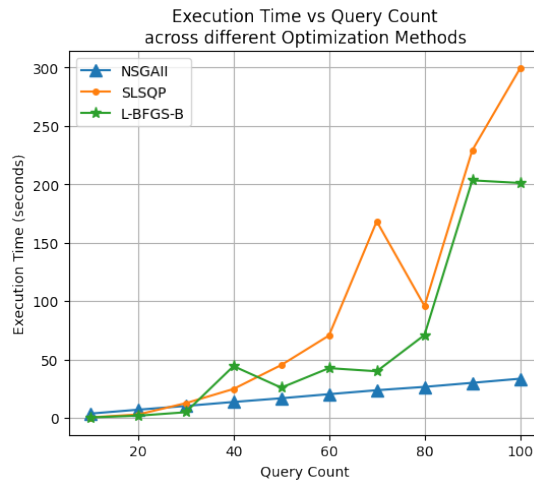


Figure 4.5: Comparison of execution times between different algorithms

4.5.4 Experiment Results

The average per-query individual objective scores (for $Q_t = 100$) shown in Fig. 4.4 suggest that the proposed optimization method achieves higher privacy protection ($V(q)$) at the cost of slightly lower processing throughput ($P(q)$) scores for half of all queries. The balanced utility per query ($U(q)$) and travel time accuracy ($A(q)$) are about the same throughout.

A large difference also exists between execution times as shown in Fig. 4.5 where SLSQP and L-BFGS-B both grow rapidly with more queries while NSGA-II grows slowly and linearly. Since this must be run regularly, shorter execution times are better for practicality and service quality.

The fitness proportionate selection method also allows NSGA-II to produce a diverse set of solutions as shown in Table 4.3, which highlights the trade-offs of focusing on one objective over others. In this example case, processing preference results in reduced privacy and accuracy, while accuracy preference trades away some processing throughput on the origin partition for higher accuracy and privacy. In this case, privacy protection preference is the middle-ground, maximizing both processing throughput and accuracy at the slight cost of privacy protection. Note that these are only the best solutions for each objective among 90 other solutions produced after 10 generations of execution, making the choice of final solution flexible enough to be tailored based on the actual distribution of user preferences among the input queries.

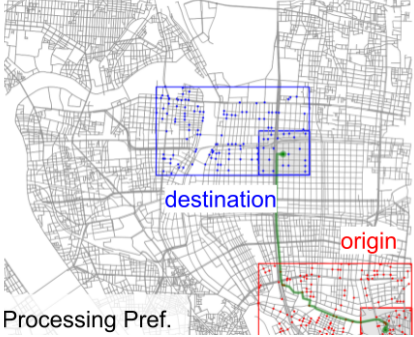
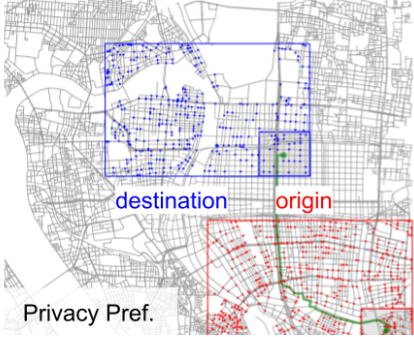
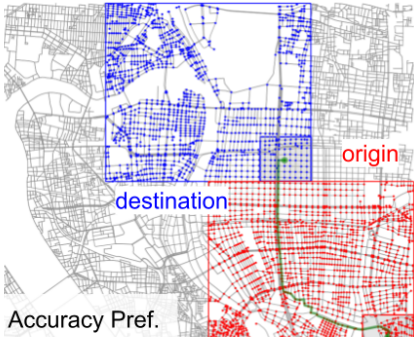
4.6 Summary

In this chapter, an adaptive user-centric approach called Privacy-Preserving Route Planning via Multi-objective Optimization (P2RoP-MO) was proposed and evaluated. The approach protects critical portions of the routes around the origin and destination locations using PIR combined with a dynamic Spatial Cloaking technique. However, as Privacy as an objective does not exist in isolation, and must therefore be considered alongside other objectives — in particular, Utility and Performance. A multi-objective optimization (MOO) problem was therefore formulated to encapsulate the processing throughput, privacy protection, and route accuracy objectives for each batch of route queries received by the system.

Afterwards, a NSGA-II-based multi-objective optimization algorithm was developed to solve it. Through simulations it was shown that the approach obtained better trade-offs between the aforementioned objectives than other MOGAs.

However, several critical issues remain. Firstly, the optimization step is rather slow as it takes 118.98 secs (30.15 secs when done in parallel) to run NSGA-II for around 10 generations on a single batch of queries. This problem is compounded by the second issue which is that optimization essentially delays *all of the queries* in the same batch. Essentially, ~ 120 seconds of optimization time would be added on top of the expected processing time for each query making it impractical for actual route planning. The final issue is that the routes in between the protected areas are not protected at all by design to reduce processing times, etc. As a consequence, this gives the adversary additional information that can ultimately be used to reveal the user's entire route. Thus, an improved approach that addresses all these issues is presented in the following chapter.

Table 4.3: Variations in solutions (a_q and r_q) for one example query

	Origin	Destination	Visualization
a_q	~ 4.87 grids/partitions		 <p>Processing Pref.</p>
r_q	20.58% of nodes		
$P(q)$	5 queries/sec	5 queries/sec	
$V(q)$	40 nodes	91 nodes	
$A(q)$	73.34%	73.34%	
a_q	~ 11.01 grids/partitions		 <p>Privacy Pref.</p>
r_q	32.30% of nodes		
$P(q)$	5 queries/sec	5 queries/sec	
$V(q)$	122 nodes	96 nodes	
$A(q)$	90.0%	90.0%	
a_q	~ 12.55 grids/partitions		 <p>Accuracy Pref.</p>
r_q	92.78% of nodes		
$P(q)$	1.45 queries/sec	5 queries/sec	
$V(q)$	122 nodes	122 nodes	
$A(q)$	90.0%	90.0%	

5 HPRoP: Hierarchical Privacy-Preserving Route Planning

In this chapter, we consider an improved approach called **Hierarchical Privacy-Preserving Route Planning (HPRoP)** that addresses the critical issues and vulnerabilities with P2RoP-MO while also improving Privacy, Utility, and Performance. This is done by using a different road network partitioning method and a novel hierarchical route planning algorithm as will be discussed in great detail throughout this chapter. In addition to this, a pair of new privacy metrics were also formulated in order to more comprehensively describe route privacy beyond that of “partial” route privacy in P2RoP-MO. A comprehensive evaluation of HPRoP is then presented towards the end of this chapter.

5.1 Models and Assumptions

This section presents the assumptions and mathematical models related to the road network graph’s partitioning and the “approximate” routes that can be derived from it.

5.1.1 Road Network Partitioning Model

Most modern route planning algorithms can already deal with very large road networks, but typically do not incorporate route privacy protection mechanisms out of the box, as they tend to significantly increase processing and communication overhead as mentioned in Sec. 2.3. However, if the route planning task can be divided, then the additional processing cost incurred by the privacy mechanism can be distributed across multiple devices instead and ultimately improve

Table 5.1: Summary of Symbols used in Sec. 5.1

Symbol	Description
$G = (V, E)$	Road network graph with road segments E and intersections V
$l_G(u, v)$	Weight of a road segment having endpoints (u, v)
s, d	Source (s) and Destination (d) vertices
$r_G(s, d)$	Path/route between s and d
$L_G(r_G(s, d))$	Total weight of path/route $r_G(s, d)$
$R_G(s, d)$	All possible paths between s and d
$\rho_G(s, d)$	<i>Shortest path</i> between s and d
$G_P = (P, C)$	Partition graph derived from G with the set of all partitions P and the connections between them C
$p = (V_p, E_p)$	A partition (i.e., a subgraph of G) consisting of road segments E_p and endpoints V_p within it
NB_{p_u}	Set of all neighboring partitions for partition p_u
ℓ	Partition level
\mathcal{L}	Maximum partition level ($0 \leq \ell \leq \mathcal{L}$)
P^ℓ	Subset of partitions in P at level ℓ
p_i^ℓ	A partition belonging under P^ℓ with a given index i
x_u	Representative vertex for the partition p_u
$c_{p_u p_v}$	A connection between partitions p_u and p_v through shortest path $\rho_G(x_u, x_v)$
$dist_G(u, v)$	<i>Shortest path</i> distance between u and v in G
$\mathcal{D}(p)$	Database of shortest paths for partition p
$\mathcal{C}(\rho_0, \dots, \rho_n)$	Arbitrary route combination heuristic
$r_G^*(u, v)$	Approximate shortest path between u and v
$\alpha(r_G^*(u, v))$	Optimal route approximation metric

RPS performance as mentioned in Sec. 3.2. The assumption, of course, is that the road network has already been divided into different *partitions*.

An arbitrary partitioning of the road network graph G is represented by a separate *partition graph* $G_P = (P, C)$ where the vertices P represent partitions and the edges C represent connections between them. Each *partition* $p \in P$ is a subgraph $p = (V_p, E_p)$ such that $V_p \subseteq V$ and $E_p \subseteq E$. Each connection $c_{p_u p_v} \in C$

is a shortest path $\rho_G(x_u, x_v)$ between the *representative vertices* $x_u, x_v \in V$ of any two neighboring partitions $p_u, p_v \in P$ where $x_u \in V_{p_u} \wedge x_v \in V_{p_v}$. Two partitions are considered “neighbors” if $\exists e = (u, v)$ s.t. $e \in E \wedge u \in V_{p_u} \wedge v \in V_{p_v}$. Finally, the set of all *neighboring* partitions for p_u is defined as $NB_{p_u} = \{p_v | p_v \in P \wedge (c_{p_u p_v} \in C \vee c_{p_v p_u} \in C)\}$

Representative vertices are ideally chosen to minimize the shortest path distance to all other vertices in the partition. Vertices with high graph centrality are good candidates, but our preliminary experiments show that it is viable to simply choose the vertex closest to the geospatial average of the coordinates of each partition’s vertices with no impact on routing performance. Thus, representative vertices were chosen via this method. It then follows that $l_{G_P}(p_u, p_v) = L_G(\rho_G(x_u, x_v))$. For simplicity, the shortest path distance between any $u, v \in G$ is represented by $dist_G(u, v)$, while $dist_{G_P}(p_u, p_v)$ refers to the shortest path distance between their *containing partitions*, $p_u, p_v \in P$.

Partitions are defined in a hierarchical manner with the maximum partition level being \mathcal{L} , and the subset of partitions for each level $\ell < \mathcal{L}$ denoted as $P^\ell \subseteq P$ such that $P^\ell \subset P^{\ell-1}$ for $\ell > 0$. A partition may also be defined as p_i^ℓ where ℓ is the partition’s level and i is the partition’s index at that level given $P^\ell = \{p_0^\ell, p_1^\ell, \dots, p_n^\ell\}$. The base level partition set P^0 contains only one partition/subgraph $p_0^0 = G$ at $\ell = 0$. Conversely, p_0^0 might be composed of several smaller partitions p_0^1, p_1^1, p_2^1 , and p_3^1 at $\ell = 1$ such that all of them are subgraphs of p_0^0 , and so on. For clarity, partition levels will henceforth be referred to by their position (i.e., higher or lower) instead of their subgraph’s size. Each partition p^ℓ has three sets of shortest path data as depicted in Fig. 5.1: (1) the shortest paths *within* the partition p^ℓ (black edges), (2) the shortest paths to its *neighboring* partitions, NB_{p^ℓ} (blue edges), and (3) the shortest paths between the *containing* partition, $p^{\ell-1}$, to its own neighbors, $NB_{p^{\ell-1}}$ (green edges). This database of shortest paths is represented by $\mathcal{D}(p^\ell)$.

5.1.2 Approximate Shortest Path Model

Our approach relaxes the shortest path problem by accepting approximate shortest paths between two areas (in this case, partitions) containing s and d in place of the exact shortest path between the two points. This, in turn, reduces the

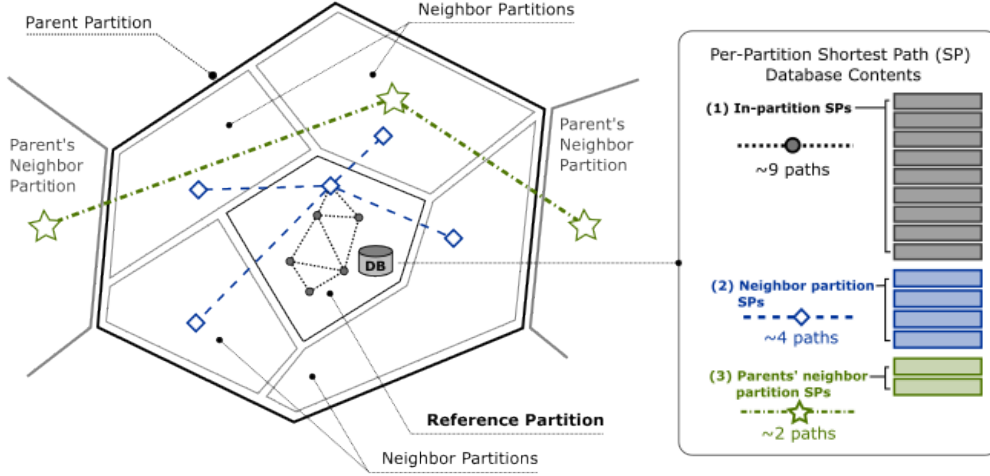


Figure 5.1: Types of shortest paths stored by each partition

number of queries required to obtain a route (hence, faster route completion times) at the cost of potentially having slightly longer paths. These are formally defined here as follows. Given an arbitrary $s, d \in V$, the exact shortest path $\rho_G(s, d)$ rarely coincides with the shortest path $\rho_{G_P}(x_s, x_d)$ where x_s and x_d are the representative vertices in the same partitions as s and d , respectively. This is because only routes between representative vertices of partitions in P can be produced from G_P , and it is highly likely that $s \neq x_s$ or $d \neq x_d$. However, it is still possible to “complete” the route by adding in the missing start and end sections. Letting $\mathcal{C}(\rho_1, \dots, \rho_n)$ be a route combination heuristic, the simplest “completed” route would be:

$$r_G^*(s, d) = \mathcal{C}(\rho_G(s, x_s), \rho_{G_P}(x_s, x_d), \rho_G(x_d, d)) \quad (5.1)$$

This can be generalized further by replacing $\rho_{G_P}(x_s, x_d)$ as follows:

$$r_G^*(s, d) = \mathcal{C}(\rho_G(s, x_s), \rho_{G_P}(x_1, x_2), \dots, \rho_{G_P}(x_{n-1}, x_n), \rho_G(x_d, d)) \quad (5.2)$$

where $x_1 = x_s$ and $x_n = x_d$. In this work, these kinds of combined paths are designated as *approximate shortest paths*, and their quality is measured based on how well they approximate their counterpart exact shortest paths. This metric is defined as the *optimal route approximation*:

$$\alpha(r_G^*(s, d)) = \frac{l_G(r_G^*(s, d))}{dist_G^*(s, d)} \quad (5.3)$$

where $l_G(r_G^*(s, d))$ is the total weight of the approximate shortest path, and $dist_G(s, d)$ is the total weight of the exact shortest path in G .

5.1.3 Assumptions

In addition to the general assumptions mentioned in Sec. 3.1, this approach adds several more. First, it is assumed that the road network can be divided into smaller subgraphs recursively for a number of times to produce a hierarchy of partitions as per Sec. 5.1.1. Each partition at the same level is also assumed to have almost the same number of vertices in order to guarantee a minimum level of privacy for the entire RPS. The rationale here is that having the same number of vertices makes it harder to determine the source and destination partitions simply by observing the PIR retrieval times for an external adversary. Unfortunately, it is impossible to guarantee that this would apply to adjacent partitions *across different service regions* because road networks inherently vary in density by a lot in the real world. For example, the road network of a city may be significantly different from the suburban areas that typically border them. Because of this, we have to assume that there is only one service region and that all route planning must occur within its confines. Each highest-level partition is assumed to be handled by a distinct physical or virtual device for the purpose of the RPS. Additionally, it is assumed that each partition can calculate, build, and maintain its own database of shortest paths $\mathcal{D}(p)$ independent of its other tasks. This per-partition database is assumed to be queryable by users in a privacy-preserving manner through PIR.

5.2 Key Idea

HPRoP expands upon the key idea behind PIR-based P2RPS first presented in Sec. 3.2 but in such a way that the privacy issue encountered in P2RoP-MO relating to unprotected intermediate routes is avoided. As mentioned before, the basic idea was to divide up the road network graph into several smaller partitions to reduce the time complexity of PIR-based route planning. However, when the smaller partitions themselves are dense subgraphs (such as in an urban setting), time complexity may still be an issue. P2RoP-MO attempted to solve this by

flexibly choosing between larger and smaller grids and reducing nodes within the partitions but was limited only to protecting “partial” routes. Recognizing this, it then becomes clear that the route planning algorithm or method itself must be modified in order to achieve the desired objectives. In this section, we discuss two modifications to Dijkstra’s algorithm that will allow it to function in a PIR-based RPS with a pre-partitioned network graph. These two algorithms serve as guides for building an intuition towards the hierarchical route planning algorithm that will be presented in Sec. 5.4.

Table 5.2: Summary of Time and Space Complexity for Sec. 5.2

	Space Complexity	Time Complexity
<i>Naive</i>	$O(V ^2 \cdot L_G^{max})$	$O(V ^2)$
<i>EPR-D</i>	$O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}_p^{adj}])$	$O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}_p^{adj}] \cdot [V + E \log V])$
<i>APR-D</i>	$O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot R_G^{*,max})$	$O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot [(P + C \log P) + 2])$

5.2.1 Exact Partial Region Dijkstra’s Algorithm (EPR-D)

As mentioned at the end of Sec. 3.2, dividing the road network into partitions allows a rudimentary PIR-based RPS to be developed such that time complexity becomes $O(|V| \cdot \mathcal{N}_p)$ where $\mathcal{N}_p = 1/|P| \cdot \sum_{p \in P} |V_p|$ is the average vertex count per partition, while the per-partition space complexity becomes $O(\mathcal{N}_p \cdot |V| \cdot L_G^{max})$. However, this still presents a problem for larger partitions in urban road networks which can be very dense, leading to high time and space complexity. This space complexity problem can be mitigated further by storing only edge weights between adjacent vertex pairs as this is the minimum information needed by Dijkstra’s algorithm. This is also known as the *adjacency matrix* representation which readily maps into a database which can then be used with *any* PIR scheme. We designated this PIR-adapted approach as **Exact Partial Region Dijkstra’s Algorithm (EPR-D)**, since it simply partitions and distributes graph data across several edge servers, and produces exact shortest paths. A notable disadvantage is its use of separate PIR queries to retrieve information for each vertex since the original algorithm tends to scan a lot of vertices which can make route completion

times very long. It is also possible to reidentify routes based on the sequence of edge servers queried by the user. This is examined further in Sec. 5.3. This is reflected in its average time complexity of $O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}_p^{adj}] \cdot [|V| + |E| \log |V|])$ where $\mathcal{N}_p^{adj} = 1/|P| \cdot \sum_{p \in P} |V_p^{adj}|$ is the average number of vertices adjacent to *other* partitions, in turn, represented by $V_p^{adj} = \{v | v \notin V_p \wedge [(u, v) \in E \wedge u \in V_p]\}$. Meanwhile, its average per-partition space complexity is $O(\mathcal{N}_p \cdot [\mathcal{N}_p + \mathcal{N}_p^{adj}])$. This is summarized in Table 5.2.

5.2.2 Approximate Partial Region Dijkstra’s Algorithm (APR-D)

A possible solution to the time complexity issue is by allowing the RPS to produce *approximate shortest paths* instead of exact shortest paths. This can be done as follows: (1) calculate an approximate route between source and destination partitions p_s and p_d using Dijkstra’s algorithm over partition graph G_P , (2) retrieve *subpaths* to both s and d from *within* their respective partitions via database lookup, and (3) merge all obtained paths to form the final route. We designated this approach as **Approximate Partial Region Dijkstra’s Algorithm (APR-D)**, since it produces approximate routes instead of exact ones. While it is significantly faster than EPR-D, it requires more space (since full routes are stored). It also has weaker privacy guarantees since routes between distant areas tend to follow the same intermediate route as will be expanded upon in Sec. 5.3. Since the database has to store complete routes, the space complexity is larger than EPR-D’s being at $O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot R_G^{*,max})$ where $\mathcal{N}_c = 1/|P| \cdot \sum_{p \in P} |C_p|$ is the average number of external connections from each partition. The different stages have different time complexities, with the first stage having $O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot [|P| + |C| \log |P|])$ and the second stage having $O(2 \cdot \mathcal{N}_p^2 + \mathcal{N}_c)$. The combined time complexity is then $O([\mathcal{N}_p^2 + \mathcal{N}_c] \cdot [(|P| + |C| \log |P|) + 2])$ which is significantly less than that of EPR-D since $|P| \ll |V|$. The complexity is summarized in Table 5.2.

5.3 Privacy Metrics

To evaluate the effectiveness of a privacy-preserving PIR-based RPS, objectively quantifiable privacy metrics must first be established in the RPS context. Thus, the two models presented in this section jointly characterize the privacy of the different kinds of information that can be leaked by an RPS in Sec. 5.1.3.

Table 5.3: Summary of Symbols used in Sec. 5.3

Symbol	Description
$\Omega(s, d)$	Endpoint location privacy metric
$\mathcal{R}_G(u, v)$	Arbitrary routing mechanism operating on some graph G
$Q^{s,d}$	Query sequence used to obtain a route from s to d
Q^*	An arbitrary query sequence for no specific route
$\Phi(Q^*)$	Route privacy metric for some candidate query sequence Q^*
$P_{s,d}$	Partition sequence derived from some query sequence $Q^{s,d}$
$k(Q_{s,d}, Q^*)$	Indicator function for checking if Q^* can replace $Q_{s,d}$ and vice versa
V_X	A subset of V containing only the representative vertices

5.3.1 Endpoint Location Privacy Model

As mentioned in Sec. 5.1.1, the partition database is used to store the shortest paths for each partition, and is queried privately using PIR in our approach. The queried partitions are assumed to be *knowable* by adversaries, but strong privacy is still guaranteed for exact locations *within* each partition. Let $\mathcal{R}_{G_P}(s, d)$ be a *routing mechanism* which returns the approximate shortest path $\rho_{G_P}(s, d)$, and suppose that the origin location s is replaced with a nearby location s' . If s' is still in the *same* partition as s (i.e., $s' \in V_{p_s}$), then $\mathcal{R}_{G_P}(s', d) = \mathcal{R}_{G_P}(s, d)$. The same applies replacing d with any $d' \in V_{p_d}$. An adversary knowing only $\mathcal{R}_{G_P}(s, d)$ would be unable to distinguish s, d from all other possible s', d' as long as $s' \in V_{p_s}$ and $d' \in V_{p_d}$. Thus, location privacy is guaranteed for s and d within p_s and p_d respectively.

The privacy for any s, d pair is then proportional to the number of possible s', d' pairs that can be drawn between V_{p_s} and V_{p_d} . This is designated as the

endpoint location privacy metric:

$$\Omega(s, d) = 1 - \frac{1}{|V_{p_s}| \cdot |V_{p_d}|} \quad (5.4)$$

where V_{p_s} and V_{p_d} give the sets of all vertices in p_s and p_d , respectively. Note that while having more vertices per partition (i.e. larger $|V_{p_s}|$ and $|V_{p_d}|$) is advantageous for endpoint location privacy, this also means higher computation overhead for PIR. Extensive preliminary experiments with the PIR scheme used by our approach showed that the retrieval times scaled almost linearly with the database size at a rate of roughly $\frac{1}{45000} \approx 2.22 \times 10^{-5}$ seconds per record. That is, a database with $|V_p|^2 \approx 10000$ routes was found to have an average retrieval time of ~ 0.25 seconds, while another with $|V_p|^2 \approx 100000$ routes took ~ 2 seconds.

5.3.2 Route Privacy Model

Endpoint location privacy assumes that a route’s origin and destination partition are already known, and thus quantifies only the privacy of the *exact* origin and destination points. This section focuses on the privacy of the *routes themselves*. As with endpoint location privacy, it is assumed that the queried partitions are knowable by adversaries. It is also assumed that they have in-depth knowledge about the algorithms used by the RPS.

Let $Q_{s,d} = \{q_0, \dots, q_n\}$ be the *query sequence* that a user must perform to obtain a route from s to d . This sequence can be transformed into a *partition sequence* $P_{s,d} = \{p_0, \dots, p_n\}$ (where $p_0 = p_s$ and $p_n = p_d$) using a function $f_{qp} : Q \rightarrow P$ that maps every element of Q to its handling partition in P . If the partitioning is hierarchical, then only the highest-level partitions are considered since they already contain the shortest path data of lower-level partitions as stated in Sec. 5.1.1.

Note that partition sequences are not simply partitions along the final route. For instance, in the case of Dijkstra’s algorithm, they can be thought of as the *entire* sequence of “scanned” vertices as depicted in Fig. 5.2. It is therefore possible for several routes to share the same partition sequence (though unlikely in the case of Dijkstra’s). This is modeled as an indicator function that identifies whether or not a candidate Q^* can replace $Q_{s,d}$ for calculating $r_{GP}(s, d)$ and vice

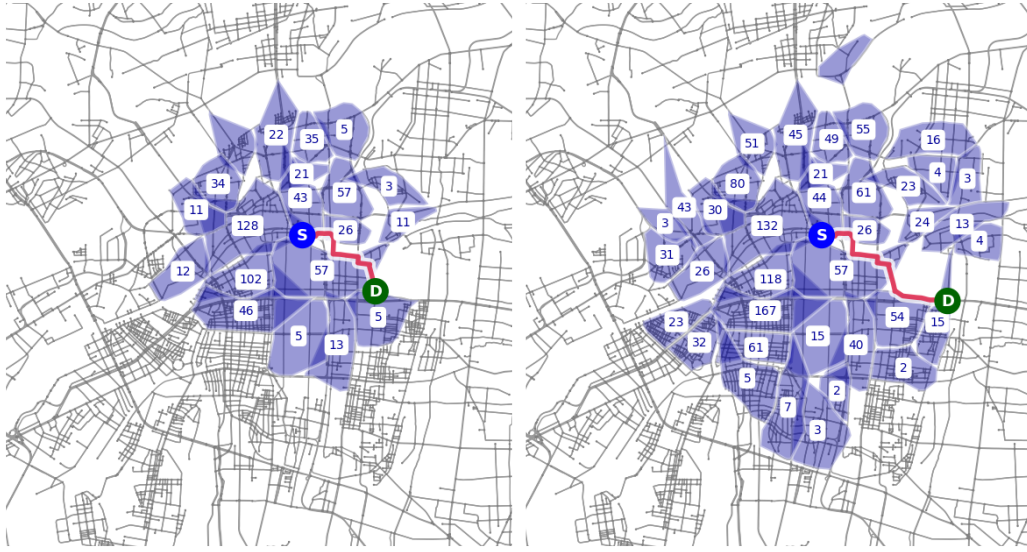


Figure 5.2: Queried Partitions using Dijkstra’s algorithm to calculate two routes with the same origin but different destinations (~ 1 km away from each other). The numbers indicate how many queries were handled by each partition.

versa:

$$k(Q_{s,d}, Q^*) = \begin{cases} 1 & \text{if } f_{qp}(Q_{s,d}) = f_{qp}(Q^*) \\ 0 & \text{otherwise} \end{cases} \quad (5.5)$$

With this, the lower bound for the total number of *distinct routes* that share Q^* is:

$$\sum_{s',d' \in V_X} k(Q_{s',d'}, Q^*) \quad (5.6)$$

where $V_X \subseteq V$ contains *only* the representative vertices (e.g., x_u, x_v) associated with the partition graph G_P as described in Sec. 5.1.1. This ensures that routes where $p_{s'} \neq p_s$ and $p_{d'} \neq p_d$ are counted with equal importance as the route where $p_{s'} = p_s \wedge p_{d'} = p_d$ — hence, the focus on *distinct routes*. Finally, the *route privacy* can then be quantified for any Q^* as follows:

$$\Phi(Q^*) = 1 - \frac{1}{\sum_{s',d' \in V_X} k(Q_{s',d'}, Q^*)} \quad (5.7)$$

5.4 Hierarchical Privacy-Preserving Route Planning

Our proposed approach, Hierarchical Privacy-Preserving Route Planning (HPRoP), is built up from several key design choices to meet particular requirements. That is, HPRoP should be able to:

- REQ1: Compose feasible approximate shortest paths by carefully choosing its component routes from the appropriate partitions,
- REQ2: Privately retrieve component route information from said partitions with minimal processing overhead,
- REQ3: Produce an approximate shortest path with good *optimal route approximation* values (i.e., $\alpha(r_G^*(s, d)) \rightarrow 1.0$),
- REQ4: Ensure a good level of privacy protection for the user’s exact origin and destination points, and intermediate route,
- REQ5: Reflect dynamic and up-to-date road conditions, and
- REQ6: Scale reasonably well with changes in client demand and computational resource availability over time and per area.

All these are brought together by a novel hierarchical route planning heuristic presented in the latter half of this section, along with other improvements to privacy and routing.

5.4.1 Private Information Retrieval (PIR)

HPRoP uses PIR as its core route privacy-preservation mechanism. The choice of implementation was the SealPIR [29] library configured to use Brakerski/Fan-Vercauteren (BFV) Homomorphic Encryption (HE) [49] with a database upper bound of $N = 216$, a plaintext modulus of $\log(t) = 12$, and a dimensionality factor, $d = 2$. This implementation was chosen specifically for its significantly reduced processing and communication overhead compared to other HE-based

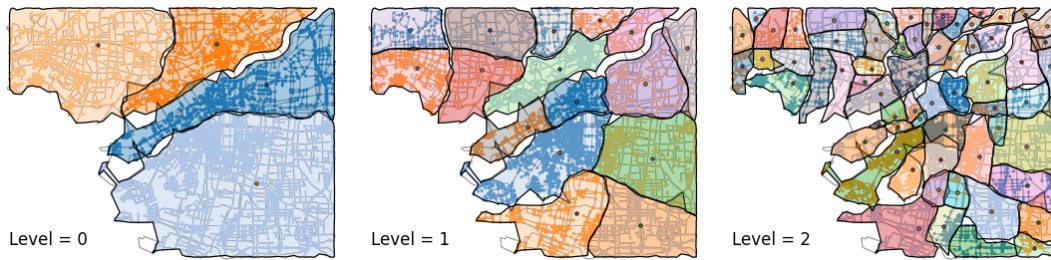


Figure 5.3: Road network graph of Osaka City, Japan hierarchically-partitioned using the Inertial Flow algorithm

ones, making it ideal for an RPS. This along with the hierarchical route planning heuristic drastically reduces the number of queries and, in effect, the route completion time.

5.4.2 Inertial Flow Partitioning

Optimal route approximation and endpoint location privacy are highly-dependent on how the service region is partitioned. Straightforward methods such as grid partitioning are simple but often result in disjoint partition subgraphs in the presence of natural barriers like rivers, etc. To mitigate this, one way would be to ensure that each partition subgraph is a strongly-connected component, ensuring high internal connectivity and reachability. Examples of road network graph aware partitioning methods include PUNCH [50], *Buffoon* [51], and Inertial Flow [52].

Inertial Flow was chosen for HPRoP as it is a relatively simple algorithm based on maximum flow which results in balanced partitions and also preserves internal connectivity. A vertex threshold of around 300 nodes per partition was chosen instead of an area-based threshold to guarantee an *Endpoint Location Privacy* of $\Omega(s, d) \approx 99.998\%$ (i.e., $< 0.002\%$ probability). Moreover, based on preliminary experiments with SealPIR, a partition database with $300^2 \approx 90000$ routes is expected to have retrieval times between 1.5 – 2.5 seconds (1.75 seconds on average) which is viable when combined with HPRoP’s reduced query counts. A queue is initialized by adding the entire road network graph to it. A graph from this queue is then used as input to the Inertial Flow algorithm to produce two balanced partition subgraphs. The simple iterative technique in [53] was

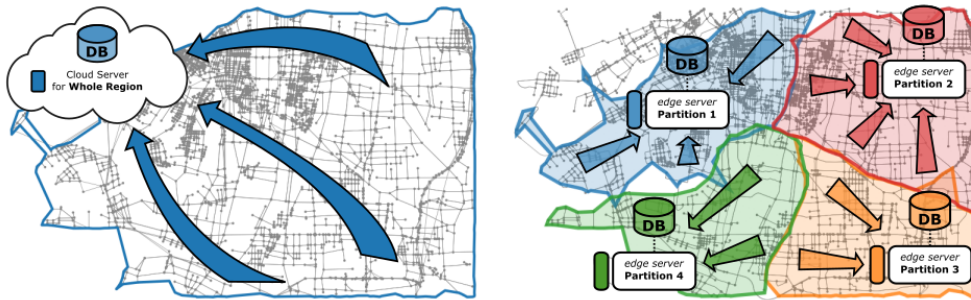


Figure 5.4: Cloud-based Architecture (Left) vs Distributed Architecture (Right)

used alongside Inertial Flow to find and apply optimal cuts during this step. If any of the subgraphs do not yet satisfy vertex threshold, they are simply added back to the queue. This entire procedure is then repeated until the queue is empty. Afterwards, every two consecutive cuts was then retroactively denoted as a separate *partition level* as shown in Fig. 5.3, and the partitions under each are then tagged accordingly. Due to the vertex threshold, the highest level partition may vary greatly from area to area. Some routes therefore require more queries to complete over other routes, which increases profiling risk. A partition level threshold $\ell_{threshold} = 3$ was therefore imposed such that higher level ones were reassigned to $\ell = \ell_{threshold}$.

5.4.3 Distributed Architecture

HPRoP leverages the hierarchically partitioned road network by delegating each partition to a different entity. In the cloud-based scenario, these entities would be server instances; while, in the edge-based scenario, these would be edge-servers throughout the smart city. The edge-based architecture presents several advantages. First, it allows PIR queries to be directed only to partitions which have the information necessary to answer them, effectively distributing the computational load of using PIR. Second, it allows the system to better scale based on the number of users, availability of computing resources, etc. which can vary greatly at different times across different parts of the city as shown in Fig. 5.4. Finally, it also makes the pre-computation of shortest paths to neighboring partitions more efficient since it can be done independently by every partition after an initial exchange of road condition information with said neighbors. This is useful for

reflecting dynamic road conditions bound to some local area.

5.4.4 Heuristic Algorithm

HPRoP uses a simple heuristic algorithm to arrive at an approximate shortest path as follows:

1. *Initialization*: Find an initial basis route between the lowest level partitions containing source and destination, then move up one level.
2. *Subroute Connection*: Connect the source and destination partitions at the current level to the basis route using subroutes.
3. *Basis Route Merging*: Merge the subroutes into the basis route.
4. Repeat steps (2) to (3) until the highest partition level is reached

Algorithm 3: Hierarchical Route Planning Heuristic

Input: Source node s , Destination node d , Current level l_c

Output: The final route r_*

```

1 begin
2    $l_o \leftarrow FindBaseLevel(s, d);$ 
3    $r_* \leftarrow RetrievePath(p_s^{l_o}, p_d^{l_o}, \text{"from source"});$ 
4   Initialize  $l_c \leftarrow l_o + 1;$ 
5   while  $l_c \leq l_{threshold}$  do
6     Initialize  $r_s^{l_c}, r_d^{l_c} \leftarrow [];$ 
7     if  $l_c < (l_{threshold} - 1)$  then
8        $r_s^{l_c} \leftarrow GetSubroute(s, l_c, r_*, \text{"from source"});$ 
9        $r_d^{l_c} \leftarrow GetSubroute(d, l_c, r_*, \text{"from destination"});$ 
10    else
11       $r_s^{l_c} \leftarrow GetEndSubroute(s, l_c, r_*, \text{"from source"});$ 
12       $r_d^{l_c} \leftarrow GetEndSubroute(d, l_c, r_*, \text{"from destination"});$ 
13       $r_* \leftarrow MergeRoutes(r_*, r_s^{l_c}, r_d^{l_c});$ 
14       $l_c \leftarrow l_c + 1;$ 
15  return  $r_*;$ 

```

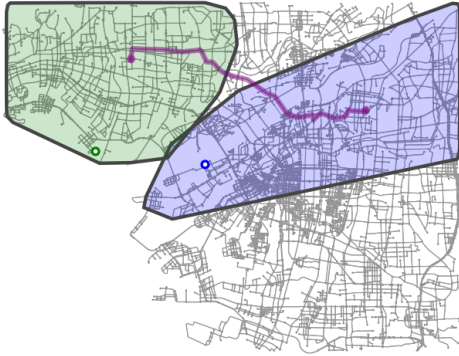


Figure 5.5: Initialization: Find SP between lowest-level partitions containing s (blue circle) and d (green) separately.

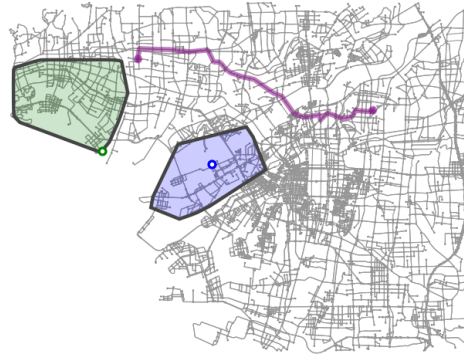


Figure 5.6: Initialization: Use discovered path as basis route (purple line) for the next level

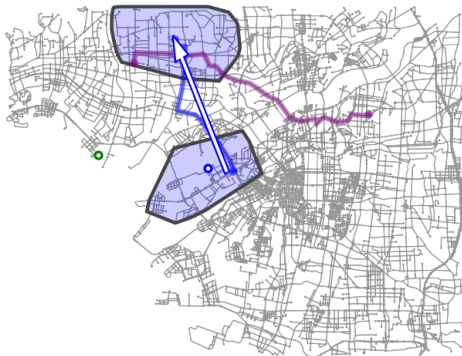


Figure 5.7: Source Subroute Conn.: Find and connect source subroute (blue line) to basis route (purple line)



Figure 5.8: Dest. Subroute Conn.: Find and connect dest. subroute (green line) to basis route (purple line)



Figure 5.9: Basis Route Merging: Merge routes and use as new basis route (purple line)

Algo. 3 runs exclusively on the client-side, sending PIR queries to edge servers handling specific partitions. Route information is obtained solely through these PIR queries, and, thus, no information is leaked by the queries themselves. However, the number of queries, their timestamps, and the partitions they were sent to are still assumed to be known to the adversary. Additionally, since the number of PIR queries contribute a lot to processing overhead, one of the algorithm’s main goals is to reduce the number of queries that need to be made in order to complete a route.

The algorithm starts with an *Initialization* step (lines 2-4 in Algo. 3) which finds an approximate shortest path between the lowest level partitions containing s and d separately as shown in Fig. 5.5. This level is denoted as the base level $l_o = \arg \min_l (p_s^l \neq p_d^l)$. For simplicity, this is just denoted as *FindBaseLevel*(s, d) in Algo. 3. The client then sends a PIR query to partition $p_s^{l_o}$, retrieving a route to partition $p_d^{l_o}$. This corresponds to *RetrievePath*(p_u, p_v, \mathfrak{D}) which retrieves the shortest path between two partitions taking into account some direction flag \mathfrak{D} . This flag simply indicates whether the path is being calculated from the source or the destination, which will be relevant later. The retrieved route is then denoted as the initial *basis route* r_* shown in Fig. 5.6. At the end of this step, the current level variable l_c is also initialized (line 4).

The main loop starts from the *Subroute Connection* steps (lines 6-9 in Algo. 3). The algorithm for Subroute Connection itself is described in Algo. 4. A *subroute* $r_x^{l_c}$ is defined as a path that connects a *basis partition* $p_x^{l_c}$ to the current basis route r_* . The basis partition given by $p_x^{l_c}$ is always the source or destination partition at level l_c containing some vertex x , and is used to obtain the source or destination sub-partitions (depending on the direction \mathfrak{D}) at line 3 of Algo. 4. For instance, the *source subroute* is obtained by finding a sequence of shortest paths from $p_s^{l_c}$ that connects to the basis route as shown in Fig. 5.7. This step also uses several important functions, such as: (1) *FindRoutePartitions*(r_*, l) which gives the sequence of partitions at level l along r_* , and (2) *DoesNotIntersect*($r_x^{l_c}, r_*$) which is “True” if $r_x^{l_c}$ and r_* have no common vertices. Starting from $p_x^{l_c}$, it builds $r_x^{l_c}$ by retrieving a path to nearby connected partitions (line 9) and then connecting them to the subroute (lines 10-13). Since $r_x^{l_c}$ might not yet intersect r_* during the initial iteration, this is repeated with an updated reference partition

(lines 14-15) until an intersection is found. Computing the *destination subroute* follows the same steps but first has to *reverse* the aforementioned sequence (lines 5-7) so that the algorithm can begin from the last route partition. This step is shown in Fig. 5.8.

The *Basis Route Merging* step (line 13 in Algo. 3) is performed once the source and destination subroutes are found. The basic idea is to find the “best” point at which the subroutes intersect with the basis route and join them there. For the source subroute, the “best” point is as far as possible from the start of the basis route; while for the destination subroute, this is as far as possible from the end of the basis route. This is illustrated in Fig. 5.9. This merged route is then used as the new basis route. The current level l_c is then updated (line 15), and the loop is restarted. The loop is terminated once l_c reaches $l_{threshold}$.

The time complexity of this algorithm depends on the required number of PIR database lookups. Finding the initial basis route and calculating the final routes

Algorithm 4: Subroute Connection

Input: Basis node x , Current level l_c , Basis route r_* , Direction \mathcal{D}

Output: Subroute at the current level $r_x^{l_c}$

```

1 begin
2   Initialize  $r_x^{l_c} \leftarrow []$ ;
3    $p_c \leftarrow p_x^{l_c}$ ; // This retrieves either source or destination sub-partition
   depending on direction  $\mathcal{D}$ 
4    $RP^{l_c} \leftarrow FindRoutePartitions(r_*, l_c)$ 
5   if  $\mathcal{D}$  is “from destination” then
6     Reverse( $RP^{l_c}$ );
7    $i \leftarrow 0$ ;
8   while DoesNotIntersect( $r_x^{l_c}, r_*$ ) and  $i < |RP^{l_c}|$  do
9      $r_{part} \leftarrow RetrievePath(p_c, RP^{l_c}[i], \mathcal{D})$ ;
10    if  $\mathcal{D}$  is “from source” then
11       $r_x^{l_c} \leftarrow r_x^{l_c} + r_{part}$ ;
12    else if  $\mathcal{D}$  is “from destination” then
13       $r_x^{l_c} \leftarrow r_{part} + r_x^{l_c}$ ;
14     $p_c \leftarrow RP^{l_c}[i]$ ;
15     $i \leftarrow i + 1$ ;
16  return  $r_x^{l_c}$ ;

```

at p_s and p_d always require a single lookup each. Meanwhile, the number of lookups at each level depends on the maximum weight of the shortest path in G_P at that level which is given by $\max_{p_u^\ell, p_v^\ell \in P^\ell} |r_{G_P^\ell}^*(p_u^\ell, p_v^\ell)|$ where $G_P^\ell \subset G_P$ containing only that level's partitions P^ℓ and connections C^ℓ . The average time complexity is then:

$$O\left([\mathcal{N}_p^2 + \sum_{\ell \in L} \mathcal{N}_c^\ell] \cdot \left[3 + \sum_{\ell \in L} 2 \cdot \max_{p_u^\ell, p_v^\ell \in P^\ell} |r_{G_P^\ell}^*(p_u^\ell, p_v^\ell)|\right]\right) \quad (5.8)$$

where \mathcal{N}_c^ℓ is the average number of connections from each partition at level ℓ . Since HPRoP precomputes and stores shortest path data for multiple levels per partition, it is necessary to account for \mathcal{N}_c^ℓ in HPRoP's space complexity:

$$O([\mathcal{N}_p^2 + \sum_{\ell \in L} \mathcal{N}_c^\ell] \cdot R_G^{*,max}) \quad (5.9)$$

5.4.5 Route Privacy Mechanism

Route privacy as defined in Sec. 5.3.2 quantifies the privacy based on how many other possible routes have a query sequence matching that of a given route, where more matches mean better privacy. That is, an adequate route privacy mechanism should: (1) maximize the matching of query sequences between all possible routes, and (2) minimize the information gain from the order of queries in the sequence itself. HPRoP already achieves the latter via hierarchical execution which can somewhat obfuscate the actual query sequence, but does not necessarily strengthen the former. To address this, the algorithm is extended to *pad* the query sequence with *dummy queries*. The basic idea is to query multiple other partitions instead of just p_s and p_d to ensure that the actual origin and destination partitions are "hidden" among them. In theory, querying more partitions would mean better route privacy at the cost of longer route completion times. Achieving full route privacy, however, would require querying all level partitions at every iteration of the algorithm at least once, which would require prohibitively long route completion times. For example, a service region with a total of 463 partitions would require roughly 810 seconds (13.5 minutes) on average to complete a single route. However, simply querying a small random subset of the aforementioned partitions will not be enough to ensure a certain level of route

privacy, since an adversary can simply use the hierarchy of partitions to check for inconsistencies in the set of queried partitions and easily identify the dummy ones. Instead, we chose to limit HPRoP to querying all other partitions *under the same parent* as the highest level partitions containing s and d . This selection method is straightforward and ensures that none of the queried partitions can easily be identified as dummy partitions. Additionally, this ensures that route privacy will be around $\Phi(Q^*) \approx 1 - 1/jk$ where j and k are the total number of partitions under the same parent partitions as p_s and p_d , respectively.

This is implemented through the *Subroute End Connection* steps (lines 10-12 in Algo. 3), while the procedure itself is presented in Algo. 5. Instead of stopping when the subroute and basis route first intersect, this algorithm continues until all other partitions sharing the same parent $p_x^{l_c-1}$ as the basis partition have been queried. This is done by repeatedly drawing a partition p_{sub} from a queue of these same-parent sub-partitions SP^{l_c} (line 10). If p_{sub} is the same as the current partition p_c , then a part of the subroute is retrieved as normal (lines 11-18). If it is a route partition, it is instead pushed back to the subpartition queue (lines 19-22). If both prior conditions are not satisfied, then a dummy query is simply sent to p_{sub} (line 24). This ensures that important queries are mixed in with dummy queries, making it more difficult to determine which ones are relevant.

5.4.6 Shortcut Connections

A simple strategy for improving algorithm performance is through pre-computing and storing shortest path data to partitions *beyond* just the adjacent ones. This reduces the number of queries to complete a route while also improving optimal route approximation for paths to further away partitions. These paths to non-adjacent partitions were therefore denoted as *Shortcut Connections*, represented as additional edges in the partition graph.

These shortcut connections, however, also increase the pre-computation time for each partition relative to how many of them need to be made. In addition, the extent of the road network graph needed for pre-computation also increases based on the distance to the partitions being connected. It is therefore more useful to limit the number of partitions to connect to and how far those partitions can be. HPRoP considers two methods for determining shortcut connections: (1) the

Algorithm 5: Subroute End Connection with Dummies

Input: Source or Destination node x , Current level l_c , Basis route r_*

Output: Source or Destination subroute at the current level $r_x^{l_c}$

```
1 begin
2   Initialize  $r_x^{l_c} \leftarrow []$ ;
3    $p_c \leftarrow p_x^{l_c}$ ;
4    $RP^{l_c} \leftarrow FindRoutePartitions(r_*, l_c)$ 
5   if  $\mathcal{D}$  is “from destination” then
6      $\lfloor Reverse(RP^{l_c});$ 
7    $SP^{l_c} \leftarrow FindSubPartitions(p_x^{l_c-1})$  as Queue
8    $j \leftarrow 0$ ;
9   while  $|SP_x^{l_c-1}| > 0$  do
10     $p_{sub} \leftarrow Pop(SP^{l_c});$ 
11    if  $p_{sub} = p_c$  DoesNotIntersect( $r_d^{l_c}, r_*$ ) then
12       $r_{part} \leftarrow RetrievePath(p_{sub}, RP^{l_c}[j]);$ 
13      if  $\mathcal{D}$  is “from source” then
14         $\lfloor r_x^{l_c} \leftarrow r_x^{l_c} + r_{part};$ 
15      else if  $\mathcal{D}$  is “from destination” then
16         $\lfloor r_x^{l_c} \leftarrow r_{part} + r_x^{l_c};$ 
17       $p_c \leftarrow RP^{l_c}[j];$ 
18       $j \leftarrow j + 1$ ;
19    else if  $p_{sub} \in RP^{l_c}$  DoesNotIntersect( $r_d^{l_c}, r_*$ ) then
20      if Index( $p_{sub}, RP^{l_c}$ )  $> j$  then
21         $\lfloor Push(p_{sub}, SP^{l_c})$ 
22         $\lfloor Shuffle(SP^{l_c})$ 
23    else
24       $\lfloor SendDummyQuery(p_{sub});$ 
25  return  $r_x^{l_c}$ ;
```

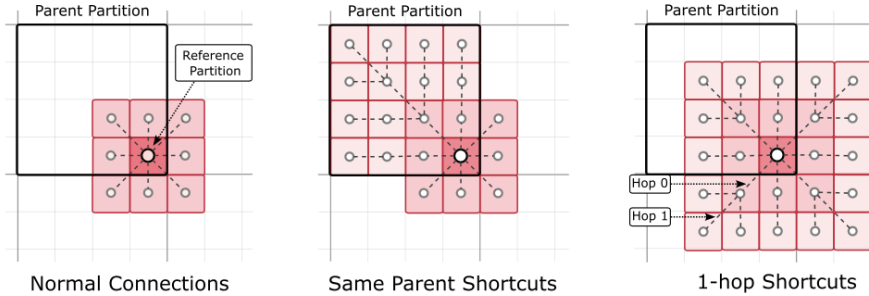


Figure 5.10: Demonstration of different shortcut connection strategies for improving the heuristic algorithm’s performance against the base case (Left), where the dark red shape represents the starting partition, and the lighter red shapes represent the partitions it connects to. The black outline represents the starting partition’s parent. (Middle) uses Same Parent Shortcuts, while (Right) uses 1-hop Neighbor Shortcuts.

Same Parent Shortcuts method, and (2) the *N-hop Neighbor Shortcuts* method. *Same Parent Shortcuts* simply connects each partition to all other partitions under the same parent partition as shown in Fig. 5.10 (Middle). *N-hop Neighbor Shortcuts* pre-computes shortcuts to other *N*-hop away partitions as shown in Fig. 5.10 (Right). Both would theoretically improve the *optimal route approximation* when calculating subroutes between non-adjacent partitions and greatly reduce the possibility of broken routes. HPRoP currently has no mechanisms to handle these other than deferring to the user’s device to perform local route calculation to bridge the final gap. The usefulness of these methods are shown through the results in Sec. 5.5.3.1.

5.5 Evaluation

In this section, the details of the evaluation framework for HPRoP are first presented prior to showing the actual evaluation results and their analysis.

5.5.1 Environment

HPRoP was implemented in Python on a Jupyter notebook for ease of testing and visualization, with the notebook itself encapsulated in a Docker container for portability. The execution environment was a dedicated Linux server running Ubuntu 20.04.1 SMP equipped with a AMD Ryzen Threadripper 3970X 32-Core processor and 256 GB RAM in total.

The service region was a rectangular geographical area of roughly 546 km^2 (i.e. 26 km in width, 21 km in height) encompassing the entire road network of Osaka City, Japan and a portion of the immediately outlying areas. Its road network graph consists of $|V| = 99,734$ vertices and $|E| = 269,614$ edges. The region is hierarchically partitioned using the Inertial Flow algorithm as shown in Fig. 5.3 based on the parameters in Sec. 5.4.2.

5.5.2 Methodology

Evaluation was done through several metrics under the following categories: (1) Utility, (2) Privacy, and (3) Performance. The Utility category pertains to the usefulness of the service, with the *Optimal Route Approximation* metric falling under this category. Since the base algorithm in Sec. 5.4.4 cannot guarantee complete routes, *Route Errors* are also included here as a metric. Route errors are then defined as the occurrence count of broken routes during testing. In turn, a *broken route* is defined as a route where a subroute connection *cannot be established* to the highest level partition containing either s or d , and thereby results in a route that cannot be completed by HPRoP’s algorithm. The Privacy category is comprised of the *Endpoint Location Privacy*, and *Route Privacy* metrics described in Sec. 5.3. The Performance category pertains to how well the service can deal with higher client demand, dynamic road conditions, etc. without service quality degradation. The *Memory Usage*, *Route Completion Time*, and *Pre-processing Time* metrics fall under this category.

Evaluation was done by comparing HPRoP to the two baseline PIR-based approaches — EPR-D and APR-D — previously presented in Sec. 5.2. For the *Utility* category, *Privacy* category, and *Route Completion Time* metrics, all three approaches were evaluated by calculating routes for different s, d pairs until 4,000

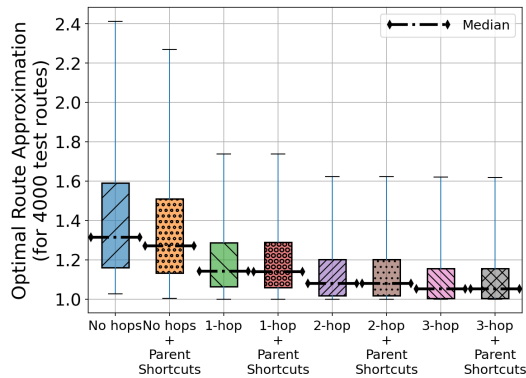


Figure 5.11: Distribution of *Optimal Route Approximation* results for different Shortcut Connection methods

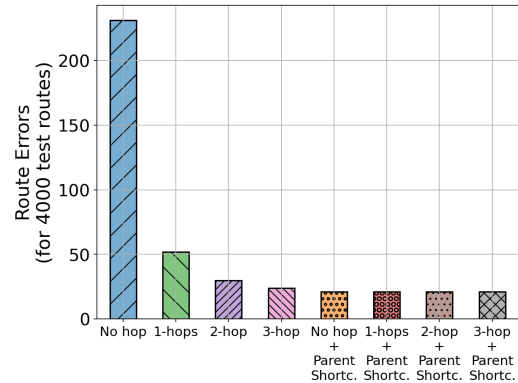


Figure 5.12: Comparison of total Route Errors for different Shortcut Connection methods

successful routes have been completed. This termination threshold was chosen to be sufficiently high enough to capture any route errors that might occur for HPRoP. It was also decided to randomly-generate the s, d pairs rather than basing on their distribution on real-world mobility data as was done for PProP-MO. In this way, the effectiveness of the algorithm is evaluated over the entire road network rather than being centered only across several “hotspots”. Note that both APR-D and EPR-D are guaranteed to produce complete routes so this metric is no longer evaluated for them. Unlike the evaluation method for PProP-MO, calculation of the test routes were done sequentially one-at-a-time rather than in parallel. This is because HPRoP no longer needs to aggregate batches of queries first prior to performing route computation. *Memory Usage* was evaluated by calculating the projected size of the per-partition databases for the highest-level partitions. Finally, *Pre-processing Time* was evaluated by measuring the time to build each partition’s database.

5.5.3 Results

5.5.3.1 Effect of Shortcut Connections

The base performance of HPRoP under different shortcut connection methods in Sec. 5.4.6 is first characterized with the goal of finding the method that maximizes optimal route approximation while minimizing both route completion

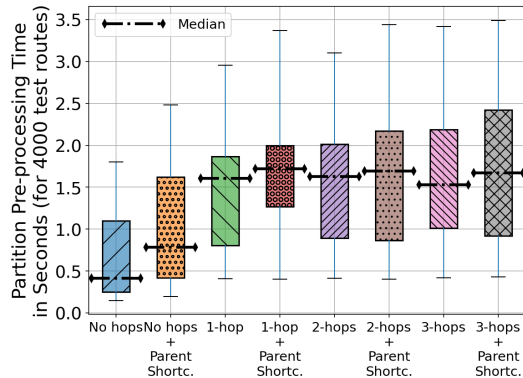


Figure 5.13: Distribution of Pre-Computation Times for different Shortcut Connection methods

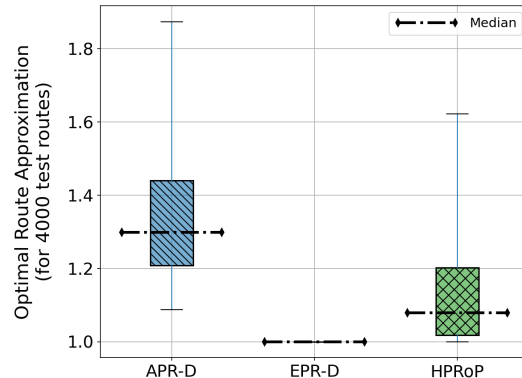


Figure 5.14: Distribution of Optimal Route Approximation results using APR-D, EPR-D, and HPRoP

time and route errors. Fig. 5.11 shows the resulting distribution of optimal route approximation values for 4,000 successful test routes under different methods. Surprisingly, *Same Parent Shortcuts* have an almost negligible effect on optimal route approximation, with *N-hop Neighbor Shortcuts* being a more effective way to increase the said metric. However, *Same Parent Shortcuts* were highly effective in preventing the occurrence of broken routes, reducing the error count to 21. Further investigation of these remaining errors showed that they were caused by choosing the same vertex as s and d which results in failure as no routing can be done by the algorithm. In short, for all 4,000 test routes, *Same Parent Shortcuts* seem to eliminate all occurrences of true broken routes — i.e., where the highest level partitions containing s or d could not be reached. This also validates the hypothesis that broken routes are caused by a lack of reachability at the final partition level that can contain more than 4 child partitions due to the deliberate choice to set $\ell_{threshold} = 3$ mentioned in Sec. 5.4.2.

Meanwhile, using *1-hop Neighbor Shortcuts* drastically improves the results of the 75-th percentile from $\alpha(r_*) \approx 1.54$ to $\alpha(r_*) \approx 1.28$, but anything beyond *2-hop Neighbor Shortcuts* is seen to have gradually diminishing returns. Finally, the overhead caused by the additional shortcut connections is evaluated based on the pre-processing time metric in Sec. 5.5.2. Fig. 5.13 shows that both methods increase the per partition pre-processing time to about ~ 1.5 seconds once *1-Hop Neighbor Shortcuts* are introduced but stabilizes around this value

even as the number of hops are further increased. In contrast, the effect of *Same Parent Shortcuts* on pre-processing time is minimal, amounting to an increase of ~ 0.1 seconds on average. Thus, both methods are equally viable in terms of this metric.

Table 5.4: Summary of Results for different Shortcut Connection Configurations

	Normal				With Same Parent Shortcuts			
	0-hop	1-hop	2-hop	3-hop	0-hop	1-hop	2-hop	3-hop
Optimal Route Approximation								
Min	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
25%	1.160	1.062	1.018	1.003	1.133	1.060	1.017	1.003
50%	1.314	1.141	1.080	1.052	1.270	1.138	1.079	1.052
75%	1.590	1.287	1.202	1.156	1.510	1.288	1.202	1.156
Max	16.256	5.210	5.371	5.371	15.424	5.210	5.371	5.371
Per-Partition Routes Calculation Time (in seconds)								
Min	0.04	0.11	0.19	0.19	0.08	0.11	0.19	0.19
25%	0.16	0.58	1.88	3.33	0.24	0.75	1.92	3.48
50%	0.20	1.43	2.49	4.41	0.47	1.51	2.90	4.75
75%	0.28	1.60	3.36	5.70	1.30	1.74	3.47	6.03
Max	1.27	3.17	5.78	10.55	3.15	3.37	6.75	11.44
Route Errors (per 4,000 successful routes)								
Total	231.0	52.0	30.0	24.0	21.0	21.0	21.0	21.0

Table 5.4 summarizes the results discussed so far. Based on these, it was decided to use *2-hop Neighbor Shortcuts* with *Same Parent Shortcuts* as the representative configuration for HPRoP as it offers good *optimal route approximation* and short pre-computation times with minimal errors.

5.5.3.2 Optimal Route Approximation

The performance of HPRoP, APR-D, and EPR-D in terms of *optimal route approximation* is shown in Fig. 5.14. As expected, EPR-D achieves a constant

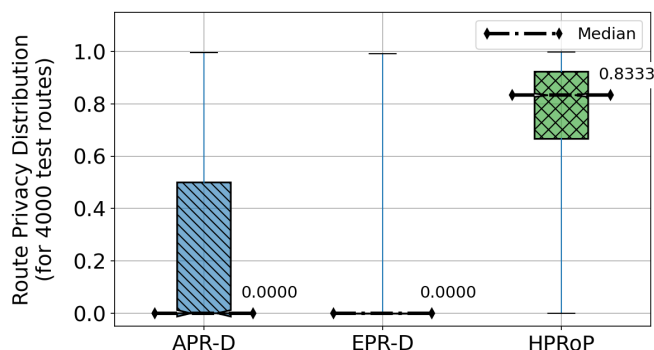


Figure 5.15: Distribution of Route Privacy $\Phi(Q^*)$ results for APR-D, EPR-D, and HPRoP

optimal route approximation value of $\alpha(r_*) = 1.0$ since it always produces exact shortest paths. APR-D produced routes with $\alpha(r_*) \approx 1.44$ for the 75-th percentile despite operating only over the partition graph. HPRoP produced even better routes with $\alpha(r_*) \approx 1.20$ for the 75-th percentile, whereas APR-D was only able to achieve this for the 25-th percentile of all results. Additionally, HPRoP achieves much better worst-case routes than APR-D.

5.5.3.3 Endpoint Location Privacy

Endpoint Location Privacy $\Omega(s, d)$ describes how well the exact s, d is kept private as described in Sec. 5.3.1. Since Inertial Flow partitioning was used, evaluation results showed that all three approaches were able to achieve an average endpoint location privacy of $\Omega(s, d) = 0.999982$. Thus, each route is *indistinguishable* from approximately 99% of all other routes between the same partitions, making all three approaches equally viable.

5.5.3.4 Route Privacy

Route Privacy $\Phi(Q^*)$ describes how well a route is kept private based on how many other routes share a query sequence similar to its own as described in Sec. 5.3.2. To evaluate this, a lookup table for the routes between each $s, d \in V_x$ such that $s \neq d$ was done separately for EPR-D, APR-D, and HPRoP to account for their unique querying behaviors as shown in Fig. 5.16.

Fig. 5.15 shows a comparison of the route privacy distributions across all three

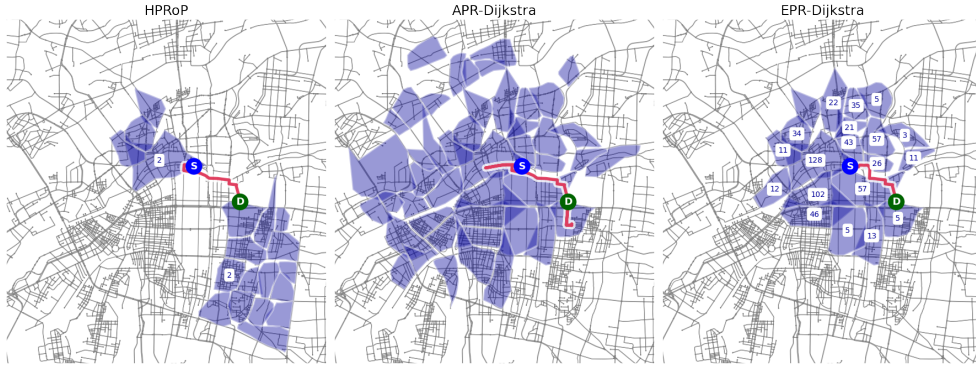


Figure 5.16: Visualization of Queried Partitions for APR-Dijkstra (Center), EPR-Dijkstra (Right) and HPRoP (Left) for the same route. Numbers indicate partitions that were queried multiple times.

approaches. EPR-D showed the worst route privacy, achieving $\Phi(Q^*) > 0$ only for the 25-th percentile of all results. APR-D performed only slightly better with $\Phi(Q^*) > 0$ for about 50% of all results, while achieving $\Phi(Q^*) \geq 0.5$ for only 25% of them. This is expected since both have no inherent privacy mechanisms, yet this does illustrate that APR-D is still better than EPR-D in terms of route privacy. HPRoP, in contrast, has route privacy of $\Phi(Q^*) \geq 0.80$ for the 50-th percentile of all results, while also achieving $\Phi(Q^*) \geq 0.50$ for the 75-th percentile, surpassing both EPR-D and APR-D. However, further analysis of the routes showed that the worst-case route privacy (i.e. $\Phi(Q^*) = 0.0$) still happens for $\sim 12.2\%$ of all test routes. This suggests that HPRoP does not fully guarantee route privacy for all cases although this can be increased further by adding more dummy queries.

Additionally, the relationship between optimal route approximation $\alpha(r_*)$ and route privacy metric $\Phi(Q^*)$ was also analyzed for HPRoP. This was done to determine whether some trade-off exists between the two metrics. The results in Fig. 5.17 show that majority of the routes have $\alpha(r_{s,d}) < 1.2$ across widely-varying levels of route privacy. This suggests that the two metrics have little effect on one another, and performing a simple Pearson correlation confirms that there is only a weak positive correlation ($\rho \approx 0.104$) between the two.

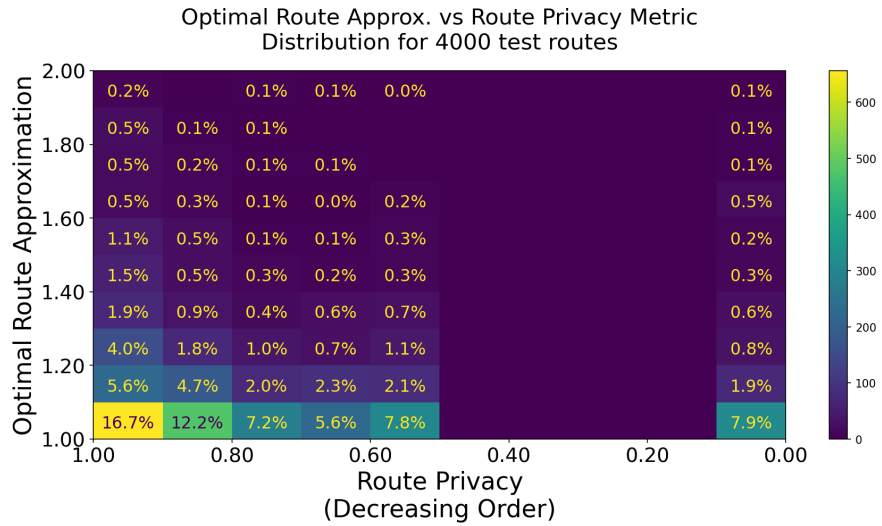


Figure 5.17: Distribution of Optimal Route Approximation and Route Privacy results using HPRoP for 4,000 test routes. Note that both axes were reoriented to show the best results on the lower left.

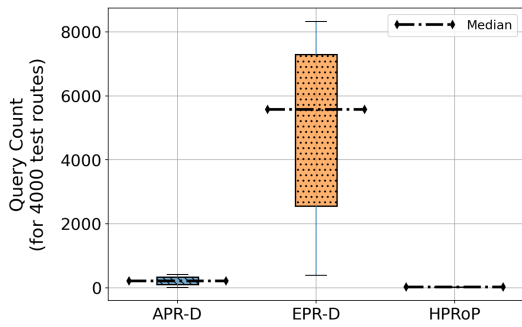


Figure 5.18: Distribution of the required number of queries for route completion

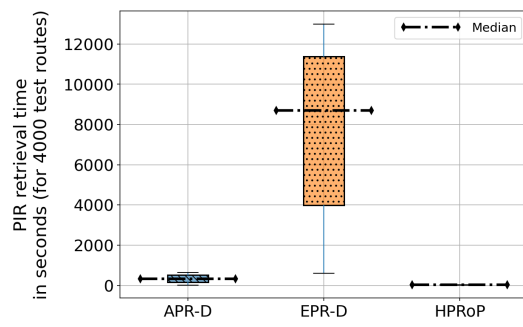


Figure 5.19: Distribution of total PIR retrieval times in seconds

5.5.3.5 Route Completion Time

Obtaining a route using any of the three algorithms (APR-D, EPR-D, and HPRoP) requires the client to make multiple PIR queries to the RPS. That is, a single PIR query is required for each vertex scan in EPR-D, while APR-D requires it for each partial route between partitions. In HPRoP, a single PIR query is required to retrieve each subroute. *Query count* is thus defined as the number of PIR queries that need to be made in order for the client to complete a route (i.e. for the underlying algorithm to terminate). *Route completion time* is the time taken to complete a route which is directly proportional to the query count. It is also equivalent to the total processing overhead for an RPS. In this work, it is calculated based on projections derived from the preliminary experiments on SealPIR database retrieval times mentioned at the end of Sec. 5.3.1. The empirically derived value of roughly $\frac{1}{45000} \approx 2.22 \times 10^{-5}$ seconds per record is then used to calculate the route completion times which are shown in Fig. 5.19. Note that we opted to use projections here instead of simulating the actual results as the latter would take a prohibitively long time to conclude in the case of EPR-D (and, to a lesser extent, APR-D). For instance, APR-D requires ~ 214 queries on average which is expected to take 5.55 minutes (333.26 seconds) per route. EPR-D is even worse, requiring $\sim 4,958$ queries on average which would take at least 2.15 hours (7,731.47 seconds) just to complete a single route. Both are clearly impractical from the perspective of any modern RPS. This is in contrast to HPRoP which requires significantly less queries on average (at ~ 25 queries) and takes only 23.55 seconds per route. This is because HPRoP’s algorithm bypasses the need to explore large sections of the partition and road network graph as it already starts with a very coarse route between the origin and destination partitions at the lowest level to guide its route search. APR-D and EPR-D, in contrast, explore outwards from the origin, checking every unexplored partition or vertex on the way as per Dijkstra’s algorithm.

5.5.3.6 Memory Usage

Memory usage depends on the size of the route databases maintained by each partition, which is very important in a distributed environment with resource-constrained devices. Fig. 5.20 shows the distribution of per partition memory

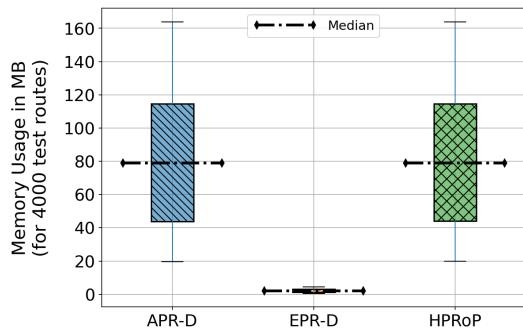


Figure 5.20: Distribution of Per-Partition Memory Usage (in MB) for the different approaches

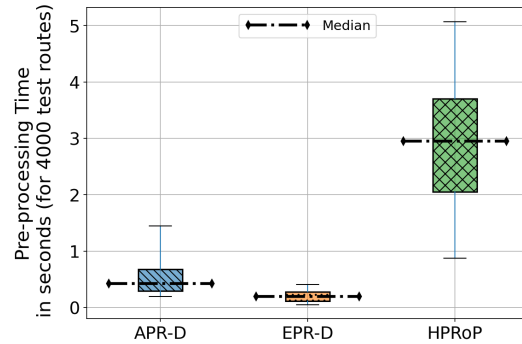


Figure 5.21: Distribution of Per-Partition Pre-processing Time results (in seconds) for the different approaches

usage across the three approaches. APR-D and HPRoP calculate and store a similar number of routes, thus requiring an allocation of around 20-160 MB per partition. EPR-D has a *significantly* smaller memory footprint at around 0.5-4 MB per partition because it essentially stores a next-hop matrix instead. In practice, however, the memory usage of all three approaches are well within the capabilities of standard edge servers.

5.5.3.7 Pre-processing Time

The pre-processing time metric is total time needed to build each partition's database during the pre-processing phase, which determines how often it can be updated during operation. As shown in Fig. 5.21, EPR-D require less than ≤ 0.5 seconds per partition on average since it only needs to calculate the routes *inside* each partition, while APR-D requires ≤ 1.5 seconds since it also needs to calculate routes to immediately neighboring partitions in addition. Meanwhile, HPRoP needs to calculate routes inside each partition, routes to neighboring partitions, and routes to other partitions under the same parent. This results in slightly longer pre-processing times at 1-5 seconds per partition. Regardless, the pre-processing time for all three approaches are clearly fast enough to accommodate frequent updates even in under dynamic road conditions.

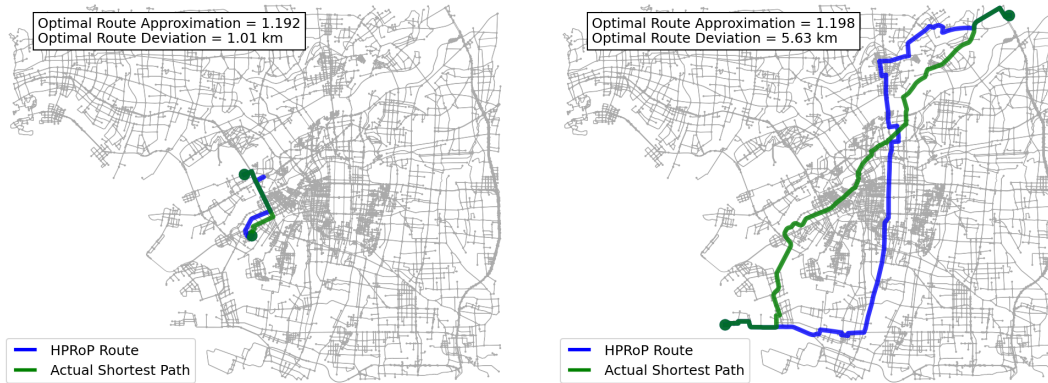


Figure 5.22: Visualization of differences in HPRoP routes at $1.19 < \alpha(r_{s,d}) < 1.2$: (Left) shows an example of minimal deviation (~ 1.0 km) from the actual SP; (Right) shows an example of maximal deviation (~ 5.63 km) from the actual SP

5.5.4 Discussions

5.5.4.1 Optimal Route Approximation and Actual Route Lengths

In evaluating the Utility of HPRoP, *Optimal Route Approximation* was used as a measure of “fitness” for the produced routes by comparing them to the actual shortest paths obtained via Dijkstra’s Algorithm. While this works well for comparisons, it also tends to conceal the total length of the actual routes. For instance, a low *Optimal Route Approximation* does not automatically mean that the produced routes are short. Thus, the difference in length between HPRoP’s route and the actual shortest path (henceforth, simply called *Optimal Route Deviation*) can vary greatly for the same *Optimal Route Approximation* value. This can be seen in Fig. 5.22 where the *Optimal Route Deviation* varies between ~ 1.0 to ~ 5.63 km despite their *Optimal Route Approximation* values being nearly similar (i.e. $\alpha(r_{s,d}) = 1.192$ and $\alpha(r_{s,d}) = 1.198$). Conversely, low *Optimal Route Deviation* does not necessarily mean better *Optimal Route Approximation*. This can be seen in Fig. 5.23 where two routes with *Optimal Route Deviation* values of 0.95 and 0.96 km have somewhat different *Optimal Route Approximation* values ($\alpha(r_{s,d}) = 1.035$ and $\alpha(r_{s,d}) = 1.162$ respectively). However, while lower *Optimal Route Approximation* values do not automatically guarantee shorter routes, it does guarantee a lower median *Optimal Route Deviation* than higher values.

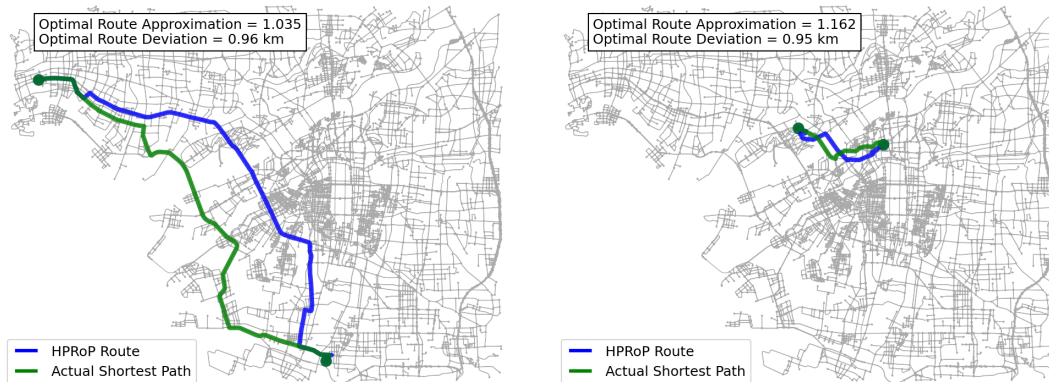


Figure 5.23: Visualization of differences in HPRoP routes with an Optimal Route Deviation between 0.95 km and 1.0 km; (Left) shows an example with good Optimal Route Approximation (~ 1.035); (Right) shows an example with bad Optimal Route Approximation (~ 1.162)

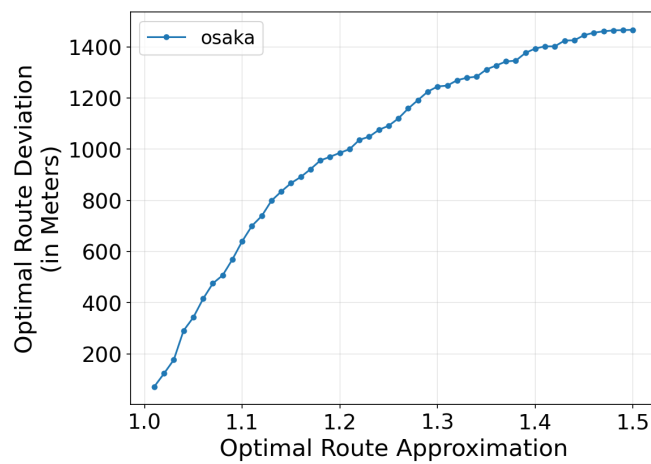


Figure 5.24: Relationship between Optimal Route Approximation and Optimal Route Deviation (difference between lengths of HPRoP's route and the actual shortest path)

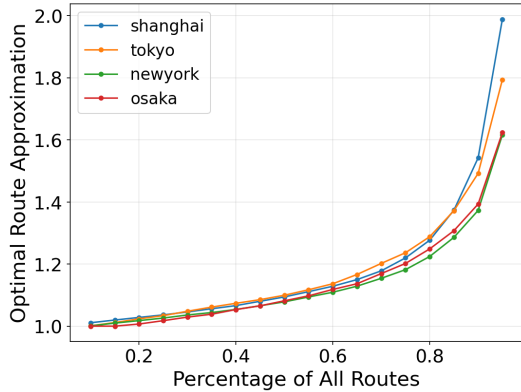


Figure 5.25: *Maximum* values for Optimal Route Approximation given a selected percentage of all produced routes

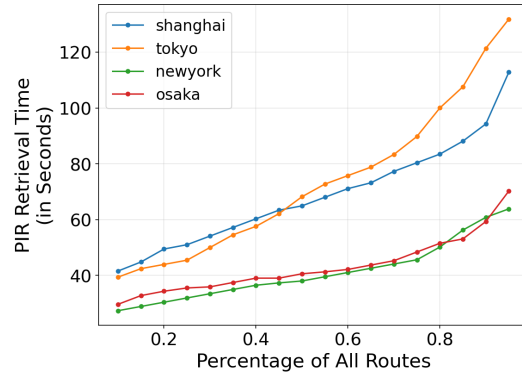


Figure 5.26: *Maximum* values for Route Completion Time given a selected percentage of all produced routes

This is seen in Fig. 5.24 where the median Optimal Route Deviation increases in a near-linear manner with increasing Optimal Route Approximation. In short, a lower Optimal Route Approximation remains preferable despite the potential variance in the range of Optimal Route Deviation values.

5.5.4.2 Scalability

While the performance results for *memory usage* and *pre-processing time* in Section 5.5 indicate that HPRoP is viable to deploy on Edge Servers in a Smart City environment, this does not necessarily mean that it scales for larger (e.g. continent-sized) road networks. We also performed some additional experiments on the road networks of two other large cities, namely: Tokyo ($34 \times 36 \text{ km} \approx 1,224 \text{ km}^2$, 228,838 vertices, 611,745 edges), and New York ($50 \times 50 \text{ km} \approx 2,500 \text{ km}^2$, 98,336 vertices, 252,810 edges). As a stress test, we also conducted an experiment which included the road network of Shanghai along with parts of Wuxi, Suzhou, and Jiaxing ($150 \times 180 \text{ km} \approx 27,000 \text{ km}^2$, 192,560 vertices, 475,142 edges). For each road network, 1,000 s, d pairs were randomly-generated and used as inputs to HPRoP. Results showed that *Optimal Route Approximation* behaved in a relatively similar manner despite the difference in road network sizes as shown in Fig. 5.25. However, a noticeable difference may be observed in the maximum *Route Completion Times* in Fig. 5.26 which clearly shows that the Tokyo and Shanghai road networks take significantly longer. Based on the

time complexity of HPRoP given by Eq. 5.8, the biggest contributor to route completion time is the number of queries in $\sum_{\ell \in L} 2 \cdot \max_{p_u^\ell, p_v^\ell \in P^\ell} |r_{G_P}^*(p_u^\ell, p_v^\ell)|$, where $r_{G_P}^*(p_u^\ell, p_v^\ell)$ is the shortest path in G_P between any p_u and p_v at level ℓ . Because of HPRoP’s hierarchical execution, for $0 < \ell < 3$, this shortest path will simply have a length of 2 because there is always a direct connection between any two partitions at those levels. However, for $\ell = 3$, using Dummy Queries means that each partition at $\ell = 3$ under the same parent *has to be queried at least once* to preserve privacy — in turn contributing to longer route completion times. Upon investigation, it was found that the maximum same parent partition count for 10% of the partitions in the Tokyo (~ 27 partitions) and Shanghai (~ 22 partitions) road networks are more than twice that of Osaka (~ 11 partitions) and New York (~ 11 partitions) — which can, in turn, explain the significant difference seen in Fig. 5.26. Note, however, that these large partition counts are mostly a consequence of setting the maximum level to $\ell = 3$ which was specifically tailored towards Osaka City’s road network. Adjusting this parameter will most likely allow HPRoP to accommodate larger and more complex road networks. Based on these preliminary results and the subsequent investigation, it can be said that HPRoP is likely to be scalable to larger road networks though further investigation is required to validate this.

5.5.4.3 Improving Route Completion Times

While HPRoP’s performance in terms of the *route completion time* showed that it performed significantly better at ~ 23.55 seconds per route than both EPR-D (at $\sim 7,731.47$ seconds per route) and APR-D (at ~ 333.26 seconds per route), it is still much slower than most modern RPS which typically have route completion times of less than one second — although it should be noted that these provide no privacy guarantees at all. However, there are several avenues to improve upon this can be explored. The first is to replace PIR with a lighter privacy-preserving database querying method like Structured Encryption (see Sec. 2.3.1) and then work-around its privacy leakage issues somehow. Another way can be to parallelize the sending of real and dummy queries at the final iteration of HPRoP to significantly cut down on the biggest source of redundant latency to route completion time. Likewise, the maximum partition level can also be adjusted

such that the maximum number of same parent partitions is significantly reduced. The last two methods, however, may have significant impacts on the *Endpoint Location Privacy* and should therefore be carefully investigated. Finally, it may also be possible to optimize the routing queries themselves in the same manner as P_{RoP}-MO in order to achieve better route completion times for the users that have a preference for it. However, as with P_{RoP}-MO, the issue with the delays introduced by batching must also be resolved in order for it to be viable.

5.6 Summary

In this chapter, a PIR-based RPS called Hierarchical Privacy-Preserving Route Planning (HP_{RoP}) was proposed which extends the key idea in Sec. 3.2 by using Inertial Flow partitioning to divide the road network and combining it with a novel hierarchical route planning heuristic algorithm to produce routes that can adequately approximate the actual shortest paths while also providing *endpoint location privacy* and *route privacy*. More importantly, HP_{RoP} addresses the issues and vulnerabilities that were present in the previous approach, P2_{RoP}-MO, by eliminating the need for an optimization step entirely and extending route privacy protection to include the whole route instead of only the parts close to the origin and destination. Furthermore, HP_{RoP} reliably produced routes with an *optimal route approximation* of $\alpha(r_*) \leq 1.2$, while also achieving near-optimal endpoint location privacy at $\Omega(s, d) \approx 1.0$ and good route privacy at $\Phi(Q^*) \geq 0.5$. In terms of performance, HP_{RoP} has a route completion time of around 23.55 seconds on average which is reasonable for a privacy-preserving RPS. Its viability for deployment in a distributed/edge-based smart city context were also shown through its relatively small memory footprint (20-160 MB for each partition's database), and short pre-processing times (2-5 seconds per partition) which are well within the capabilities of conventional edge servers.

6 Conclusion

6.1 Summary

As RPS become increasingly ingrained in the daily activities of modern-day commuters, vehicle owners, and even automated vehicles as essential tools for navigating urbanized spaces, making them more resilient and robust to the actions of malicious entities has also become very crucial. Privacy-preserving RPS is one of the steps towards achieving this by protecting the privacy of users' route-related information. The key idea behind the approaches presented in this dissertation is that of the PIR-based RPS which enables users' client devices to retrieve of pre-calculated routes in some remote database without the server discovering the route retrieved. This, however, is not practical as it is because of the time and space complexity required to deal with large, dense road network graphs that are typical of modern cities.

To address this problem, two approaches are presented in this dissertation: (1) an initial approach called *Practical PIR-based Route Planning via Multi-Objective Optimization* (P2RoP-MO), and (2) an improved approach called *Hierarchical Privacy-Preserving Route Planning* (HPRoP). The initial approach, *P2RoP-MO*, uses multi-objective optimization to balance the trade-off between Privacy, Utility, and Performance in order to make the PIR-based RPS more practical. It was able to achieve acceptable trade-offs between said objectives but was bogged down by its slow, batch-based optimization step, and did not protect the intermediate section of the route by design. Thus, the improved approach, HPRoP, was developed which uses a different road network partitioning method called Inertial Flow and combines that with a novel hierarchical route planning heuristic algorithm to produce near-optimal routes with good *endpoint location privacy* and good *route privacy*. Besides addressing the issues with P2RoP-MO, HPRoP also reliably

produced routes with an *optimal route approximation* of $\alpha(r_*) \leq 1.2$ alongside near-optimal endpoint location privacy ($\Omega(s, d) \approx 1.0$) and good route privacy ($\Phi(Q^*) \geq 0.5$). It was also able to achieve this with an average route completion time of around 23.55 seconds while taking up a relatively small memory footprint (20-160 MB for each partition’s database), and having short pre-processing times (2-5 seconds per partition). This makes it more suitable for a real-world smart city context with existing distributed/edge-based server infrastructure.

6.2 Future Work

While HPRoP is a good step towards realizing P2RPS, a significant number of improvements and alternative approaches remain to be explored. In terms of improving the Performance of HPRoP, the ideas presented in Sec. 5.5.4.3 can be considered. These improvements include: (1) using lighter privacy-preserving database query mechanisms, (2) parallelizing algorithm execution, and (3) optimizing the service region partitioning parameters. More efficient data structures for storing route information should also be explored since the current route databases contain a lot of redundant information in terms of shared sections — eliminating these could result in better execution times. The hierarchical routing algorithm itself can be replaced with a more sophisticated routing algorithm such as *Contraction Hierarchies* [24] which speeds up routing via shortcut edges on the network graph itself. This, however, requires a radical overhaul of the road network partitioning and indexing of routes that may be incompatible with HPRoP. Next, the Utility (or quality) of HPRoP’s routes can be improved by considering machine learning based predictive techniques [26, 27] for guiding the hierarchical algorithm in choosing the best sequence of partitions to query to result in a close-to-optimal route. In terms of improving Privacy, more sophisticated techniques for choosing dummy queries can be considered, as well as realizing out-of-order query execution for full route privacy. Finally, HPRoP’s limitation of being constrained to a single fixed service region can also be addressed in a scalable manner by treating each “service region” as a separate partition and simply performing privacy-preserving routing on that macro-level partition graph instead.

Acknowledgements

First and foremost, I would like to express my utmost gratitude to my supervisor, Professor Keiichi Yasumoto, who has provided a huge amount of support and being instrumental in making this research — and even just the chance to take this Ph. D. course in Japan — possible in the first place. He has helped in finding the key insights and refining the fundamental ideas that constitute the backbone of this research. He has inspired me to *see* seemingly impossible research problems as merely “complex and difficult” and, having found initial solutions, motivated me to seek out progressively better ones. I am truly grateful for everything.

I would also like to thank all the thesis committee members — Professor Fujikawa, Professor Kadobayashi, Professor Suwa, and Professor Matsuda — for providing me invaluable feedback and insights on how the quality of the research (as well as this dissertation) can be improved to the utmost. Their comments have been very helpful in making me recognize and address the “blind spots” in my research’s methodology and subsequent results caused by being overly familiar with the material. To that end, I would also like to thank our international and domestic collaborators — Professor Das, Professor Bhattacharjee, Professor Dubey, Professor Yamana, and Professor Yamaguchi — with whom the project from which the initial idea behind this research originated in the first place. Collaborating with them facilitated an exchange of ideas and information between fields that I normally would not have known about and explored beforehand. I would also like to thank both former and current faculty members of the Ubiquitous Computing Systems Laboratory — Professor Matsui, Professor Nakamura, Professor Mizumoto, Professor Fujimoto, and Professor Arakawa — for fostering and nurturing a very active research environment. Likewise, I would also like to thank my colleagues — Jose Paolo Talusan, Research Dawadi, and Professor Hyuckjin Choi — who have been very great and supportive friends during my

time in the laboratory, and our laboratories secretaries — Nao Yamauchi, Megumi Kanaoka, and Eri Ogawa — for all their help and advise in administrative matters and Japan life.

Most importantly, I would like to thank my family back in the Philippines: to my parents — Jerome and Grace — who supported me in my decision to pursue my Ph. D. degree and an atypical future career in research, and to my siblings — Paolo, Charlene, and Angela — for supporting me (and one another) and sticking together through great and not-so-great times.

Lastly, I would like to thank Gema for her unwavering support and overflowing patience on my admittedly *overly long* journey of seeing this research through to the very end.

Bibliography

- [1] S. Gambs, M.-O. Killijian, and M. N. del Prado Cortez, “Show me how you move and i will tell you who you are,” in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, pp. 34–41, 2010.
- [2] V. Primault, A. Boutet, S. B. Mokhtar, and L. Brunie, “The long road to computational location privacy: A survey,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2772–2793, 2018.
- [3] Y. Liang, Y. Liu, and B. B. Gupta, “Pprp: preserving-privacy route planning scheme in vanets,” *ACM Transactions on Internet Technology*, vol. 22, no. 4, pp. 1–18, 2022.
- [4] J. Zhou, S. Chen, K.-K. R. Choo, Z. Cao, and X. Dong, “Epn: Efficient privacy preserving intelligent traffic navigation from multiparty delegated computation in cloud-assisted vanets,” *IEEE Transactions on Mobile Computing*, 2021.
- [5] M. Li, Y. Chen, S. Zheng, D. Hu, C. Lal, and M. Conti, “Privacy-preserving navigation supporting similar queries in vehicular networks,” *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 2, pp. 1133–1148, 2020.
- [6] B. Baruah and S. Dhal, “A security and privacy preserved intelligent vehicle navigation system,” *IEEE Transactions on Dependable and Secure Computing*, 2022.
- [7] U. I. Atmaca, C. Maple, G. Epiphaniou, and M. Dianati, “A privacy-preserving route planning scheme for the internet of vehicles,” *Ad Hoc Networks*, vol. 123, p. 102680, 2021.

- [8] E. Ghosh, S. Kamara, and R. Tamassia, “Efficient graph encryption scheme for shortest path queries,” in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pp. 516–525, 2021.
- [9] C. Zhang, L. Zhu, C. Xu, K. Sharif, C. Zhang, and X. Liu, “Pgas: Privacy-preserving graph encryption for accurate constrained shortest distance queries,” *Information Sciences*, vol. 506, pp. 325–345, 2020.
- [10] C. Liu, L. Zhu, X. He, and J. Chen, “Enabling privacy-preserving shortest distance queries on encrypted graph data,” *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 192–204, 2018.
- [11] B. Chor and N. Gilboa, “Computationally private information retrieval,” in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pp. 304–313, 1997.
- [12] D. J. Wu, J. Zimmerman, J. Planul, and J. C. Mitchell, “Privacy-preserving shortest path computation,” *arXiv preprint arXiv:1601.02281*, 2016.
- [13] K. Mouratidis, “Strong location privacy: A case study on shortest path queries,” in *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pp. 136–143, IEEE, 2013.
- [14] L. Zhang, J. Li, S. Yang, and B. Wang, “Privacy preserving in cloud environment for obstructed shortest path query,” *Wireless Personal Communications*, vol. 96, no. 2, pp. 2305–2322, 2017.
- [15] F. Farokhi, I. Shames, and K. H. Johansson, “Private routing and ride-sharing using homomorphic encryption,” *IET Cyber-Physical Systems: Theory & Applications*, vol. 5, no. 4, pp. 311–320, 2020.
- [16] C. Benevolo, R. P. Dameri, and B. D’auria, “Smart mobility in smart city,” in *Empowering organizations*, pp. 13–28, Springer, 2016.
- [17] Z. Fan, X. Song, R. Jiang, Q. Chen, and R. Shibasaki, “Decentralized attention-based personalized human mobility prediction,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 4, pp. 1–26, 2019.

- [18] J. Feng, C. Rong, F. Sun, D. Guo, and Y. Li, “Pmf: A privacy-preserving human mobility prediction framework via federated learning,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 4, no. 1, pp. 1–21, 2020.
- [19] S. Sheng, E. Pakdamanian, K. Han, Z. Wang, J. Lenneman, and L. Feng, “Trust-based route planning for automated vehicles,” in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, pp. 1–10, 2021.
- [20] C. Samal, L. Zheng, F. Sun, L. J. Ratliff, and A. Dubey, “Towards a socially optimal multi-modal routing platform,” *arXiv preprint arXiv:1802.10140*, 2018.
- [21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.
- [22] A. V. Goldberg and C. Harrelson, “Computing the shortest path: A search meets graph theory,” in *SODA*, vol. 5, pp. 156–165, Citeseer, 2005.
- [23] T. Ikeda, M.-Y. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, and K. Mitoh, “A fast algorithm for finding better routes by ai search techniques,” in *Proceedings of VNIS’94-1994 Vehicle Navigation and Information Systems Conference*, pp. 291–296, IEEE, 1994.
- [24] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, “Exact routing in large road networks using contraction hierarchies,” *Transportation Science*, vol. 46, no. 3, pp. 388–404, 2012.
- [25] D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck, “Customizable route planning in road networks,” *Transportation Science*, vol. 51, no. 2, pp. 566–591, 2017.
- [26] M. Wilbur, C. Samal, J. P. Talusan, K. Yasumoto, and A. Dubey, “Time-dependent decentralized routing using federated learning,” in *2020 IEEE 23rd International Symposium on Real-Time Distributed Computing (ISORC)*, IEEE, 2020.

- [27] J. P. V. Talusan, M. Wilbur, A. Dubey, and K. Yasumoto, “Route planning through distributed computing by road side units,” *IEEE Access*, vol. 8, pp. 176134–176148, 2020.
- [28] M. Chase and S. Kamara, “Structured encryption and controlled disclosure,” in *Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16*, pp. 577–594, Springer, 2010.
- [29] S. Angel, H. Chen, K. Laine, and S. Setty, “Pir with compressed queries and amortized query processing,” in *2018 IEEE symposium on security and privacy (SP)*, pp. 962–979, IEEE, 2018.
- [30] M. H. Mughees, H. Chen, and L. Ren, “Onionpir: Response efficient single-server pir,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2292–2306, 2021.
- [31] S. J. Menon and D. J. Wu, “Spiral: Fast, high-rate single-server pir via fhe composition,” in *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 930–947, IEEE, 2022.
- [32] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, “One server for the price of two: Simple and fast single-server private information retrieval,” *Cryptology ePrint Archive*, 2022.
- [33] A. Beimel, Y. Ishai, and T. Malkin, “Reducing the servers computation in private information retrieval: Pir with preprocessing,” in *Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20*, pp. 55–73, Springer, 2000.
- [34] A. C.-C. Yao, “How to generate and exchange secrets,” in *27th annual symposium on foundations of computer science (Sfcs 1986)*, pp. 162–167, IEEE, 1986.
- [35] M. O. Rabin, “How to exchange secrets with oblivious transfer,” *Cryptology ePrint Archive*, 2005.

- [36] Confidential Computing Consortium, “A technical analysis of confidential computing v1.3.” [https://confidentialcomputing.io/wp-content/uploads/sites/85/2023/01/CCC-A-Technical-Analysis-of-Confidential-Computing-v1.3_Updated_November_2022.pdf]. Accessed: Oct. 27, 2023.
- [37] M. Sabt, M. Achemlal, and A. Bouabdallah, “Trusted execution environment: what it is, and what it is not,” in *2015 IEEE Trust-com/BigDataSE/Ispa*, vol. 1, pp. 57–64, IEEE, 2015.
- [38] M. Henson and S. Taylor, “Memory encryption: A survey of existing techniques,” *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–26, 2014.
- [39] P. Jauernig, A.-R. Sadeghi, and E. Stapf, “Trusted execution environments: properties, applications, and challenges,” *IEEE Security & Privacy*, vol. 18, no. 2, pp. 56–60, 2020.
- [40] F. Brasser, U. Müller, A. Dmitrienko, K. Kostianen, S. Capkun, and A.-R. Sadeghi, “Software grand exposure: {SGX} cache attacks are practical,” in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.
- [41] C. Dwork, A. Roth, *et al.*, “The algorithmic foundations of differential privacy,” *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.
- [42] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
- [43] T. Yamada, S. Kataoka, and K. Watanabe, “Heuristic and exact algorithms for the disjunctively constrained knapsack problem,” *Information Processing Society of Japan Journal*, vol. 43, no. 9, 2002.
- [44] Ministry of Land, Infrastructure, Transport and Tourism, Japan, “Fy2015 road traffic census summary.” [<http://www.mlit.go.jp/road/census/h27/>]. Accessed: Oct. 4, 2020.

- [45] PTV Group, “Visum and vissim traffic simulators.” |<https://www.ptvgroup.com/en/>|. Accessed: Oct. 4, 2020.
- [46] Kinki Regional Development Bureau, Ministry of Land, Infrastructure, Transport and Tourism. |<https://www.kkr.mlit.go.jp/plan/pt/>|. Accessed: Oct. 4, 2020.
- [47] K. Schittkowski, “Nonlinear programming methods with linear least squares subproblems,” in *Evaluating Mathematical Programming Techniques*, pp. 200–213, Springer, 1982.
- [48] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, “Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 23, no. 4, pp. 550–560, 1997.
- [49] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *Cryptology ePrint Archive*, 2012.
- [50] D. Delling, A. V. Goldberg, I. Razenshteyn, and R. F. Werneck, “Graph partitioning with natural cuts,” in *2011 IEEE International Parallel & Distributed Processing Symposium*, pp. 1135–1146, IEEE, 2011.
- [51] P. Sanders and C. Schulz, “Distributed evolutionary graph partitioning,” in *2012 Proceedings of the fourteenth workshop on algorithm engineering and experiments (ALENEX)*, pp. 16–29, SIAM, 2012.
- [52] A. Schild and C. Sommer, “On balanced separators in road networks,” in *International Symposium on Experimental Algorithms*, pp. 286–297, Springer, 2015.
- [53] G. Aggarwal, S. Gollapudi, and A. K. Sinop, “Sketch-based algorithms for approximate shortest paths in road networks,” in *Proceedings of the Web Conference 2021*, pp. 3918–3929, 2021.
- [54] F. Tiausas, K. Yasumoto, J. P. Talusan, H. Yamana, H. Yamaguchi, S. Bhattacharjee, A. Dubey, and S. K. Das, “Hprop: Hierarchical privacy-preserving

route planning for smart cities,” *ACM Trans. Cyber-Phys. Syst.*, vol. 7, oct 2023.

- [55] F. Tiausas, J. P. Talusan, Y. Ishimaki, H. Yamana, H. Yamaguchi, S. Bhattacharjee, A. Dubey, K. Yasumoto, and S. K. Das, “User-centric distributed route planning in smart cities based on multi-objective optimization,” in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 77–82, 2021.
- [56] J. P. Talusan, F. Tiausas, S. Stirapongsasuti, Y. Nakamura, T. Mizumoto, and K. Yasumoto, “Evaluating performance of in-situ distributed processing on iot devices by developing a workspace context recognition service,” in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 633–638, IEEE, 2019.
- [57] M. J. Islam, J. P. Talusan, S. Bhattacharjee, F. Tiausas, S. M. Vazirizade, A. Dubey, K. Yasumoto, and S. K. Das, “Anomaly based incident detection in large scale smart transportation systems,” in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 215–224, IEEE, 2022.
- [58] M. J. Islam, J. P. Talusan, S. Bhattacharjee, F. Tiausas, A. Dubey, K. Yasumoto, and S. K. Das, “Scalable pythagorean mean based incident detection in smart transportation systems,” *ACM Transactions on Cyber-Physical Systems*, 2023.

Publication List

Journals

- [1] F. Tiausas, K. Yasumoto, J. P. Talusan, H. Yamana, H. Yamaguchi, S. Bhattacharjee, A. Dubey, and S. K. Das, “Hprop: Hierarchical privacy-preserving route planning for smart cities,” *ACM Trans. Cyber-Phys. Syst.*, vol. 7, oct 2023
(for *Chapter 5*)

International Conferences

- [1] F. Tiausas, J. P. Talusan, Y. Ishimaki, H. Yamana, H. Yamaguchi, S. Bhattacharjee, A. Dubey, K. Yasumoto, and S. K. Das, “User-centric distributed route planning in smart cities based on multi-objective optimization,” in *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 77–82, 2021
(for *Chapter 4*)

Other Publications

- [1] J. P. Talusan, F. Tiausas, S. Stirapongsasuti, Y. Nakamura, T. Mizumoto, and K. Yasumoto, “Evaluating performance of in-situ distributed processing on iot devices by developing a workspace context recognition service,” in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 633–638, IEEE, 2019
- [2] M. J. Islam, J. P. Talusan, S. Bhattacharjee, F. Tiausas, S. M. Vazirizade, A. Dubey, K. Yasumoto, and S. K. Das, “Anomaly based incident detection in large scale smart transportation systems,” in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 215–224, IEEE, 2022
- [3] M. J. Islam, J. P. Talusan, S. Bhattacharjee, F. Tiausas, A. Dubey, K. Yasumoto, and S. K. Das, “Scalable pythagorean mean based incident detection in smart transportation systems,” *ACM Transactions on Cyber-Physical Systems*, 2023