B.Sc. in Computer Science and Engineering Thesis

An Efficient Approach for Real-Time Crowdsourced Package Delivery using Public Transport Networks

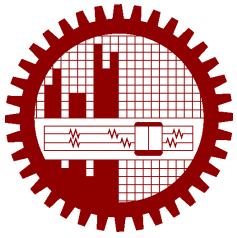
Submitted by

Fariha Tabassum Islam

201305007

Supervised by

Tanzima Hashem



Department of Computer Science and Engineering

Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

October 2018

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "An Efficient Approach for Real-Time Crowdsourced Package Delivery using Public Transport Networks", is the outcome of the investigation and research carried out by us under the supervision of Tanzima Hashem.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Fariha Tabassum Islam
201305007

CERTIFICATION

This thesis titled, **"An Efficient Approach for Real-Time Crowdsourced Package Delivery using Public Transport Networks"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in October 2018.

Group Members:

Fariha Tabassum Islam

Supervisor:

Tanzima Hashem
Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

First and Foremost, I am grateful to my thesis supervisor, Dr. Tanzima Hashem, who has constantly guided and supported me in every step of my thesis. She has helped me in easy and hard times in completing the thesis with her knowledge, patience and kindness and motivated me to continue my thesis. I could not have completed my thesis without her guidance and support.

I am grateful to Saiedur Rahaman and Dr. Flora Salim for their guidance and sharing resources.

I am grateful to my parents who constantly inspired and encouraged me throughout my thesis.

Dhaka
October 2018

Fariha Tabassum Islam

Contents

List of Figures

List of Tables

List of Algorithms

ABSTRACT

The surge in popularity of e-commerce has increased the demand of fast and cost-efficient package delivery and this demand has caused the rise of crowdsourced package delivery. In this thesis, we introduce a novel concept of crowdsourced package delivery using the public transport networks. People usually travel in public transport networks for work, grocery-shopping, taking child to school etc. They can easily carry a package during their journies and transfer them to other passengers. Using the public transport network for the crowdsourced package delivery increases the number of workers available for package delivery. In this thesis, we propose an efficient approach to find the package delivery route with the minimum delivery time based on the available journey plans of the travellers in the public transport networks. The underlying idea of our approach is to incrementally find a shortest path in the public transport network for the source and destination locations of the pakage until the path matches with the available journey plans. To compute the package delivery path in real time, we propose heuristics to estimate the shortest path for the package delivery in the public transport network and to match the package delivery path with the journey plans of multiple passengers. Our approach allows package transfers between the public transport passengers and thereby increase the success rate of the crowdsourced package delivery. We evaluate our approach in experiments using real public transport network and synthetic user journey datasets generated from real check-in data. We find that crowdsourced package delivery using public transport networks is a feasible solution and show that our approach incurs less processing overhead than the optimal one in return of decreasing the package delivery rate and increasing the package delivery time slightly.

Chapter 1

Introduction

In recent years, the need of fast and cost-efficient package delivery has risen dramatically with the growth of e-commerce. The number of packages delivered per day is more than ever. Traditional pickup and delivery method with a fixed set of workers cannot satisfy the large need of package delivery. Furthermore, traditional methods require a large number of vehicles on the road for delivering packages and they incur large fuel-costs, pollute the environment and cause extra traffic-load in peak hours. For example, in 2017, United Package Service (UPS) held 22% of the global market share of express and courier service providers [1]. Their daily global delivery volume is 22 million packages and documents and their delivery fleet includes over 112 thousand vehicles in 2017 [2]. On the other hand, crowdsourced package delivery provides the benefit of speedy and low-cost delivery, does not pollute the environment and reduces the traffic load. Hence, many package delivery services [3–10] have started using crowdsourcing [11], like using taxi drivers, commuters, travelers as worker to deliver packages for minimal incentives.

Existing crowdsourced package delivery methods use various transport mediums like private car, taxi even walking. However, existing crowdsourced delivery services have following limitations:

i A package might be heavy for a single worker to carry on and it might be difficult to match a package delivery request with more than one worker's travel plan at the same time.

ii A package may need space, which is not available in a taxi or a private car. Thus workers who do not have adequate space required for a package becomes ineligible for the package delivery.

iii A package delivery request may not match with a single worker's travel plan and require transfer among multiple workers, which might become hard to arrange.

We propose a solution that overcome the above mentioned issues. In this thesis, we introduce a

novel concept of crowdsourced package delivery using the public transport network.

Everyday many people use public transports for their daily routine works. Now-a-days, these public transport network services provide people the facility to pre-plan their journeys optimally given starting points, destinations and departure time ranges. Daily many people travel some routine paths using public transports for work, grocery-shopping, taking kid to school etc. They might be interested in carrying packages for delivery during their journey if they are rewarded properly in return. If these passengers are targeted as workers, then we can use their journey plans to calculate delivery routes of package and the package delivery chance increases. Furthermore, the utility of the public transport capacity will be increased and there will be less vehicles for the particular purpose of package delivery which in turn will reduce traffic loads and carbon emissions.

Since in public transports many passengers travel together, it would not be hard to manage multiple workers for heavy packages. Furthermore, passengers have overlaps in their journey plans in public transports, which increases the probability to match a package delivery request with transfers. In addition, in cities of developed country the public transports are underutilized, except in peak hours, which also solve the problem of required space for a package.

In this thesis, we propose a method to find the routes for delivering packages with the help of public transport passengers by using their journey plans. In our system, passengers who are interested in becoming crowdsourced delivery workers share their journey plans . Each worker has a maximum detour limit, which represents how much the worker is willing to deviate from his original plan in time. On the other hand, each package is delivered from a source location to a destination location and has additional information like the package pick up time.Given all these information, package delivery routes are calculated that include the passengers who will carry a package and the time and place of the package transfers among the passengers. Since package deliveries must be fast and cost-efficient, we focus on minimizing each package delivery time. By minimizing the package delivery time, fast package delivery is ensured.

The challenges for finding the package delivery path with the minimum delivery time is to explore a large number of possible options from a huge set of journey plans in the large public transport network. Exploring all possible options for every package delivery request would incur extremely high processing overhead. To adderss this issue, we develop a heuristic approach to find the delivery path for the package using the journey plans in the large public transport network with reduced processing overhead in return of increasing the packet delivery time and decreasing the delivery rate slightly. The underlying idea of of our heuristic solution is to incrementally find shortest paths for the package package delivery and then to check whether there exist a match of the shortest path with the journey plans of the travellers in the public transport network. To increase the efficiency for finding the shortest paths, we summarize the public transport network (PTN) graph and run the shortest path algorithm in the summarized graph.

The paths found in the summary graph in used to construct a smaller search space. The shortest paths found from the refined search space is matched with available journey plans using greedy approach because finding the best set of travellers to carry a package is unduly expensive. The matching approach allows detour if there is no user in some part of the path.

The contributions of this thesis are summarized as follows:

1. We introduce the concept of package delivery with crowd using the public transport network and formulate the problem.

2. We present an efficient approach for finding the package delivery path using the journey plans of the crowd in the public transport network. We propose a heuristic algorithm to incrementally find shortest paths for package delivery in the public transport network. In our heuristic, we summarize the public transport network graph in a novel way to refine the search space for finding the shortest paths with reduced processing time and high accuracy. We also develop a heuristic to match the shortest path with the journey plans of the travellers in the public transport network to find a delivery route for the package delivery.

3. We evaluate the performance of our heuristic approach with real and synthetic datasets and compare our approach with optimal method in extensive experiments.

The rest of the thesis is organized as follows: Chapter 2 covers the brief discussion of existing works related to our work. Chapter 3 includes necessary notations and formal problem statement. In Chapter 4, we give a brief overview of our approach to solve the problem and then each step of our solution is explained in details. In Chapter 5, we thoroughly evaluate the performance of our approach and feasibility of the concept. We conclude the thesis in chapter 6 and provide future working direction.

Chapter 2

Literature Review

In this chapter, we mention the existing research works that are related to our work. In Section 2.1 research works related to crowdsourced package delivery are discussed. We summarize the public transport graph and find shortest path to solve our problem. In Section 2.2 we discuss existing works on graph summarization. Section 2.3 describes existing works related to k-shortest paths algorithm.

2.1 Package Delivery with Crowd

Among traditional delivery related works, a scheduling of pickup and delivery problem with package transfers among delivery-vehicles has been proposed in [12] with constraints like, vehicle capacity, pickup and delivery time-window with the objective of delivering maximum number of packages in the minimum time. The authors solve the pickup and delivery problem more efficiently by planning route using exchange of packages among delivery-vehicles. But this approach of delivery is not as flexible and cheap as crowdsourced delivery.

Many works have been done on crowdsourcing in recent few years focusing on different aspects like sensing [13–15], privacy [16], task assignment [15, 17, 18], route-recommendation to crowd-worker [19] and delivery [20–25]. Since our work focuses on crowdsourced package delivery, we are only concerned about the works on package delivery with crowd. The existing works on crowdsourced delivery can be compared based on four criteria which are discussed as follows:

i **Future locations of crowd delivery workers known or unknown:** Among existing works on crowdsourced delivery, in [21–23] the future locations of delivery workers are not known in advance. We know the future locations in advance in [24, 25] and both cases are considered in [20]. Workers' future locations are needed for calculating delivery

route. In those works where future locations are unknown, they are predicted from human mobility pattern or historical data.

In the paper [20], when workers' future locations are known, a routing graph is formed from that information and an optimal delivery path is found from that graph for each package. When the future locations of workers are not known, the authors predict those locations from past geo-tagged tweets and ranks each worker based on their probability to reach destination. A package is repeatedly transferred to higher ranked users until it reaches destination.

A distribution method is proposed in [26] to deliver packages using local pre-existing mobility routine. In this paper, human mobility is predicted from cell tower data history and that prediction is used to plan delivery routes for packages. The quality of delivery service depends on the accuracy of the prediction.

The authors in [21] propose to deliver packages using taxi-drivers while attending passenger requests. They pre-calculate offline optimal paths from historical taxi data for all pair of interchange stations. A package is assigned to a nearby taxi if it offers less expected delivery time than calculated. The delivery depends on availability of taxi in real time.

In the paper [22], human mobility pattern is traced from workers' daily mobility data (e.g. cell tower data) and package delivery route optimization is done using the concept of message routing in pocket switched network. The delivery depends on the availability of workers near package safety boxes in real time.

Thus, the problem with predicting future location of worker is that, uncertainty and unexpected long delay is introduced in the package delivery system. Delivering packages in certain time ranges may fail for some packages and those packages cannot be identified in advance. Hence, it becomes hard to respond to unexpected delays.

In both [24, 25], package delivery requests are matched with drivers whose future travel plans are known. Since knowing the future locations of workers prevents the limitations like, unexpected delays, our work takes the journey plans of the worker as input.

ii **Transfer of packages:** The concept of transferring package from worker to worker is important for crowdsourced delivery, because workers may not be available for all pair of sources and destinations. Without transfer, all packages cannot reach destination. In both [24, 25] transfer of package is not available which reduces delivery chances.

Package transfer is implemented in existing literature in two ways: using safety boxes and worker to worker direct transfer. In [21, 22, 26], packages are kept in interchange stations or in safety boxes or in exchange points for transfer. Workers pickup packages from safety boxes and delivers to other safety boxes. The problem with safety boxes is that, they are not available everywhere. Hence, the possible locations of transfers become limited. In [20], package is transferred from worker to worker directly. Since

Paper	Criteria			
	Future location of worker	Transfer of packages	Detour limit	Objective
[20]	Unknown or known	Worker to worker	Same for all	Minimize each package delivery time
[26]	Unknown	Safety box	No detour	Minimize each package delivery time
[21]	Unknown	Safety box	Same for all	Minimize each package delivery time
[22]	Unknown	Safety box	No detour	Balance higher profit, lower cost and higher quality of service
[25]	Known	No transfer	Different for all	Minimize system-wide delivery cost
[24]	Known	No transfer	Different for all	Maximizing delivery task assignment and minimizing delivery cost
Our work	*Known*	*Worker to worker*	*Different for all*	*Minimize each package delivery time*

Table 2.1: Comparison of existing literature on crowdsourced package delivery

direct transfer overcomes the shortcoming of safety boxes, we use worker to worker direct transfer method in our thesis.

iii **Detour limit:** Detour limit of a worker represents how far a worker is prepared to stray from his original path to deliver a package. This concept is essential for delivering all requested packages for the same reason as package transfer. The works in [22, 26] do not allow detour of workers.

The detour limit can be same for all workers or different for every worker. The concept of same detour limit for all workers is used in [20, 21]. However, this does not reflect the individual requirement of each worker. In both [24, 25], every worker inputs his own detour limits. We allow individual detour limit for each worker because it better reflects the demand of each worker.

iv **Objective:** The existing literature of crowdsourced delivery focus on a variety of objectives. The objectives of the works in [20, 21, 26] are same, which is minimizing each package delivery time. They try to find the optimal or near-optimal delivery path using various approach. However, finding a optimal path for a package delivery may use some workers such that no path or a worse path is found for another delivery. Thus, this objective does not reflect practical necessity. The authors in [25] aims to minimize the total delivery cost. The work in [22] aims to find a balance among higher profit, lower delivery cost and higher quality of service. The work in [24] has the objective of maximizing delivery task assignments and minimizing delivery cost.

In package delivery, faster delivery is very important. In this thesis, we minimize each package delivery time to improve the quality of service.

From the Table 2.1, it can be seen that any single work in the above mentioned crowdsourced delivery related works does not include all the best features. But, we combined all of them in our thesis. Thus, our work overcomes the limitations of existing works.

2.2 Graph Summarization

Our solution approach summarizes the PT network graph to reduce search space for route finding. The PT network is different from road network [27] since the former one contains a timetable. We could not fit any existing paper on summarizing the PT network to our purpose. The paper [28] converts a time-table based transport network graph into a multilevel graph where each higher level contains less nodes than the lower levels. Here, the search space is reduced only if both the source and the destination are in higher level. Thus, this paper does not fit our purpose.

2.3 The k-Shortest Paths Problem

In our approach, an algorithm for finding k-shortest paths is run on summary graph. The k-shortest paths (KSP) problem is classified into two types: KSP where paths must be loopless and KSP with no restriction on paths (loop is allowed).

- *The k-shortest loopless paths problem:* For this problem, the algorithm by Yen [29] is the state-of-art algorithm with the worst-case time complexity $\mathcal{O}(kn(SP(s,d))$, where $SP(s,d)$ represents the worst case time complexity of solving the shortest path problem. Since the time complexity to calculate the shortest path is $\mathcal{O}(m + n\log n)$ at minimum, the worst-case time complexity of Yen's algorithm is $\mathcal{O}(kn(m + nlogn))$ at best. Each work in [30–32] provides a more efficient implementation of Yen that does not improve over the worst-case time complexity, but computational experiments reveal their better performances.

- *The k-shortest paths problem with cycle:* For this problem, the algorithm by Eppstein [33] has an excellent worst-case time complexity of $\mathcal{O}(m + n\log n + k)$. Though the work in [34] does not improve Eppstein's worst-case time complexity, it performs better than Eppstein in practice for $k \leq 100$. The work in [35] provides a new algorithm for KSP problem and [36] provides an improvement of Eppstein's algorithm. These two

works do not improve the worst-case time complexity but perform better than Eppstein in practice. One performs better than the other in some cases.

Therefore, computing KSP without restriction is faster than loopless KSP. Thus, after finding KSP using algorithms of second type, paths with loops can be detected and removed to find k'-shortest loopless paths, where $k' \leq k$. For $k \leq 100$, the performances of algorithms in [34], [35] and [36] are not much different. Since, the KSP found in the summary graph do not need to be loopless and we do not need high value of k, any of them can be used.

2.3.1 The k-Shortest Paths Problem in the Public Transport Network

In our solution approach, k-shortest paths need to be found in the PT network graph given the source, the destination and the start time. As the edge-costs are time-dependent in PT network graph, KSP algorithms mentioned earlier cannot be used directly. The paper [37] proposes adaptations of some existing algorithms to find KSP in the transport network graph, which includes both the road network and the PT network. The authors in this paper proposes the adaptation of Yen's [29] (or Lawler's [30]) algorithm to find k-shortest loopless paths. They proposed to adapt the REA [35] algorithm to find k-shortest paths without restriction to cycle. They also provided an iterative variant of REA and suggests a heuristic approach to detect and suppress loops while searching.

Though it is possible to use an adaptation from [37] for our approach, but this paper focuses on general transport network graph. The paper in [38] provides a better option because, k-shortest paths are found in timetable-based PT networks by adapting the algorithm in [34]. It represents the PT network in the time-expanded model and in the time-dependent model and finds KSP based on the earliest arrival time, the shortest travel time and the minimum number of transport changes in a path. Among them, KSP based on the earliest arrival time fits our purpose. In addition, this paper has impressive experimental results. Thus, the algorithm of this paper for time-dependent approach based on earliest arrival time in used in our solution. The algorithm is slightly modified while using, since the PT network graph is represented in more condensed form than time-dependent model in our thesis.

Chapter 3

Problem Formulation

In this chapter, we define the basic concepts and introduce notations and assumptions used throughout this thesis, and formally define our problem.

3.1 Preliminaries

DEFINITION 1 (Public transport network): The graph $G = (V, E, T)$ represents the public transport network where each node $v_k \in V$ represents a stop (e.g. bus stop, train station) and each edge $e_{ij} \in E$ represents a direct connection between two adjacent stops, v_i and v_j. Each node v_k has a location $loc(v_k)$. The edge cost $c(e_{ij}) = c(v_i, v_j)$ represents the travel-time from v_i to v_j using the public transport. The timetable of connection e_{ij} is represented by $T_{ij} = \{(t^1_{v_i}, t^1_{v_j}), \ldots, (t^n_{v_i}, t^n_{v_j})\}$, where $t^k_{v_i} < t^{k+1}_{v_i}$ and $t^k_{v_j} - t^k_{v_i} = t^{k+1}_{v_j} - t^{k+1}_{v_i} = c(e_{ij})$ for $k \in [1, n]$.

Figure 3.1 shows a map of the public transport network in Melbourne city which is converted to graph.

DEFINITION 2 (Package delivery request): Let, $\mathbb{D}$ be a set of package delivery requests. A package delivery request $D_i \in \mathbb{D}$ is composed of the 3-tuple: $< P_i, l_s, l_d >$ where P_i represents the package to deliver from location l_s to location l_d and t_b represents the time delivery begins.

DEFINITION 3 (Journey edge): Each journey edge j_k is a $5-$tuple $< v_{dept}, v_{arr}, t_{dept}, t_{arr}, r_{id} >$, where t_{dept} represents the departure time from stop $v_{dept} \in V$, t_{arr} represents the arrival time at stop $v_{arr} \in V$ and r_{id} represents the transport id.

DEFINITION 4 (User participation request): Let, S be a set of user participation requests. A user participation request $S_i \in S$ is composed of the following 3-tuple: $< u_i, d_{dt}, J >$ where

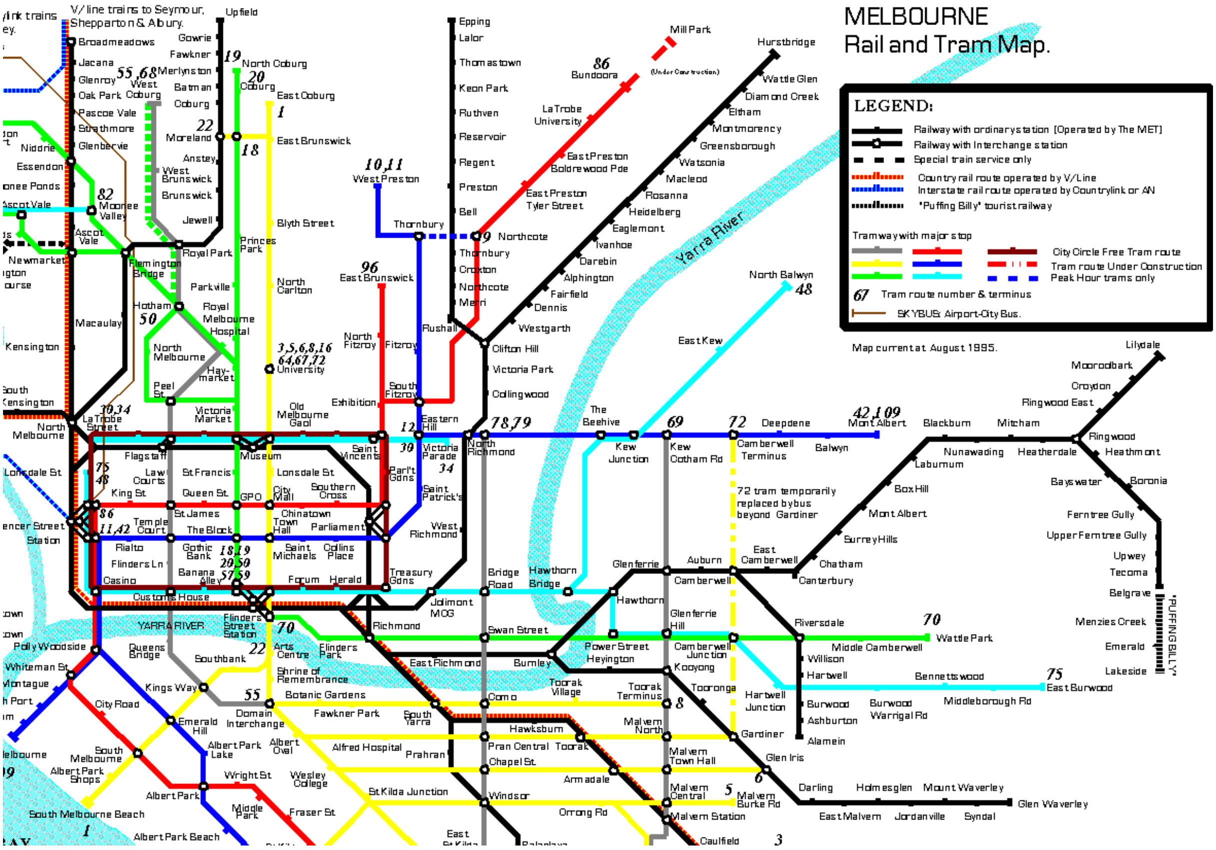


Figure 3.1: The public transport network map of Melbourne city

user $u_i \in U$ is a passenger who uses the public transport network, d_{dt} represents the detour time limit of user u_i. The journey plan $J = (j_1, j_2, \ldots, j_n)$ is a sequence of journey edges traveled in order by user u_i.

DEFINITION 5 (Package delivery path): A package delivery path $L_k^i = (j_1, j_2, \ldots, j_n)$ is the kth shortest delivery path, which consists of a sequence of journey edges traveled in order by package P_i.

DEFINITION 6 (Package delivery route): A delivery route $Q_i = (q_1, q_2 \ldots, q_n)$, represents a sequence of route edges traveled in order by the package P_i. A route edge q_k is 2-tuple: $< j_k, u_k >$, where j_k is a journey edge and $u_k \in U$ is the passenger who will carry the package along the journey edge.

3.2 Problem Statement

Thus, the problem of crowdsourcing package delivery using public transport network can be stated as follows: *Given a public transport network graph G, a set of package delivery requests D and a set of user participation requests S, compute a package delivery route for each delivery request that minimizes each package delivery time while maintaining the detour limit of users.*

Chapter 4

Our Solution

In this chapter, we introduce our solution approach to the problem formulated in Chapter 3, which is delivering packages using the public transport network with the help of its passengers. To solve this problem we incrementally find heuristic at most t shortest paths for a package delivery request from the public transport network graph. Our approach is made computationally efficient by reducing the search space for finding paths in the public transport network graph. We find at most t delivery paths for each package delivery to ensure we can find users to match the path of a package for delivery. When we find a delivery path for a package, we match the path with the journey plans of users and keep repeating this process until a package delivery route for that package is found such that user specified detour limit is maintained.

This chapter is organized as follows: Section 4.1 gives the overview of our approach to compute a delivery route for a package delivery. Each step of refining the search space in the original graph is described in Section 4.2. The steps needed for finding package delivery paths for a package and matching user participation requests with those paths to find the package delivery route are explained in Section 4.3.

4.1 Overview

To solve the problem we first focus on finding package delivery paths for each package without considering the journey plans of the workers. Finding the delivery path with the shortest delivery time is computationally expensive, since the PTN graph is very large. In addition, the need to consider timetable makes it more expensive since each edge in this graph contains multiple schedules and the package delivery time depends on when the delivery starts. We need a way to make the computation time small and practical. Thus, we summarize the PTN graph and find up to k shortest paths for a package delivery in the summarized graph. These shortest paths enable us to find a refined search space in the PT graph that is likely to include the k shortest

paths for the package delivery in the original graph.

Because we did not consider user participation requests while finding a route for delivery, a case may occur where no user is available for the path with the shortest delivery time or this path cannot maintain detour limit constraints of users. This problem is solved by finding at most top-k paths for delivery for each package.

After finding each path, that path is matched with user journey plans to find suitable users and transfer points that maintain all constraints and a package delivery route is formed for each package.

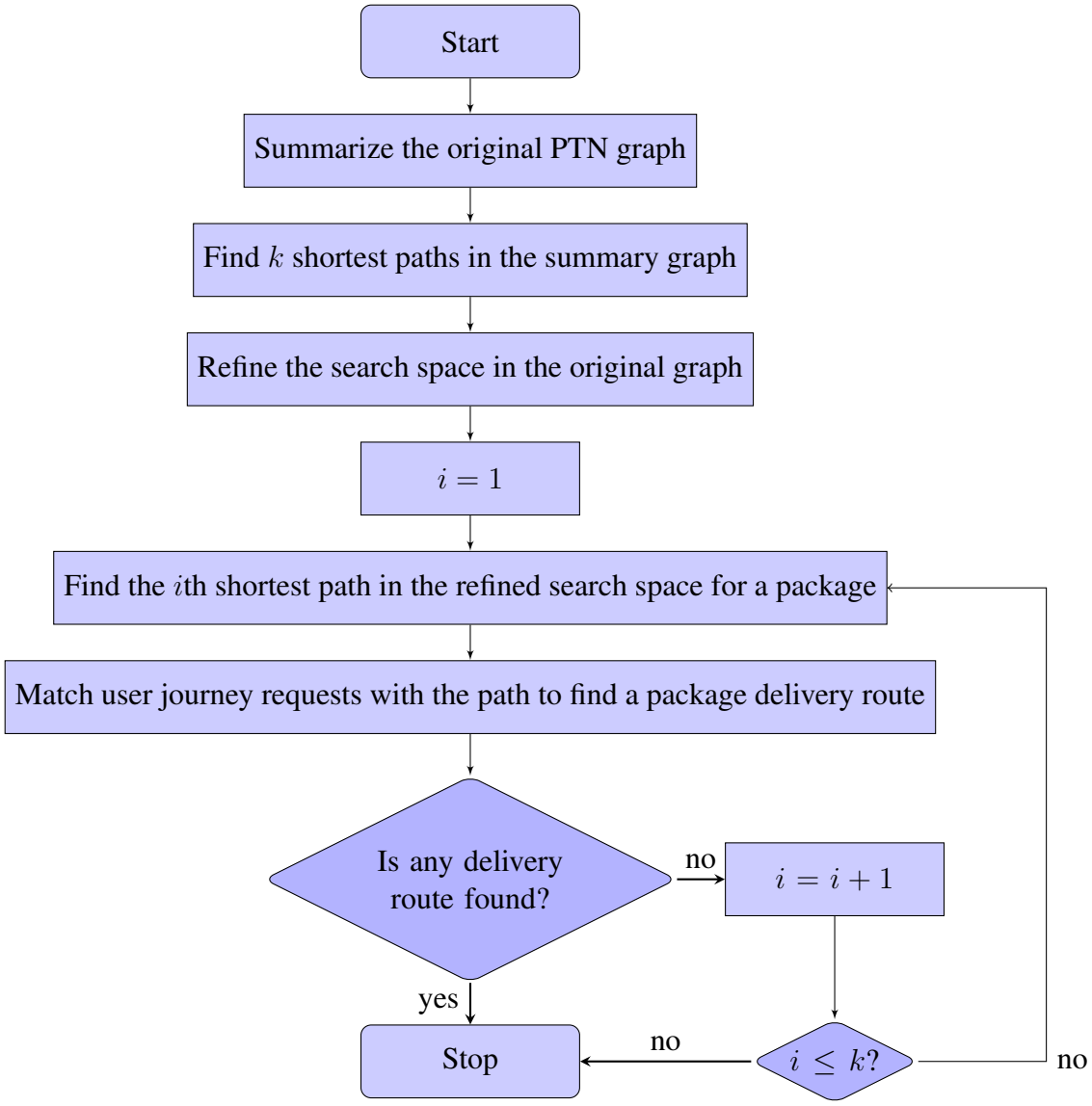


Figure 4.1: Overview of our approach for crowdsourced package delivery using the public transport network

Figure 4.1 gives an overview of our solution approach. Our solution approach finds the delivery route for each package iteratively. Each iteration can be divided into two phases. First, we

find a delivery path in the PTN graph for each package. Then we match that path with the journey plans of users and find users and transfer points for that package. Since the PTN graph is large with thousands of nodes and millions of edges, to find delivery path efficiently we first summarize the original PTN graph. The summary graph is pre-computed. Then we find at most t shortest paths in the summary graph, which helps us to refine the search space in the original graph using the upper bound. We find heuristic at most t-shortest paths incrementally from the refined search space. When a path is found, they are matched with user journey plans to find a package delivery route for each package that includes users and journey edges. If a delivery route is not found for the path, then the next shortest delivery path within the refined search space is calculated and matching is repeated. This process continues until a package delivery route is found or a pre-defined number of iterations are completed.

4.2 Steps of Refining the Search Space

We present the steps of refining the search space in the original PTN graph in this section. These steps are designed such that real time efficient computation of package delivery paths are possible.

4.2.1 Summarizing the PTN Graph

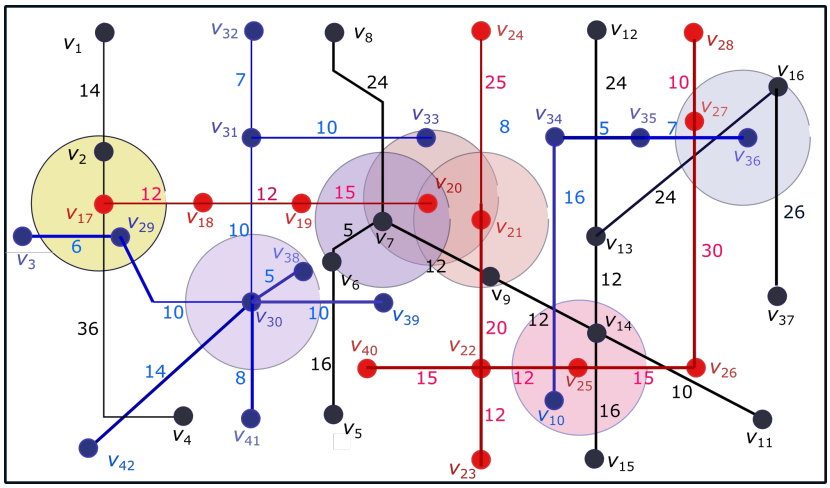
Summarizing the PTN graph is the first step in refining the search space.

Given PTN graph $G(V, E)$ is summarized into the graph $G_S = (V_S, E_S)$ with less nodes and less edges to reduce the search space. A node v_i in G is a supernode s_i in G_S, if the total number of outgoing edges from v_i and other nodes within the distance r from v_i is greater than T_h. Thus, a supernode s_i is defined with respect to a node v_i. It can be represented as a 2-tuple: $s_i =< v_i, N_i >$ where $v_i \in V$ is the center of the supernode s_i and $N_i \subset V$ is a set of other nodes of G that are within the distance r from v_i. N_i is called the set of neighbour nodes of v_i. A node in G can be part of multiple supernodes. The vertices of the summarized graph represent the supernodes and there is an edge between two supernodes if any of the following rules are satisfied.

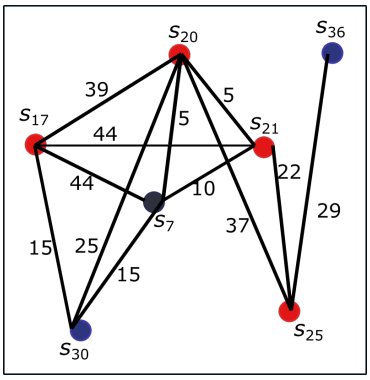
The graph summarization heuristic follows the following rules:

1. *If two different supernodes s_i and s_j, share a common node v_k, then s_i and s_j has an edge in G_S. The cost of this edge (s_i, s_j) is*

$$c(s_i, s_j) = c(s_i.v_i, v_k) + c(v_k, s_j.v_j) \tag{4.1}$$



(a) Original graph G of a transport network. Each color represents different transportation mode. Circles wiht radius r have been drawn around nodes eligible for supernodes for $T_h = 4$.



(b) Summarized graph G_S of G. Each node v_i in G eligible for being supernode is converted into s_i in G_S. Edges have been added according to rules in Section 4.2.1

Figure 4.2: Example of constructing a summary graph from the original PT network graph.

2. *If two nodes v_l and v_m from two different supernodes s_i and s_j respectively, has an edge $e_{lm} \in E$ in G, then s_i and s_j has an edge $e_{ij} \in V_S$ in G_S. The cost of this edge (s_i, s_j) is*

$$c(s_i, s_j) = c(s_i.v_i, v_l) + c(v_l, v_m) + c(v_m, s_j.v_j) \tag{4.2}$$

3. *If two nodes v_l and v_m from two different supernodes s_i and s_j respectively, have distance less than r, then s_i and s_j have an edge in G_S. The cost of this edge (s_i, s_j) is*

$$c(s_i, s_j) = c(s_i.v_i, v_l) + c(v_l, v_m) + c(v_m, s_j.v_j) \tag{4.3}$$

4. *If two nodes from two different supernodes s_i and s_j are connected via another series of nodes v_{k_1}, v_{k_2}, ..., v_{k_i} in G, where v_{k_i} is neither a supernode nor included in the supernode of other nodes, then s_i and s_j has an edge in G_S. The cost of this edge (s_i, s_j) is*

$$c(s_i, s_j) = c(s_i.v_i, v_{k_1}) + c(v_{k_1}, v_{k_2}) + ... + c(v_{k_i-1}, v_{k_i}) + c(v_{k_2}, s_j.v_j) \tag{4.4}$$

If two supernodes share an edge according to multiple rules, then the rule that results in minimum cost is chosen. If there is no edge between v_i and v_j while calculating edge cost in the summary graph, then we assume $c(v_i, v_j) = walkingTime(loc(v_i), loc(v_j))$, where *walkingTime*$(location1, location2)$ returns the approximate time to walk from $location1$ to $location2$.

The construction of the summary graph is explained using Figure 4.2. For this example, we assume edge cost is represented in minutes and it takes 5 minutes to walk distance r.

In Figure 4.2, the supernode s_{17} in G_S is defined with respect to v_{17} in G and includes nodes v_2

, v_{29} which are within r distance from v_{17}. The supernode s_{17} is eligible as a supernode because v_{17}, v_{29} and v_2 have total 5 outgoing edges from the circle with radius r which is greater than the threshold T_h, where $T_h = 4$. Here, v_{17} is the center node and $s_{17}.N_{17} = \{v_{29}, v_2\}$.

There is an edge between s_7 and s_{21} in Figure 4.2, because the circles of v_7 and v_{21} share a common node v_{20} in G. Here, $c(s_7, s_{21}) = c(v_7, v_{20}) + c(v_{20}, v_{21}) = 5 + 5 = 10$.

The supernodes s_{21} and s_{25} are connected with an edge in G_S, because v_9 included in supernode s_{21} and v_{14} included in supernode s_{25} share an edge in G. Hence, $c(s_{21}, s_{25}) = c(v_{21}, v_9) + c(v_9, v_{14}) + c(v_{14}, v_{25}) = 5 + 12 + 5 = 22$.

Again, the supernodes s_{30} and s_7 have an edge between them, because though v_{38} included in supernode s_{30} and v_6 included in supernode s_7 are not connected, the distance between v_{38} and v_6 is less than r in G. Thus, $c(s_{30}, s_7) = c(v_{30}, v_{38}) + c(v_{38}, v_6) + c(v_6, v_7) = 5 + 5 + 5 = 15$.

The supernode s_{17} has an edge with s_{20}, because v_{17} and v_{20} in G are connected via a series of vertices v_{18} and v_{19}, where v_{18} and v_{19} are neither supernode nor included in the supernode of other nodes. Here, $c(s_{17}, s_{20}) = c(v_{17}, v_{18}) + c(v_{18}, v_{19}) + c(v_{19}, v_{20}) = 12 + 12 + 15 = 39$.

Each edge in the summary graph represents a path from the center of one supernode to another. The path corresponding to each edge is saved in the summary graph. A saved path w_{ij} =*saved-path*(s_i, s_j)=*saved-path*(e_{ij}) is a 4-tuple: $< s_i, s_j, SP, d >$, where $s_i, s_j \in V_S$, $SP = (s_i.v_i, v_{k_1}, \ldots, v_{k_n}, s_j.v_j)$ is a sequence of nodes from the center of supernode s_i to the center of supernode s_j and d is the path-cost and equal to the edge cost, $d = c(s_i, s_j)$. W is a set of saved paths.

We introduce two notations $src_{sn}(v_i)$ and $dest_{sn}(v_i)$, $\forall v_i \in V$. The notation $src_{sn}(v_i)$ represents a set of supernodes that includes v_i in the summary graph or it represents the nearest supernode which can be reached from v_i. $dest_{sn}(v_i)$ is represents a set of supernode that includes v_i in the summary graph or it represents the nearest supernode from which v_i can be reached. This is called the mapping of a node of the original graph to one or more supernodes of the summary graph.

The algorithm to summarize an original PT network graph is explained below that maintains all the rules and assumptions mentioned earlier.

Algorithm 1 SummarizeGraph

Input: PT network graph $G(V, E)$, Radius r, Threshold T_h
Output: Summary graph $G_S(V_S, E_S)$
1. $G \leftarrow$ AddWalkingEdgesInOriginalGraph(G, r_{walk})
2. $X \leftarrow$ FindSupernodes(G, r, T_h)
3. **for all** $s_i \in X$ **do**
4. **for all** $v_k \in s_i.v_i \cup s_i.N_i$ **do**
5. add s_i in $src_{sn}(v_k)$
6. add s_i in $dest_{sn}(v_k)$
7. **end for**
8. **end for**
9. $G_S \leftarrow$ ConstructSummaryGraph(G, X)
10. MapNonSupernodesToSupernodes(G, G_S)
11. SortNodeIdToSupernodeAndCost(G, G_S)
12. **return** G_S

Algorithm 1 shows the pseudocode for converting the original PTN graph to a summarized graph. Given a graph G, radius r and threshold T_h as input, it returns the summary graph G_S. First, we add walking edges in the original graph when the distance between two nodes is less than a predefined threshold, r_{walk}, $r_{walk} \geq r$ in Line 1. Then, using Function *FindSupernodes*(G, r, T_h) the supernodes are identified and included in a set. In Line 3-8, the mapping of the nodes included in the supernodes are updated. After that, the *4 rules* of edges are applied and a summary graph $G_S = (V_S, E_S)$ is constructed by the Function *ConstructSummarryGraph*($G(V, E), X$). The nodes that were not included in any supernode are now associated with a supernode in the Function *MapNonSupernodesToSupernodes*($G(V, E), G_S(V_S, E_S)$).

Algorithm 2 AddWalkingEdgesInOriginalGraph

Input: PTN graph $G(V, E)$, distance threshold r_{walk}
Output: Updated PTN graph $G(V, E)$
1. **for all** $v_i \in V$ **do**
2. **for all** $v_j \in V$ **do**
3. $d \leftarrow$ *distance*(v_i, v_j)
4. **if** $d < r_{walk}$ **then**
5. add walking edge between v_i, v_j in E
6. **end if**
7. **end for**
8. **end for**
9. **return** G

In Algorithm 2, the pseudocode of the Function *AddWalkingEdgesInOriginalGraph*(G, G_S) is shown. An edge is added between two nodes in the PTN graph if their distance is less than a threshold r_{walk}.

Algorithm 3 FindSupernodes

Input: Graph $G(V, E)$, Radius r, Threshold T_h
Output: A set of supernodes, X

1. $X \leftarrow \emptyset$
2. **for all** $v_i \in V$ **do**
3. $src_{sn}(v_i) \leftarrow \emptyset$
4. $dest_{sn}(v_i) \leftarrow \emptyset$
5. $N_i \leftarrow$ FindNearbyNodes(v_i, r)
6. $g \leftarrow$ NumberOfOutgoingEdgesOfCandidateSN(v_i, N_i)
7. **if** $g > T_h$ **then**
8. make a supernode $s_i =< v_i, N_i >$
9. add s_i to X
10. **end if**
11. **end for**
12. **for all** $s_i \in X$ **do**
13. **for all** $s_j \in X$ **and** $i \neq j$ **do**
14. **if** $\{s_j.v_j\} \cup s_j.N_j$ is a subset of $\{s_i.v_i\} \cup s_i.N_i$ **then**
15. remove s_j from X
16. **end if**
17. **end for**
18. **end for**
19. **return** X

The Function *FindSupernodes*(G, r, T_h) is elaborated in Algorithm 3. The loop in Line 2-11 finds the set of nodes that fulfill the conditions of becoming supernodes. To identify a supernode, first the Function *FindNearbyNodes*(v_i, r) finds the set of nearby nodes of v_i within r radius. The Function *NumberOfOutgoingEdgesOfCandidateSN*(v_i, N_i) calculates into g the original number of outgoing edges of the node cluster $\{v_i\} \cup N_i$. If g is greater than the threshold, a supernode is added in the set X. The loop in Line 12-18 refines the set X by removing those supernodes, the nodes of which are a subset of the nodes of another supernode. Finally, the function returns the set of supernodes X for the summary graph of the original graph G.

Algorithm 4 AddEdge

Input: Tail vertex v_i, Head vertex v_j, Edge-weight $cost$, Graph $G(V, E)$
Output: Graph $G(V, E)$

1. **if** there is no edge between v_i, v_j **then**
2. add edge between v_i, v_j with edge-weight $cost$
3. **else if** $cost < c(v_i, v_j)$ **then**
4. change edge-weight of e_{ij} into $cost$ in E
5. **end if**
6. **return** G

In Function *AddEdge*$(v_i, v_j, cost, freq, G)$ shown in Algorithm 4, an edge is added with given cost if an edge does not exist between the tail vertex and the head vertex. if an edge already

exists for a particular pair, then edge cost is only updated when new edge cost is less than current edge cost.

Algorithm 5 ConstructSummaryGraph

Input: Original graph $G(V, E)$, A set of supernodes X
Output: Summary graph $G_S(V_S, E_S)$

1. $V_S \leftarrow \emptyset$
2. $E_S \leftarrow \emptyset$
3. **for all** $s_i \in X$ **do**
4. add s_i to V_S
5. **end for**
6. $W \leftarrow \emptyset$
7. **for all** $s_i \in V_S$ **do**
8. $W \leftarrow$DijkstraModified($G(V, E), G_S(V_S, E_S), s_i, W$)
9. **end for**
10. **for all** $s_i \in V_S$ **do**
11. **for all** $s_j \in V_S$ **do**
12. **for all** $v_m \in s_i.N_i \cup \{s_i.v_i\}$ **do**
13. **for all** $v_n \in s_j.N_j \cup \{s_j.v_j\}$ **do**
14. **if** $v_m = v_n$ **then**
15. $cost \leftarrow c(v_i, v_m) + c(v_m, v_j)$// common node, *rule 1*
16. AddEdge$(v_m, v_n, cost, G_S)$
17. **end if**
18. **end for**
19. **end for**
20. **end for**
21. **end for**
22. **return** G_S

The Function *ConstructSummaryGraph*$(G(V, E), X)$ shown in Algorithm 5 constructs the summary graph given the original graph and the set of supernodes. The set V_S is populated in Line 3-5. In Line 8, the Function *DijkstraModified*(s_i) is called for each supernode of the summary graph. This function finds and saves path from supernode s_i to all other supernodes if a path exists. This function indirectly applies the *rule* 2-4 in the summary graph. The *rule* 1 of adding edges in summary graph is implemented in Line 10-21.

The Function *DijkstraModified*$(G(V, E), G_S(V_S, E_S), s_i, W)$ shown in Algorithm 6 finds and saves path from supernode s_i to all other supernodes if a path exists given the original graph, the summary graph, the source supernode and the current list of supernodes. This is the dijkstra algorithm but the only difference is that it tracks the paths between two supernodes. The variable $queue$ is a priority queue of 2-tuple:$< v, d >$ where $v \in V$ and d is the path-cost from the center of source supernode to node v. The variable $traversed$ is a list of nodes that contains the nodes that have been traversed inside this function. The Function *parent*(v) returns the parent node of the node v. The $queue$ is initialized in Line 4-6. The tuple with the minimum cost is removed from the $queue$ in Line 9. If the node u in this tuple is included in atleast one supernode, then in

Algorithm 6 DijkstraModified

Input: Original graph $G(V, E)$, Summary graph $G_S(V_{S,S})$, supernode s_i, A list of saved paths, W

Output: An updated list of saved paths, W

1. $traversed \leftarrow \emptyset$
2. $queue \leftarrow \emptyset$
3. $source \leftarrow s_i.v_i$
4. **for all** $v_i \in V$ **do**
5. put $< v_i, \infty >$ to $queue$
6. **end for**
7. put $< source, 0 >$ to $queue$
8. **while** $queue$ is not empty **do**
9. $< u, d_u >\leftarrow$remove the tuple with smallest cost from $queue$
10. add u to $traversed$
11. **if** $src_{sn}(u) \neq \emptyset$ **then**
12. **for** $s_k \in src_{sn}(u)$ **do**
13. **if** $s_k = source$ **then**
14. *continue*
15. **end if**
16. $nodes \leftarrow$ExistsSNBetweenTwoSNs($G(V, E), s_i.v_i, u$)
17. **if** $nodes \neq \emptyset$ **then**
18. $edgeCost \leftarrow 0$
19. **if** $u = s_k.v_K$ **then**
20. $edgeCost \leftarrow d$
21. **else**
22. $edgeCost \leftarrow$ GetEdgeCost($G, u, s_k.v_k$)
23. add $s_k.v{-}_k$ to $nodes$
24. **end if**
25. AddEdge($G_S, s_i, s_k, edgeCost$)
26. add $< s_i, s_k, nodes, edgeCost >$ to W
27. **end if**
28. **end for**
29. **end if**
30. **for** $v \in$ GetOutGoingEdgeNodes(u) **do**
31. **if** $v \notin traversed$ **then**
32. $< v, d_v >\leftarrow$ GetDist($queue, v$)
33. **if** $d_v > d_u + c(u, v)$ **then**
34. UpdateQueue($queue, v, d_u + c(u, v)$)
35. parent(v) $\leftarrow u$
36. **end if**
37. **end if**
38. **end for**
39. **end while**
40. **return** W

Line 12-28 the path from node $source$ to u is saved in the summary graph corresponding to the edge between s_i and s_k, $\forall s_k \in src_{sn}(u)$. The function *ExistsSNBetweenTwoSNs*$(G, s_i.v_i, u)$ in Line 16 finds that path if it exists. The function *GetEdgeCost*$(G, u, s_k.v_k)$ in Line 22 returns the edge cost between two nodes in a graph. In Line 30-38, the non-traversed node of outgoing edges are added to the $queue$ if going to that node via u results in less cost. Finally, this function returns the updated list of saved paths.

Algorithm 7 ExistsSNBetweenTwoSNs

Input: Original graph $G(V, E)$, Source node v_{src}, Destination node v_{dest}
Output: A sequence of nodes, $nodes$

1. $nodes \leftarrow \emptyset$
2. $child \leftarrow v_{dest}$
3. **while** $parent(child) \neq \emptyset$ **do**
4. add $child$ to $nodes$
5. $child \leftarrow parent(child)$
6. **end while**
7. add v_{src} to $nodes$
8. Reverse($nodes$)
9. **return** $nodes$

In Algorithm 7, the Function *ExistsSNBetweenTwoSNs*$(G(V, E), G_S(V_S, E_S), s_i, W)$ is elaborated. This function takes a graph, a source node and a destination node as input and returns the sequence of nodes in the path from source node to destination node calculated using dijkstra.

It should be noted that $src_{sn}(v_i)$ and $dest_{sn}(v_i)$ is not updated for those nodes v_i in the original graph G that are not converted into supernodes or included in any supernode while constructing the summary graph. Algorithm 8 updates them for those nodes. For each such two paths have to be saved, one path starts from this node and ends in its nearest supernode and another path starts from its nearest supernode and ends in this node. The first path is calculated in Line 2-16 and the second path is calculated in Line 17-31. The function *dijkstra*$(src, dest, G)$ returns the shortest path and the shortestt path cost from source node to destination node in a given graph. The supernode for which the shortest path cost is minimum is selected. L_{min} contains the shortest path among all the calculated path. Finally, $src_{sn}(v_i)$ is updated with that supernode. The value of $dest_{sn}(v_i)$ is updated in the same way.

The mapping of nodes that do not belong to any supernode is explained with an example. In Figure 4.2, the node v_1 does not belong to any supernode. Among the supernodes, s_{17} can be reached in the shortest time in the worst case. Thus, $src_{sn}(v_1) = s_{17}$.

Algorithm 8 MapNodesToSupernodes

Input: PTN graph $G(V, E)$, Summary graph $G_S(V_S, E_S)$
Output: Two Lists of saved paths Q_{src}, Q_{dest}

1. **for all** $v_i \in V$ **do**
2. **if** $src_{sn}(v_i) = \emptyset$ **then**
3. $d_{min} \leftarrow +\infty$
4. $L_{min} \leftarrow \emptyset$
5. $s_{min} \leftarrow \emptyset$
6. **for all** $s_k \in V_S$ **do**
7. $< nodes, d > \leftarrow$ dijkstra($v_i, s_k.v_k, G$)
8. **if** $d < d_{min}$ **then**
9. $d_{min} \leftarrow d$
10. $L_{min} \leftarrow nodes$
11. $s_{min} \leftarrow s_k$
12. **end if**
13. **end for**
14. add s_{min} to $src_{sn}(v_i)$
15. add $< v_i, s_{min}.v, L_{min}, d_{min} >$ to Q_{src}
16. **end if**
17. **if** $dest_{sn}(v_i) = \emptyset$ **then**
18. $d_{min} \leftarrow +\infty$
19. $L_{min} \leftarrow \emptyset$
20. $s_{min} \leftarrow \emptyset$
21. **for all** $s_k \in V_S$ **do**
22. $< nodes, d > \leftarrow$ dijkstra($s_k.v_k, v_i, G$)
23. **if** $d < d_{min}$ **then**
24. $d_{min} \leftarrow d$
25. $L_{min} \leftarrow nodes$
26. $s_{min} \leftarrow s_k$
27. **end if**
28. **end for**
29. add s_{min} to $dest_{sn}(v_i)$
30. add $< s_{min}.v, v_i, L_{min}, d_{min} >$ to Q_{dest}
31. **end if**
32. **end for**
33. **return** Q_{src}, Q_{dest}

The construction of the summary graph and the mapping of nodes of the original graph to supernodes of the summary graph explained in this section are pre-computed to save computational time.

4.2.2 Finding k Shortest Paths in the Summary Graph

For a particular package delivery request, k shortest paths are found in the summary graph. The objective of this step is to reduce search space for finding up to k shortest delivery paths with

respect to delivery time in the original graph explained later in Section 4.2.3.

First, the nearest node from the source location, $v_s \in V$ and the nearest node from the the destination location, $v_d \in V$ of the package delivery request is identified. Then, the source node v_s and the destination node v_d of a package delivery are mapped to two supernodes of the summary graph, respectively. If one node can be mapped to multiple supernodes, any one of them can be chosen. This does not affect the final result for finding heuristic k shortest paths in the original graph, because rather than using this k shortest paths in the summary graph directly, we use them to decide the search space. If one of the source and the destination nodes is mapped but the other is not mapped, then no path from source to destination is possible. If both of them are not mapped, then the search space cannot be refined for that package and existing k *shortest paths algorithm* is used in that case. If both of them are mapped, then a k*-shortest paths algorithm* is executed in the summary graph from source supernode to destination supernode. Since, the edge-cost in the summary graph represents the minimum travel-time between the centers of two supernodes, k_s shortest paths calculated here are independent of the delivery start time. The paths calculated in this step is used to refine the search space in the original graph.The value of k is user defined.

This step is executed in real time. Because the summary graph is significantly smaller than the original graph, this step takes considerably small computational time.

4.2.3 Refining the Search Space in the Original Graph

In this step, we refine the search space in the original graph. This refined search space reduces the computational cost of finding at most k shortest paths but the optimality of the k paths is sacrificed. The refined search space is different for every package delivery request.

For refining the search space, we use the k shortest paths found in section 4.2.2 which consist of edges from the summary graph. Each edge in the summary graph has a corresponding saved path. So, the nodes in the saved paths are included in the refined search space. *If the source node is a non-supernode then the nodes in the path from the source node to its nearest supernode is also added. If the destination node is a non-supernode then the nodes in the path from the nearest supernode to the destination node is added too.*

The Algorithm 9 shows the pseudocode for refining the search space. It returns the updated list of nodes to be searched in the PTN graph given the list of paths found in section 4.2.2, the source node and the destination node. Each node included in each of the k-paths are added in the refined search space. The nodes included in the mapped supernodes are also added in that space because the those nodes are within the walking distance from the source node or the destination node.

Algorithm 9 RefineSearchSpace

Input: A list of path L, source node v_{src}, destination node v_{dest}
Output: The list of nodes included in refined search space V_r

1. $V_r \leftarrow \emptyset$
2. **for all** $s_i \in src_{sn}(v_{src})$ **do**
3. **for all** $v_k \in s_i.v_i \cup s_i.N_i$ **do**
4. add v_k in V_r
5. **end for**
6. **end for**
7. **for all** $s_i \in dest_{sn}(v_{dest})$ **do**
8. **for all** $v_k \in s_i.v_i \cup s_i.N_i$ **do**
9. add v_k in V_r
10. **end for**
11. **end for**
12. add $dest_{sn}(v_{dest})$ to V_r
13. **for all** path $l \in L$ **do**
14. **for all** edge $e \in l.edges$ **do**
15. **for all** $v_k \in$ *saved-path*$(e).SP$ **do**
16. add v_k in V_r
17. **end for**
18. **end for**
19. **end for**
20. **return** V_r

4.3 Matching Crowd with Package Delivery Requests

In this section, our approach to match user participation requests with package delivery requests is explained. For each package delivery request, we incrementally find at most k shortest path within the refined search space and for each path we match it with user journey plans to find a delivery route. If a delivery route is found, no more paths are calculated.

4.3.1 Finding the ith Shortest Path within the Refined Search Space

The first step of the iteration is finding the ith shortest path in the refined space. This step is computationally feasible because the refined space is much smaller than the original PTN graph.

We find the i shortest path within the refined search space incrementally, starting from $i = 1$. This ith path may be the jth shortest path in the original search space, where $j \geq i$. We use the state-of-the-art k shortest paths algorithm in [34]. Since the algorithm in [34] finds looped paths too, we provide the parameter t to the algorithm. Then this algorithm will find at most k shortest loopless paths where $k \leq t$.

4.3.2 Matching the ith Shortest Path with Crowd

The last step of the iteration is matching the ith shortest path in the refined space with the user participation requests. In a PTN, we have to consider many user participation requests for a delivery path. For this reason, it is not feasible to find the best set of users possible for a package delivery request computationally. As a result, we use a greedy approach to match the package delivery request with given user participation requests within practical computational time.

To match the ith path $L_i = (j_1, \ldots, j_n)$ wihtin the refined search space with user journey plans, first the overlap between the path and the journey plans needs to be found. The journey edge j_k overlaps with the journey plan of a user participation request S_i if any of the following conditions is satisfied:

1. *If the journey edge j_k is not an walking-edge, then an journey edge j_l of a user participation request S_i overlaps j_k if $j_k.v_{dept} = j_l.v_{dept}$, $j_k.v_{arr} = j_l.v_{arr}$, $j_k.t_{dept} = j_l.t_{dept}$ and $j_k.t_{arr} = j_l.t_{arr}$.*
2. *If the journey edge j_k is an walking-edge, then an journey edge j_l of a user participation request S_i overlaps j_k if $j_l.t_{dept} \geq j_{k-1}.t_{arr}$ and $j_l.t_{arr} \leq j_{k+1}.t_{dept}$.*

We represent the overlap between a journey edge j_k and user participation requests as a 2-tuple: $< j_k, Z_k >$, where Z_k is a set of user participation requests that overlap with j_k. We can show by a simple calculation how much computational expensive finding the optimal matching for a package delivery request can be. Assume, the ith shortest path consists of 20 journey edges and each edge overlaps with 10 user participation requests. So, considering detour, the number of possible combination of user participation requests is 11^{20}.

After finding the overlaps, we choose an user for each journey edge of the delivery path from the overlapped users in a greedy way. The following rules are applied when choosing a user:

1. *The user who carries the package in the first journey edge of the delivery path have to pick it up from the source location. This user's detour limit must allow this detour.*
2. *The user who carries the package in the last journey edge of the delivery path have to drop it at the destination location. The detour limit of this user must allow this detour.*
3. *if a user u_{k-1} carries the package along journey edge j_{k-1} and another user u_k carries the package along journey edge j_k, then u_{k-1} must change transport at the end of edge j_{k-1}.*
4. *If a journey edge j_k in the delivery path does not have any user to carry the package, then the user who carries it in the previous journey edge j_{k-1} may detour and carry the package if any of the following rules hold:*

(a) *if the edge of the user journey plan that overlaps with the journey edge j_{k-1} is not the last journey edge of the user journey plan and j_k is a walking edge, then the user can detour if s/he has enough time to carry the package in edge j_k and then come back at the joining node of j_{k-1} and j_k to continue his journey.*

(b) *if the edge of the user journey plan that overlaps with the journey edge j_{k-1} was the last journey edge of the user journey plan, then the user can detour if the time needed to carry the package along journey edge j_k is within the availble detour limit of the user. If the journey edge j_k is the last edge of the delivery path, then the available detour limit must include the time needed to drop off the package from destination node to destination location.*

If a delivery route can be constructed that abides by these rules from the ith delivery path then we terminate the iteration. Otherwise we calculate the next shortest path within the refined space and try to construct a feasible delivery route.

Chapter 5

Experiment

In this chapter, we evaluate the performance of our approach to find package delivery routes for a set of delivery requests which minimizes each package delivery time. Since, the problem solved by existing literature do not exactly match with our problem formulation, we compare our approach with a straightforward approach where delivery paths are found incrementally using the state-of-the-art k *shortest paths algorithm* [34] and the greedy matching approach explained in Section 4.3.2. We call this approach baseline approach. We compare these two approaches extensively by varying many parameters.

We run experiments using both real world dataset and synthetic dataset. For the public transport network graph, we use PTV GTFS [39] dataset which contains different public transport networks, e.g. train, bus, tram, night-bus etc. In the experiments shown in this chapter, we use the train network which contains 219 nodes, 512 edges and 472636 timetables. The user participation requests and the package delivery requests are generated synthetically. The user participation requests are generated from a probability distribution which is extracted from the Global-scale Check-in [40] dataset. This dataset contains global check-in information for many users. From this dataset, we extract the check-ins made inside Victoria, Australia and use this data to generate practical user participation requests. From the dataset we find the top 100 places with the most check-ins and use their probability distribution to generate package delivery requests. Using the check-in dataset ensures the practicality of the synthetic dataset.

For running the experiments, we use an Intel Core i5 machine with 1.60 GHz CPU and 8GB RAM. We identify three performance matrices to compare the impact of different parameters: the runtime, the delivery ratio and the average delivery time per package. The runtime is the processing time needed to calculate the solution and the delivery ratio means the percentage of packages for which a delivery route was found. We run the experiments for n package delivery requests and m user participation requests where each user allows d_{dt} min detour. We vary these three parameters, the number of user participation requests, detour limit of users in minutes and the number of package delivery requests. We also vary the parameter t explained

in Section 4.3.1.

This chapter is organized as follows: Section 5.1 shows the results of varying the parameter t and explains the results. Section 5.2 shows the results of varying the parameter d_{dt} which is the detour limit of each user and discusses the results. Section 5.3 shows the experiment results where m, the number of user participation requests were varied and explains them and Section 5.4 shows the experiment results where n, the number of package delivery request were varied and explains them.

5.1 Effect of the Parameter t

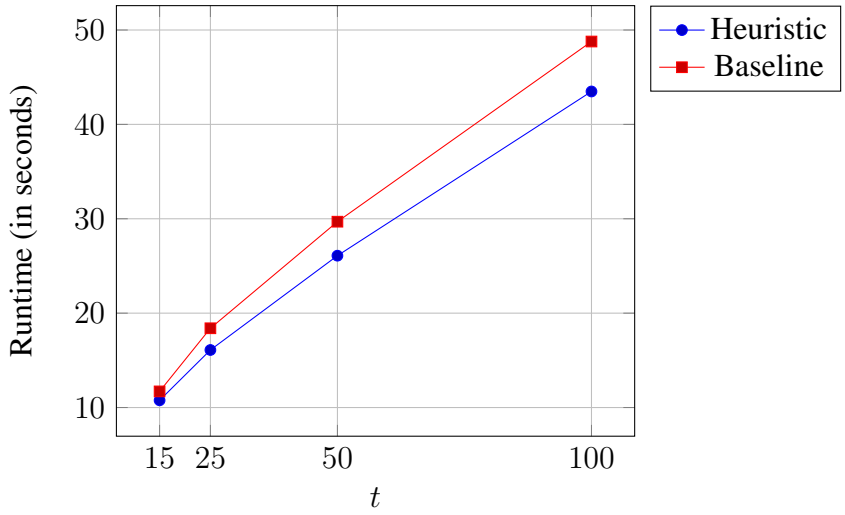


Figure 5.1: Comparison of runtime in seconds for our heuristic approach and the baseline approach for different parameter settings of t. Here, $m = 50000$, $n = 100$ and $d_{dt} = 30$.

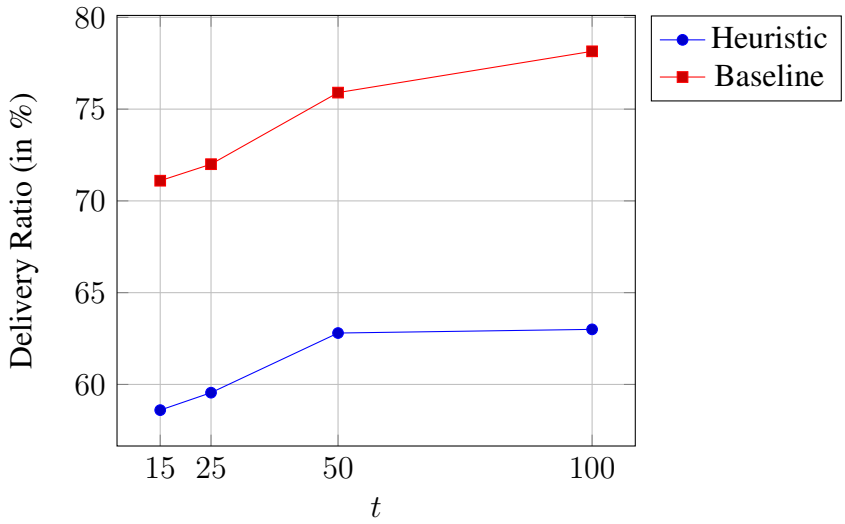


Figure 5.2: Comparison of delivery ratio in percentage for our heuristic approach and the baseline approach for different parameter settings of t. Here, $m = 50000$, $n = 100$ and $d_{dt} = 30$.

In this section, we observe the impact of the parameter t in the runtime and the delivery ratio. When t increases, the number of delivery paths calculated may increase.

In Figure 5.1, we see that the runtime of our approach is lower than the approach where optimal k *shortest paths algorithm* is used. Hence, our solution approach improves on performance. We notice that the runtime increases as the value of t increases. It is because more delivery paths are considered for each package and so computational time increases. In Figure 5.2, we observe that the delivery ratio is slightly lower than the original approach. This is the trade-off for reducing computational time. Here, the delivery ratio increases with the increase in the value of t. Since more delivery paths are considered, the chance of finding a delivery route for each package increases. Thus, more packages can be delivered.

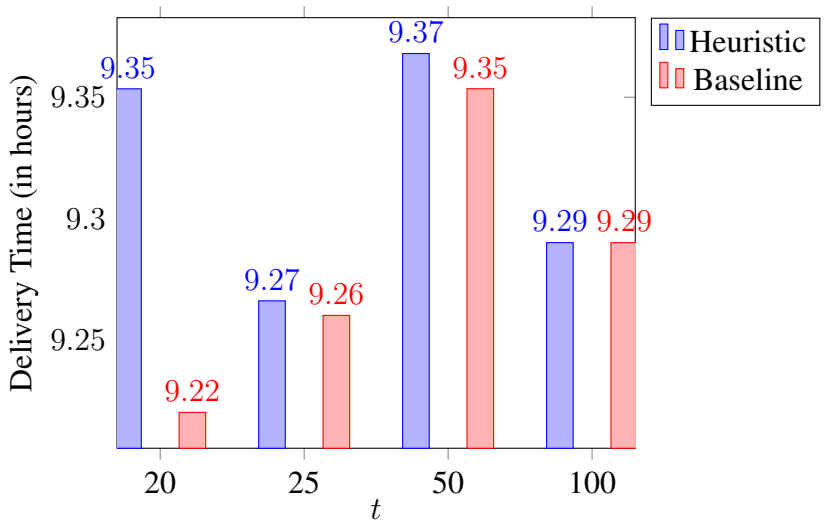


Figure 5.3: Comparison of delivery time in hours for our heuristic approach and the baseline approach for different parameter settings of t. Here, $m = 50000$, $n = 100$ and $d_{dt} = 30$.

In Figure 5.3, y-axis represents the average delivery time needed to deliver a package. We observe that the average time it takes to deliver a package is almost same in our approach and the baseline approach. We see that it takes approximately 9.5 hours to deliver a package, which is a feasible delivery time. Thus, our heuristic does not perform bad in terms of service quality.

5.2 Effect of Detour Limit

We study the impact of detour limit in this section. If the detour limit increases then we have more flexibility in choosing users for a delivery route and detouring a user. We choose 15, 30, 45, 60 as the values of d_{dt}. All usera allowing detour limit more than 60 minutes or 1 hour is not realistic.

We observe that in Figure 5.4, the runtime decreases with increase in detour limit and in Fig-

ure 5.5, the delivery ratio increases with increase in detour limit. These results follow basic intuition. If detour limits increases then the algorithm will have more flexibility in choosing a user to detour. As a result, more packages are delivered. Again, because there is more flexibility in choosing users, we find a delivery route sooner and the iteration terminates earlier. As a result, the runtime decreases.

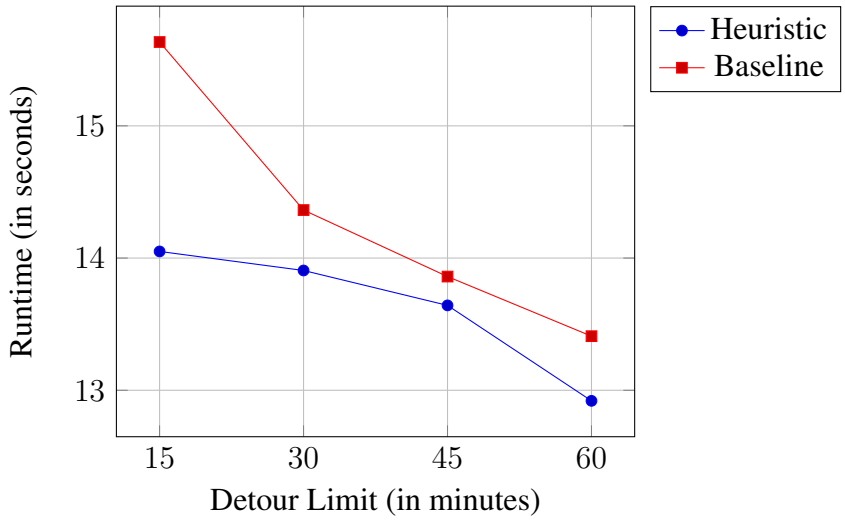


Figure 5.4: Comparison of runtime in seconds for our heuristic approach and the baseline approach for different parameter settings of detour limit, d_{dt} in minutes. Here, $m = 50000$, $n = 100$ and $t = 50$.

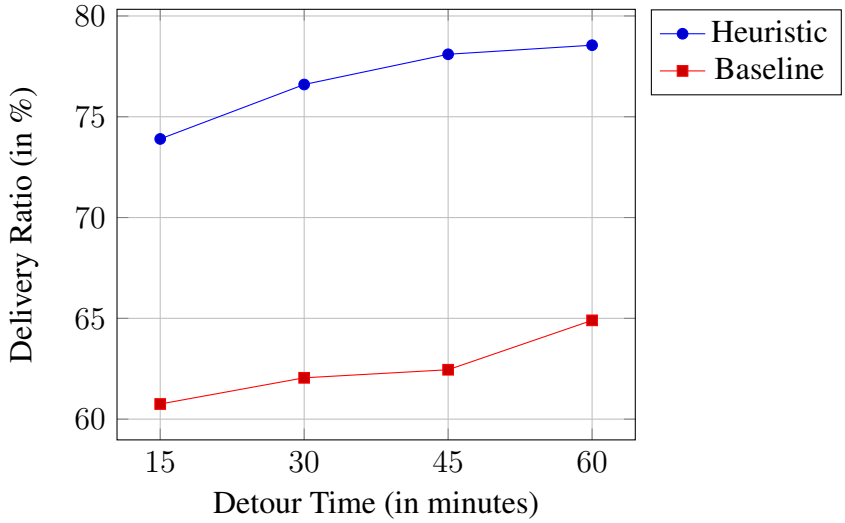


Figure 5.5: Comparison of delivery ratio in percentage for our heuristic approach and the baseline approach for different parameter settings of detour limit, d_{dt} in minutes. Here, $m = 50000$, $n = 100$ and $t = 50$.

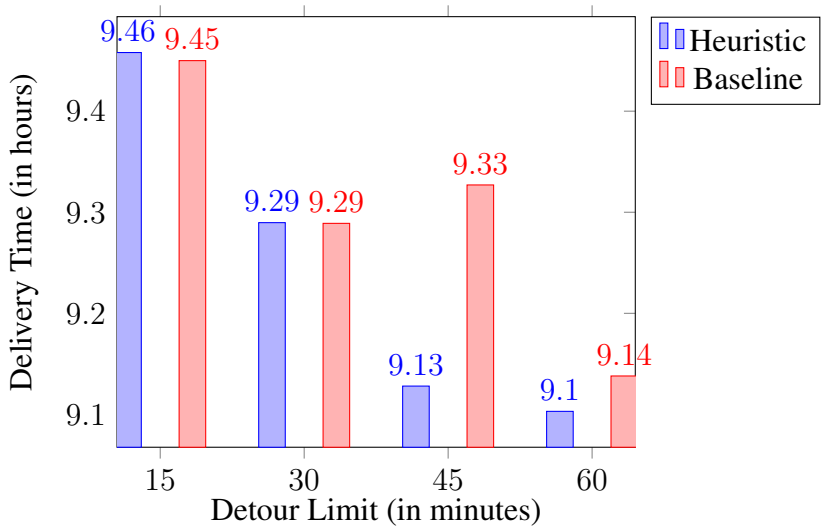


Figure 5.6: Comparison of delivery time in hours for our heuristic approach and the baseline approach for different parameter settings of detour limit, d_{dt} in minutes. Here, $m = 50000$, $n = 100$ and $t = 50$.

In Figure 5.6, we notice that average delivery time decreases with increase in detour limit. This is because relaxed detour limit allow us to find delivery routes for shorter paths.

5.3 Effect of the Number of User Participation Requests

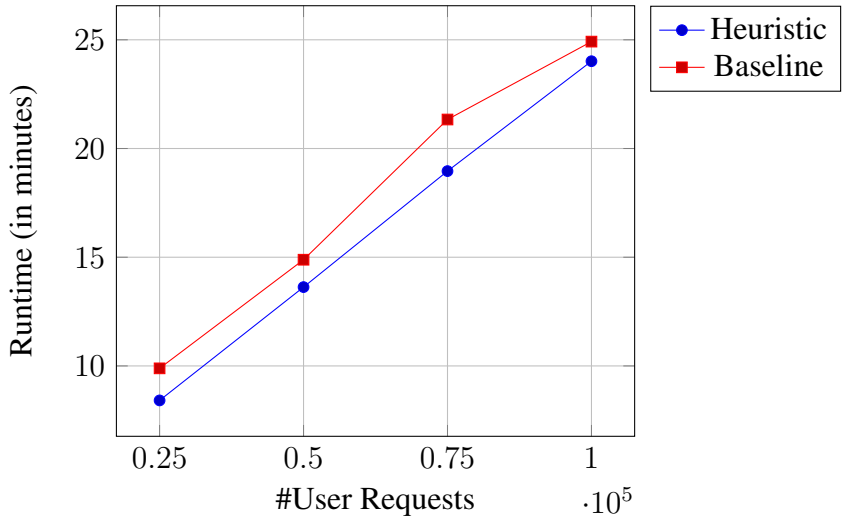


Figure 5.7: Comparison of runtime in seconds for our heuristic approach and the baseline approach for different parameter settings of the number of user participation requests. Here, $n = 100$, $t = 50$ and $d_{dt} = 30$.

We evaluate the impact of varying the number user participation requests (m) in this section. If m increases then there is more options in choosing users for a delivery route. We choose 25000,

50000, 75000, 100000 as the values of m. The choice of the values is realistic because daily a large number of people use the train network.

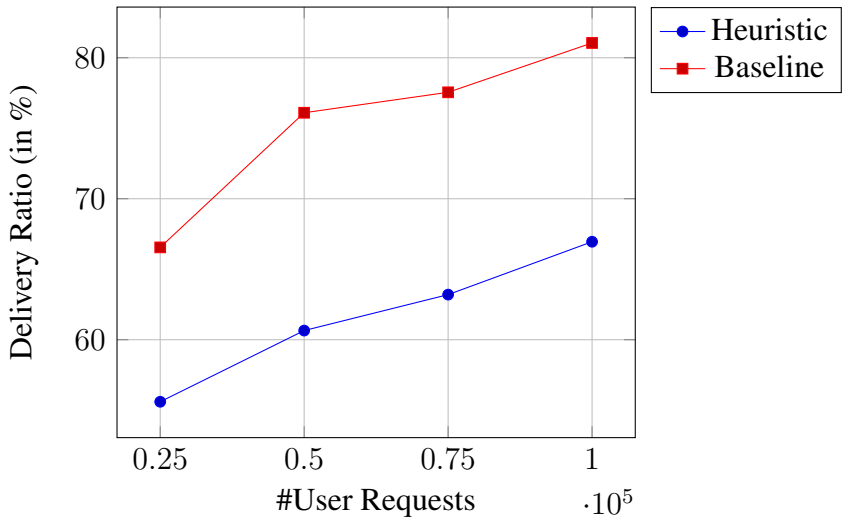


Figure 5.8: Comparison of delivery ratio in percentage for our heuristic approach and the baseline approach for different parameter settings of the number of user participation requests. Here, $n = 100$, $t = 50$ and $d_{dt} = 30$.

In fig. 5.8, we notice that the runtime increases with increase in the number of user participation requests. In Figure 5.5, we observe that the delivery ratio increases with the increase in the number of user participation requests. These results can be explained intuitively. If the number of user participation requests increases then there will be more user available to carry a package. As a result, more packages are delivered. In addition, because the algorithm have to match more users with a single delivery path, the runtime decreases.

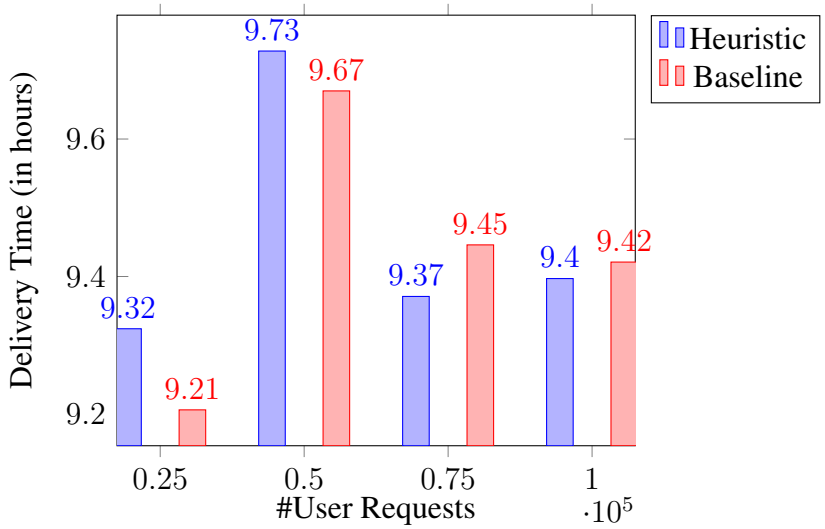


Figure 5.9: Comparison of delivery time in hours for our heuristic approach and the baseline approach for different parameter settings of user participation requests. Here, $n = 100$, $d_{dt} = 30$ and $t = 50$.

In Figure 5.9, we observe that the change in the number of user participation requests does not affect the average package delivery time much. We can also see that sometimes our approach results in slightly better average delivery time. This can happen because the baseline algorithm is not optimal.

5.4 Effect of the Number of Package Delivery Requests

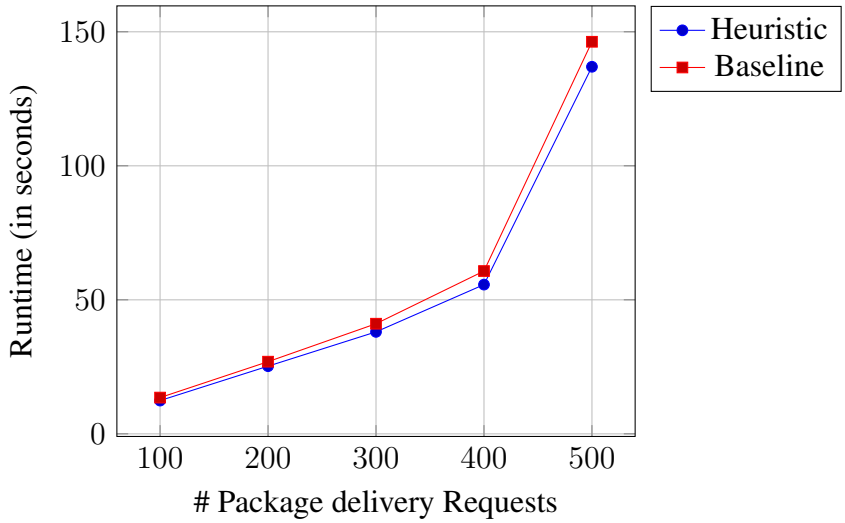


Figure 5.10: Comparison of runtime in seconds for our heuristic approach and the baseline approach for different parameter settings of the number of package delivery requests. Here, $m = 50000$, $t = 50$ and $d_{dt} = 30$.

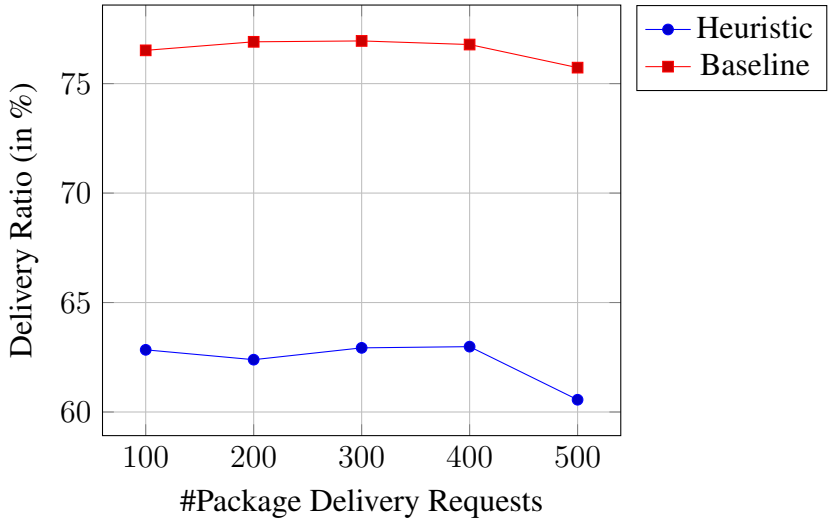


Figure 5.11: Comparison of delivery ratio in percentage for our heuristic approach and the baseline approach for different parameter settings of the number of package delivery requests. Here, $m = 50000$, $t = 50$ and $d_{dt} = 30$.

In this section, we study how varying the number user participation requests (n) impacts the performance metrics. If m increases then there is more options in choosing users for a delivery route. We choose 50, 100, 200 and 400 as the values of n.

In Figure 5.10, we notice that the runtime increases with the increase in the number of package delivery requests. This is expected because more packages need to be matched means more computation. In Figure 5.11, we observe that first delivery ratio does not change with increase in package delivery requests but then the delivery ratio starts dropping. From this graph, we can conclude that for a particular number of journey plans available, there is a limit in the number of packages that can be delivered without degrading performance. If the number of packages to deliver exceeds that limit then delivery ratio starts dropping.

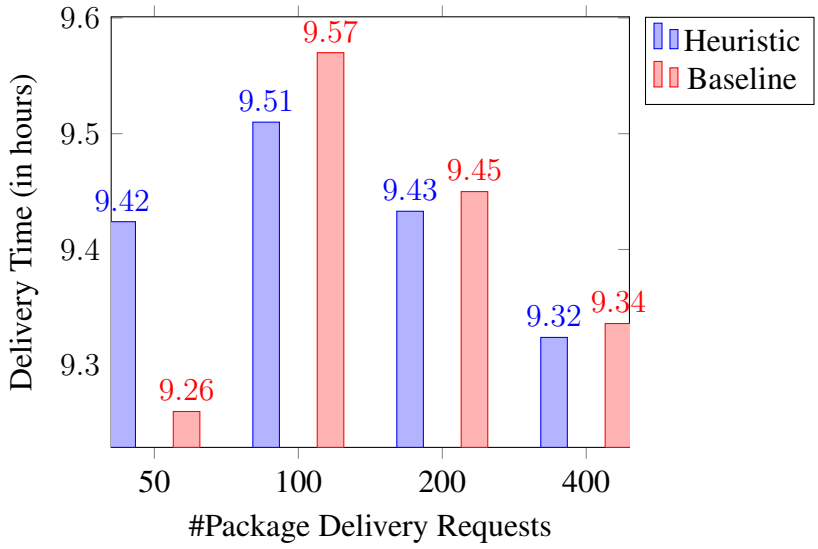


Figure 5.12: Comparison of delivery time in hours for our heuristic approach and the baseline approach for different parameter settings of the number of the package delivery requests. Here, $n = 100$, $d_{dt} = 30$ and $t = 50$.

In Figure 5.12, we observe that the change in the number of package delivery requests does not affect the average package delivery time much.

Chapter 6

Conclusion

We have introduced a novel problem, crowdsourced package delivery utilizing the public transport network in our thesis. We propose a heuristic approach to solve this problem computationally efficiently with detour constraint that reflect real life scenario. In our approach we minimize each package delivery time since fast delivery is important delivery services. We propose a novel graph summarizing method for the large PTN graph to efficiently find paths in that graph. In addition, we use a greedy matching approach to match the heuristically found paths with passengers journey plans to make the computational time of matching feasible.

We evaluated our approach extensively with real world dataset and synthetic dataset. The synthetic data was generated from real world probability distribution to ensure its practicality. Since our problem formulation is different from existing works, we compared our heuristic approach with the optimal approach. From the experiments, we observe that there is a trade-off between computational efficiency and the number of successful delivery. However, we interestingly notice that there is no trade-off in terms of package delivery time. The service quality of the delivery system does not reduce in the heuristic approach. From the experiments we can also conclude that if there is enough travellers willing to deliver a package and the number of packages needed to deliver is within reasonable limit, then crowdsourced package delivery via the public transport network is practical.

In the future, we aim to find delivery routes with more constraints, e.g. transport capacity that reflect practical scenario. In this thesis we have minimized the delivery time of each package. An even more practical approach will be to minimize aggregate package delivery time which reflect the service quality of the system or to minimize aggregate detour made by workers which reflects service cost of the system. Another working direction is to work on matching multiple users to single package in case of heavy packages.

References

[1] "Couriers and local delivery service providers' global market share in 2017." https://www.statista.com/statistics/236309/market-share-of-global-express-industry/. Accessed: 2018-03-26.

[2] "UPS fact sheet." https://www.pressroom.ups.com/pressroom/ContentDetailsViewer.page?ConceptType=FactSheets&id=1426321563187-193. Accessed: 2018-03-26.

[3] "amazonFLEX." https://flex.amazon.com/. Accessed: 2018-04-5.

[4] "deliv." https://www.deliv.co/about/. Accessed: 2018-04-5.

[5] "POSTMATES." https://about.postmates.com/. Accessed: 2018-04-5.

[6] "instacart." https://www.instacart.com/. Accessed: 2018-04-5.

[7] "deliveroo." https://deliveroo.co.uk/. Accessed: 2018-04-5.

[8] "UBEReats." https://www.ubereats.com/. Accessed: 2018-04-5.

[9] "DoorDash." https://www.doordash.com/about/. Accessed: 2018-04-5.

[10] "Dada." https://www.imdada.cn/. Accessed: 2018-04-5.

[11] J.-F. Rougès and B. Montreuil, "Crowdsourcing delivery: New interconnected business models to reinvent delivery," in *IPIC*, pp. 28–30, 2014.

[12] B. Coltin and M. M. Veloso, "Scheduling for transfers in pickup and delivery problems with very large neighborhood search," in *AAAI*, pp. 2250–2256, 2014.

[13] D. Zhang, H. Xiong, L. Wang, and G. Chen, "Crowdrecruiter: selecting participants for piggyback crowdsensing under probabilistic coverage constraint," in *UbiComp*, pp. 703–714, 2014.

[14] D. Zhang, L. Wang, H. Xiong, and B. Guo, "4w1h in mobile crowd sensing," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 42–48, 2014.

[15] S. He, D. Shin, J. Zhang, and J. Chen, "Toward optimal allocation of location dependent tasks in crowdsensing," in *INFOCOM*, pp. 745–753, 2014.

[16] L. R. Varshney, "Privacy and reliability in crowdsourcing service delivery," in *SRII Global Conference*, pp. 55–60, 2012.

[17] M. Xiao, J. Wu, L. Huang, Y. Wang, and C. Liu, "Multi-task assignment for crowdsensing in mobile social networks," in *INFOCOM*, pp. 2227–2235, 2015.

[18] C. Chen, S. Cheng, A. Gunawan, A. Misra, K. Dasgupta, and D. Chander, "TRACCS: A framework for trajectory-aware coordinated urban crowd-sourcing," in *HCOMP*, 2014.

[19] Y. Li, M. L. Yiu, and W. Xu, "Oriented online route recommendation for spatial crowdsourcing task workers," in *SSTD*, pp. 137–156, 2015.

[20] A. Sadilek, J. Krumm, and E. Horvitz, "Crowdphysics: Planned and opportunistic crowdsourcing for physical tasks," in *ICWSM*, pp. 536–545, 2013.

[21] C. Chen, D. Zhang, X. Ma, B. Guo, L. Wang, Y. Wang, and E. H. Sha, "Crowddeliver: Planning city-wide package delivery paths leveraging the crowd of taxis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 6, pp. 1478–1496, 2017.

[22] C. Zhang, Z. Du, M. D. Parmar, and Y. Bai, "Pocket-switch-network based services optimization in crowdsourced delivery systems," *Computers & Electrical Engineering*, vol. 62, pp. 53–63, 2017.

[23] J. McInerney, A. Rogers, and N. R. Jennings, "Learning periodic human behaviour models from sparse data for crowdsourcing aid delivery in developing countries," *CoRR*, vol. abs/1309.6846, 2013.

[24] D. S. Setzke, C. Pflügler, M. Schreieck, S. Fröhlich, M. Wiesche, and H. Krcmar, "Matching drivers and transportation requests in crowdsourced delivery systems," in *AMCIS*, 2017.

[25] A. Arslan, N. Agatz, L. Kroon, and R. Zuidwijk, "Crowdsourced delivery: A dynamic pickup and delivery problem with ad-hoc drivers," 2016.

[26] J. McInerney, A. Rogers, and N. R. Jennings, "Learning periodic human behaviour models from sparse data for crowdsourcing aid delivery in developing countries," in *UAI*, pp. 401–410, 2013.

[27] H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," in *Algorithm Engineering*, vol. 9220, pp. 19–80, 2016.

[28] F. Schulz, D. Wagner, and C. D. Zaroliagis, "Using multi-level graphs for timetable information in railway systems," in *ALENEX*, pp. 43–59, 2002.

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

[30] E. L. Lawler, "A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem," *Management Science*, vol. 18, no. 7, pp. 401–405, 1972.

[31] A. Perko, "Implementation of algorithms for *K* shortest loopless paths," *Networks*, vol. 16, no. 2, pp. 149–160, 1986.

[32] E. de Queirós Vieira Martins and M. M. B. Pascoal, "A new implementation of yen's ranking loopless paths algorithm," *4OR*, vol. 1, no. 2, pp. 121–133, 2003.

[33] D. Eppstein, "Finding the k shortest paths," *SIAM Journal on computing*, vol. 28, no. 2, pp. 652–673, 1998.

[34] E. d. Q. V. Martins and J. L. E. Dos Santos, "A new shortest paths ranking algorithm," *Investigção Operacional*, vol. 20, no. 1, pp. 47–62, 1999.

[35] V. M. Jiménez and A. Marzal, "Computing the K shortest paths: A new algorithm and an experimental comparison," in *WAE*, pp. 15–29, 1999.

[36] V. M. Jiménez and A. Marzal, "A lazy version of eppstein's *K* shortest paths algorithm," in *WEA*, pp. 179–190, 2003.

[37] G. Scano, M.-J. Huguet, and S. U. Ngueveu, "Adaptations of k-shortest path algorithms for transportation networks," in *IESM*, pp. 663–669, 2015.

[38] S. Wang, Y. Yang, X. Hu, J. Li, and B. Xu, "Solving the k-shortest paths problem in timetable-based public transportation systems," *Journal of Intelligent Transportation Systems*, vol. 20, no. 5, pp. 413–427, 2016.

[39] "PTV GTFS dataset." https://www.data.vic.gov.au/data/dataset/ptv-timetable-and-geographic-information-2015-gtfs. Accessed: 2018-10-19.

[40] "Global-scale Check-in Dataset." https://sites.google.com/site/yangdingqi/home/foursquare-dataset. Accessed: 2018-10-19.

Generated using Undegraduate Thesis LaTeX Template, Version 1.4. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on October 25, 2018 at 7:39am.