

UNIVERSITÀ DEGLI STUDI DI BRESCIA



**Università
di Brescia**

DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

CICLO
XXXVIII

Last Mile Delivery in Routing Problems: New Paradigms, Models, and Algorithms

Dottorando:
Filippo RANZA

Relatore:
Prof.ssa Renata MANSINI

Settore scientifico disciplinare:
MATH/06 - Ricerca Operativa

Abstract

Last-Mile Delivery denotes the last leg of the delivery process, where goods are transferred from a local depot to the final recipients. Its inefficiencies, driven by the need for a large number of relatively small vehicles and high labor intensity, make it the most expensive part of the overall delivery process. Since its introduction, the Vehicle Routing Problem, modeling the routing of a fleet of vehicles starting from a depot, visiting the customers exactly once, and returning to the depot, has played a pivotal role in Operations Research in general and Last-Mile Delivery in particular (Dantzig and Ramser, 1959). Over the decades, researchers and practitioners alike have been looking for innovative extensions and improvements to the original problem. Such improvements include new models and algorithms to solve existing problems more efficiently, and novel delivery paradigms to extend the delivery mechanisms beyond traditional truck-based operations.

In this thesis, we explore two new extensions to Last-Mile Delivery. The first part of this thesis is devoted to Public Transport Crowdsipping, a delivery paradigm where public transport users take part in the delivery process. In such a system, automated parcel lockers are installed in public transport stations. Some of these lockers, likely the ones outside the city, are loaded by the logistics service provider with parcels that have to be delivered to lockers inside the city. Commuters, called crowdshippers, transfer parcels, during their regular journey, from locker to locker until each parcel's destination locker is reached. Enabling this service requires interconnected planning decisions. The first one is the strategic selection of the right stations to install lockers and the identification of their number in each station. To this end, we propose a mixed integer programming model that jointly maximizes customer satisfaction and the number of stations with installed lockers, under budget and commuter-flow constraints. Extensive computational analysis highlights that the two objectives are not in sharp contrast. The second key operational decision addresses how parcels are routed through the network. This involves two tightly coupled choices: assigning each parcel to the initial locker where the logistics provider deposits it, and selecting and synchronizing the crowdshippers who will carry the parcel across successive transfers. To achieve this, we introduce a mixed integer programming model, strengthened with valid inequalities, to find the optimal scheduling. This model can find optimal solutions for small-sized instances, with solution times ranging from minutes to almost an hour. However, it struggles with larger scale instances, reaching a point where for the solver it is impossible to identify an integer solution within the time limit. Therefore, to solve real-world sized instances, we introduce a custom Adaptive Large Neighborhood Search metaheuristic. Such an algorithm finds high-quality solutions to large scale instances, always outperforming the model. Moreover, in small scale instances its solutions are of comparable or equal quality to the ones found by a state-of-the-art MILP solver. Detailed computational experiments show the positive effects of the valid inequalities and that the proposed algorithm outperforms the model in medium and large sized instances. A thorough sensitivity analysis identifies the relationship between the model's parameters and the effectiveness of the proposed delivery method. The comparison between a traditional delivery plan and a crowdsipping one shows that crowdsipping can help to reduce the overall environmental impact, given that a large enough number of parcels is delivered by crowdshippers. Finally, we propose a non-compact set covering formulation, solved using a Branch-and-Price algorithm, for parcel scheduling and locker selection. This latter method achieves a significant speed up with respect to both the compact MILP formulation and the ALNS algorithm, solving exactly instances of large

size in a short amount of time, finding optimal solutions within minutes on very large scale instances.

The second part of this thesis focuses on mitigating the social externalities associated with last-mile delivery. Beyond air pollution, last-mile traffic contributes to noise exposure and increases the risk of accidents due to heavy vehicle flows in urban areas. Such issues impact the population disproportionately, with people living near major roadways receiving a larger share of the disturbance. Therefore, we propose two extensions of the vehicle routing problem, both aiming at steering heavy vehicular traffic from populated areas. The first extension minimizes the total delivery time and the overall penalty costs for visiting city districts during peak hours. To solve this problem, we propose a bi-objective mixed integer linear program. To represent the concept of peak hours, we divide the day in multiple time windows, each with a different associated penalty for each district. Such penalties are higher the more a time window is associated with a peak hour in the relative district. Moreover, an optional constraint can be used to allow the visit of a district only within one time window. To assess this model and its impact on a real-world case, we conduct a comprehensive computational study based on synthetic instances generated over San Francisco (USA) map. Test results show a mild increase in routing cost as penalties are minimized. However, forcing to visit a district only in one time window has a major impact on both the routing and penalty costs. Final insights highlight the middle ground between the delivery time minimization and the off-peak hour policy. The second extension actively steers traffic away from densely populated areas by jointly minimizing the total delivery costs and the disturbance over k most impacted streets. We define the impact over a street as the number of inhabitants multiplied by the number of times that street is traversed by a vehicle in the solution. To this end, we propose a compact bi-objective mixed integer linear model, formulated on a multigraph, minimizing the k largest values of a set of variables. The formulation, taken from the literature, is compared with three other well-known formulations in order to assess its efficacy in a VRP context. Moreover, to solve large-scale instances, we propose a novel Kernel Search metaheuristic with a new way to construct the initial kernel and a new way to sort variables. Such Kernel Search iterates over the bucket set multiple times, each time adapting its metaparameters based on the behavior in the previous iteration. Experimental analysis, run on synthetic instances built on top of the well-known Solomon instances, shows that it can outperform a state-of-the-art solver in both solution quality and computational time. To evaluate the effect of the proposed model in a real-world scenario, we constructed a large set of synthetic instances based on the cities of Charleston and San Francisco (USA). Using census data and street information from an online repository, we validate the proposed model's ability to re-route heavy vehicular traffic away from densely populated areas. Results show that it is possible to identify a reasonable middle ground between routing cost minimization and disturbance avoidance. In fact, we can observe that a small percentage increase in the routing cost allows for a significant decrease in population disturbance, whereas larger increases cause a proportionally less significant reduction. Moreover, we observe that such results are highly dependent on the city's characteristics, where densely populated cities with tight street network can benefit more of such a model with respect to sparsely populated areas with few alternative routes between two points.

Sommario

La *Logistica dell'Ultimo Miglio* identifica l'ultimo segmento della logistica, in cui i beni sono trasferiti da un magazzino locale fino al destinatario. Tale segmento è anche il più oneroso dell'intero processo distributivo: le sue inefficienze, dovute alla necessità di impiegare un elevato numero di veicoli relativamente piccoli e all'elevata intensità di manodopera, lo rendono la parte più costosa della catena logistica. Sin dalla sua concezione, il *Vehicle Routing Problem*, la formalizzazione del problema di instradare una flotta di veicoli che partono da un deposito, visitano i clienti una e una sola volta, e tornano al deposito, ha avuto un ruolo fondamentale nella ricerca operativa (Dantzig and Ramser, 1959). Negli ultimi decenni, la logistica dell'ultimo miglio è stata ampiamente studiata per ridurre gli elevati costi e inefficienze, dovuti all'elevato numero di veicoli usati in rapporto alla quantità di merci trasportate e alla conseguente manodopera. Tra i miglioramenti si possono citare sia estensioni modellistiche e algoritmiche, volte a risolvere più efficientemente problemi già noti, sia nuovi paradigmi di consegna che estendono il meccanismo distributivo oltre la tradizionale operatività basata su camion, come l'introduzione di robot a guida autonoma o droni nel sistema di consegne, oppure cambiamenti metodologici volti a trovare soluzioni di maggiore qualità in minor tempo a problemi già noti. In questa tesi si esplorano due estensioni alla logistica dell'ultimo miglio. Nella prima parte viene trattato il Crowdsipping su trasporto pubblico, ossia un innovativo sistema di consegne in cui i pendolari prendono parte al processo di spedizione. Per applicare tale sistema, nelle stazioni della rete urbana vengono installati *locker* per il deposito e il ritiro di pacchi. Alcuni di questi, idealmente quelli in aree periferiche, vengono selezionati dal fornitore di servizio logistico, e all'inizio di ogni servizio di spedizione sono caricati con pacchi provenienti dal magazzino. Dunque, i pendolari, chiamati *crowdshippers*, durante il loro percorso ordinario, trasportano i pacchi da un locker al successivo finché la destinazione di ogni pacco è raggiunta. La messa in pratica di tale sistema richiede diverse decisioni interconnesse. In primo luogo, è necessario determinare in quali stazioni installare i locker, e in queste quanti locker installare. Per questo fine, nella tesi viene proposto un modello di programmazione lineare misto intera compatto che massimizza congiuntamente la soddisfazione dei clienti e il numero di stazioni con locker, rispettando vincoli di budget e il flusso di crowdshipper tra ogni coppia di stazioni. Con l'ausilio di prove computazionali effettuate su un grande numero di istanze sintetiche, è stato possibile determinare che i due obiettivi non sono in forte contrasto. La seconda decisione necessaria per l'implementazione di un sistema di crowdsipping riguarda l'instradamento dei pacchi nella rete. Tale pianificazione è composta da due sotto decisioni: come prima cosa è necessario determinare, per ogni pacco, quale sia la stazione di partenza ideale, quindi a tale stazione va individuato il percorso che il pacco seguirà nella rete. Queste scelte vanno fatte tenendo conto della disponibilità di crowdshipper, inclusa la sincronizzazione nel caso in cui un pacco venga spedito da più di una persona, e della capacità dei locker nelle varie stazioni. Questo problema viene formalizzato in un modello di programmazione lineare misto intera compatto, rinforzato con disuguaglianze valide, che minimizza i costi di spedizione totali. Il modello è in grado di trovare soluzioni ottime per istanze di piccola dimensione in tempi che variano da pochi minuti fino a circa un'ora, ma incontra difficoltà su istanze più grandi, fino a casi in cui il risolutore non riesce ad identificare una soluzione intera entro il limite di tempo. Per trattare istanze di grandi dimensioni, utili in applicazioni pratiche, successivamente viene proposta una metaeuristica Adaptive Large Neighborhood Search. Tale algoritmo è in grado di produrre soluzioni di alta qualità su istanze di grande scala, superando sistematicamente le

prestazioni del modello compatto; inoltre, su istanze di piccola scala, la qualità delle soluzioni è comparabile o uguale a quella ottenuta da un risolutore MILP di riferimento. I risultati delle analisi computazionali mostrano l'impatto positivo delle disuguaglianze valide e confermano che l'algoritmo proposto supera il modello su istanze di media e grande dimensione. Una successiva analisi di sensitività mostra le relazioni tra i parametri del modello e l'efficacia del metodo, evidenziando che il sistema può aiutare a ridurre l'impatto ambientale della logistica dell'ultimo miglio, a patto che un numero sufficientemente elevato di pacchi venga trasferito da crowdshipper; a tal fine, un confronto tra un piano di consegna tradizionale ed uno basato su crowdshipping evidenzia una potenziale riduzione dell'impatto complessivo quando tale condizione è soddisfatta. Alla fine di questa prima parte, viene proposto un modello non-compatto alternativo a quello precedente, basato su una formulazione di tipo set covering, risolto tramite *Branch-and-Price* per la schedulazione dei pacchi e la selezione dei locker. Tale approccio mostra un notevole incremento delle prestazioni rispetto sia al modello compatto, sia alla metaeuristica, ottenendo un significativo miglioramento dei tempi computazionali e risolvendo esattamente istanze di grande dimensione in tempi ridotti, fino a trovare soluzioni ottime in pochi minuti anche su istanze di scala molto elevata. Nella seconda parte della tesi vengono discusse estensioni del tradizionale vehicle routing problem mirate alla minimizzazione delle esternalità imposte alla popolazione. Infatti, il traffico pesante causato dai servizi di consegna non è solo fonte di inquinamento atmosferico, ma anche di rumore ed elevato rischio di incidenti. Si nota inoltre che queste problematiche affliggono la popolazione in maniera sproporzionata, dove i residenti in aree vicine a grandi arterie stradali sono coloro che subiscono maggiormente gli effetti. Pertanto, in questa tesi vengono proposte due estensioni, entrambe mirate a deviare il traffico pesante da aree densamente popolate. La prima punta a disincentivare l'ingresso in certe aree della città durante le ore di picco. A tale fine viene proposta una formulazione bi-obiettivo che minimizza congiuntamente il costo di trasporto e le penalità per l'ingresso in specifici quartieri durante determinate fasce orarie. Per rappresentare il concetto di ore di picco, la giornata viene suddivisa in più finestre temporali, ciascuna associata ad una penalità diversa per ogni distretto; tali penalità risultano tanto più elevate quanto più la relativa finestra temporale è coerente con una fascia di picco per quel distretto. Un vincolo opzionale consente di transitare in ogni quartiere durante una sola fascia oraria, limitando così la flessibilità dell'operazione. L'efficacia di questo modello viene validata su istanze sintetiche costruite usando la rete stradale e la distribuzione di popolazione di San Francisco (USA). I risultati mostrano un incremento moderato dei costi di instradamento al diminuire delle penalità; tuttavia, l'introduzione del vincolo opzionale ha un impatto rilevante sia sui costi di trasporto sia su quelli di penalità. Nel complesso, l'analisi evidenzia l'esistenza di un punto intermedio tra la minimizzazione dei tempi e costi di consegna e l'adozione di una politica di consegna orientata alle fasce orarie non di punta. La seconda estensione punta attivamente a deviare il traffico dalle vie più densamente abitate, minimizzando congiuntamente il costo di trasporto e il disturbo sulle k strade maggiormente interessate. L'impatto su una strada è definito come il numero di abitanti lungo strada moltiplicato per il numero di volte che questa strada è attraversata nella soluzione. A tal fine, viene proposto un modello bi-obiettivo compatto, formulato su un multigrafo, che minimizza i costi di spedizione e, congiuntamente, i k valori peggiori di una collezione di variabili che rappresentano l'impatto sulle strade. La formulazione impiegata per minimizzare i k casi peggiori, tratta dalla letteratura, viene confrontata con altre tre formulazioni note al fine di valutarne l'efficacia in un contesto di vehicle routing problem. Per risolvere istanze di grandi dimensioni, viene proposta una innovativa metaeuristica Kernel Search, la quale utilizza nuovi metodi per costruire il kernel di partenza ed ordinare le variabili fuori dal kernel. Le variabili

fuori il kernel vengono iterate più volte e, ad ogni iterazione, i metaparametri dell'euristica vengono adattati in base al comportamento osservato nell'iterazione precedente. L'analisi sperimentale, condotta su istanze sintetiche costruite a partire dalle note istanze di Solomon, evidenzia che l'approccio proposto può superare un risolutore di riferimento sia in qualità di soluzione sia in tempo computazionale. L'impatto di tale modello è inoltre validato su istanze sintetiche costruite sulla base delle mappe e popolazione di Charleston e San Francisco (USA), combinando dati censuari e informazioni stradali ottenute da un repository online. I risultati confermano la capacità del modello di reinstradare il traffico pesante lontano dalle aree densamente popolate e mostrano che è possibile individuare un compromesso ragionevole tra minimizzazione dei costi di consegna ed riduzione del disturbo: piccoli incrementi percentuali nel costo di instradamento possono produrre riduzioni significative dell'impatto sulla popolazione, mentre incrementi maggiori generano benefici progressivamente meno marcati. Infine, si osserva che tali effetti dipendono in misura rilevante dalle caratteristiche della città: contesti molto densi e con una rete stradale fitta offrono maggiori possibilità di deviazione e possono beneficiare maggiormente del modello, mentre aree più scarsamente popolate e con poche alternative di percorso presentano un margine di miglioramento più limitato.

Contents

Abstract	i
Sommario	iii
1 Introduction	1
I Crowdsipping	4
2 Locker Selection Problems for Public Transport-based Crowdsipping	5
2.1 Introduction	6
2.2 Problem Description	8
2.3 Mathematical Formulation	8
2.3.1 Mixed Integer Programming Model	9
2.4 Solution Methodologies for Multi Objective Programs	11
2.4.1 The weighted sum method	11
2.4.2 The ϵ -constraint method	13
2.5 Experimental Analysis	14
2.5.1 Benchmark Instances	15
2.5.2 Sensitivity Analysis & Managerial Insights	15
2.6 Cycles in CCNDP-RC solutions	20
2.7 Conclusions	22
3 Public Transport-based Crowdsipping	23
3.1 Introduction	24
3.2 Literature Review	27
3.3 Problem Definition and Notation	32
3.4 Mathematical Formulation	36
3.4.1 Valid Inequalities	39
3.5 An ALNS Algorithm for the PTCP	40
3.5.1 Introduction to ALNS	40
3.5.2 The proposed ALNS Algorithm	41
3.6 A Branch-and-Cut Algorithm for PTCP	47
3.6.1 Introduction to the Branch-and-Cut Algorithm	47
3.6.2 Separation Procedure	48
3.7 Experimental Results	49
3.7.1 Instance Generation and Experimental Setup	49

3.7.2	MILP Formulation Results: Base CPLEX versus Implemented Branch-and-Cut	50
3.7.3	ALNS Results	55
3.7.4	Tuning of ALNS Parameters	58
3.7.5	Sensitivity Analysis of System Efficiency	63
3.7.6	Sensitivity to Travel Delays and Path Length Variability	68
3.8	Environmental Impact Comparison: Crowdsourcing versus Traditional Delivery Systems	72
3.9	Conclusions	74
4	A Column Generation Algorithm for PTCP	75
4.1	Problem Definition	76
4.1.1	Set Covering Formulation for PTCP	78
4.2	Introduction to Column Generation and Branch-and-Price	79
4.2.1	Why Branch-and-Price?	79
4.2.2	Column Generation	80
4.2.3	Branch-and-Price	81
4.3	Branch-and-Price Algorithm	82
4.3.1	Derivation of the Pricing Problem	82
4.3.2	Pricing Problem	84
4.3.3	Feasibility Cuts	88
4.3.4	Branching Rules	88
4.4	Experimental Results	90
4.4.1	Instance Generation	90
4.4.2	Analysis of Total Solution Time	90
4.4.3	Impact of the Percentage of Backup Parcels	94
4.4.4	Analysis of the Branch-and-Bound Tree	95
4.4.5	Analysis on the Number of Generated Columns	96
4.4.6	Comparison of the Solution Methods for PTCP	98
4.5	Conclusions	100

II Equitable Vehicle Routing Problems 101

5	An Off-Peak Hours Delivery Policy to Mitigate Last-Mile Delivery Social Impact	102
5.1	Introduction	103
5.2	Problem Description	104
5.3	Mathematical Formulation	105
5.4	Instance Generation	107
5.5	Experimental Analysis	108
5.5.1	Benchmark Instances	108
5.5.2	Managerial Insights and Sensitivity Analysis	109
5.6	Conclusions	112

6	On the Equitable Vehicle Routing Problem	113
6.1	Introduction	114
6.2	Literature Review	116
6.2.1	Fair and Equitable Vehicle Routing Problems	116
6.2.2	On the k -sum Optimization Problems	119
6.2.3	Kernel Search Algorithms	119
6.3	Problem Definition	121
6.4	Minimization of the k Largest Variables in a Linear Program	122
6.5	Mathematical Formulation	124
6.6	Solution Methodology	126
6.7	Real-World Instances	126
6.8	Sensitivity Analysis	129
6.8.1	Sensitivity analysis on main parameters	129
6.8.2	Managerial insights	140
6.8.3	Statistical analysis of the results	142
6.9	A Kernel Search for k -EqVRP	144
6.9.1	An Introduction to Kernel Search	144
6.9.2	The Kernel Search Matheuristic	145
6.9.3	Initial VRP solution	146
6.9.4	Variable Ranking	149
6.9.5	Kernel Search Algorithm	149
6.10	Computational Results	153
6.10.1	Instances generation	154
6.10.2	Comparison of the k -sum formulations	155
6.10.3	TS-ALNS	157
6.10.4	Gurobi performance on the k -sum formulation	160
6.10.5	SAKS performance	161
6.11	Conclusions	165
7	Conclusions	166

List of Algorithms

2.1	Pareto front with Weighted sum method	12
3.1	Adaptive Large Neighborhood Search	40
3.2	ALNS within ISA Framework	42
3.3	Parcel path finding	45
3.4	Branch-and-Cut Algorithm	48
3.5	Search for violated (3.25)–(3.26)	49
4.1	Column Generation Algorithm	81
4.2	Generic Branch-and-Price Algorithm	82
4.3	Pricer	84
4.4	Compute Path	86
4.5	Build Path	87
6.1	Kernel Search Matheuristic	145
6.2	TS–ALNS for VRP	147
6.3	$RankScore(sol, \xi)$	150
6.4	Self-Adjusting Kernel Search (SAKS)	151

List of Figures

2.1	Evolution of f_1^ϵ for each value of α	16
2.2	Percentage of pick up points for each α and ϵ	17
2.3	Percentages of stations hosting lockers and stations chosen as drop-off points for each value of α and ϵ	18
2.4	Example of a Pareto Frontier	19
2.5	Value of the weighted sum $z^* = \lambda f_1 + (1 - \lambda)f_2$ for the frontier in Figure 2.4	20
2.6	Example of a solution without Constraints (2.9).	21
3.1	Example of a real network and its associated transfer graph.	33
3.2	Union of crowdshipper delivery graphs G_1 , G_2 , and G_3 into final graph G . . .	34
3.3	Example of a parcel transfer with crowdshippers' flow indicated by a black arrow and their id.	35
3.4	Visual comparison of the solutions obtained with (a) ALNS and (b) CPLEX for the same instance.	58
3.5	Comparison of Time-to-Best metrics: (a) by cooling rate, (b) by time limit. . .	61
3.6	Comparison of Time to Best statistics: (a) by Histogram, (b) by Empirical Cumulative Density	62
3.7	Comparison of Critical Distance: (a) by Cooling Rate, (b) by Time Limit. . .	63
3.8	Distribution of parcel waiting times at exchange stations for paths involving multiple crowdshippers.	69
3.9	Percentage of parcels missing planned transfers at exchange stations as a function of the incurred travel delay.	70
3.10	Number of parcels delivered in presence of travel delays for instances involving $ P = 30$ parcels, $ K = 30$ available crowdshippers, $ N = 44$ stations and $ L = 7$ public transportation lines.	71
4.1	Distribution of the ratios between the number of generated columns and number of pricing problems	97
5.1	Example of an instance with 15 nodes.	109
5.2	Worsening of f_1 with constraints (5.11).	109
5.3	Worsening of f_2 with constraints (5.11).	110
5.4	Example of a solution when optimizing (a) f_1 without constraints (5.11), (b) f_1 , and (c) f_2	111
5.5	worsening of f_1 for different degradation of f_2	112
6.1	Map of San Francisco	127

6.2	Map of Charleston	128
6.3	Distribution of optimal objective function value of $f_1 (f_2^*, 0)$, when changing the value of k , considering Charleston and San Francisco.	129
6.4	Example of a Multigraph	130
6.5	Distribution of the optimal objective function value of $f_2 (f_1^*, 0)$ when changing the value of k , considering Charleston and San Francisco.	131
6.6	Distribution of equity value improvement, in the two cities, when comparing $f_2 (f_1^*, \epsilon)$ with $f_2 (f_1^*, 0)$	132
6.7	Distribution of traveling time worsening, in the two cities, when comparing the value of f_1 computed on the optimal solution of $f_2 (f_1^*, \epsilon)$ with $f_1 (f_2^*, \infty)$	133
6.8	Solution in Charleston with $\epsilon = 0$	134
6.9	Solution in Charleston with $\epsilon = 1$	135
6.10	Solution in San Francisco with $\epsilon = 0$	136
6.11	Solution in San Francisco with $\epsilon = 1$	137
6.12	Distribution of the value of f_1 in the solutions of $f_2 (f_1^*, \epsilon)$ with respect to t_{max} , in the two cities	138
6.13	Distribution of the value of f_2 in the solutions of $f_2 (f_1^*, \epsilon)$ with respect to t_{max} , in the two cities	138
6.14	Distribution of the value of f_2 in the solutions of $f_2 (f_1^*, \epsilon)$ with respect to t_{max}	139
6.15	Distribution of the routing time by the average distance between nodes	139
6.16	Distribution of the equity value by the average distance between nodes	140
6.17	Pareto frontier for each k value - city of Charleston ($\bar{t} = 600s$, $t_{max} = 4h$, and $\bar{p} = 3$)	141
6.18	Pareto Frontier by k in San Francisco ($\bar{t} = 600s$, $t_{max} = 4h$, and $\bar{p} = 3$)	142
6.19	Critical Distance plots on the equity value	144
6.20	Example of an LP relaxation of an instance with 51 nodes	146
6.21	Example of an instance made from random (R) points	154
6.22	Regression of the normalized improvements	159

List of Tables

2.1	Consumption of budgets B_1 and B_2 .	18
3.1	Comparison between the study in Wyrowski et al. (2024) and the present one.	31
3.2	Sets and Parameter	36
3.3	Decision Variables	37
3.4	Base CPLEX: solution statistics	51
3.5	Implemented Branch-and-Cut: solution statistics	52
3.6	Statistics on generated number of cuts	53
3.7	Valid inequalities incorporated into the model from scratch: solution statistics	54
3.8	CPLEX and ALNS results for the instances solved to optimality	55
3.9	ALNS results	56
3.10	Correlation matrix	57
3.11	ALNS solutions statistics	59
3.12	Average objective value obtained from ALNS with various values for the cooling rate and the time limit.	59
3.13	ALNS time to best statistics	60
3.14	Sensitivity analysis – percentage of parcels delivered by crowdshippers across different cost configurations	64
3.15	Sensitivity analysis – objective function value across different cost configurations	65
3.16	Sensitivity analysis – objective function value across different time-slot configurations	66
3.17	Sensitivity analysis – percentage of parcels delivered by crowdshippers across different time-slot configurations	67
3.18	Percentage of parcel paths for each length, grouped by network size $ N $ and number of available crowdshippers $ K $	68
3.19	Travel time comparison between the Traditional Delivery System and the Crowdshipping System under varying numbers of crowdshippers	73
4.1	Sets	77
4.2	Parameters	78
4.3	Variables	78
4.4	Label Fields	85
4.5	Total run time statistics for the three configurations by number of crowdshippers	91
4.6	Total run time statistics for the three configurations by number of parcels	92
4.7	Column selection rule ranking by $ K $ and $ P $	92

4.8	Comparison of column selection rules by $ K $ and $ P $	93
4.9	Impact of backup parcels on algorithmic performance	94
4.10	Statistics about the number of BnB nodes for the three configurations	95
4.11	Statistics on the number of generated columns for the three configurations .	96
4.12	Comparison of solution methods for PTCP	99
6.1	Comparison of the literature on Vehicle Routing Problems	118
6.2	Comparison of Related Kernel Search Literature	121
6.3	Instance generation parameters.	128
6.4	Value of f_2 when traversing each option between A and B under the change of k	130
6.5	Optimal solutions for the optimization $f_1 (f_2^*, 0)$ for different values of k . .	131
6.6	Pearson correlation index between parameters and objective function values	143
6.7	k-worst value statistics for the four formulations	155
6.8	Formulation Rankings	156
6.9	Pairwise formulation comparisons: number of wins, losses, and ties	157
6.10	p -values for t -test and Wilcoxon tests	157
6.11	TS-ALNS results and computational time with $I = 10, 25, 50, 100$	158
6.12	TS-ALNS results and computational time with $I = 200, 300, 400$	158
6.13	Gurobi results in 1 hour compared to TS-ALNS	160
6.14	Gurobi k-sum results	160
6.15	Configuration Parameters	161
6.16	SAKS Threshold values	162
6.17	KS results with different configurations	162
6.18	Configuration ranking	162
6.19	Binomial tests	163
6.20	Wilcoxon tests	164
6.21	Gaps between KS and Gurobi	164
6.22	Gaps between KS and Gurobi Best Solutions	165

Chapter 1

Introduction

Last-Mile Delivery denotes the final transport step that brings goods from local depots to end customers in urban and suburban areas. It has become a central component of parcel logistics as electronic commerce reshapes shopping habits and increases delivery volumes worldwide. The global Last-Mile Delivery market was estimated at about 133 billion US dollars in 2022 and is expected to almost double by 2030, with annual growth close to 9 percent (Research, 2023). In major markets, last-mile activities account for roughly half of total shipping or logistics costs, despite representing only a short segment of the physical journey (DHL, 2023). This cost share reflects dense stop patterns, high labor and vehicle costs, repeated delivery attempts, and the need to respect tight delivery time windows. The Vehicle Routing Problem, first introduced by Dantzig and Ramser (1959), played a central role in Operations Research and Last-Mile Delivery. It consists of finding minimum-cost routes of a set of vehicles with limited capacity, starting from a depot, visiting customers exactly once, and returning to the depot.

In this thesis, we focus on two distinct innovations for last-mile delivery. The first one, explored in Part I, deals with public transport crowdshipping, a novel delivery paradigm where public transport users are included in the delivery process, in exchange for a small remuneration. Automatic parcel lockers are installed in public transport stations. Some of these lockers, likely outside the city boundary, are loaded in the morning with parcels by a logistics service provider. Then, commuters, called *crowdshippers*, on their regular journey, transfer parcels from the initial locker to the destination one selected by the recipient. It is possible to deliver parcels in more than one step. In this case, parcels are left at lockers acting as transshipment points. A backup service is available to deliver parcels that cannot be handled by crowdshippers. Such a paradigm presents several advantages over traditional delivery systems. First, it can reduce the number of delivery vehicles on the road by shifting part of the workload to commuters and vehicles already traveling for other purposes. Also, it may serve as an incentive for people to commute by public transport, as to commute using public transport, since the overall cost for a user crowdshipping is smaller. However, this delivery method presents several challenges. Some of these decisions are strategic, such as the placement of parcel lockers at public transport stations, while others are operational, such as routing parcels through the network. Others relate to the legal framework needed to allow regular commuters to handle parcels, as well as the economic design required to keep the initiative attractive to commuters while remaining profitable for the operator. These aspects are beyond the scope of this thesis, therefore they will not be discussed further. In this thesis, we analyze the strategic decision concerning the optimal installation of automated

parcel lockers in a network, and the operational problem of routing parcels through the network using crowdshippers.

The second novelty, presented in Part II, proposes two distinct extensions of the Vehicle Routing Problem. Both extensions optimize, alongside the delivery costs, delivery plans' social impact. Most VRPs optimize for costs, and some authors proposed extensions to take into account environmental impact, workers' equity, and customer well-being, among others. However, most studies fail to take into account the overall social impact, measured as the disturbance imposed on citizens not involved in any part of the process. The first novel formulation introduces a penalty associated with certain city districts in certain time windows, aiming at, for instance, discouraging traffic near schools when children are arriving or leaving. The second one uses a compact k -sum formulation to steer traffic away from densely populated areas.

Contributions

The main contributions of this thesis are as follows.

1. We have formulated, modeled, and solved a number of problems connected to Last-Mile Delivery and solved several novel problems in Last-Mile Delivery.
2. We have developed, tuned, and analyzed several tailored solution procedures for medium- and large-sized instances based on variants of matheuristics (Kernel Search) and metaheuristics (ALNS and Tabu Search).
3. We have developed exact approaches based on Branch and Cut with several classes of valid inequalities and Branch and Price.
4. We have generated new realistic and synthetic benchmark instances for the studied problems. We have conducted extensive computational and sensitivity analysis on these instances, understanding models and algorithmic properties and drawing useful managerial insights.

Structure of the Thesis

This thesis is divided into two parts and 7 chapters.

Part I focuses on Public Transport Crowdshipping. It studies delivery systems where public transport users participate in parcel delivery in exchange for a monetary incentive. The main contributions of this part concern optimization models and exact and heuristic algorithms for strategic and operational planning. Part II focuses on the social impact of urban parcel delivery. It introduces routing models that explicitly account for equity and social externalities, together with dedicated solution methods.

Chapters 2–6 are based on published or submitted research papers. Each chapter is self-contained. It presents a formal problem definition, a review of the relevant literature, and the proposed models and algorithms. When a solution methodology is introduced, it is preceded by a short methodological overview.

Chapter 2 studies an automated parcel locker location problem in a public transport network. The chapter introduces a mixed-integer optimization model to select locker locations that maximize service quality and customer satisfaction while controlling installation costs. Computational results show that a limited number of well-placed lockers can significantly improve the feasibility of crowdshipping operations. Chapter 3 addresses the operational

crowdshipping problem. It models parcel routing and transfer decisions over a public transport network, accounting for passenger flows, locker capacities, and timing constraints. The chapter proposes exact and heuristic solution approaches and evaluates their performance on realistic instances. Chapter 4 focuses on the exact solution of the operational crowdshipping problem introduced in Chapter 3. It presents an alternative non-compact mathematical formulation that enables a tailored branch-and-price algorithm. Computational results show that the proposed approach substantially improves scalability compared to a standard mixed-integer programming formulation. Chapter 5 introduces an off-peak delivery framework for urban logistics. The chapter proposes a model that promotes deliveries during off-peak hours time windows to reduce congestion and emissions. Results show that modest temporal shifts can lead to significant reductions in traffic-related externalities with limited impact on operational costs. Chapter 6 presents the k Equitable Vehicle Routing Problem. The chapter introduces a bi-objective routing model that jointly minimizes travel cost and social impact. Equity is captured through a k -sum objective that limits exposure in the most affected urban areas. A dedicated kernel search algorithm is proposed, and computational experiments show substantial improvements over general-purpose solvers.

Part I

Crowdshipping

Chapter 2

Locker Selection Problems for Public Transport-based Crowdshipping

The content of the following chapter is adapted and extended from Alessandro Gobbi, Renata Mansini, Lorenzo Moreschini, and Filippo Ranza. Optimal locker placement for budget-constrained crowdshipping in public transport networks. IFAC-PapersOnLine, 59(10):3041–3046, 2025, presented at the 11th IFAC Conference on Manufacturing Modelling, Management and Control – IFAC MIM2025.

Contents

2.1	Introduction	6
2.2	Problem Description	8
2.3	Mathematical Formulation	8
2.3.1	Mixed Integer Programming Model	9
2.4	Solution Methodologies for Multi Objective Programs	11
2.4.1	The weighted sum method	11
2.4.2	The ϵ -constraint method	13
2.5	Experimental Analysis	14
2.5.1	Benchmark Instances	15
2.5.2	Sensitivity Analysis & Managerial Insights	15
2.6	Cycles in CCNDP-RC solutions	20
2.7	Conclusions	22

Summary Crowdshipping is a novel delivery paradigm where regular commuters are involved in the delivery process. The idea is to exploit the spare capacity of already traveling vehicles to reduce traffic, with the associated benefits thereof. Commuters who deliver parcels, called crowdshippers, receive a small remuneration for their service. Such remunerations are, in many cases, designed to be large enough to encourage people to participate but still small enough to avoid professionalization, i.e. people crowdshipping as their main or sole source of income. These systems can, in general, be classified in two macro categories:

Private-transport based crowdshipping, where crowdshippers commute with their own private vehicle, and Public-transport based crowdshipping, where transits are done via a public transport system.

Many public-transport based crowdshipping systems rely on a network of automated parcel lockers, installed in, or nearby, public transport stations. Parcels are placed by a logistics service provider in lockers in selected stations, so that crowdshippers can later transfer them, potentially in more than one step, to the locker selected by the recipient. By carefully selecting which stations will receive parcels from the logistics service provider, which are used as intermediate delivery points, and which are used by the customers to retrieve their parcels, it is possible to limit heavy vehicular traffic to selected areas outside the city and deliver parcels only through public transport vehicles. In this chapter, we address the strategic design of such a locker network inside a public transport system, where the optimal number of lockers to install in each station has to be found within budget constraints. Such an optimal design must take into account both the available traffic within the network, and the customers' demand outside it. To this end, we propose a bi-objective mixed-integer programming model that jointly maximizes customer satisfaction and the spatial coverage of the locker network, while explicitly considering fixed installation, variable operational costs, and constraints imposed by the available traffic flow.

2.1 Introduction

Large metropolitan areas pose considerable difficulties for parcel delivery companies. Congested roads, scarce parking, and unpredictable access to customers often lead to inefficiencies such as delays, rising costs, and reduced service quality. To overcome these issues, many delivery systems now rely on designated collection points, where recipients pick up parcels instead of receiving them at home.

Nowadays, customers' expectations are shifting towards more flexible, faster, and more reliable services. Emerging logistics models aim to satisfy these requirements by providing tailored, efficient delivery solutions that not only improve customer satisfaction but also reduce the environmental impact of last-mile operations (Bonomi et al., 2022b). In this context, the strategic decision of the location of parcel lockers becomes a critical factor in optimizing delivery systems. With customers seeking faster, greener, and more flexible services, strategically placing lockers can substantially enhance the efficiency of the delivery process. Several authors have addressed this issue, employing various solution methodologies to determine optimal locker placements. For instance, in Moslem and Pilla (2023), the authors use the Spherical Fuzzy Analytic Hierarchy Process to identify six metrics for preferred locations, and five hierarchical scores to evaluate the quality of each potential position. Their study identifies public transport stops - like bus stops and subway stations - as the most suitable locations. Ottaviani et al. (2023) combines Mixed Integer Linear Programming (MILP) with greedy heuristics to minimize lockers' installation costs, all while ensuring client satisfaction by imposing dedicated constraints.

Lockers installed in densely populated or other hot-spots enable crowdshipping-based last-mile delivery systems, where commuters transfer parcels during their usual journey. In fact, lockers in areas with a larger population size, near by amenities, and close to bus stops experience a larger customer demand (Ma et al., 2024). However, parcel lockers can face some resistance by customers, despite their environmental advantages (Karlı and Karlı, 2026). Crowdshipping is inspired by the *hub-and-spoke* model, where parcels are stored at intermediate points before final delivery by various fleets. Bruno et al. (2025) optimize the

selection of pick-up point (i.e. lockers installation points) locations in order to maximize customers choosing self collection. Pan et al. (2020) propose a two-echelon optimization model to minimize locker installation, operational, and delivery costs, relying on vans to transport parcels from depots to lockers, and then onward to final customers. Yang et al. (2020) introduce a Bi-Level model to minimize the lockers' installation cost, while the lower level minimized the customer's pick up costs. Ozyavas et al. (2024) study a hybrid locker-based and hub-and-spoke system, where lockers serve as storage for the company and pick-up points for customers, while residual deliveries are handled by cargo bikes. Lin et al. (2022) maximize locker placement profit by accounting for fixed costs and revenue, with customer pick-up preferences estimated via a threshold Luce model. Li and Li (2025) propose to use the well-known p -median problem to identify the locker location given minimizing the detour length of people traveling by taxi. Wang et al. (2024) optimize mobile locker routes with recourse strategies for uncertain demands and pick-up times. Deutsch and Golany (2018) study the location problem of parcel lockers to facilitate a last mile delivery system to maximize the total profit. Commuter-based crowdsipping systems using public transport are quite interesting since they offer a sustainable alternative to private transport-based models, reducing traffic and environmental impact (Cebeci et al., 2024; Buldeo Rai et al., 2018). Moreover, such systems also meet the European Commission's directive for efficient, multimodal transport systems to enhance life quality and cut emissions (European Commission, 2020). Such approaches typically do not require commuters to change their usual routes, such a constraint is in place to make the system more attractive for the commuters. Studies like Gatta et al. 2019a and Gatta et al. 2019c, highlight that locker placement is a most critical aspect for system efficiency with a large number of users. Cheng et al. (2024) presents two MILP models to minimize locker placement costs and van travel times, assuming the locker locations and parcel destinations are predefined. Tests on Copenhagen data highlight the benefits of public transport-based crowdsipping. Zhang et al. (2023) considers a crowdsipping network involving commuters using public transport and regular delivery drivers as well. Such a problem aims to minimize locker placement costs and parcel movements using data derived from travel surveys in the Greater Toronto Area. The interested reader can refer to Boysen et al. (2021) for an in-depth analysis of the latest advances in last-mile delivery and parcel lockers.

In this chapter, we study the strategic decision of optimal design and sizing of a crowdsipping system where commuters are responsible for transferring parcels between lockers located at public stations. The system is structured so that logistics service providers (LSPs) drop off parcels at easily accessible stations (*drop off points*), for instance outside the city or near their depot. Then commuters transfer, in one or more steps, to the stations selected by the recipients as *pick-up points*. In particular, we focus on the perspective of an LSP that must decide whether and how much to invest in the realization and management of such a delivery system. The LSP faces both one-time costs to purchase and install lockers, as considered for example in Kahr (2022), and recurring daily costs to transfer parcels from the depot to drop-off points. Therefore, it is crucial to understand how to manage the available resources and assess their impact on the size of the network. For the operational details of such a network, please refer to Chapter 3.

This chapter contributes in several respects:

1. We introduce a novel problem concerning the joint optimization of locker placement and network size, based on estimated commuters flows in a public transportation system.

2. We propose and solve a new bi-objective MILP model that simultaneously maximizes customer satisfaction and the number of stations equipped with lockers.
3. We generate and analyze synthetic benchmark instances that vary in terms of transport lines, station numbers, and traffic flows, thereby testing the robustness of the model under different conditions.
4. We provide a general decision-support tool to assess the feasibility of commuter-based crowdsipping systems.

The remainder of this chapter is organized as follows: Section 2.2 introduces the problem in detail, and Section 2.3 presents the MILP formulation. Then, Section 2.4 discusses solution methodologies for multi-objective optimization, Section 2.5 reports computational experiments and managerial insights, and finally Section 2.7 concludes with final remarks and possible directions for future research.

2.2 Problem Description

Let $G = (N, A)$ be a directed complete graph, where N represents the set of stations and A is the set of connections among them. Such connection can be either direct physical links or indirect paths in the network. Crowdshippers collect parcels at their boarding station and deposit them at their destination station. A parcel can be delivered in more than one segment, by more than one crowdshipper. A crowdshipper can carry at most one parcel. Each arc $(i, j) \in A$, is associated with a value $\tau_{ij} \in \mathbb{N}$, which represents the average number of crowdshippers traveling from station i to station j in each day. Let C be the set of customers, and $N^c \subseteq N$ the stations where each $c \in C$ is willing to collect their parcel. Let $s_{ic} \in \mathbb{N}$ be the satisfaction score for station $i \in N^c$. Lockers are installed in fixed-capacity *racks* of size $\eta \in \mathbb{N}^+$. To install lockers in a station $i \in N$, the LSP incurs a fixed setup cost $a_i \in \mathbb{N}$ and a proportional cost $b_i \in \mathbb{N}$ per rack, with a maximum of $m_i \in \mathbb{N}$ racks per station. The total installation cost across all selected stations is limited to $B_1 \in \mathbb{N}$. Choosing station $i \in N$ as a drop-off point incurs a daily cost $c_i \in \mathbb{N}$ for the LSP, with total costs bounded at $B_2 \in \mathbb{N}$. The goal of the Commuter-based Crowdsipping Network Design Problem with Resource Constraints (CCNDP-RC) is to determine the optimal selection of drop-off and pick-up points, as well as the identification of the number of racks to install at each station, to maximize customer satisfaction and ensure the widest possible distribution of pick-up points.

2.3 Mathematical Formulation

The model has to make two connected decisions: how many racks to install in each station in the network and the role each station with at least one rack plays in the delivery system. Among these stations, we identify three classes:

- **Drop-off Stations:** these stations are used in the morning by the LSP to drop parcels that will by delivered be crowdshippers.
- **Pick-up Stations:** these stations are used by the recipients to pick up their parcels.
- **Intermediate Stations:** these stations are used by crowdshippers as transshipment points when a delivery requires more than one person to be completed.

Moreover, these decisions have to be taken by considering the commuters flow, station physical limit, and the customers needs as well.

To define a unified model that makes all these decisions at once, we propose a network flow design problem. Such network uses three types of nodes: regular nodes representing the public transport stations, a fictitious node s representing the LSP's depot, and a fictitious node t representing the destination of the flow. Parcels flow originating the fictitious node s is moved through the network by crowdshippers to reach a fictitious node t . Nodes receiving flow directly from s can be identified as the drop-off ones, whereas nodes sending flow directly into t identify the pick-up stations. With the exception of s and t , the flow entering and leaving any node $i \in N$ has to remain constant. The parcels' flow between two stations has to respect the commuters capacity τ_{ij} , and the lockers' capacity in node i and j .

2.3.1 Mixed Integer Programming Model

Formally, we define an extended directed graph $\bar{G} = (\bar{N}, \bar{A})$, where $\bar{N} = N \cup \{s, t\}$ and $\bar{A} = A \cup \{(s, i) | i \in N\} \cup \{(i, t) | i \in N\}$.

The MILP model uses the following decision variables:

- $r_i \in \mathbb{N}$: the number of racks to be installed in station $i \in N$;
- $o_i \in \mathbb{N}$: the order of visit of station $i \in N$ in the path from s to t ;
- $z_i \in \{0, 1\}$: 1 if at least one rack is installed in station $i \in N$ and 0 otherwise;
- $e_{ij} \in \{0, 1\}$: 1 if at least one parcel is moved from node i to node j with $(i, j) \in \bar{A}$;
- $f_{ij} \in \mathbb{N}$: the number of parcels directly transferred from node i to node j with $(i, j) \in \bar{A}$;
- $y_{ic} \in \{0, 1\}$: 1 if customer $c \in C$ picks up their parcel in station $i \in N^c$, and 0 otherwise.

We formulate the CCNDP-RC as follows:

$$\max f_1 := \sum_{c \in C} \sum_{i \in N^c} s_{ic} y_{ic} \quad (2.1)$$

$$\max f_2 := \sum_{i \in N} e_{it} \quad (2.2)$$

$$z_i \leq r_i \leq m_i z_i \quad i \in N \quad (2.3)$$

$$\sum_{j \in N | (i,j) \in A} f_{ij} = \sum_{j \in N | (j,i) \in A} f_{ji} \quad i \in N \quad (2.4)$$

$$z_i \leq \sum_{j \in N | (i,j) \in A} f_{ij} \leq \eta r_i \quad i \in N \quad (2.5)$$

$$e_{si} \leq f_{si} \leq \eta m_i e_{si} \quad i \in N \quad (2.6)$$

$$e_{it} \leq f_{it} \leq \eta m_i e_{it} \quad i \in N \quad (2.7)$$

$$e_{ij} \leq f_{ij} \leq \tau_{ij} e_{ij} \quad (i, j) \in A \quad (2.8)$$

$$o_j \geq o_i + 1 - |\bar{N}|(1 - e_{ij}) \quad (i, j) \in \bar{A} \quad (2.9)$$

$$\sum_{c \in C | i \in N^c} y_{ic} = f_{it} \quad i \in N \quad (2.10)$$

$$\sum_{i \in N^c} y_{ic} \leq 1 \quad c \in C \quad (2.11)$$

$$\sum_{i \in N} a_i z_i + \sum_{i \in N} b_i r_i \leq B_1 \quad (2.12)$$

$$\sum_{i \in N} c_i e_{si} \leq B_2 \quad (2.13)$$

$$r_i, o_i \in \mathbb{N} \quad i \in N \quad (2.14)$$

$$z_i \in \{0, 1\} \quad i \in N \quad (2.15)$$

$$e_{ij} \in \{0, 1\} \quad (i, j) \in \bar{A} \quad (2.16)$$

$$f_{ij} \in \mathbb{N} \quad (i, j) \in \bar{A} \quad (2.17)$$

$$y_{ic} \in \{0, 1\} \quad c \in C, i \in N^c \quad (2.18)$$

The primary objective function (2.1) maximizes the total customer satisfaction, whereas the secondary objective function (2.2) maximizes the number of stations used as pick-up points. Objective function (2.1) tends to select stations in densely populated areas, and configures all the stations as both pick-up and drop-offs, without opening any intermediate stations. Of course, such a decision is the one maximizing lockers utility since all of them provide service to the final customers. However, such a solution has two main drawbacks. First, it creates a delivery network that does not use crowdshippers to deliver parcels, going against the high level objective of the study. Second, it precludes service from less densely populated areas of the city. Thus comes into play objective function (2.2), that forces the model to select a larger number of pick-up points. This objective incentivize the model to select as pick-up points also stations away from densely populated areas, and so to keep under control the handling costs by the LSP the model also selects stations as intermediate transfer.

For each station $i \in N$, Constraints (2.3) impose that a station becomes part of the crowdshipping network ($z_i = 1$) if and only if it hosts at least one rack ($r_i \geq 1$), and limit to m_i the number of racks that can be installed. Constraints (2.4) ensure flow conservation by setting that the flow entering is equal to the flow leaving each station $i \in N$. Nodes s and t are not included into these constraints since parcels flow originates and terminates in them. Constraints (2.5) impose consistency between the number of racks installed and the parcel flows handled at a station, ensuring that capacity is not exceeded. Constraints (2.6) mark a station $i \in N$ as a drop-off point ($e_{si} = 1$) if and only if this node receives a non-zero flow from the source node $s \in \bar{N}$. Analogously, Constraints (2.7) mark a station $i \in N$ as a pick-up point ($e_{it} = 1$) if and only if this node sends a nonzero flow to the sink node $t \in \bar{N}$. For each arc $(i, j) \in A$, Constraints (2.8) set each binary variable e_{ij} to one if and only if the flow between nodes i and j is non-zero. Constraints (2.9) impose an ordering on the nodes guaranteeing that nonzero flow only enters nodes with a higher rank than the origin. These constraints follow the Miller-Tucker-Zemlin formulation for the Traveling Salesman Problem (Miller et al., 1960) and are needed to prevent the creation of ‘virtual flow’, i.e. flow circulating between intermediate nodes that is not coming from s and not going into t . Such flow is harmful since it causes an artificial increase in the number of installed lockers, thereby increasing the costs, without generating extra profit or service quality. Constraints (2.10) ensure that each unit of flow going from a station $i \in N$ to the sink node t corresponds to a customer reaching that station to pick up its parcel, and Constraints (2.11) guarantee that each customer can be associated with at most one pick-up point. Constraints (2.12) and (2.13) impose the spending limits for rack installation and parcel transportation to the drop-off points (i.e. stations $i \in N$ identified by $e_{si} = 1$). Finally, Conditions (2.14) to (2.18) define the model variables.

2.4 Solution Methodologies for Multi Objective Programs

Usually, Mixed Integer Linear Program (MILP) problems have one objective function, representing the sole goal of the decision maker. However, in certain situations it is not possible to represent the whole complexity of the decision making process through a unique objective function. For example, consider a Vehicle Routing Problem (VRP) where it is required to minimize the delivery costs and maximize the customer satisfaction. It is clear that it is not meaningful to incorporate such diverse quantities in one unique objective. In these situations two, or more, concurrent objective functions are required to fully represent the problem at hand. While the *best* possible solution of a single objective function is clear - it is the optimal value of the objective function - it is impossible to define the *best* solution in multi-objective problems, since there are many solutions such that it is impossible to improve one objective function without spoiling the value of any other one. Such solutions are called Pareto efficient solutions, and their set is defined as the Pareto front.

For simplicity, in the following we assume to have two maximization objective functions. Such simplifications are not restrictive, since all the methodologies presented hereafter are easy to extend to more than two objective functions, and to adapt to minimization functions requires only sign management. The reader interested to a more in-depth discussion on multi objective problem is referred to Ehrgott (2005). Moreover, in this section we will focus only on general purpose exact methods using MILPs and solvers.

In the following the generic optimization problem (2.19)–(2.22) will be used to explain the solution methods.

$$\max f_1 := c_1^T x \quad (2.19)$$

$$\max f_2 := c_2^T x \quad (2.20)$$

s.t

$$Ax \leq b \quad (2.21)$$

$$x \in \mathcal{X} \quad (2.22)$$

2.4.1 The weighted sum method

Given the Bi-Objective MILP in (2.19)–(2.22), the weighted sum method defines a new objective function as the weighted sum of two previous ones, as follows:

$$\max f := \alpha_1 f_1 + \alpha_2 f_2 \quad (2.23)$$

s.t

$$Ax \leq b \quad (2.24)$$

$$x \in \mathcal{X}. \quad (2.25)$$

Controlling the values of the coefficients, it is possible to both scale the functions so that they represent the same unit of measure and to set the priority of the two objective functions. For instance, if f_1 and f_2 both range over the unit interval, using $\alpha_2 \gg \alpha_1$, will prioritize f_2 over f_1 . The vice versa is also true. However, since the two objective function might take values that are orders of magnitude apart, it can be hard to correctly find such parameters α_1 and α_2 that impose a correct scaling and a correct priority, especially in the general case. When this is the case, it is possible to normalize them, as shown in (2.26).

$$\hat{f}_k = \frac{f_k - f_k^{\min}}{f_k^{\max} - f_k^{\min}} \quad k = 1, 2 \quad (2.26)$$

Notice that such a normalization guarantees that:

$$\hat{f}_k : \mathcal{X} \rightarrow [0, 1] \quad k = 1, 2, \quad (2.27)$$

ensuring that the two objective function take comparable value, at least in terms of size. However, it is worth noting that the normalization approach can be computationally expensive, since it requires the optimization of four distinct optimization problems, to identify the value of $f_k^{\min}$ $k = 1, 2$, before it is possible to tackle the bi-objective problem.

In summary, the weighted sum is straightforward to implement and, if run without normalization, efficient to solve since the two objective functions are optimized as one. However, it can be hard to control the trade-off between the two objective functions, since the impact of the two coefficients is highly dependent on the values taken by the two objective functions. Moreover, if the two objective functions take values that differ by one or more order of magnitude, it might be necessary to normalize the two objective function, which adds computational burden to the optimization.

On the Identification of the Pareto front

The weighted sum method by itself cannot identify all point of the Pareto front.

It is straightforward to identify the Pareto front using the weighted sum method. By defining the objective function as

$$f_\lambda := \lambda f_1 + (1 - \lambda) f_2 \quad (2.28)$$

with $\lambda \in [0, 1]$, it is possible to find the Pareto front, with precision ϵ and tolerance δ using Algorithm 2.1.

Algorithm 2.1 Pareto front with Weighted sum method

```

1: function WEIGHTED-SUM-PARETO-FRONT(f: the Pareto front, p: MILP to solve,  $\lambda_1, \lambda_2$ ,
    $\epsilon, \delta$ )
2:    $z_{\lambda_1}^* \leftarrow$  solve p with objective function  $f_{\lambda_1}$  and add value to f
3:    $z_{\lambda_2}^* \leftarrow$  solve p with objective function  $f_{\lambda_2}$  and add value to f
4:    $\triangleright$  Memoization can be used to avoid solving the same problem twice.
5:   if  $|z_{\lambda_1}^* - z_{\lambda_2}^*| \geq \epsilon$  and  $|\lambda_1 - \lambda_2| \geq \delta$  then
6:      $\lambda_3 \leftarrow \frac{\lambda_1 + \lambda_2}{2}$ 
7:     WEIGHTED-SUM-PARETO-FRONT(p,  $\lambda_1, \lambda_3, \epsilon, \delta$ )
8:     WEIGHTED-SUM-PARETO-FRONT(p,  $\lambda_2, \lambda_3, \epsilon, \delta$ )
9:   end if
10: end function
    
```

Algorithm 2.1 describes a recursive procedure that adaptively identifies the Pareto front by solving weighted-sum formulations at different points in the weight space.

The procedure starts by solving the MILPs at two extreme weights, $\lambda_1 = 0$ and $\lambda_2 = 1$, corresponding to the optimization of each single objective. The resulting solutions $z_{\lambda_1}^*$ and

$z_{\lambda_2}^*$ are stored in the set $\mathbf{f}$, which represents the current approximation of the Pareto front. In a recursive algorithm it is possible for the same sub-problem to be solved more than once. An uncaredful implementation of Algorithm 2.1, might optimize the same problem up to three times (the first time, in the left, and in the right recursive call). To avoid such redundant computation, it is possible to store previously solved problems, and then when the same problem arises again its previous solution can be retrieved and used for the next steps. Such caching technique is known as *memoization*, and it is commonly used as an acceleration method for recursive algorithms.

If the objective values corresponding to the two weights differ by more than a threshold ϵ , and the distance between the weights is larger than a tolerance δ , the algorithm considers the midpoint weight $\lambda_3 = \frac{\lambda_1 + \lambda_2}{2}$. The algorithm is recursively applied to the two sub-intervals $[\lambda_1, \lambda_3]$ and $[\lambda_2, \lambda_3]$. Such a recursion allows the algorithm to automatically explore in more detail areas of the weight space where the objective function changes significantly, while areas that are *flatter* are explored less intensively.

This recursive refinement continues until no significant improvement can be obtained, either because the objective values between two neighboring solutions are sufficiently close (within ϵ) or because the weight interval is too small (below δ). The final set $\mathbf{f}$ constitutes an approximation of the Pareto front obtained through a sequence of weighted-sum MILP solves.

The algorithm is intended to be called with $\mathbf{f} = \emptyset$, $\lambda_1 = 0$, and $\lambda_2 = 1$.

Supported and Unsupported Pareto Efficient Solutions

Let $\mathcal{Y} = \{f_1(x), f_2(x)\} : x \in \mathcal{X}\}$ be the solution space of (2.19)–(2.22). If the feasible region $\mathcal{X}$ is non-convex, the solution space $\mathcal{Y}$ is non-convex as well. A practical consequence is that there is no guarantee that the Pareto front will be convex. Indeed, the Pareto front is constituted by so-called *supported* and *unsupported* solutions. Supported solutions are ones forming a convex subset of the Pareto front, and can be identified by linear scalarization techniques. Unsupported, being part of a non-convex set, ones are instead those solutions that cannot be identified by linear scalarization techniques. Therefore, the Pareto front identified with Algorithm 2.1 can provide a good approximation of the true front. In the general case, little can be said about the quality of such an approximation.

2.4.2 The ϵ -constraint method

Given the generic Bi-Objective MILP problem in (2.19)–(2.22), the ϵ -constraint method works as follows. First, the main objective function f_1 is optimized as a standalone objective function, subject only to the original problem's constraints, like shown in (2.29)–(2.31).

$$\max f_1 := c_1^T x \quad (2.29)$$

s.t

$$Ax \leq b \quad (2.30)$$

$$x \in \mathcal{X} \quad (2.31)$$

Then, the secondary objective function f_2 is optimized, as shown in (2.32)–(2.35), subject to the original constraint plus the ϵ -constraint (2.34).

$$\max f_2 := c_2^T x \quad (2.32)$$

s.t

$$Ax \leq b \quad (2.33)$$

$$c_1^T x \geq (1 - \epsilon)f_1^* \quad (2.34)$$

$$x \in \mathcal{X} \quad (2.35)$$

Such a constraint allows the optimization of f_2 upto a maximum worsening of the value of f_1 of $(1 - \epsilon)$ with respect to its optimal value f_1^* . In many cases, a third problem is optimized, in which the value of f_1 is optimized again, without allowing worsening to the value of f_2 . This ensures that the identified solution will be Pareto efficient.

In the applications this method is appreciated for several reasons. For starters, it allows the decision maker to explicitly control the order of the optimization by choosing which objective function is optimized first and which is optimized later. More importantly, this technique allows to explicitly decide how much to spend on f_2 in terms of f_1 , allowing an explicit trade-off that is independent from the specific instance. Last but not least, with the epsilon constraint it is never a problem if the two objective functions have incomparable values, both in terms of metric and in terms of absolute value. However, this method always requires solving two, or three if the Pareto efficient solution is desired, optimization problems in order to identify the final solution.

Notice that the ϵ -constraint method is a generalization of the lexicographic approach, where the objective functions are solved in hierarchical order but without allowing any worsening to the previously optimized one.

On the Identification of the Pareto front

It is quite simple to find the Pareto front with the ϵ -constraint method as well. Given f_1^* , the optimal unconstrained value of f_1 , and $f_1^*(f_2^*, \infty)$, the Pareto efficient value of f_1 at f_2^* , it is possible to compute ϵ_{max} , which is smallest the value of ϵ that allows to find the optimal value of f_2 , as:

$$\epsilon_{max} = \frac{f_1^* - f_1^*(f_2^*, \infty)}{f_1^*}. \quad (2.36)$$

Considering that the smallest value of ϵ is 0, it is straightforward to adapt Algorithm 2.1 to work with the ϵ -constraint method. As such, it will not be presented here. Unlike the weighted sum, the ϵ -constraint method can explore the whole Pareto frontier, allowing the identification of both supported and unsupported solutions.

2.5 Experimental Analysis

In this section, we present the results obtained by solving the proposed model on a large set of synthetic instances. All tests have been executed on a computer equipped with an AMD Ryzen 5 5600U CPU, 40 GB of RAM, running Debian 12. The code is implemented in Python 3.11, and Gurobi 11.0.3 has been used as the back-end MILP solver. The Bi-objective MILP model is solved using the ϵ -constraint method, where f_1 is optimized as primary objective function and f_2 is the secondary. All solutions analyzed in the following are Pareto efficient with respect to f_1 .

2.5.1 Benchmark Instances

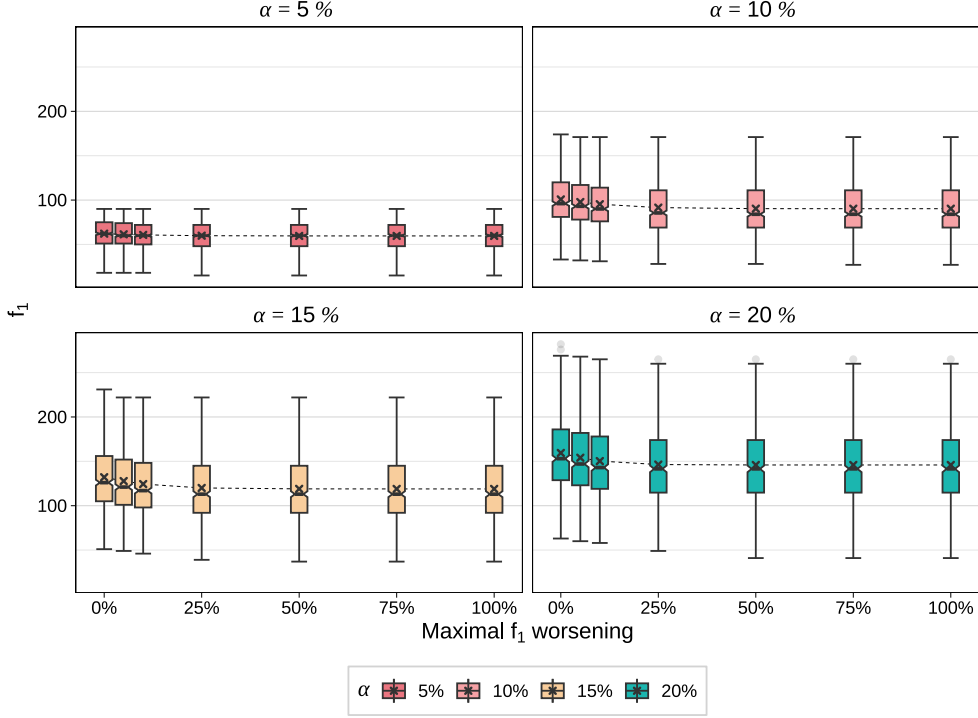
We generated eight distinct public transport networks over a rectangular area of $5 \times 10 \text{ km}^2$, each consisting of between two and six lines and comprising 51 to 132 stations. For every network, 80 test instances were created, each characterized by a different distribution of customers and traffic patterns. In total, this yielded a comprehensive test set of 640 instances. In particular, each instance has customer set $C \in [2000, 5000]$, and the traffic flow $\tau_{ij} \in [60000, 90000]$. To simulate customer distribution in each network, we randomly place 15 *focal points* across the map: 75% of these were designated as *attraction points* (e.g., schools, supermarkets), representing desirable areas for residential development, whereas the remaining 25% served as *repulsion points* (e.g., landfills), discouraging settlement nearby. Candidate customer positions were then evaluated based on their proximity to attraction and repulsion points and accepted if they are not too close to repulsion point and close enough to attraction points.

For each customer $c \in C$, N^c includes all stations located within a 1 km radius from it. The score s_{ic} takes a value equal to 3 if the distance between customer c and station $i \in N^c$ is lower than 250 m, 2 if the distance is between 250 and 500 m, 1 if the distance is between 500 m and 1 km. The LSP depot and the clients are located in the area, and the delivery cost c_i to reach station i is an increasing stepwise function of the distance between station i and the depot, with a minimum value of 10 and a maximum value of 100. The number of lockers η in each rack is set to 10 and the number of racks m_i that can be installed in a station i ranges between 2 and 5. We assume that the company has an annual budget B , which is divided into two parts: B_1 for the installation of the racks and B_2 for the daily delivery operations, referred to as the *daily budget*. The total budget is therefore $B = B_1 + 365B_2$, where B_2 is determined as $\lfloor \frac{\alpha}{365} B \rfloor$ for $\alpha \in \{5\%, 10\%, 15\%, 20\%\}$, and B_1 is computed as $B_1 = B - 365B_2$. In our experiments, we set $B = 500000$, and tested scenarios with $a_i = \frac{1}{100} B$, and $b_i = \frac{1}{50} B$ for each station $i \in N$. Finally, we computed Pareto-efficient solutions for $\epsilon = 0, 0.05, 0.1, 0.25, 0.5, 0.75$, and 1.

2.5.2 Sensitivity Analysis & Managerial Insights

All test instances were solved to optimality. The average solution time was 8.2 seconds, with a maximum of 299.8 seconds. In the following f_1^* and f_2^* represent the optimal value of f_1 and f_2 when optimized a first objective function. On the other hand, the notation f_1^ϵ and f_2^ϵ represent the Pareto efficient solution when optimizing f_1 (resp. f_2), with the extra ϵ -constraint on the maximal worsening on the value of f_2 (resp. f_1).

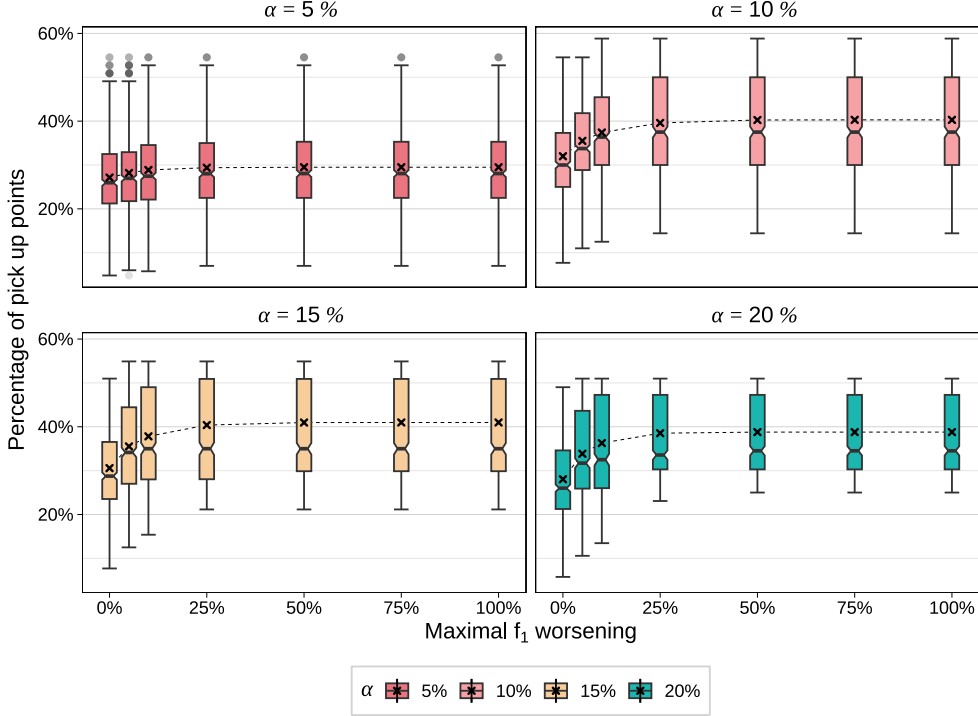
Each boxplot in Figure 2.1 shows the distribution of f_1^ϵ for different values of the daily budget and ϵ . Each boxplot illustrates the variation across instances, with the cross inside the box denoting the average objective value. The average values of f_1^ϵ are 60.4, 93.7, 123, and 150 for α equal to 5%, 10%, 15%, and 20%, respectively. The results show that the average value of f_1^ϵ increases as the daily budget B_2 grows: this means that allocating a larger portion of the total budget to daily deliveries yields better rewards than increasing the number of lockers designated only for pick-ups. It can be observed that the maximization of f_2 has a limited impact on the optimal value of f_1^ϵ compared to f_1^* since the average worsening from $\epsilon = 0\%$ to $\epsilon = 100\%$ is quite small, independently of the value α . In particular, with $\alpha = 5\%$, the gap between the average values of $f_1^{0\%}$ and $f_1^{100\%}$ is minimal and equals 4.1%, whereas the maximal gap is equal to 10.4% when $\alpha = 10\%$. This behavior is explained by the following aspects: the number of drop-off lockers is always equal to the number of pick-up lockers (due to the flow conservation constraints), and the larger B_1 the smaller B_2 becomes,


 Figure 2.1: Evolution of f_1^ϵ for each value of α .

leading to an inevitably small number of drop off lockers. Moreover, if the budget B_2 is too small, the low number of drop-off lockers that can be opened becomes a bottleneck for the system, reducing the total number of parcels that can be shipped, thus preventing the increase in f_2 . On the other hand, when B_2 is too large, the available budget to install intermediate lockers becomes a bottleneck; when this value is limited, it is also difficult to increase the number of pick-up points. This analysis highlights the need for the decision maker to carefully allocate economic resources among B_1 and B_2 . When the budget B_2 is too low, the capacity to deliver a large number of parcels is limited, making the crowdsourcing system ineffective; however, excessively high values for B_2 reduce the possibility of opening more pick-up points in the entire area and also reduce the effectiveness of crowdsourcing to contrast pollution and traffic.

Figure 2.2 shows the average percentage of pick-up points out of the total, for different values of ϵ . This value is the ratio between f_2^ϵ and the number of stations in each network. We observe that for $\epsilon \geq 25\%$, the average values are mostly steady, regardless of the allotted daily budget. All charts show nearly constant average values: allowing for further deterioration in the value of f_1 leads to improvements for f_2 by at most 1.4% in the second chart where the daily budget is set to 10% of B . This means that pursuing a widespread crowdsourcing network is not in contrast with the primary goal of maximizing customer satisfaction. On the other hand, the maximum gaps between $f_2^{0\%}$ and f_2^* range from 7.1% when the daily budget is 5% of B to 27% when the daily budget is 20% of B .

Since the total flow of parcels through customers is equal to the number of lockers at drop-

Figure 2.2: Percentage of pick up points for each α and ϵ .

off points, a smaller number of drop-off points results in fewer lockers assigned to pick-up points, thereby significantly impacting the system's expansion. For this reason, for $\alpha = 5\%$ we found the lowest average values for f_2 . Instead, in instances with $\alpha = 20\%$, the larger value of B_2 allows us to open more drop-off lockers, and consequently a larger number of pick-up lockers. It should be noted that this reasoning also explains the greater variance observed in the charts, as indicated by the wider boxplots, for higher values of B_2 . When the number of drop-off lockers is sufficiently high, more pick-up stations can be opened.

It is interesting to observe that neither low daily budgets ($\alpha = 5\%$) nor high daily budgets ($\alpha = 20\%$) provide the highest percentage of pick-up points: in fact, the highest percentage (out of all values of ϵ) is found for $\alpha = 15\%$ and equals 38.2%, closely followed by the value 37.9% achieved for $\alpha = 10\%$.

Even when the largest part of the total budget B is assigned to the locker installations ($\alpha = 5\%$), it is useless to open a large number of pick-up points as the number of drop-off lockers is small, reducing the total number of parcels that flow through the network. This analysis highlights the importance of properly distributing resources between fixed and variable costs.

For each value of α and ϵ , Figure 2.3 shows the percentage of stations chosen to install at least one rack of lockers (represented by triangles), and the number of stations selected as drop-off points (represented by dots). As expected, the number of drop-off points grows as the amount dedicated to the daily budget increases. Interestingly, when α is fixed, the percentage of drop-off stations is mostly constant, fluctuating by only 0.8% when $\alpha = 5\%$,

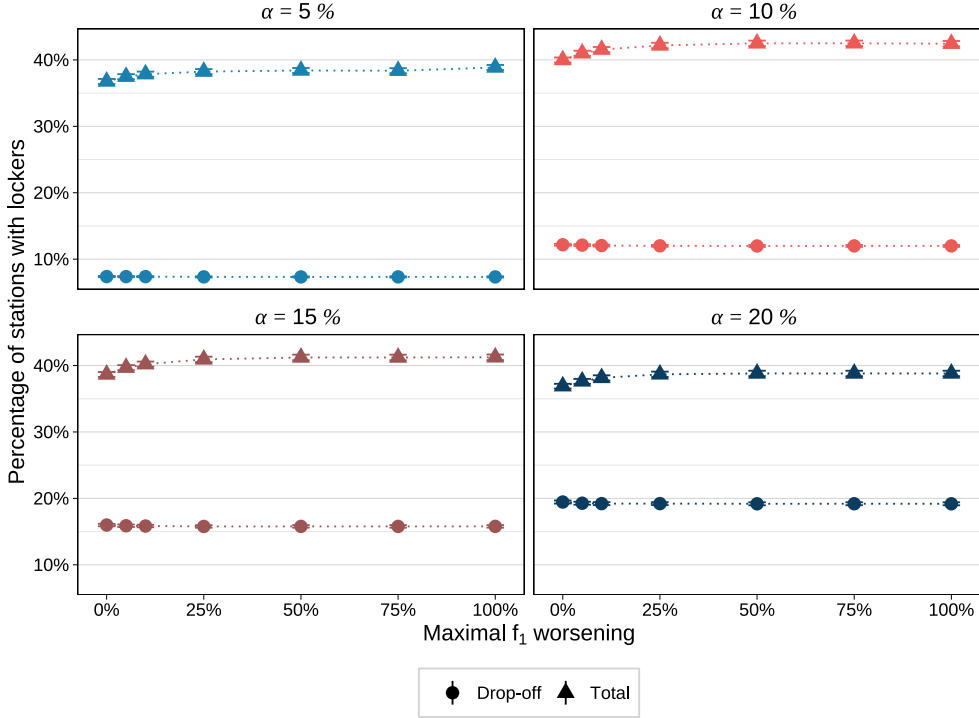


Figure 2.3: Percentages of stations hosting lockers and stations chosen as drop-off points for each value of α and ϵ .

and up to 1.5% when $\alpha = 20\%$.

Regarding the total number of stations selected to host at least one locker rack, we observe that this value is minimized when $\alpha = 20\%$. Although a higher number of drop-off points is required to deliver more parcels, the remaining budget for purchasing and installing lockers is lower compared to the other scenarios.

α	B_1 consumption			B_2 consumption		
	Mean	Min	Max	Mean	Min	Max
5%	88.3%	86.6%	89.1%	97.5%	97.5%	97.6%
10%	96.4%	94.4%	97.3%	97.3%	97.2%	97.4%
15%	97.7%	95.2%	98.7%	97.9%	97.9%	98.0%
20%	97.9%	96.0%	98.7%	98.6%	98.6%	98.6%

Table 2.1: Consumption of budgets B_1 and B_2 .

Table 2.1 shows, for each value of α , the average consumption of budgets B_1 and B_2 across all values of ϵ . The columns Mean, Min, and Max show the mean, minimum, and maximum consumption average for each budget. B_2 consumption never falls below 97% and is almost constant. This behavior aligns with the trend in the number of drop-off points shown in

Fig. 2.3, emphasizing their strong correlation. In addition, the almost complete consumption of B_2 confirms the intuition that optimal solutions aim to open as many drop-off points as possible. Moreover, as the percentage of the total budget allocated to B_1 decreases, its consumption increases, due to two distinct phenomena. Higher values of B_2 lead to lower B_1 values (owing to the relation with B), increasing consumption percentages even when the actual paid amount of B_1 does not change much. In addition, higher daily budgets enable more drop-off points, which require additional pick-up points and intermediate lockers, as the number of lockers for pick-ups equals those for drop-offs due to flow conservation constraints. Combining these two observations, it is possible to understand why less than 90% of B_1 is used when $\alpha = 5\%$, whereas when $\alpha = 20\%$ the consumption of B_2 is at least 96%. This also explains the smaller number of stations selected when the daily budget is set to $5\%B$ or $20\%B$, as shown in Fig. 2.3. In the first case, the limited number of drop-off points restricts the network's throughput, thus limiting the number of required pick-up points. In the other case, the high value of B_2 reduces the available B_1 for activating pick-up points, although the system could benefit from additional lockers (note that for $\alpha = 20\%$, B_1 consumption is nearly complete).

Overall, this analysis highlights the need for a careful allocation of the total budget among fixed and variable costs. In fact, if too large a percentage of B is allocated to B_1 or B_2 , our simulations show that various types of bottlenecks can occur, negatively impacting the performance of the crowdsipping system.

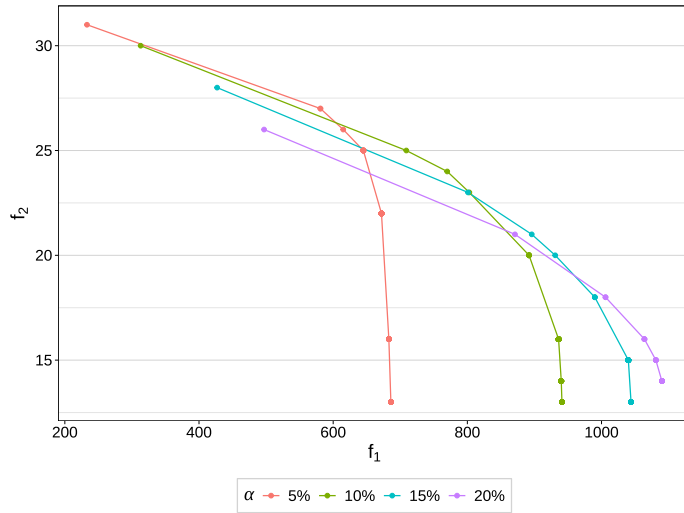


Figure 2.4: Example of a Pareto Frontier

Figure 2.4 shows the example of a Pareto frontier of a benchmark instance, under the change of the value α . The line represents the frontier, whereas the dots represent the distinct solutions. These frontier, identified using Algorithm 2.1, have same general evolution, and they all display the same concave shape common to a Pareto frontier of a max-max bi-objective problem. On the left hand side of the line, it is possible to observe the unconstrained optimal value of f_2 that generates a quite poor quality value of f_1 . At the other end, one can observe the unconstrained optimal value of f_1 with a quite poor value of f_2 . The intermediate points find a balance between the two objective function. The algorithm was

configured with $\epsilon = 0.01$ and $\delta = 0.01$, resulting in a total 128 different problems solved for each instance. The algorithm found 7, 8, 7, and 6 distinct solutions for $\alpha = 5\%$, 10%, 15%, and 20% respectively.

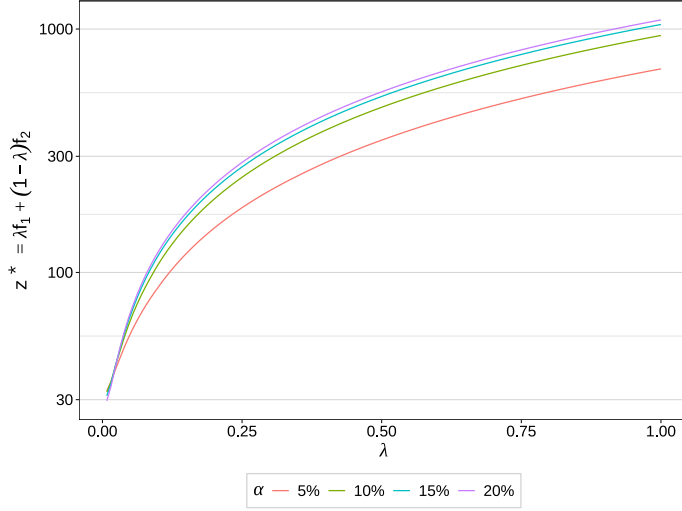


Figure 2.5: Value of the weighted sum $z^* = \lambda f_1 + (1 - \lambda)f_2$ for the frontier in Figure 2.4

In Figure 2.5 we present the value of weighted sum obtained by Algorithm 2.1 while identifying the frontier of Figure 2.4. Since the values of f_1 and f_2 are on quite different orders of magnitude, the y-axis, representing z^* , is on a logarithmic scale. As expected, as λ grows, so does z^* since the impact of f_1 becomes more and more relevant. Moreover it is possible to notice a clear change of the gradient around $\lambda = 0.25$. This is connected to the change in gradient that is possible to observe in Figure 2.4. The latter is connected to the behavior analyzed in Figure 2.2, where we observed that large f_1 values are connected with relatively good value of f_2 .

2.6 Cycles in CCNDP-RC solutions

In Figure 2.6 we present the optimal solution of a benchmark instance obtained without Constraints (2.9). Such a solution presents virtual flow between nodes 7 and 8, with their cycle highlighted in red.

Virtual flow can present itself whenever Constraints (2.8) provide a loose upper bound on the flow leaving a node i . Whenever the input flow in a node is less than the actual available capacity $\eta m_i r_i$, then Constraint (2.8) does not prohibit the generation of extra units of flow. Until the flow respects the conservation Constraints (2.4), for the solver it is possible to create virtual flow that circulates among the nodes without any changes the optimal solution. It is a simple case of multiple optimal solutions. Moreover, if budget constraints (2.12) is also loose, then it is also possible for the solver to increase the value of r_i , loosening further constraints (2.8), allowing for extra virtual flow.

Notice that minimizing the value $\sum_{i \in N} a_i z_i + \sum_{i \in N} b_i r_i$, without allowing any worsening to f_1 and f_2 , only partially addresses this issue. Even if such action will reduce the value r_i to the bare minimum, it will only partially tighten the right-hand side of (2.8). Take as example

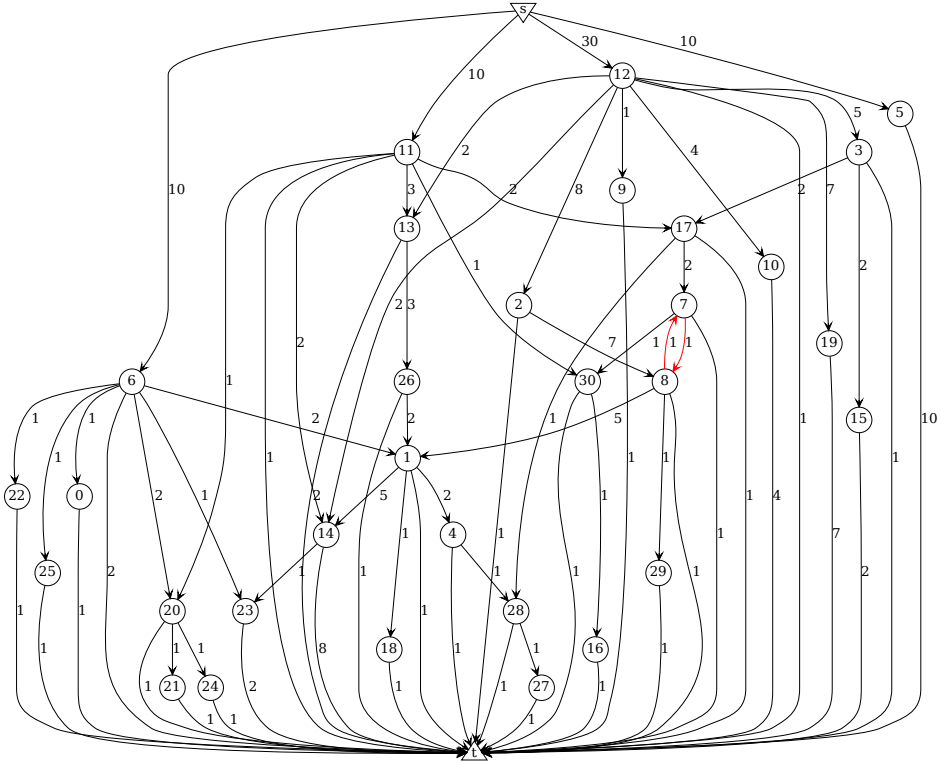


Figure 2.6: Example of a solution without Constraints (2.9).

the solution in Figure 2.6. In there, the minimal value of r_7 is 1. Therefore, the gap between the real outgoing flow and the actual capacity is 8, thus leaving some *free space* that can be filled with virtual flow.

Another corrective action could be to penalize the flow in the objective function. However, such a correction introduces a new set of issues to the optimization. Indeed, adding a penalty will not only change the objective function value, but might potentially change the optimal solution as well. Take into account f_2 . This objective function counts the number of nodes that send flow directly to t . It is easy to imagine, fact also confirmed by Figure 2.6, that many of these nodes provide more than one unit of flow to t . Therefore, their number is, in general, smaller than the total flow in the network. In such cases penalized f_2 takes negative values if there is flow, while it takes value 0 if there is no flow. It is evident that without this penalty the solution will take that not only is larger than zero, but its flow will be larger than 0. With a similar line of reasoning, it is possible to observe that a similar behavior is possible also when penalizing f_1 . Therefore, such penalization would require a correct weighting the penalization coefficient, which consists in an optimization problem of its own.

Last but not least, taking into account that any flow not originating from s and not targeting t will not alter neither of f_1 nor f_2 , and that a flow network is an acyclic directed graph, imposing a partial order to the nodes cannot change the value of the objective function and cannot change the optimal network layout. Therefore, including (2.9) cannot worsen the

objective solution, but it will actively remove any cycle in the optimal network.

2.7 Conclusions

This chapter has addressed the strategical decision of designing and dimensioning a commuter-based crowdshipping system integrated into the public transport infrastructure of a metropolitan area. A mixed-integer linear programming (MILP) model was developed to select the location and capacity of lockers at transport stations, with two concurrent objectives that maximized customer satisfaction and the extension of the distribution network. The proposed formulation has proven to be an effective tool for decision support, enabling logistics service providers to assess both the feasibility and the expected benefits of implementing a commuter-based crowdshipping network under a variety of conditions. The computational study has shown that the model can provide detailed insights into the allocation of financial resources between fixed installation costs and recurring operational expenses. By explicitly representing the trade-off between service quality and network coverage, the analysis offers valuable guidance for strategic decisions concerning investments in sustainable last-mile delivery systems.

Chapter 3

Public Transport-based Crowdshipping

This chapter is extended from Mikele Gajda, Olivier Gallay, Renata Mansini, and Filippo Ranza. Optimizing last-mile delivery through crowdshipping on public transportation networks. Transportation Research Part C: Emerging Technologies, 179:105250, 2025. The original work was presented at the International Conference of Optimization and Decision Sciences (ODS) 2023.

Contents	
3.1	Introduction 24
3.2	Literature Review 27
3.3	Problem Definition and Notation 32
3.4	Mathematical Formulation 36
3.4.1	Valid Inequalities 39
3.5	An ALNS Algorithm for the PTCP 40
3.5.1	Introduction to ALNS 40
3.5.2	The proposed ALNS Algorithm 41
3.6	A Branch-and-Cut Algorithm for PTCP 47
3.6.1	Introduction to the Branch-and-Cut Algorithm 47
3.6.2	Separation Procedure 48
3.7	Experimental Results 49
3.7.1	Instance Generation and Experimental Setup 49
3.7.2	MILP Formulation Results: Base CPLEX versus Implemented Branch-and-Cut 50
3.7.3	ALNS Results 55
3.7.4	Tuning of ALNS Parameters 58
3.7.5	Sensitivity Analysis of System Efficiency 63
3.7.6	Sensitivity to Travel Delays and Path Length Variability 68
3.8	Environmental Impact Comparison: Crowdshipping versus Traditional Delivery Systems 72
3.9	Conclusions 74

Summary In this chapter, we introduce a novel last-mile delivery paradigm that employs commuters traveling on public transportation (PT) networks, to transfer parcels between parcel lockers installed in the stations. At the beginning of each delivery period, parcels are transported by a logistics service provider (LSP) to selected stations and assigned to commuters willing to act as crowdshippers. These commuters carry parcels along their regular journeys through the public transport network without deviating from their planned routes. A backup service ensures the final delivery of parcels that cannot be completed via crowdshipping. We call this problem the Public Transportation-based Crowdshipping Problem (PTCP), whose goal is defined as the task of determining the optimal schedule and routing of parcels, with the objective of minimizing total delivery costs.

We propose a compact Mixed-Integer Linear Programming (MILP) formulation, strengthened with valid inequalities, to represent the PTCP. To solve large-scale instances that cannot be handled efficiently by commercial MILP solvers, we design an Adaptive Large Neighborhood Search (ALNS) metaheuristic. Computational experiments conducted on an extensive set of problem instances confirm the effectiveness of the proposed heuristic compared with exact optimization. Furthermore, sensitivity analyses highlight the significant influence of crowdshipping and backup delivery costs on the overall efficiency of the system.

3.1 Introduction

In the past ten years, e-commerce and on-demand services have experienced significant growth. Industrial reports, like Chevalier (2024), foresee these services to grow further over the next few years. This rise has placed an ever-increasing strain on urban centers and has resulted in last-mile logistics becoming a significant contributor to urban issues such as pollution, noise, accidents, and congestion, as discussed by Demir et al. (2015); Savelsbergh and Van Woensel (2016); Manerba et al. (2018b). Moreover, last-mile delivery accounts for more than a quarter of the overall delivery cost, making it a major expense for logistics companies and retailers alike (Ranieri et al., 2018; Côté et al., 2024b). Therefore, innovative paradigms, that are both economically feasible and sustainable, to tackle last-mile delivery are in need (Mansini et al., 2024). In the last decades, several solutions have been proposed to address these issues (Boysen et al., 2021). Among these alternatives, many contributions focus on drones and autonomous robots as new technologies available for widespread delivery (Boysen et al., 2018; Gajda et al., 2024). In this chapter, we investigate a crowdshipping model based on public transportation (PT), a novel delivery paradigm in which regular commuters are involved in the delivery process. We focus on urban settings equipped with automated parcel lockers (APLs) strategically placed at selected stations of public transport systems, such as subways, buses, and trams. The proposed framework is designed for urban and suburban logistics networks with dense APL deployment and high commuter availability, making same-day delivery feasible. It is particularly suitable for e-commerce platforms, postal services, and logistics providers aiming to consolidate deliveries at APLs instead of distributing parcels to individual households.

PT commuters register on a digital platform, indicating their availability in terms of days, times, and travel routes. A matching system assigns parcels to available crowdshippers in accordance with their regular journeys. Each parcel is associated with both an origin station and one or more feasible destination stations. The logistics service provider (LSP), responsible for traditional home delivery, specifies the stations through which each parcel can enter the PT network, while the recipient identifies a set of preferred pick-up stations within a reasonable distance from home. Parcels are then transported by crowdshippers

along their routine commuting paths, either directly to the recipient's selected destination station or through intermediate transfer points such as exchange stations, where parcels may be passed between different commuters. Since crowdshippers cannot deviate from their planned routes, some deliveries require multiple transfers before the parcel reaches its final destination station. The PT-based crowdshipping framework therefore enables LSPs to avoid distribution within dense city centers by directing parcels to APLs at peripheral stations, reducing congestion and environmental impact in urban cores. The system is designed for small and medium-sized parcels, such as typical e-commerce goods weighting a few kilograms, while excluding perishable or fragile products. Parcels allocated to crowdshipping are pre-qualified by the LSP to comply with size, weight, and volume requirements, ensuring compatibility with the slots of standard APLs.

Each service day, early in the morning and before peak commuter hours, the LSP determines the optimal starting station for each parcel. Parcels are transported by truck and delivered to selected APLs within the PT network, ensuring they are ready for pick up when crowdshippers arrive. A pool of crowdshippers, with known travel routes and journey timings, is available to handle the delivery of each parcel between its source and destination station. Deliveries must be completed within the same operational day, aligning with urban logistics practices, without strict time windows. Once a parcel reaches its destination APL, the last-mile process is considered complete, as the destination APL serves as the customer's designated endpoint. By treating the destination APL as the terminal node in the delivery chain, we align with urban logistics trends where customers opt for self-service pick-up points, thereby reducing failed deliveries and improving flexibility.

In this chapter, we analyze a new problem to optimize parcel delivery through a PT network. We jointly optimize two main segments: from the LSP's central warehouse to the APL at the source station, and from there to the destination APL, intentionally decoupling it from the retrieval phase between the destination APL and the final customer (recipient). The problem looks for:

- (i) the selection of a starting station (*source station*) for each parcel. Each source is selected from a subset of stations identified by the LSP and will receive its parcels directly from the central warehouse by the LSP truck.
- (ii) the selection of the final station (*destination station*) for each parcel. Each destination is selected from a subset of stations specified by the customer following their preferences.
- (iii) the assignment of parcels to crowdshippers, together with their routing and scheduling across the PT network, subject to APL capacity constraints at each station. Since crowdshippers must follow their regular commuting paths without deviations, the formulation also accounts for the possibility that a parcel undergoes multiple transfers and temporary storage at intermediate stations before reaching its final destination.

Suitable parcels that cannot be included in daily planning due to capacity constraints or other limitations are delivered via a backup service arranged by the LSP. This ensures that all parcels reach their destination at the end of the day. The problem aims to minimize the total cost, which includes the remuneration of crowdshippers, the cost of transferring parcels from the warehouse to the source stations, the cost of parcel handling at the stations, and the backup cost for undelivered parcels.

From an economic standpoint, the advantages of such a delivery paradigm are evident. Crowdshippers receive a fixed travel credit in their customer accounts, which can be redeemed for discounted or subsidized travel with the LSP. On the other hand, through crowd-

shipping, LSPs can optimize delivery routes and vehicle capacity, thereby reducing operational costs such as fuel and labor. Additionally, this system supports an eco-friendly approach: recipients contribute by opting for crowdsipping rather than requesting door-to-door delivery.

In this chapter, we provide the following contributions:

- Although the issue of crowdsipping in urban PT networks is a known paradigm in the literature (Filippi and Plebani, 2021; Boysen et al., 2021; Wyrowski et al., 2024), We propose a new problem in relation to the state of the art. First, unlike other formulations, PTCP explicitly allows crowdshippers to transfer parcels to any subsequent station on their journey. This feature is integrated into a multi-step routing system, where a parcel can be delivered by more than one crowdshipper if necessary and economically feasible. Second, the problem evaluates the overall cost-effectiveness of the backup service with respect to the use of crowdshippers. In some cases, relying on the backup service may be preferable if transferring the parcel through multiple crowdshippers results in higher costs. Although none of these features is novel on its own, to the best of our knowledge, this is the first time they have been integrated into a single problem.
- We develop a mixed-integer linear programming (MILP) formulation for the PTCP. Our model minimizes overall costs by deciding the source and destination stations for each parcel, assigning parcels to crowdshippers, and routing them through the PT network while guaranteeing the less expensive option between using crowdshippers or the backup service.
- We strengthen the proposed mathematical formulation by introducing several valid inequalities. We compare their impact when all are included from scratch versus when they are dynamically separated during the solution process.
- We develop an Adaptive Large Neighborhood Search (ALNS) metaheuristic algorithm with two destroy and four repair operators that can efficiently handle large-scale instances derived from different PT networks and commuter traffic flows.
- We generate a comprehensive set of problem instances reflecting different PT system structures to assess the applicability of our model and the performance of our solution approaches across various scenarios.
- We provide relevant managerial insights for operators in the logistics sector. In particular, our sensitivity analysis results indicate that several factors are crucial for the success of a PT-based crowdsipping system. Specifically, these systems are highly influenced by cost structure, crowdshipper availability, and system scale. Effective implementation requires a careful balance among these factors, maintaining a large crowdshipper network while optimizing the backup delivery system to ensure reliability.
- Although the proposed model is deterministic, we analyze the impact of potential delays and disruptions on travel times, which may affect parcel pick-up schedules and render some delivery plans infeasible. Specifically, we perform a sensitivity analysis to assess how delays along a transit line influence the final solution.

3.2 Literature Review

Transportation research has extensively examined the critical role of last-mile delivery in the movement of goods. This final leg of the delivery process has traditionally relied on delivery vans, yet it remains the most inefficient and expensive component, accounting for over 40% of total supply chain costs in the United States (Joeress et al., 2016). The rapid growth of e-commerce, coupled with increasing customer expectations for faster and more flexible deliveries, has intensified pressure on last-mile logistics systems (Boysen et al., 2021). Contemporary challenges include high delivery costs, restricted delivery time windows, and a proliferation of delivery vehicles on urban roads, contributing to congestion, emissions, noise pollution, and increased accident risks (Department of Transportation, 2019).

The need for innovative last-mile delivery solutions has led researchers to rethink traditional paradigms, exploring approaches like crowdshipping, which leverages commuter mobility for parcel delivery. Recent studies propose a conceptual framework to integrate an urban parcel delivery system with public transport by using a crowdshipping system Zhang and Cheah (2025); Zingler and Vidyarthi (2025). In parallel, urbanization and sustainability challenges have driven interest in underground freight transportation (UFT) to reduce congestion and improve efficiency. Powell et al. (2024) propose UFT with electric autonomous vehicles in tunnels, modeling it as a fixed-charge multicommodity flow problem. Their study shows that a 45-mile network in Chicago could divert 42% of packages from roads and reduce costs by 40%. Zhang et al. (2022a) analyze 222 studies on underground logistics, focusing on metro-based networks. Their review identifies key research areas like system planning, vehicle design, and operations, while noting a gap between theory and practice, echoing Powell et al. (2024) on the need for government support.

Integrating freight movement with existing PT assets is another promising strategy. Freight on transit (FoT) systems aim to utilize the surplus capacity on buses, trams, or even metro vehicles to transport cargo. In the context of FoT, Elbert and Rentschler (2022) discuss various operational models and optimization techniques for sharing vehicles and infrastructure between passenger and freight services. This review describes the evolution of FoT models, from small pilot projects (e.g., off-peak metro deliveries) to more comprehensive mathematical frameworks that account for emissions, congestion, and stochastic demand. While mixed-integer programming is able to optimize shared vehicle routing, challenges like passenger resistance and scheduling conflicts require adaptive pricing models that can adjust to changing conditions. Further exploring this integration concept, Derse and Van Woensel (2024) present a conceptual framework for integrated people-and-freight transportation, combining multiple approaches at the strategic, tactical, and operational levels. This study highlights the socio-economic benefits of such a system, including reductions in urban traffic density by 18–24%, while identifying policy gaps and passenger dissatisfaction as key barriers to adoption. These findings underscore the need for data-driven public-private partnerships to address conflicting interests.

Modular vehicle routing studies have explored the consolidation of passenger and freight demand into a single vehicle platoon. In this context, Hatzenbühler et al. (2023) demonstrates how combining passenger and freight demand can result in 48% cost savings through the Modular Multipurpose Pick-up and Delivery Problem framework. A case study in Stockholm confirms a 60% reduction in empty vehicle kilometers, with demand consolidation contributing an additional 9% efficiency gain. Despite challenges like limited depot space and regulations, the study supports the idea that transit and logistics operators should collaborate, especially through adaptive pricing and infrastructure adjustments.

Within this evolving landscape of infrastructural and technological innovations, crowdshipping emerges as a promising paradigm that addresses many last-mile challenges through a fundamentally different approach. Unlike traditional solutions, crowdshipping, also known as crowd logistics, leverages the existing mobility patterns of individual commuters to facilitate parcel delivery. The operational framework of crowdshipping generally manifests mainly through two different models. The first approach involves occasional drivers who integrate parcel deliveries into their personal vehicle commutes, accepting minor detours to transport goods. These drivers utilize spare cargo space in private vehicles, adapting their routine travel patterns to additionally accommodate the delivery they are assigned. The second model, and the primary focus of this analysis, exploits the potential of PT systems. Here, commuters using trains, buses, or metros voluntarily participate as couriers, transporting parcels between stations based on their commute. Crowdshipping represents a shift toward more flexible, adaptive logistics systems that can potentially reduce both costs and environmental impacts by utilizing excess capacity in existing PT networks. This model aligns with the broader principles of the sharing economy while tackling key last-mile delivery challenges. As Carbone et al. (2017) highlight, crowdshipping presents notable advantages over traditional delivery models, particularly in urban density, delivery flexibility, and sustainability performance.

Research on occasional drivers includes several studies. In Archetti et al. (2016), the authors introduce the Vehicle Routing Problem with Occasional Drivers (VRPOD), which involves both regular vehicles and non-professional delivery couriers, and results in cost savings and efficiency improvements. In this work, occasional drivers are required to deviate from their usual routes to deliver packages in exchange for a small financial reward. The authors propose a multi-start matheuristic that integrates both metaheuristic and exact optimization strategies. The approach starts by generating diverse initial solutions using a sequential greedy insertion algorithm, alongside integer programming to assign customers to either regular or occasional drivers based on criteria such as distance, demand, and cost. An internal tabu search is then applied, using moves like 1-move, swap-move, in-move, and out-move, to iteratively improve the solution while avoiding cycling due to temporary tabu restrictions. The study explores various compensation methods for occasional drivers and demonstrates that this crowdsourced delivery system can lead to more cost-efficient solutions. More specifically, significant cost savings can be achieved when a large number of flexible occasional drivers are available for deliveries. Building on this, Arslan et al. (2019) extend the analysis by considering a dynamic version of the problem, where an online crowdsourcing platform continuously receives new delivery tasks and driver trip announcements over time. A rolling horizon framework is developed to reoptimize the system whenever new information becomes available. In addition, their heuristic accelerates the routing subproblem by pruning infeasible routes early and storing only a limited number of the most promising candidate routes, particularly under strict route duration constraints for backup vehicles. This approach reduces computational complexity without compromising solution quality. Results indicate that employing occasional drivers can significantly enhance cost-efficiency in last-mile delivery operations, potentially achieving system-wide vehicle-mile savings of up to 37% compared to conventional delivery systems relying solely on dedicated vehicles. Dayarian and Savelsbergh (2020) analyze a dynamic setting that accounts for uncertainty in both future demand (online orders) and delivery capacity (owned and crowd-sourced). A specific constraint imposes that the delivery of an order must occur within a fixed time window after the order is placed. The authors developed two variants (static and dynamic) of a fast algorithm capable of making real-time decisions based on incoming

demand and delivery capacity. In contrast, Di Puglia Pugliese et al. (2023) introduce the VR-POD with Time Windows (VRPODTW), considering uncertain travel times and penalties for missed deliveries. The problem is formulated using a chance-constrained stochastic model to develop a distribution plan that minimizes the total costs, including routing expenses for regular drivers, compensation for occasional drivers, and penalties for violating customer time windows. Additionally, the authors propose a robust model and apply two decomposition techniques: Benders decomposition and column-and-row generation, leveraging the problem's two-stage nature. In Archetti et al. (2021b), the online VRPOD is studied, where a company's fleet is supplemented by occasional drivers available within specific time windows. The research focuses on addressing dynamic customer requests, assuming a prior knowledge of crowdshippers' availability. The objective function aims to minimize overall distribution costs and incorporates a penalty function for time window violations. Unlike previous studies, such as Archetti et al. (2016) and Dayarian and Savelsbergh (2020), the occasional drivers can handle multiple deliveries during their assigned routes. Despite concerns about the negative impacts of delivery vans in urban areas, most research on crowdshipping systems has largely focused on operational challenges and cost efficiency. Although these studies have optimized logistics processes, they have not addressed the impact of additional vehicles and their detours on urban traffic, congestion, and pollution. Thus, a research gap remains, focusing on strategies that enhance cost-effectiveness while mitigating environmental concerns in crowdshipping systems.

Gatta et al. (2018) highlight the environmental and economic impacts, revealing potential reductions in particulate matter and greenhouse gas emissions through a PT-based crowdshipping system integrated with APLs. Results indicate significant environmental benefits, such as an average annual reduction of 239 kg in particulate matter emissions. Gatta et al. (2019b) identify key factors influencing crowdshipping, highlighting that in-station APLs are preferred over external ones, even more than monetary incentives. The authors also emphasize the importance of delivery scheduling flexibility for consumers and the reluctance of crowdshippers to significantly alter their commuting routes. Similarly, Filippi and Plebani (2021) analyze a metro-based crowdshipping service in Brescia, Italy, finding that young and retired metro users are willing to participate without monetary incentives. The authors assess the economic impact of combining metro-based crowdshipping with traditional delivery, using a MILP model. The results show that this integration could reduce delivery costs by up to 66% and vehicle miles traveled by 17% compared to standard van deliveries. Fessler et al. (2022) find that younger individuals, students, and employed people are most willing to carry parcels on PT in Copenhagen. Higher compensation increases participation, while larger parcels decrease willingness.

The concept of PT-based crowdshipping is closely linked to the successful implementation of APLs and their location (Gatta et al., 2019b). The adoption of APLs provides a convenient delivery method, with growing popularity worldwide (Zurel et al., 2018). Seghezzi et al. (2022) analyze the cost benefits of APLs. An analytical model compares the cost of home delivery and delivery through APLs. Results show that APLs can significantly reduce delivery costs by 60% averaging 1.2 € per parcel versus 2.9 € for home delivery. Vakulenko et al. (2018) highlight that the most important factor for customers is the location of APLs, preferring those close to their destination or on their way to work/home. Balancing this customer-centric approach with environmental concerns, Bonomi et al. (2022c) explore the strategic positioning of APLs. This study shows that while the placement of APLs is crucial, the eco-conscious behavior of consumers has the highest impact on reducing emissions. This outcome depends on two key parameters: the maximum distance individuals are willing to

travel to reach a locker and that they are willing to cover by foot or bicycle. Zurel et al. (2018) analyze implications of APLs on logistics and postal systems. The authors unveil APLs' cost efficiency and sustainability benefits by comparing the integration of these systems across different countries and focusing on regulatory considerations accompanying the widespread adoption of APLs.

Relevant research on the operational challenges of PT-based crowdsipping has emerged in recent years, and it is within this specific area of study that our project is situated. Within the context of Singapore's PT system, Zhang et al. (2022b) assess the viability and benefits of the PT-based crowdsipping delivery method. Focusing on the congested urban environment of Singapore, the authors assess the feasibility and impact of crowdsipping on e-commerce logistics. A parcel allocation problem is introduced to match parcels with suitable public bus journeys, but no mathematical formulation is provided. When an insufficient number of crowdshippers are available, traditional delivery vans are used to complete the deliveries. A simple heuristic algorithm is proposed to solve the problem. The findings reveal that crowdsipping can significantly reduce kilometers traveled and air emissions by up to 17%, while the delivery cost per parcel improves by up to 29%, even when limited to off-peak hours. Kızıl and Yıldız (2023) propose a slightly different variant of the problem discussed by Zhang et al. (2022b). The main idea is to place APLs in PT stations and other strategically located spots around the city, known as *satellites*. In this framework, crowdshippers are employed to transfer parcels from the satellites to the PT stations, while the transfers between PT stations are handled by the PT vehicles (i.e., trains) independently of the crowdshippers. If no crowdshipper is available, a backup transfer vehicle is used instead. The authors formulate a two-stage stochastic problem that accounts for uncertainties in demand and crowdshipper capacity. To efficiently solve realistic instances of the problem, they propose a Branch-and-Price algorithm enhanced with a new branching rule. This approach can achieve up to an 84.4% reduction in run time, with an average reduction of 61.5% across all problem instances, compared to the standard Branch-and-Price implementation. Different scenarios involving 10 to 80 crowdshippers and 149 network nodes are considered (39 PT connection locations and 110 satellites outside the PT system). The best algorithm can optimally solve the instance with 80 crowdshippers in 7,356 seconds (approx. 123 minutes). Mohri et al. (2025) propose a crowdsipping system where parcels are either allocated to crowdshippers, with an expected cost related to the probability for the job not being accepted by any commuter, or to a backup service. In this system a parcel assigned to crowdshippers is delivered by exactly one person, without any transshipments.

A more complete analysis of the PT-based crowdsipping paradigm is provided by Wyrowski et al. (2024). The authors examine a problem where metro users can transfer parcels between stations. They provide a mathematical formulation for the problem and independently test five different objective functions: the maximization of (i) the profit (difference between the income for the delivered parcels and the crowdshipper remuneration cost); (ii) the number of delivered parcels; and (iii) the crowdshipper involvement; the minimization of (iv) the total number of entrainments (joint travel of a crowdshipper and a parcel from one station to another) and (v) the maximum length of entrainment sequences. Unlike the present work, their framework does not consider the first leg of delivery, which involves transshipment of parcels from the warehouse to the starting stations, nor a backup system for parcels that remain unserved by the crowdshippers. We account for both such aspects dealing with the selection of a starting station for each parcel and the assignment of parcels to crowdshippers depending on the initial cost of the logistics service provider to deliver the parcel, as well as the backup cost. Additionally, in the formulation by Wyrowski et al. (2024),

a crowdshipper can only transfer a parcel from their current location to the immediate next location on their path. This simplification restricts the set of feasible paths for parcels. In our formulation, we explicitly consider the possibility of a crowdshipper delivering a parcel not only to the next station on their route but to any subsequent station being the final destination or any exchange station. This assumption leads us to introduce a discrete time setting instead of the continuous one used in (Wyrowski et al., 2024). Wyrowski et al. (2024) propose a decomposition heuristic to solve the problem, which involves generating numerous parcel paths and selecting the best subset using a MILP model. The authors recognize that their heuristic is generally outperformed by the pure solver approach and is only a viable option when the number of crowdshippers exceeds twice the number of parcels to be delivered. Although their work introduces a first attempt to tackle the complex PT-based crowdsipping problem by focusing on parcel movement through the PT stations, it does not fully address the integrated parcel delivery system. In Table 3.1, we summarize the main differences between the study in (Wyrowski et al., 2024) and the present work.

Table 3.1: Comparison between the study in Wyrowski et al. (2024) and the present one.

	(Wyrowski et al., 2024)	This Chapter
Backup cost		✓
Loading and delivery costs		✓
Multiple crowdshippers	✓	✓
Valid inequalities	✓	✓
Delivery time windows	✓	
Different objective functions	✓	
Selection of the starting station		✓
Selection of the destination station		✓
Explicit time synchronization		✓
Time management	Continuous	Discrete
Main objective function	Profit maximization	Cost minimization
Parcel transfer	Subsequent station	Any exchange station
Heuristic framework	MILP decomposition	ALNS
Maximal number of parcels	100	250
Maximal number of crowdshippers	300	300
Maximal number of stations	147	44

While recent literature on crowdsipping delivery systems has been expanding and addressing various issues, this work introduces a new MILP formulation for the global PT-based crowdsipping delivery problem. Our approach manages synchronization between crowdshippers coordinated to transport the same parcel, while minimizing the overall system cost.

In this work, we develop an efficient ALNS (Ropke and Pisinger, 2006) metaheuristic to deal with real-size instances of the problem. ALNS-based metaheuristics, as highlighted in a recent survey by Windras Mara et al. (2022), have proven to be very effective and adaptable for various combinatorial optimization problems, especially in vehicle routing. The survey highlights the algorithm’s ability to handle tightly constrained problems, shown by its successful use in several two-echelon VRPs (Grangier et al., 2016; Hemmelmayr et al., 2012; Jie et al., 2019) and large-scale crowdsourcing routing problems in the food delivery industry with large vehicle networks (Liu et al., 2019), among others.

3.3 Problem Definition and Notation

We define the parcel delivery system over a public transport (PT) network consisting of a set $L = \{1, \dots, l_{\max}\}$ of $l_{\max}$ transportation lines and a set $\bar{N}$ of *network stations* equipped with APLs with capacity C_i , representing the number of parcels that can be stored at station $i \in \bar{N}$. Each transportation line $l \in L$ serves a subset of network stations $\bar{N}_l \subseteq \bar{N}$. We denote as $\bar{S} \subset \bar{N}$ the set of *source stations*, indicating the subset of network stations where parcels can enter the network. Which of these stations are actually used as entry points depends on the specific delivery plan and the LSP's scheduled stops. Similarly, the subset $\bar{D} \subset \bar{N}$ defines the set of *destination stations*, where recipients can collect parcels, with the understanding that the final selection of these stations will depend on the choices made by the final recipients. We assume $\bar{S} \cap \bar{D} = \emptyset$. This is not a restrictive assumption: if a station is both a source and a destination, the parcel is directly delivered by the truck during its stop, making crowdsipping unnecessary. Some stations serve as *exchange stations*, enabling passengers and, consequently, parcels to switch between different transportation lines. A station $i \in \bar{N}$ is an exchange station if there exists a set $L_i \subseteq L$ such that $|L_i| \geq 2$ and $\cap_{l \in L_i} \bar{N}_l \neq \emptyset$. Note that the set of network stations $\bar{N} \setminus (\bar{S} \cup \bar{D})$ includes all exchange stations that are not part of $\bar{S}$ or $\bar{D}$, as well as additional network stations (not necessarily exchange stations) that serve as entry or exit points for crowdsippers. Each station $i \in \bar{S}$ has two associated cost coefficients: an activation cost v_i if the LSP selects the source station as entry point, and a cost σ_i proportional to the number of parcels loaded by the LSP at that station.

A set K of *crowdsippers* operates within the PT network. We assume that crowdsippers follow the shortest path (in terms of travel time) between their entry and exit stations without deviating from their planned route. The sequence of network stations traversed by a crowdsipper k , where parcels can be picked up or delivered, is denoted as $N_k \subset \bar{N}$ for each $k \in K$. A crowdsipper can transport parcels only if their route meets specific conditions: they must either pick up a parcel at their entry station, which must be a network station, or their route must pass through at least one exchange station. If the crowdsipper picks up the parcel at an exchange station, it must be delivered either to the crowdsipper exit station or another exchange station. We indicate as $K_{ij} \subseteq K$ the subset of crowdsippers who can transfer a parcel from station i to station j , being two stations of their planned route. Each crowdsipper $k \in K$ receives a fixed remuneration $\rho > 0$ for their service. To explicitly prevent professionalization, our system employs non-monetary and distance-independent incentives: crowdsippers earn rewards exclusively as public transit fare credits (e.g., discounted or free tickets), avoiding cash exchanges. This ensures that participation remains casual and aligns with existing mobility patterns of the commuters, rather than encouraging full-time roles.

To formalize the problem, we introduce three key graph representations that progressively refine the structure of the initial PT network:

- **Transfer Graph:** We first construct a *transfer graph* $\bar{G} = (\bar{N}, \bar{A})$, which models the real PT network by considering only the network stations equipped with APLs. We define as $\bar{A}_l = \{(i, j) | i, j \in \bar{N}_l\}$, the set of arcs representing all possible parcel transfers on the line l and $\bar{A} = \bigcup_{l \in L} \bar{A}_l$ all possible transfers on the PT network. Each arc $(i, j) \in \bar{A}$ corresponds to the shortest path between network stations i and j in the real network.

In Figure 3.1(a), we show an example of a small PT network consisting of three transportation lines (blue, red, and green) and 11 stations, where the nodes drawn as squares

represent network stations equipped with APLs, whereas nodes represented as circles are stations without lockers. Arcs with the same color and pattern belong to the same PT line. Note that when referring to a transportation line, we consider both directions of travel, which typically operate on two separate tracks. Stations 2 and 3 are the exchange stations where lines blue and red and lines blue and green intersect, respectively. The corresponding transfer graph (represented in Figure 3.1b) is obtained by excluding the stations without lockers and constructing all possible connections between the network stations belonging to each line.

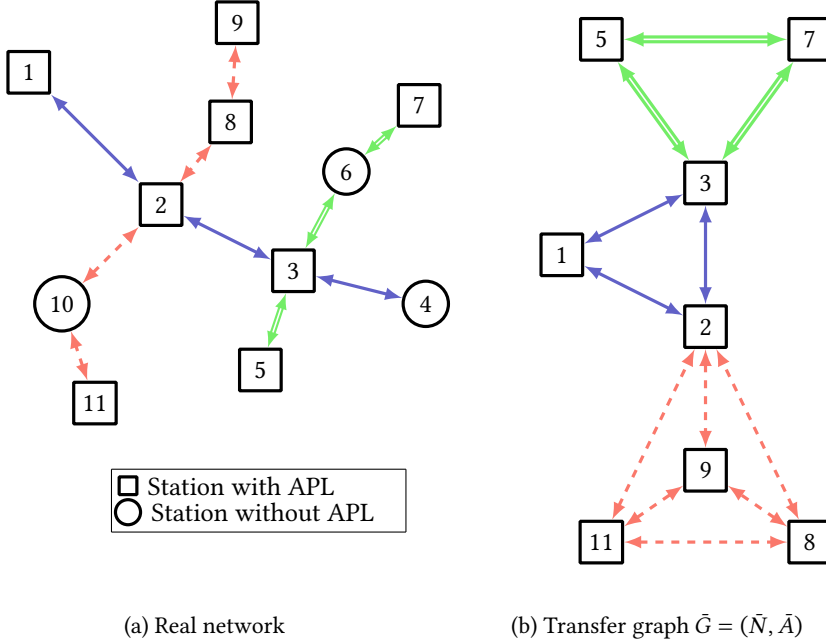


Figure 3.1: Example of a real network and its associated transfer graph.

- **Crowdsipper Delivery Graph:** Given the transfer graph, each crowdsipper $k \in K$ operates on a *delivery graph* $G_k = (N_k, A_k)$, where $N_k \subset \tilde{N}$ consists of the network stations that the crowdsipper can interact with, meaning the stations from which parcels can be picked up or delivered without deviating from the planned route. The set $A_k = \{(i, j) \in \tilde{A} \mid i, j \in N_k\}$ represents the set of feasible parcel transfers for that crowdsipper.

In the left part of Figure 3.2, we show the individual delivery graphs associated with three crowdsippers traveling on their regular journeys over the transfer graph reported in Figure 3.1(b). The routes of the three crowdsippers are: $\{9, 2, 3, 7\}$, $\{1, 3, 7\}$, and $\{11, 2, 1\}$, respectively. The arrows in each of Figures 3.2(a) to 3.2(c) show all the possible ways a crowdsipper can transfer parcels. For example, Figure 3.2(a) represents the delivery graph of the first crowdsipper. Since this crowdsipper follows the path $\{9, 2, 3, 7\}$, a parcel can be transferred from station 9 to stations 2, 3, or 7; from station 2 to stations 3 or 7; or from station 3 to station 7. Notice that the transfers do not include station 8, even though it is a network station. This is because the station 8 is neither an entry nor an exit station for that commuter, nor does it serve as an

exchange station. Similarly, Figures 3.2(b) and 3.2(c) represent the delivery graphs for the second and third crowdshippers, respectively.

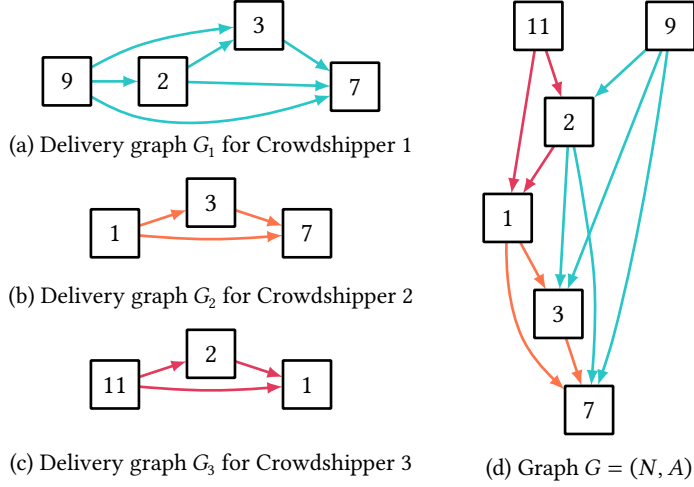


Figure 3.2: Union of crowdshipper delivery graphs G_1 , G_2 , and G_3 into final graph G .

- **Global Delivery Graph:** The final directed graph $G = (N, A)$ is the global delivery graph obtained as the union of the delivery graphs of all crowdshippers. The node set $N = \cup_{k \in K} N_k$ includes all stations where at least one crowdshipper can perform a transfer, and the arc set $A = \cup_{k \in K} A_k$ represents all feasible transfers in the system. For each arc $(i, j) \in A$ we indicate as t_{ij} the cumulative travel time across the shortest path between i and j in the transfer graph. Given $G = (N, A)$, we redefine the sets of the source and destination stations, $\tilde{S}$ and $\tilde{D}$, to include only the stations used by at least one crowdshipper: $S = N \cap \tilde{S}$, $D = N \cap \tilde{D}$, with $S \cap D = \emptyset$.

Figure 3.2(d) shows the global delivery graph obtained as union of the three crowdshippers delivery graphs reported in Figures 3.2(a)–3.2(c).

Finally, let P be the set of parcels to be delivered. One or more crowdshippers can transfer parcels. Each parcel $p \in P$ is associated with a set of feasible destination stations $D_p \subseteq D$ selected by the corresponding recipient and a backup cost γ_p , incurred if the parcel is handled by the LSP, which uses a vehicle to retrieve and transport parcels that the crowdsipping system does not deliver. Moreover, once a source station $i \in S$ for a parcel is selected, a fixed cost v_i is incurred, representing the transportation cost of the parcel from the depot to station i by the LSP vehicle, along with a unit cost σ_i for loading the parcel in the APL. To synchronize the parcel transfer between consecutive crowdshippers while complying with the capacity constraints for the APL at each network station, the delivery day is divided into equally sized time slots. Let T represent the set of available time slots. We assume that each parcel can be picked up from or dropped off at a network station no more than once per time slot. This time discretization allows for accurately calculating the total capacity used at each station during a given time slot and ensures that a parcel is handled by at most one crowdshipper per time slot. To simplify the problem formulation, we assume that the time slot at which each crowdshipper starts from their initial station is known in advance, meaning we are aware of when each crowdshipper begins their journey. We also assume that

the average travel time between any station and the next one is known. These assumptions make the problem fully deterministic and allow us to handle time within the model without relying on dedicated time variables. To this end, we denote by K_i^t the set of crowdshippers present at network station $i \in N$ at time slot $t \in T$.

The PTCP aims to find the optimal assignment of parcels to crowdshippers and their optimal route through the network, while determining both the source and destination station for each parcel and complying with capacity constraints of APL at network stations. The problem is framed from the perspective of the LSP, seeking to minimize the total costs. These costs include the remuneration paid to crowdshippers, the backup costs for delivering parcels to their destinations using the LSP vehicles, and the costs for loading parcels at the source and exchange stations.

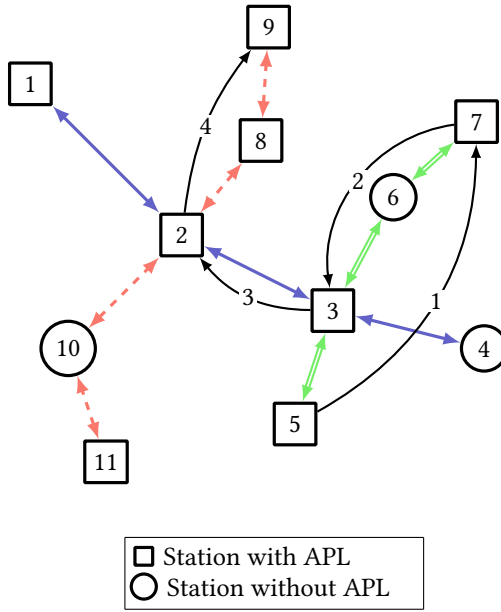


Figure 3.3: Example of a parcel transfer with crowdshippers' flow indicated by a black arrow and their id.

It is worth noting that the backup service is not only employed when no crowdshippers are available to complete a parcel transfer, but also when relying on a large sequence of crowdshippers would result in excessive costs, making vehicle-based transportation a more convenient alternative. As an illustrative example, consider the parcel transfer depicted in Figure 3.3, where the parcel starts at source station 5 and is delivered to destination station 9 via a route involving sequentially four distinct crowdshippers. Crowdshipper 1 travels directly from station 5 to 7 without line changes; crowdshipper 2 takes the parcel from station 7 to exchange station 3; crowdshipper 3 travels from station 3 to station 2 leaving the parcel there, whereas crowdshipper 4 enters station 2, retrieves the parcel, and leaves it at destination station 9. Assuming that the remuneration for each crowdshipper is $\rho = 2$ and that the backup cost for the parcel is $\gamma_p = 6$, using the backup service becomes more cost-effective than relying on crowdshippers. Moreover, this simplified comparison considers only the remuneration of crowdshippers, neglecting additional costs such as the potential activation cost of station 5 and the expense of loading the parcel into the APL.

The sets and parameters required for the formal definition of the problem are summarized in Table 3.2.

Table 3.2: Sets and Parameter

Sets	
P	Set of parcels to be delivered
K	Set of available crowdshippers
N	Set of network stations visited by at least one crowdshipper
A	Set of arcs connecting nodes in N
S	Set of source stations
D_p	Set of possible destination stations for parcel $p \in P$ provided by the recipient
D	Set of possible destination stations for all parcels, $D = \cup_{p \in P} D_p$
A_k	Set of arcs in A traversed by crowdshipper $k \in K$
T	Set of discrete time slots
T'	Set of discrete time slots without the last one
K_i^t	Set of crowdshippers visiting stations $i \in N$ at time slot $t \in T$
K_{ij}	Set of crowdshippers moving from station $i \in N$ to station $j \in N$ at any time
Parameters	
ρ	Reward granted by the LSP to crowdshippers
v_i	Fixed cost to send a delivery vehicle to source station $i \in S$
σ_i	Unitary cost to load a parcel into source station $i \in S$
γ_p	Backup cost for parcel $p \in P$
C_i	Maximum capacity of the APL located at station $i \in N$

3.4 Mathematical Formulation

In this section, we introduce a new compact Mixed-Integer Linear Programming (MILP) formulation for the PTCP. The sets of decision variables required to formulate the model are defined in Table 3.3.

Table 3.3: Decision Variables

x_{ij}^{kp}	Binary variable taking value 1 if crowdshipper $k \in K$ transfers parcel $p \in P$ from station $i \in N$ to station $j \in N$; 0 otherwise.
z_k	Binary variable taking value 1 if crowdshipper $k \in K$ is selected; 0 otherwise.
y_{kp}	Binary variable taking value 1 if parcel $p \in P$ is assigned to crowdshipper $k \in K$; 0 otherwise.
o_{ipt}	Binary variable taking value 1 if parcel $p \in P$ is at station $i \in N$ at time slot $t \in T$; 0 otherwise.
a_{ipt}	Binary variable taking value 1 if parcel $p \in P$ arrives at station $i \in N$ at time slot $t \in T$; 0 otherwise.
l_{ipt}	Binary variable taking value 1 if parcel $p \in P$ leaves station $i \in N$ at time slot $t \in T$; 0 otherwise.
s_{ip}	Binary variable taking value 1 if parcel $p \in P$ starts its route at source station $i \in S$; 0 otherwise.
d_{ip}	Binary variable taking value 1 if parcel $p \in P$ terminates its route at destination station $i \in D$; 0 otherwise.
e_i	Binary variable taking value 1 if source station $i \in S$ is selected; 0 otherwise.
q_p	Binary variable taking value 1 if parcel $p \in P$ is delivered by the backup service; 0 otherwise.

The MILP for the PTCP is stated as follows:

$$\min \rho \sum_{k \in K} z_k + \sum_{i \in S} v_i e_i + \sum_{i \in S} \sigma_i \sum_{p \in P} s_{ip} + \sum_{p \in P} \gamma_p q_p \quad (3.1)$$

s. to:

$$\sum_{i \in S} s_{ip} = 1 - q_p \quad p \in P \quad (3.2)$$

$$\sum_{i \in D_p} d_{ip} = 1 - q_p \quad p \in P \quad (3.3)$$

$$\sum_{p \in P} s_{ip} \leq C_i e_i \quad i \in S \quad (3.4)$$

$$\sum_{j \in N} \sum_{k \in K_{ij}} x_{ij}^{kp} - \sum_{j \in N} \sum_{k \in K_{ji}} x_{ji}^{kp} = s_{ip} - d_{ip} \quad p \in P, i \in N \quad (3.5)$$

$$x_{ij}^{kp} \leq y_{kp} \quad p \in P, k \in K, (i, j) \in A_k \quad (3.6)$$

$$\sum_{p \in P} y_{kp} \leq z_k \quad k \in K \quad (3.7)$$

$$l_{ipt} = \sum_{j \in N} \sum_{k \in K_i^t} x_{ij}^{kp} \quad p \in P, i \in N, t \in T \quad (3.8)$$

$$a_{ipt} = \sum_{j \in N} \sum_{k \in K_i^t} x_{ji}^{kp} \quad p \in P, i \in N, t \in T \quad (3.9)$$

$$o_{ip(t+1)} \geq a_{ipt} \quad i \in N, p \in P, t \in T' \quad (3.10)$$

$$o_{ip(t+1)} \geq o_{ipt} - l_{ipt} \quad i \in N, p \in P, t \in T' \quad (3.11)$$

$$o_{ip(t+1)} \leq 1 - l_{ipt} \quad i \in N, p \in P, t \in T' \quad (3.12)$$

$$\sum_{i \in N} o_{ipt} \leq 1 - q_p \quad p \in P, t \in T \quad (3.13)$$

$$\sum_{p \in P} o_{ipt} + \sum_{p \in P} a_{ipt} \leq C_i \quad i \in N, t \in T \quad (3.14)$$

$$o_{ip1} = s_{ip} \quad p \in P, i \in N \quad (3.15)$$

$$x_{ij}^{kp} \in \{0, 1\} \quad (i, j) \in A_k, k \in K, p \in P \quad (3.16)$$

$$z_k \in \{0, 1\} \quad k \in K \quad (3.17)$$

$$a_{ipt}, l_{ipt}, o_{ipt} \in \{0, 1\} \quad i \in N, p \in P, t \in T \quad (3.18)$$

$$s_{ip} \in \{0, 1\} \quad i \in S, p \in P \quad (3.19)$$

$$d_{ip} \in \{0, 1\} \quad i \in D_p, p \in P \quad (3.20)$$

$$y_{kp} \in \{0, 1\} \quad k \in K, p \in P \quad (3.21)$$

$$q_p \in \{0, 1\} \quad p \in P \quad (3.22)$$

$$e_i \in \{0, 1\} \quad i \in S \quad (3.23)$$

The objective function (3.1) minimizes the overall cost that comprises the remuneration for selected crowdshippers, the cost to send delivery vehicles to source stations, the cost for the loading of the parcels, and the backup cost for the parcels undelivered through crowdshipping. Constraints (3.2) and (3.3) ensure that, if the LSP does not deliver parcel $p \in P$ through the backup service (i.e., $q_p = 0$), the parcel must have exactly one source and one destination stations. Constraints (3.4) set variable e_i equal to 1 if station $i \in S$ is used as a source station by at least one parcel. Constraints (3.5) deal with three distinct cases:

- $s_{ip} - d_{ip} = 1$. In this case, station i is the source station for parcel p ($s_{ip} = 1$ and $d_{ip} = 0$). The constraints impose that parcel p must leave station i without entering it.
- $s_{ip} - d_{ip} = -1$. In this case, station i is the destination station for parcel p ($s_{ip} = 0$ and $d_{ip} = 1$). The constraints impose that parcel p must enter i without leaving it.
- $s_{ip} - d_{ip} = 0$. This configuration holds when $s_{ip} = d_{ip} = 0$, so station i is an exchange station for parcel p . In this case, the constraints impose flow conservation.

According to constraints (3.6), crowdshipper $k \in K$ can transfer parcel $p \in P$ from station i to j only if parcel p has been assigned to them ($y_{kp} = 1$). Constraints (3.7) guarantee that each crowdshipper $k \in K$ (if selected) can deliver at most one parcel. Constraints (3.8) set variable l_{ipt} to 1 if parcel p leaves station i at time t . Since variables l_{ipt} are binary, these constraints prohibit a parcel from leaving a station more than once. Similarly, constraints (3.9) set variable a_{ipt} to 1 if parcel p arrives at station i at time t and exclude the parcel from entering station i more than once at time t . Constraints (3.10) force variable $o_{ip(t+1)}$ to 1 if variable $a_{ipt} = 1$, thus specifying that parcel p is located at station i at time $t + 1$, if the parcel arrived at this station at time t . Constraints (3.11) force the location of parcel p at station i at time $t + 1$ if the parcel has not been moved from station i at time t ($l_{ipt} = 0$) and it was already located there at time t ($o_{ipt} = 1$). Constraints (3.12) force variable $o_{ip(t+1)}$ to 0 if parcel p leaves station i at time t , otherwise the constraint is not binding. Notice that

constraints (3.10) and (3.12) imply that binary variables a_{ipt} and l_{ipt} are mutually exclusive. If parcel p is managed by crowdshippers (i.e., $q_p = 0$), constraints (3.13) impose that it can be located in at most one station for each time slot. Otherwise, the LSP will take care of the delivery and the parcel will not travel through the network $\sum_{i \in N} o_{ipt} = 0$. Constraints (3.14) are capacity constraints, ensuring that the number of parcels located at station i plus the number of parcels arriving at the same station at time t has to be lower than or equal to the station capacity. Constraints (3.15) set the location of parcel p at time 1 at its source station i when $s_{ip} = 1$. Finally, constraints (3.16 - 3.23) are binary and non-negativity conditions on variables.

3.4.1 Valid Inequalities

Our formulation (3.1)–(3.23) involves a large set of variables and constraints driven by the number of network stations, parcels, crowdshippers, and the selected time slots discretization. While complete and correct, we introduce the following valid inequality classes to strengthen the relationships among the variables, a common technique to improve the model's structure and optimization performance.

$$\sum_{p \in P} s_{ip} \geq e_i \quad i \in N \quad (3.24)$$

$$\sum_{i \in D} d_{ip} \leq \sum_{k \in K} y_{kp} \quad p \in P \quad (3.25)$$

$$\sum_{i \in S} s_{ip} \leq \sum_{k \in K} y_{kp} \quad p \in P \quad (3.26)$$

$$\sum_{i \in N} \sum_{t \in T} l_{ipt} \leq \sum_{k \in K} y_{kp} \quad p \in P \quad (3.27)$$

$$\sum_{i \in N} \sum_{t \in T} a_{ipt} \leq \sum_{k \in K} y_{kp} \quad p \in P \quad (3.28)$$

$$y_{kp} \leq 1 - q_p \quad p \in P, k \in K \quad (3.29)$$

$$s_{ip} + d_{ip} \leq 1 \quad i \in N, p \in P \quad (3.30)$$

$$\sum_{i \in N} o_{ipt} \leq \sum_{i \in S} s_{ip} \quad p \in P, t \in T \quad (3.31)$$

$$\sum_{i \in N} o_{ipt} \leq \sum_{i \in D} d_{ip} \quad p \in P, t \in T. \quad (3.32)$$

Inequalities (3.24) impose that if the source station $i \in S$ is selected ($e_i = 1$), at least one parcel must start its path at station i . Inequalities (3.25) and (3.26) establish that neither a destination station nor a source station can be selected for parcel $p \in P$ if the parcel has not been assigned to a crowdshipper. Similarly, inequalities (3.27) and (3.28) allow a parcel p to leave (resp. arrive) at a station only if a crowdshipper handles this parcel. Each inequality (3.29) ensures that parcel $p \in P$ cannot be assigned to crowdshipper $k \in K$ if the parcel has already been allocated to the backup service (i.e., $q_p = 1$). Inequalities (3.30) state that, for a given parcel p , since a station i cannot be jointly its source and destination stations, variables s_{ip} and d_{ip} cannot be both equal to 1. Finally, valid inequalities (3.31) and (3.32) ensure that the location variables o_{ipt} for parcel $p \in P$ at time slot $t \in T$ can only be active at a station $i \in N$ if both a starting and a destination stations have been assigned to this parcel.

3.5 An ALNS Algorithm for the PTCP

This section we first provide a general introduction to the Adaptive Large Neighborhood Search (ALNS) metaheuristic, then we introduce the proposed ALNS to solve the PTCP.

3.5.1 Introduction to ALNS

First introduced in Ropke and Pisinger (2006) and Pisinger and Ropke (2007), the Adaptive Large Neighborhood Search (ALNS) metaheuristic in recent years has become one of the most commonly used metaheuristic algorithms to solve complex combinatorial problems, thanks to its adaptability and ease of implementation. An ALNS algorithm starts from an initial feasible solution, then it iteratively modifies the current solution by applying a destroy and repair operators to it. The destroy operator modifies the current solution by rendering it either infeasible or of worst quality. On the other hand a repair operator either restores the feasibility of the current solution or tries to improve the value of the current solution. An ALNS algorithm makes use of multiple destroy and repair operators. At each iteration, random destroy and repair operators are selected, usually with a roulette wheel selection. Given Ω^- and Ω^+ the set of destroy and repair operators respectively, this selection method creates $|\Omega^-| \times |\Omega^+|$ local search algorithms, each one with its own distinct neighborhood of the current solution. The different operators contribute to expand the effective size of the neighborhood, without the need to design and implement complex local search heuristics.

Algorithm 3.1 Adaptive Large Neighborhood Search

```

1: Construct a feasible solution  $x$ 
2:  $x^* \leftarrow x$ 
3:  $t \leftarrow$  initial temperature
4: while  $t \geq t_{min}$  do
5:    $d \leftarrow$  random destroy operator from  $\Omega^-$  using scores  $\pi_i^-$ 
6:    $r \leftarrow$  random repair operator from  $\Omega^+$  using scores  $\pi_i^+$ 
7:   Generate new solution  $x' \leftarrow r(d(x))$  applying destroy and repair operators
8:   if Accept  $x'$  as current solution ? then
9:      $x \leftarrow x'$ 
10:  end if
11:  if  $f(x') < f(x^*)$  then ▷ Assume a minimization problem.
12:     $x^* \leftarrow x'$ 
13:  end if
14:  Update score  $\pi_{(r)}^+$  and  $\pi_{(d)}^-$  of the selected operators
15:   $t \leftarrow cr * t$ 
16: end while
17: return  $x^*$ 

```

In Algorithm 3.1⁰ we present the general outline of the method. First, an initial solution is constructed, usually with a constructor operator, and it is saved as the current incumbent solution x^* . In the version proposed by Ropke and Pisinger (2006), the ALNS is controlled with a Simulated Annealing. The temperature is used to determine the termination criteria, as shown in Line 4, and as an acceptance criterion for worsening solutions. However, it is possible to embed the ALNS inside other metaheuristics like a Tabu search. During each iteration, a random destroy and a random construct operators are selected. These operators are

applied to the current solution x to generate a new solution x' . Then the algorithm decides to accept or reject the new solution. A common acceptance criteria is to always accept an improving solution, and to accept a worsening solution with a probability dependent on the current temperature. At this point the algorithm checks if the new solution is better than the incumbent, and in that case updates it. After checking the solution value the algorithm updates the scores for the selected operators according to the obtained improvements. At the end of each iteration the algorithm proceeds with the temperature schedule.

We chose ALNS as heuristic framework thanks to its ease of implementation and its ability to handle diverse destroy and repair operators. This makes it well-suited to our problem, despite differences from (Ropke and Pisinger, 2006) (e.g., APL capacities, time-dependent crowdshipper availability). ALNS effectively explores large solution spaces, balancing intensification and diversification to address the trade-offs in PTCP.

3.5.2 The proposed ALNS Algorithm

The ALNS framework iteratively applies destroy and repair operators to refine solutions. Destroy operators remove parcel assignments, while repair operators reassign them more effectively. An adaptive scoring mechanism prioritizes effective operators. This process is embedded in an Iterated Simulated Annealing (ISA) framework, which regulates solution acceptance via a temperature-based criterion, aiding in escaping local optima while balancing exploration and exploitation. The stopping condition is the maximum computation time.

In the following, we introduce the concept of *parcel path* to denote the sequence of steps a parcel takes from its source station to its destination, with its length indicating the number of crowdshippers involved. Shorter paths are more cost-effective. Our ALNS prioritizes crowdshipper-based paths over backup services, when they incur higher costs due to additional personnel and vehicles. This principle guides the development of our destroy and repair operators, which aim to minimize the use of backup services while efficiently using available crowdshippers.

Algorithm 3.2 outlines our ALNS within an ISA framework, initializing a baseline starting solution and assigning an initial score of 1 to each operator. The outer loop runs until the time limit is reached, initializing operator scores before executing the simulated annealing temperature schedule. In each iteration of the inner loop, a destroy and a repair operator are selected via roulette wheel selection (Line 6). The selected destroy operator removes a predetermined number of parcel assignments (Line 7), reducing node capacities to prevent regenerating the same solution (Line 8). The selected repair operator then reinserts as many parcels as possible while minimizing costs (Line 9). Finally, each node's capacity is reset to its original value. In Lines 11–23, the algorithm decides whether to accept the new solution. Lower-cost solutions are always accepted, while worse solutions are accepted probabilistically based on the current temperature. If the new solution improves the incumbent, the latter is updated and a reheating policy is applied. The new solution is compared to the current solution based on cost difference, and a temperature-dependent criterion is used to determine if the new solution should be accepted. The temperature is systematically reduced by the so called cooling rate factor cr (Line 24). Finally, in Lines 25–30, operator scores are reset to their initial value after each *Limit* iterations.

During iterations, operator scores are updated based on solution quality (Lines 13–20):

- New best solution: Operators are rewarded by η_{best} when finding a new globally improved solution.

Algorithm 3.2 ALNS within ISA Framework

```

1: Initialize solution  $sol$ 
2: while stopping criteria not met do
3:   operator scores  $\omega_d, \omega_r \leftarrow 1$ , temperature  $temp \leftarrow T_0$ 
4:   iterations  $\leftarrow 0$ 
5:   while  $temp > t_{\min}$  do
6:     Get destroy and repair operators  $(d, r)$  via roulette wheel method on  $\omega_d, \omega_r$ 
7:      $sol' \leftarrow$  apply  $d$  to remove  $q$  parcels from  $sol$  ▷ Destroy phase
8:     Reduce node capacity according to the destroy operator
9:     ▷ Avoid to recreate the previous solution
10:     $sol' \leftarrow$  apply  $r$  to reinsert removed parcels into  $sol'$  ▷ Repair phase
11:    Reset node capacity
12:    Compute cost difference  $\delta = \text{cost}(sol') - \text{cost}(sol)$ 
13:    if  $\delta < 0$  or  $U(0, 1) < \exp(-\frac{\delta}{temp})$  then
14:      ▷ Acceptance rule, with  $U(0, 1)$  Uniform random in  $[0, 1]$ .
15:       $sol \leftarrow sol'$ 
16:      Update scores:
17:      if  $S'$  is a new best solution then
18:         $\omega_d \leftarrow \omega_d + \eta_{\text{best}}, \omega_r \leftarrow \omega_r + \eta_{\text{best}}$  ▷ Reward for new best solution
19:        Reheat:  $temp \leftarrow temp + \Delta t$  ▷  $\Delta t = (\bar{z} - z^*) \exp(-\frac{temp}{T_0})$ 
20:      else if  $S'$  is better than current solution then
21:         $\omega_d \leftarrow \omega_d + \eta_{\text{incumbent}}, \omega_r \leftarrow \omega_r + \eta_{\text{incumbent}}$ 
22:        ▷ Reward for new incumbent
23:      else
24:         $\omega_d \leftarrow \omega_d + \eta_{\text{accepted}}, \omega_r \leftarrow \omega_r + \eta_{\text{accepted}}$ 
25:        ▷ Reward for accepted solution
26:      end if
27:    end if
28:    Cool temperature:  $temp \leftarrow cr * temp$  ( $\alpha \in (0, 1)$ )
29:    if iterations = Limit then
30:      ▷ Reset operator scores after a certain number of iterations
31:      operator scores  $\omega_d, \omega_r \leftarrow 1$ ,
32:      iterations  $\leftarrow 0$ 
33:    else
34:      iterations  $\leftarrow$  iterations + 1
35:    end if
36:  end while
37: end while
38: return best solution found

```

- New incumbent solution: Operators are increased by $\eta_{\text{incumbent}}$ when finding a solution better than the current one but not the best.
- Accepted solution: Operators are rewarded by η_{accepted} even if they do not provide any solution improvement.

In the remainder of this section, we describe the destroy and repair operators tailored to PTCP, highlighting their role in addressing the problem's specific characteristics. Then,

we present the constructive initial heuristic to generate the starting solution, ensuring a high-quality baseline for ALNS.

Destroy Heuristics

The destroy heuristics remove existing parcel routes and allocate them to the backup set. Let $H \in \mathbb{N}^{|N| \times |T|}$ be the load matrix where h_{it} denotes parcels at station i at time slot t . Each of the proposed destroy operators influence station prioritization through four metrics:

- **Absolute Load:** sort stations according to the value $\alpha_i = \max_{t \in T} h_{it}$, targeting peak congestion;
- **Relative Load:** sort stations according to the value $\rho_i = \alpha_i / C_i$, identifying peak utilization rate;
- **Average Load:** sort stations according to the value $\mu_i = \sum_{t \in T} h_{it} / |T|$, identifying sustained usage;
- **Random:** sort stations according to the value $v_i = U(0, 1)$, enabling diversification.

The chosen metrics help identify stations under high stress or congestion. Specifically, the *Absolute Load* metric targets stations with peak demand, the *Relative Load* metric highlights stations operating near capacity, and the *Average Load* metric reflects sustained high usage over time. These metrics directly inform the destroy operators by prioritizing the removal of parcel paths from the most burdened stations. Destroy operators (D1–D2) apply these metrics to select stations for parcel removal.

(D1) Parcel Paths Elimination. This heuristic targets parcel paths that originate from a selected station, thereby encouraging the formation of completely new paths. The procedure is as follows:

- (a) **Station Selection:** Select the top-ranked station using one of the four metrics.
- (b) **Path Sorting:** Select the longest the parcel paths originating from the selected station.
- (c) **Determination of Paths to Remove:** Let $\bar{\pi}$ denote the number of parcel paths from the selected station. Define $\pi^* = \min(\bar{\pi}, \pi)$, where π is the pre-specified number of paths to eliminate.
- (d) **Path Removal and Update:** Remove the π^* longest parcel paths and assign them to a backup service. Update the load matrix H accordingly.

(D2) Capacity Reduction. This procedure reduces the effective capacity of a selected subset of stations, forcing the reassignment of parcel paths that no longer comply with the updated capacity constraints:

- (a) **Station Selection:** Select the highest scoring stations using one of the four metrics and select the top π stations.
- (b) **Capacity Adjustment:** For each selected station i , update its capacity by setting $\tilde{C}_i = \max(\alpha_i - \tilde{\pi}, 0)$, where α_i is the original capacity of station i and $\tilde{\pi}$ is the specified reduction amount.
- (c) **Path Sorting and Removal:** For each selected station, rank the incident parcel paths by length. Iteratively remove the longest paths until the load at station i , for every time slot t , satisfies $h_{it} \leq \tilde{C}_i$.

Selection and ranking are all done using dedicated priority queues. This implementation choice allows the algorithm to identify the top ranking objects (i.e. the longest paths, the highest scoring stations,...) without the need to sort the entire list of objects. Such an optimization is particularly useful when the cardinality of the objects list is significantly larger than the number of items to select. All priority queues are cached, so the computational cost to allocate memory and initialize the data structure is paid only once during the overall algorithm initialization.

Repair Heuristics

The repair heuristics aim to deliver backup set parcels through crowdsipping by constructing feasible delivery paths using a label-setting algorithm. For each undelivered parcel in the backup set, the method aims at (i) assigning a reasonable source station and (ii) computing a feasible path from that source station to the destination, considering the availability of crowdshippers. Two distinct cost functions are used in the labeling process: one that minimizes the number of crowdshippers employed, and another that penalizes overused crowdshippers to promote assignment diversification. Regardless of the specific variant, if a feasible path is found, the parcel is marked as delivered; otherwise, it remains in the backup set.

The proposed label-setting algorithm, presented in Algorithm 3.3, is an extension of Dijkstra's algorithm, and associates with each station j a label given by the tuple (c_j, t_j, k, i) , where c_j is the cumulative cost to reach station j ; t_j is the arrival time at station j of crowdshipper k who traveled from station i ; k identifies the crowdshipper; i is the predecessor station in the path. At the source station, the label is initialized as $(0, 0, \emptyset, \emptyset)$. Each station is assigned at most one label corresponding to the lowest cost (or best) path found so far. When processing a station i with label $(c_i, t_i, \cdot, \cdot)$, the algorithm considers each feasible move to a neighboring station j . For every crowdshipper k not yet employed in the current path and able to depart after time t_i , a new label is computed as $c_j = c_i + f_k(i, j)$ and $t_j = t_i^k + t_{ij}$, where: t_i^k is the arrival time of crowdshipper k at station i ; t_{ij} is the travel time from station i to station j , and $f_k(i, j)$ is a cost function associated with the arc (i, j) for crowdshipper k . A new label replaces the current one at station j if it is the first label or if it has a lower cost than the one previously recorded. The algorithm terminates when a label for a destination station is fixed.

We consider two alternative cost functions:

- (i) $f_k(i, j) = 1$ for all $k \in K$ and $(i, j) \in A$, which minimizes the number of crowdshippers used;
- (ii) A cost function that counts the number of times crowdshippers were employed in previous solutions. This second function encourages diversification.

In all cases, the cost of the final solution is evaluated according to Eq. (3.1).

The repair procedure is applied sequentially to all undelivered parcels that are in the backup set. The order in which parcels are processed follows one of the following strategies:

- *As-Is*: Process parcels in their original order.
- *Random*: Process parcels in a randomly shuffled order.
- *Max Backup*: Process parcels in descending order of their backup costs, thus prioritizing those that are most expensive to handle via the backup service.

Algorithm 3.3 Parcel path finding**Require:** *origin*: path source, *Finals*: parcel possible destinations

```

1: Initialize origin station cost to 0, and all other stations to  $+\infty$ 
2: Initialize the station queue q sorted by station's cost
3: while q is not empty do ▷ Set all labels
4:   curr  $\leftarrow pop(q)$ 
5:   successors  $\leftarrow$  free crowdshippers leaving curr after time[curr]
6:   for next  $\in$  successors do
7:     if cost[curr] + cost[next] < cost[dst[next]] then
8:       ▷ Set label if it cheaper to move parcel in that way
9:       cost  $\leftarrow cost[curr] + cost[next]$ 
10:      time  $\leftarrow time[curr] + time[next]$ 
11:      label  $\leftarrow (cost, time, crs[next], position[curr])$ 
12:      Set label for node dst[next]
13:      Push label into q
14:     end if
15:   end for
16: end while
17: f  $\leftarrow \emptyset$ , cost  $\leftarrow +\infty$ 
18: for final  $\in$  Finals do ▷ Look for the cheapest destination station
19:   if cost[final] < cost then
20:     cost  $\leftarrow cost[final]$ 
21:     f  $\leftarrow final$ 
22:   end if
23: end for
24: path  $\leftarrow []$ 
25: curr  $\leftarrow f$ 
26: while curr  $\neq origin$  do ▷ Rebuild the path
27:   prev  $\leftarrow prev[curr]$ 
28:   Push curr into path
29:   curr  $\leftarrow prev$ 
30: end while
31: Push origin into path
32: Reverse path
33: return path, cost

```

- *Min Backup*: Process parcels in ascending order of their backup costs, thereby prioritizing parcels that require minimal backup adjustments and can yield early, incremental improvements.

These various processing orders allow the heuristic to target different aspects of solution quality, ranging from aggressively reducing high backup costs to quickly capturing low-hanging improvements, while the baseline (*As-Is*) and randomized orders help ensure diversification. Based on the above, four repair heuristic operators are proposed:

(R.1) Parcel Distribution + Shortest Path

- (a) Station Assignment: An undelivered parcel in the backup set is assigned to a source station using function $p_i(n)$ that represents the cost of loading $n+1$ parcels

at station i . A priority queue is initialized with all stations that are not full, sorted by $p_i(n)$ value. For each parcel, the station with the lowest loading cost is selected; if the station is not yet full after the assignment, it is reinserted into the queue. If no station is left, the process stops.

- (b) Path Computation: Using the label-setting algorithm, a path is computed for each parcel from the assigned source station to its destination to minimize the number of crowdshippers employed.

(R2) Delivery Cost Minimization + Shortest Path

- (a) Station Assignment: stations are first sorted in ascending order of their base loading cost $p_i(0)$. Undelivered parcels are then assigned to source stations by matching each parcel with the nearest feasible destination from the current station until the station reaches its capacity. This process is repeated with the next station.
- (b) Path Computation: The labeling algorithm is used to compute each parcel's shortest path (in terms of the number of crowdshippers).

(R3) Parcel Distribution + Uniform Crowdshipper Load

- (a) Station Assignment: As in variant R1.
- (b) Path Computation with Load Balancing: Each arc cost in the labeling algorithm is set to the number of times the corresponding crowdshipper has been used in previous solutions. This penalizes overused crowdshippers, thus favoring paths that involve less frequently employed resources.

(R4) Delivery Cost Minimization + Uniform Crowdshipper Load

- (a) Station Assignment: As in variant R2.
- (b) Path Computation with Load Balancing: As in variant R3.

Constructive Initial Heuristic

The Constructive Initial Heuristic (CIH) generates a starting solution used as a baseline in the ALNS. CIH sequentially improves the solution starting from a full backup set containing all the parcels to be delivered. In more detail, CIH:

- (1) Uses Repair Heuristic R1 (Parcel Distribution + Shortest Path) for parcel assignment.
- (2) Processes parcels in the **As-Is** order.
- (3) Employs a uniform arc cost $f_k(i, j) = 1$ to minimize the number of crowdshippers.

All parcels are initially placed in the backup set, and CIH iteratively assigns them to the source stations using the introduced labeling algorithm to compute feasible delivery parcel paths. If no feasible path is found for a parcel, it stays in the backup set. If a feasible path is found, the parcel is removed from the backup set, which ensures that the obtained starting solution is at least as good as the one involving all the parcels in the backup set.

3.6 A Branch-and-Cut Algorithm for PTCP

The valid inequalities introduced in Section 3.4.1 are polynomial in the size of N , P , K , and T . As such, it is either possible to insert them from scratch, i.e. as regular constraints, in the model or to separate them dynamically during the Branch-and-Cut.

3.6.1 Introduction to the Branch-and-Cut Algorithm

Historically, the first formally described algorithm to solve generic integer problem was the cutting planes method proposed by Gomory (1958). The algorithm begins by solving the linear programming relaxation of the problem. If such a solution is not integer-feasible, the method derives a new constraint satisfied by all integer solutions but violated by the current fractional one. Such a constraint is called a valid inequality or a cut. This cut is then added to the model, thereby excluding the fractional solution. The process is repeated until an integer solution is obtained or no further useful cuts can be found. The cutting planes is an exact method, meaning that, sooner or later, it will find the optimal integer solution or prove that the problem is infeasible. However, new cuts have the tendency to be less and less effective in removing fractional solutions from the feasible region. This leads to a potentially slow convergence.

The branch-and-bound algorithm, introduced by Land and Doig (1960), systematically explores the solution space of an integer program through a search tree. At each node, the problem is relaxed into a linear program whose solution yields a bound on the objective value. If this solution is fractional, the algorithm selects a fractional variable and branches on it by imposing two additional constraints: one requiring the variable to be at least the ceiling of its current value, and the other forcing it to be at most the floor. These constraints generate two subproblems, which inherit all additional constraints from the parent problem. If instead the relaxation is infeasible or its bound is worse than the best known integer solution, the node is pruned and no descendants are created. The procedure repeats until every node has either been explored or pruned, ensuring that the best integer solution is eventually identified. While branch-and-bound can often find feasible integer solutions quickly, certifying their optimality may require examining an exponential number of nodes in the worst case.

To overcome the issues from these methods, and exploit their strengths, Padberg and Rinaldi (1987) proposed the branch-and-cut algorithm. A simple pseudocode for the method is presented in Algorithm 3.4.

Algorithm 3.4 provides a high-level pseudocode of the branch-and-cut algorithm. The procedure begins by initializing the branch-and-bound tree with a root node containing the linear relaxation of the MILP. The algorithm then iteratively selects a node to process, according to a chosen search strategy such as depth-first, breadth-first, or best-bound. At the selected node, the LP relaxation is solved. If it is infeasible, typically due to previously imposed branching constraints, or if its objective value does not improve upon the current incumbent, the node is pruned. If the relaxation yields an integral solution (up to a numerical tolerance), this solution updates the incumbent. When the solution is fractional, the algorithm attempts to strengthen the relaxation by adding cutting planes. These may be general-purpose cuts, such as Gomory cuts, or problem-specific valid inequalities (e.g., as described in Section 3.4.1). Heuristic rules and effectiveness estimates are typically used to decide whether cuts should be added. If cuts are introduced, the LP relaxation is re-optimized, and this process may be repeated multiple times. If no suitable cuts are found, the algorithm

Algorithm 3.4 Branch-and-Cut Algorithm

```

1: Initialize search tree with root node (relaxation of MILP)
2: while Open nodes exist do
3:   Select node from tree
4:   Solve LP relaxation at current node
5:   if Infeasible or Bound  $\geq$  best solution then
6:     Prune the node
7:   else if Solution integral then
8:     Update best solution
9:   else
10:    Try to add cutting planes
11:    if Cuts added then
12:      Re-optimize LP at current node
13:    else
14:      Branch on fractional variable, add children to tree
15:    end if
16:  end if
17: end while
18: return Best Solution or Infeasible

```

branches on a fractional variable, commonly chosen using strategies such as pseudo reduced costs, and generates two new subproblems that are added to the tree. This process continues until all nodes are either solved or pruned. At termination, the algorithm returns the optimal integer solution if one exists, or proves that the MILP is infeasible.

To improve the efficiency of a branch-and-cut algorithm, implementations often incorporate heuristic methods, both general-purpose and problem-specific, to quickly generate feasible integer solutions. These solutions provide a primal bound that aids pruning and can serve as a valid incumbent when the algorithm is run under strict time limits.

3.6.2 Separation Procedure

In this section, we explain the procedure used to identify and separate the violated valid inequalities.

First, the algorithm checks if it is worthwhile to search for new valid inequalities to separate. To do so, the algorithm checks if the objective value at the current cutting planes iteration has improved from the value at the previous iteration. If it is the case, it means that the previous cuts have been useful, so it might be useful to search and add new ones. Otherwise, the previous cuts didn't contribute strengthening the linear relaxation, so it is unlikely that new cuts might help. If the algorithm proceeds to search for violated valid inequalities, it searches, for one class of valid inequalities at a time, the most violated one, and adds it to the model. Violation is checked up to a parameter ϵ . Some of the cuts are processed together to improve the computational efficiency of the method. For instance, Algorithm 3.5 shows the pseudocode used to identify violated (3.25) and (3.26). Since both these valid inequalities share a similar underlying structure of checking that the sum of location-related variables is less than the number of crowdshippers used to transfer a parcel, the same algorithm can do both the checks without any significant modification.

Algorithm 3.5 Search for violated (3.25)–(3.26)**Require:** $V: V \in |N| \times |P|$ containing the value of either d_{ip} or s_{ip}

```

1:  $violation \leftarrow 0$ 
2:  $idx \leftarrow -1$ 
3: for  $p \in P$  do
4:    $lhs \leftarrow 0$ 
5:   for  $i \in N$  do                                      $\triangleright$  For all  $i \notin D$  or  $i \notin S$   $V_{ip} = 0$ 
6:      $lhs \leftarrow lhs + V_{ip}$ 
7:   end for
8:    $rhs \leftarrow 0$ 
9:   for  $k \in K$  do
10:     $rhs \leftarrow rhs + y_{kp}$ 
11:  end for
12:   $d = lhs - rhs$ 
13:  if  $d > \epsilon$  and  $d > violation$  then              $\triangleright \epsilon$  is a small number used to prevent numerical
    issues
14:     $violation \leftarrow d$ 
15:     $idx \leftarrow p$ 
16:  end if
17: end for
18: return  $violation, idx$ 

```

Algorithm 3.5 is quite straightforward. It gets as input either the values of d_{ip} or s_{ip} , and it initialize the largest violation score and the current violation parcel to placeholder values. Then, for each parcel compute the left hand side and the right hand side of (3.25) or (3.26) respectively. The currently largest violation is checked against the current one, if the current one is larger than the largest violation and the most violated parcel are updated. At the end, the largest violation and the violating parcel are returned to the caller. If the violation score corresponds to the initial placeholder, the caller knows that the current valid inequality is never violated.

A similar procedure is implemented to check for the violation of all the other valid inequalities. Due to their similarities with Algorithm 3.5, the procedures will not be presented.

3.7 Experimental Results

In this section, we first introduce the procedure used to generate the synthetic instances, followed by a description of the experimental setup. Next, we discuss the results obtained from the introduced strengthened MILP, as well as the performance of the proposed ALNS algorithm. Finally, we present and analyze the sensitivity of some key problem parameters.

3.7.1 Instance Generation and Experimental Setup

We generate synthetic instances based on three key components: the transportation network, the crowdshippers, and the parcel demand. We randomly create different PT networks depending on the number of lines and the total number of stations. Each transport line is generated as a random curve on the plane, ensuring that it intersects with at least one other

curve and contains no self-intersections. These intersections define exchange stations between lines, while the remaining stations are randomly distributed along the available lines. The travel time between consecutive stations is an integer value in minutes distributed in the range [5, 15]. We generated four PT networks with the following configurations in terms of number of lines and stations: (4, 24), (6, 36), (8, 40), (8, 44). The first network contains 3 exchange stations, the second contains 6, and the last two contain 9 each. We created six instance classes, each defined by a given number of crowdshippers and parcels. Each class includes four families, with six instances per family all sharing the same transportation network. In all instances, the number of time slots is set to 85, with each time slot lasting five minutes.

Each station is associated with two probabilities: the likelihood of being selected as a starting point by a crowdshipper and the likelihood of being chosen as a destination station by a customer. These probabilities are higher for traffic-intensive stations, such as terminal or exchange stations. A random variable with a realistic time pattern distribution determines when crowdshippers arrive at their starting stations. A random point on the plane is chosen as the depot for the LSP, and the distance between the depot and each station is used to compute the backup cost, which is regularized to be an integer between 12 and 20. Each instance is solved three times using the CPLEX solver: first on the base model without the valid inequalities, then with the valid inequalities separated dynamically, and finally on the model strengthened with valid inequalities added from scratch. The solver is configured with a 1-hour time limit, with no restrictions on the maximum number of threads or memory available. Each instance is also solved using our ALNS algorithm with different configuration parameters. These parameters cover possible combinations of the cooling rate value, set to [0.75, 0.8, 0.85, 0.9], with the time limit threshold, set to 1, 5, and 10 minutes, respectively.

The destroy heuristic (D1) is configured with $q = [10, 15, 20]$, whereas for the destroy heuristic (D2) both q and $\tilde{q}$ can take value between 1 and 4. To ensure the robustness of the results, each instance-configuration combination is solved five times. The reported results always present the family-wise average. All experiments were executed on a computer running Debian 12.1, with an AMD Ryzen 5 5600U CPU, with 6 cores and 12 threads, and equipped 40 Gb of RAM. The ALNS algorithm is implemented in Rust 1.70, the MILP model is implemented in Java 20, and solved using CPLEX 22.1. All valid inequalities are inserted with CPLEX's flag `UseCutForce` that forces the solver to insert the proposed valid inequalities in the current and all subsequent cutting plane iteration.

3.7.2 MILP Formulation Results: Base CPLEX versus Implemented Branch-and-Cut

This section presents the results of our computational experiments when solving the proposed mathematical formulation. We begin by discussing the results obtained with CPLEX without any valid inequality, then we describe the outcomes from CPLEX with the valid inequalities separated dynamically (as discussed in Section 3.6), finally we present the results from CPLEX with the valid inequalities added from scratch. Henceforth, we will refer to a solution that makes use of the backup service to deliver parcels as a *default solution*. It is important to note that finding a default solution is always straightforward, as it involves simply summing the lowest backup costs for each parcel.

Table 3.4 presents a summary of the results obtained from running CPLEX without incorporating our valid inequalities. The table is composed of 11 columns representing the following information. The first three columns identify the instance family, specifying the

number of stations ($|N|$), crowdshippers ($|K|$), and parcels ($|P|$) in the instance family, respectively. The fourth column (**Gap**) shows the family average MIP gap (between the incumbent integer solution and the best bound) provided by CPLEX. The fifth column (**Opt.**) shows the number of family instances (out of 6) solved to optimality. The sixth column (**Nodes**) shows the average number of nodes explored in the Branch-and-Bound tree. The seventh column (**Obj.**) shows the average objective function value. The column (**Bound**) shows the average bound value. The ninth column (**Time[s]**) indicates the execution time, in seconds, whereas the tenth column (**TTB[s]**) displays the average time to best, in seconds. Finally, the last column (**Def.**) shows the number of instances in the family for which the solver reaches the time limit without improving the default solution.

Table 3.4: Base CPLEX: solution statistics

$ N $	$ K $	$ P $	Gap	Opt.	Nodes	Obj.	Bound	Time[s]	TTB[s]	Def.
24	75	50	11.01%	1	437.17	613.17	546.28	3029.36	1311.72	0
36	75	50	8.20%	0	1019.67	654.3	600.66	3600.17	1338.78	0
40	75	50	1.28%	3	2455.33	671.67	663.08	2648.68	124.25	0
44	75	50	1.42%	3	2949.83	692.83	683.29	2000.27	181.11	0
24	100	75	14.51%	0	1.17	951.17	814.54	3600.15	2037.67	0
36	100	75	30.53%	0	0.00	1063.33	730.80	3600.19	1138.09	2
40	100	75	7.50%	0	77.67	999.17	924.50	3600.29	1609.20	0
44	100	75	3.93%	0	178.67	1059.33	1018.00	3600.45	1449.46	0
24	150	120	35.11%	0	0.00	1809.83	1173.93	3600.25	46.66	5
36	150	120	36.30%	0	0.00	1862.00	1185.85	3600.38	0.09	6
40	150	120	16.94%	0	0.00	1833.83	1524.86	3600.33	66.89	5
44	150	120	10.99%	0	0.00	1859.67	1653.38	3600.41	532.68	3
24	200	160	38.00%	0	0.00	2501.67	1552.05	3600.45	120.83	5
36	200	160	33.78%	0	0.00	2363.33	1564.88	3600.63	0.13	6
40	200	160	17.72%	0	0.00	2495.67	2052.65	3600.56	164.28	5
44	200	160	16.75%	0	0.00	2487.50	2070.05	3600.60	205.58	5
24	250	225	30.48%	0	0.00	3348.33	2327.01	3600.72	0.13	6
36	250	225	63.00%	0	0.00	3536.50	1293.66	3601.43	0.19	6
40	250	225	16.30%	0	0.00	3437.00	2876.30	3600.83	0.21	6
44	250	225	27.87%	0	0.00	3552.33	2572.97	3601.07	0.21	6
24	300	250	34.61%	0	0.00	3853.00	2518.59	3600.85	0.16	6
36	300	250	100.00%	0	0.00	3740.67	0.00	3602.82	0.24	6
40	300	250	85.53%	0	0.00	3782.33	538.19	3602.15	565.39	5
44	300	250	86.21%	0	0.00	3861.00	530.60	3602.20	0.26	6
Averages			30.33%	0.29	340.02	2209.62	1309.06	3482.07	453.82	3.71

By examining Table 3.4, we observe that only 7 out of 144 instances were solved to optimality within the one hour time limit. Even the family with the smallest average solution time, namely (44, 75, 50), still takes more than 37 minutes on average to find a solution. Looking at the **Nodes** column, we observe that in 17 out of 24 families, the solver is unable to explore the solution tree beyond the root node. Many instance families display very small time-to-best (TTB) values, often below one second. Such findings can be explained by the fact that the solver only finds the default solution. However, in several families, the solver managed to find better solutions than the default one, resulting in more reasonable TTB values. Finally, the MIP gap values are generally high, with an average value of 30.33%. The minimum gap observed is 1.28%, while the maximum reaches 100% (for family (36, 300, 250)),

the solver is unable to find a feasible solution for any instance). The solver struggles to find improving solutions for numerous large instances, and in several cases, CPLEX could not even find the optimal solution of the linear relaxation. Although the solver performs better with smaller instances, even in these cases computational times are close to or reach the time limit. These suggest that CPLEX alone is not sufficient for handling large-scale instances, therefore heuristic methods should be considered to improve the solution process.

Table 3.5: Implemented Branch-and-Cut: solution statistics

$ N $	$ K $	$ P $	Gap	Opt.	Nodes	Obj.	Bound	Time[s]	Impr.	TTB[s]	Def.
24	75	50	6.65%	0	877.17	610.83	570.27	3621.28	4.39%	818.85	0
36	75	50	3.86%	0	835.5	649.7	623.92	3626.62	0.70%	311.80	0
40	75	50	0.76%	4	249.00	671.67	666.48	2008.88	0.51%	208.67	0
44	75	50	0.57%	4	242.17	692.83	689.06	1552.07	0.00%	604.80	0
24	100	75	8.09%	0	81.50	936.50	861.71	3619.18	5.79%	1872.32	0
36	100	75	15.58%	0	3.83	944.50	787.18	3619.48	7.71%	2114.86	0
40	100	75	4.53%	0	10.83	1000.17	954.93	3624.80	3.29%	1956.47	0
44	100	75	1.92%	0	31.83	1054.17	1033.95	3624.16	1.57%	1374.48	0
24	150	120	34.00%	0	0.00	1809.83	1194.37	3600.27	1.74%	45.29	5
36	150	120	36.00%	0	0.00	1862.00	1191.43	3600.38	0.47%	0.09	6
40	150	120	16.80%	0	0.00	1833.83	1527.58	3600.36	0.18%	66.44	5
44	150	120	10.53%	0	0.00	1849.83	1652.97	3600.43	-0.02%	1201.96	2
24	200	160	37.85%	0	0.00	2501.67	1555.80	3600.47	0.24%	122.44	5
36	200	160	33.38%	0	0.00	2363.33	1574.24	3600.69	0.60%	0.13	6
40	200	160	17.34%	0	0.00	2495.67	2062.36	3600.63	0.47%	185.06	5
44	200	160	16.49%	0	0.00	2487.50	2076.63	3600.63	0.32%	219.17	5
24	250	225	30.17%	0	0.00	3348.33	2337.39	3600.72	0.45%	0.14	6
36	250	225	63.00%	0	0.00	3536.50	1293.66	3653.59	0.00%	0.20	6
40	250	225	16.29%	0	0.00	3437.00	2876.77	3949.77	0.02%	0.20	6
44	250	225	27.75%	0	0.00	3552.33	2576.98	3823.56	0.16%	0.21	6
24	300	250	34.61%	0	0.00	3853.00	2518.59	3737.91	0.00%	0.17	6
36	300	250	100.00%	0	0.00	3740.67	0.00	3603.02	N.A.	0.26	6
40	300	250	85.53%	0	0.00	3782.33	538.19	3602.21	0.00%	580.21	5
44	300	250	86.21%	0	0.00	3861.00	530.60	3602.22	0.00%	0.26	6
Averages			28.66%	0.33	94.81	2203.20	1320.69	3503.52	1.28%	474.90	3.58

Table 3.5 summarizes the corresponding results obtained by our implementation of the Branch-and-Cut obtained from running the solver CPLEX with our valid inequalities separated dynamically. Table 3.5 consists of 12 columns following the structure of Table 3.4 with the addition of column “**Impr.**” showing the average percentage improvement in the objective value compared to the initial one found by Base CPLEX. This column provides various insightful observations. Primarily, our cuts are more effective with smaller instances, providing only minimal enhancements for larger instances. Furthermore, as shown in Table 3.6, no valid inequalities are added in the final three instance families, as the solver is unable to compute the linear relaxation within the time limit. In one instance with 44 nodes, 150 crowdshippers, and 120 parcels, the introduction of valid inequalities resulted in a slight deterioration of the objective bound. This is presumably due to the substantial number of constraints added, resulting in a slower optimization process and hence worsening the results. For class $|N| = 36$, $|K| = 300$, and $|P| = 250$, CPLEX never solved the linear relaxation, hence it is impossible to compute the improvement from the Base CPLEX. For this reason, the corresponding cell contains the placeholder N.A. (Not Available), and the average presented at the bottom of the table is computed without considering this value. The use of valid

inequalities allows to solve an additional instance to optimality with respect to Base CPLEX. We observe that, on average, the valid inequalities increase the average TTB by 4.65%. On the other hand, the valid inequalities improve the MIP gap by 1.67%. It suggests that the introduction of valid inequalities forces CPLEX to spend more time on the Branch-and-Cut algorithm, thereby reducing the time available for the internal heuristics, which results in a generally worse TTB. However, the new information given to the Branch-and-Cut allows the solver to find better linear relaxation to the problem, proving the effectiveness of our valid inequalities. With respect to Base CPLEX, we can observe that it has been possible to find solutions that are not the default one for a larger number of instances: 86 against 89. An interesting result to discuss is the average number of nodes explored by the solver for small size classes when compared to Base CPLEX. With a few exceptions, this number is smaller. This can be attributed to the fact that the added cuts enabled the solver to close subproblems without necessitating branching.

In Table 3.6, we summarize the statistics on the number of generated cuts. The table consists of nine columns, where the first three identify the instance size, the fourth column displays the average number of added cuts for inequalities (3.24). Similarly, from the fifth to the ninth column, we provide the average numbers of added cuts for valid inequalities (3.25) and (3.26) jointly applied for each instance, valid inequalities (3.27) and (3.28), (3.29), (3.30), and (3.31) and (3.32), respectively.

Table 3.6: Statistics on generated number of cuts

$ N $	$ K $	$ P $	(3.24)	(3.25)–(3.26)	(3.27)–(3.28)	(3.29)	(3.30)	(3.31)–(3.32)
24	75	50	28.00	6171.67	11224.50	10035.33	40.50	95830.83
36	75	50	117.83	3146.50	4345.83	13789.16	96.00	43205.50
40	75	50	7.17	572.50	873.67	2001.17	12.17	15703.50
44	75	50	1.00	1553.50	1930.00	4235.33	15.17	19614.67
24	100	75	0.00	1393.83	2578.50	1747.17	6.50	14601.83
36	100	75	0.50	183.17	321.83	91.50	0.50	1252.67
40	100	75	6.50	226.50	331.83	513.33	2.17	2548.83
44	100	75	6.17	431.33	564.67	815.83	1.33	4024.50
24	150	120	0.00	129.83	145.00	18.83	0.00	252.50
36	150	120	0.00	98.83	134.33	87.00	0.00	192.17
40	150	120	0.00	58.17	68.17	161.17	0.00	153.67
44	150	120	0.17	63.33	74.33	48.33	0.00	140.67
24	200	160	0.00	152.33	184.00	18.17	0.00	152.67
36	200	160	0.00	153.33	192.00	121.83	0.00	293.17
40	200	160	0.00	76.50	95.00	190.33	0.00	303.83
44	200	160	0.00	76.83	109.33	525.50	0.00	727.50
24	250	225	0.00	206.67	225.50	204.50	0.00	338.67
36	250	225	0.00	26.33	41.00	0.83	0.00	21.33
40	250	225	0.00	121.83	135.17	462.50	0.00	458.00
44	250	225	0.00	81.17	91.83	1074.17	0.00	982.67
24	300	250	0.00	195.33	227.00	60.83	0.00	111.00
36	300	250	0.00	0.00	0.00	0.00	0.00	0.00
40	300	250	0.00	0.00	0.00	0.00	0.00	0.00
44	300	250	0.00	0.00	0.00	0.00	0.00	0.00

It is interesting to note that the smaller the size of the instance, the larger the number of

added cuts. This can be explained by the fact that, given the time limit, in small instances, the solver explored a large number of Branch-and-Cut nodes. On the other hand, no cuts were generated for the last three families where the solver cannot even find the optimal solution of the continuous relaxation within the time limit. By examining the columns of Table 3.6, we observe that cuts like (3.31) and (3.32) were generally applied the most. Cuts (3.27) and (3.28) were also applied frequently. This is likely because these cuts operate on larger variable sets, such as the location variables o_{ipt} and the movement variables a_{ipt} and l_{ipt} . The least used cuts are (3.24) and (3.30).

Table 3.7: Valid inequalities incorporated into the model from scratch: solution statistics

$ N $	$ K $	$ P $	Gap	Opt.	Nodes	Obj.	Bound	Time[s]	Impr.	TTB[s]	Def.
24	75	50	6.51%	2	3348.17	610.67	571.87	3023.85	0.37%	420.53	0
36	75	50	5.58%	0	3363.67	571.33	538.06	3600.22	1.28%	635.36	0
40	75	50	0.00%	6	1118.50	671.67	671.67	731.54	0.00%	98.87	0
44	75	50	0.17%	5	2186.83	692.83	691.64	1000.00	0.00%	144.39	0
24	100	75	8.21%	0	601.33	937.17	860.88	3600.38	1.47%	329.14	0
36	100	75	7.03%	0	251.50	871.50	809.49	3600.61	17.45%	1886.17	0
40	100	75	3.45%	0	1899.83	993.83	959.24	3600.47	0.54%	894.71	0
44	100	75	1.51%	1	1054.17	1054.00	1038.21	3538.96	0.52%	586.34	0
24	150	120	15.44%	0	7.33	1521.50	1274.84	3600.61	16.02%	1278.46	1
36	150	120	8.36%	0	1.17	1418.50	1297.43	3601.13	23.88%	2077.48	0
40	150	120	3.54%	0	243.17	1633.00	1576.18	3601.30	11.03%	2091.54	0
44	150	120	1.83%	0	539.83	1730.67	1697.84	3601.23	6.88%	1084.99	0
24	200	160	21.39%	0	0.00	2154.83	1672.92	3600.51	13.92%	1607.61	2
36	200	160	19.51%	0	0.00	2116.33	1669.03	3600.72	10.49%	1402.88	3
40	200	160	5.20%	0	0.00	2248.83	2131.94	3600.86	9.86%	2415.93	0
44	200	160	4.33%	0	0.00	2241.67	2142.71	3601.31	9.86%	2869.15	0
24	250	225	23.77%	0	0.00	3230.00	2446.32	3601.06	3.58%	514.53	5
36	250	225	88.25%	0	0.00	3536.50	435.27	3601.92	0.00%	0.21	6
40	250	225	26.45%	0	0.00	3334.33	2441.27	3602.15	2.94%	1078.69	4
44	250	225	34.96%	0	0.00	3310.00	2057.00	3601.49	6.88%	1783.58	2
24	300	250	30.90%	0	0.00	3853.00	2661.80	3601.13	0.00%	0.18	6
36	300	250	88.77%	0	0.00	3740.67	423.81	3603.31	0.00%	0.25	6
40	300	250	85.57%	0	0.00	3782.50	570.62	3602.81	0.00%	0.26	6
44	300	250	69.56%	0	0.00	3769.83	1065.55	3602.98	2.37%	599.50	5
Averages			23.35%	0.58	606.01	2084.44	1321.12	3340.22	5.80%	990.91	1.92

Table 3.7 presents the results obtained when valid inequalities (3.24)–(3.32) are incorporated directly into the model. These results are compared with the two previously considered approaches: solving the model using CPLEX without valid inequalities (Base CPLEX) and with a Branch-and-Cut algorithm that dynamically separates these inequalities. Column **Opt.** reports the number of instances solved to optimality within the time limit, totaling 14. In particular, all instances previously solved optimally by both Base CPLEX and Branch-and-Cut algorithm with dynamically separated inequalities were also solved by the current version that incorporates inequalities from scratch. For the configuration with $|N| = 40$, $|K| = 75$ and $|P| = 50$, incorporating valid inequalities directly into the model proves more effective in finding the optimal solution in all instances. On average, the incumbent solution improves by 5.80% (column **Impr.**) compared to Base CPLEX, significantly outperforming the 1.28% improvement achieved with dynamically separated inequalities. The improvement is most significant for medium-sized instances ($|K| = 100$ and 150) with a peak at 23.88%, while for smaller instances ($|K| = 75$), it is negligible or absent. This occurs because the solver finds

the same incumbent solution, but struggles to prove optimality without valid inequalities. Regarding MIP gaps (column **Gap.**), two key insights emerge: (1) the average gap is lower than both Base CPLEX and Branch-and-Cut with dynamically added inequalities, and (2) no instances reach the worst-case 100% gap, with even the largest instances ($|K| = 300$) successfully solving their linear relaxations. In terms of computing times, the time to the best solution (column **TTB[s.]**) is generally higher than for the other two approaches. However, this must be interpreted in the context of the number of instances solved. Since fewer instances are solved to their default value, the solver must invest more effort in refining better incumbent solutions, which naturally takes more time. This is confirmed by the overall solution time (column **Time[s.]**), which is lower than for both the other two approaches, demonstrating that incorporating valid inequalities directly into the model improves efficiency.

3.7.3 ALNS Results

In this section, we analyze the performance of our ALNS algorithm. Table 3.8 shows the

Table 3.8: CPLEX and ALNS results for the instances solved to optimality

$ N $	$ K $	$ P $	ID	CPLEX Obj.	ALNS Obj.	ALNS Gap	CPLEX Time [s]	ALNS Time [s]
24	75	50	3	638	638	0.00	377	9.0
24	75	50	6	629	632	0.41	3364	269.0
40	75	50	1	750	750	0.00	30	0.0
40	75	50	2	617	618	0.16	93	20.0
40	75	50	3	694	695	0.17	3178	81.0
40	75	50	4	647	647	0.00	94	233.0
40	75	50	5	708	712	0.56	73	1.0
40	75	50	6	614	614	0.00	918	57.0
44	75	50	2	716	716	0.00	48	1.0
44	75	50	3	715	715	0.00	72	1.0
44	75	50	4	646	648	0.31	1248	5.0
44	75	50	5	684	685	0.15	58	0.0
44	75	50	6	718	718	0.00	972	0.0
44	100	75	2	1143	1143	0.00	3231	3.0

comparison between the optimal solution values found by CPLEX with the valid inequalities inserted from scratch, and the best value returned by our ALNS algorithm. The table includes only the fourteen instances that were solved to optimality. All these results come from small instances: thirteen from instances with $|K| = 75$ crowdshippers and $|P| = 50$ parcels to deliver, and only one from an instance with $|K| = 100$ crowdshippers and $|P| = 75$ parcels to deliver. The first column indicates the number $|N|$ of stations in the network, the second column represents the number $|K|$ of crowdshippers, the third column the number $|P|$ of parcels, while the fourth column represents the instance id. The fifth and sixth columns show the CPLEX and ALNS objective values, respectively. The last two columns display the execution times (in seconds) for CPLEX and ALNS, respectively. By observing the **ALNS Gap** column, we see that our ALNS algorithm found the optimal value in seven instances. For the instances where our ALNS did not find the optimal value, in six cases, the gap between the optimal value and the heuristic solution is below 1%. From this initial analysis, we can see that our ALNS algorithm can find near-optimal solutions in the vast majority of cases, and in a fraction of the time required by a state-of-the-art solver.

Table 3.9 summarizes the results obtained from employing the ALNS algorithm with the best configuration for each instance family. Our tests reveal that, for each family, the configuration achieving the best results consistently uses a time limit of 600 seconds and a cooling rate fixed to 0.9. The table consists of nine columns. The first three columns identify the instance family, specifying the number of nodes ($|N|$), crowdshippers ($|K|$) and parcels ($|P|$), respectively. The following two columns (**TTB[s]** and **ITB**) show the average time to the best solution, in seconds, and the average number of ISA iterations to the best solution, respectively. The next column (**Obj.**) shows the average objective function value. The last three columns (**N. Sols**, Δ **Bound**, and Δ **Obj.**) display respectively the average number of improving solutions found during the search, the average difference in percentage between the best solution from ALNS and the best known lower bound from CPLEX, and the average difference in percentage between the ALNS solution value and the CPLEX's one.

Table 3.9: ALNS results

$ N $	$ K $	$ P $	TTB[s]	ITB	Obj.	N. Sols	Δ Bound	Δ Obj.
24	75	50	206.35	14948.47	613.63	15.97	7.60%	0.46%
36	75	50	11.89	353.56	684.83	8.33	9.76%	5.41%
40	75	50	65.80	7153.37	672.70	11.00	0.93%	0.15%
44	75	50	19.03	1754.80	693.43	9.83	1.48%	0.86%
24	100	75	236.20	7523.03	940.90	16.77	9.19%	0.47%
36	100	75	55.32	827.07	933.37	13.37	18.57%	-1.18%
40	100	75	150.98	7433.87	996.10	13.13	4.31%	-0.41%
44	100	75	111.30	10530.87	1054.47	12.70	1.98%	0.03%
24	150	120	307.69	4078.70	1450.83	21.17	21.47%	-19.84%
36	150	120	219.16	1159.50	1482.97	18.60	24.47%	-20.36%
40	150	120	242.29	6007.00	1636.03	15.77	7.10%	-10.79%
44	150	120	196.41	9468.60	1725.00	16.47	4.36%	-6.75%
24	200	160	271.20	1826.20	1988.03	19.83	27.78%	-20.53%
36	200	160	287.85	838.40	1871.80	22.03	18.90%	-20.80%
40	200	160	259.53	3335.20	2240.30	17.53	8.63%	-10.23%
44	200	160	262.97	7454.17	2208.77	19.83	6.36%	-11.21%
24	250	225	312.18	1314.33	2789.67	22.00	19.35%	-16.68%
36	250	225	283.21	327.80	2912.67	20.20	125.15%	-17.64%
40	250	225	240.26	1704.13	3112.23	17.87	8.19%	-9.45%
44	250	225	267.37	4537.57	3219.60	22.20	24.94%	-9.37%
24	300	250	328.80	912.77	3130.80	25.07	24.31%	-18.74%
36	300	250	318.09	297.00	2895.57	25.90	N.A.	-22.59%
40	300	250	271.92	1154.87	3417.20	20.00	534.94%	-9.65%
44	300	250	275.87	3686.90	3521.30	19.37	563.64%	-8.80%
Averages			216.74	4109.47	1925.01	17.68	64.07%	-9.48%

An important aspect to highlight concerns the time to the best solution (**TTB[s]**). It is worth noting that the average TTB is consistently below the five-minute mark (300 seconds), and even below one minute for some instances. Nevertheless, the best configurations are always the ones with a 10-minute time limit. Likely, a longer time limit allowed the algorithm to find better solutions in later phases of the optimization without sacrificing lucky runs, in which the incumbent was found in the initial phases. We observe in the Δ **Bound** column

that, in general, small gaps from the objective bound exist when the MIP gap itself is small. This indicates that our heuristic approach is able to find solutions that are close to optimal, especially for small instances. For the instance family $|N| = 36$, $|K| = 300$, and $|P| = 250$, the placeholder N.A. (Not Available) is due to the absence of any linear relaxation coming from CPLEX, and the average presented at the bottom of the table is computed without considering this value. Finally, we observe from the Δ **Obj.** column that CPLEX found better average solutions than our ALNS approach in only five instance families. Except for one case, all these gaps are equal to or below 1%. Additionally, there is a noticeable trend of the gap improving as the instance sizes increase.

Table 3.10: Correlation matrix

	$ N $	$ K $	TTB[s]	ITB	N. Sols
$ N $	-	0.00	-0.35	0.06	-0.34
$ K $	0.00	-	0.79	-0.50	0.84
TTB[s]	-0.35	0.79	-	-0.17	0.93
ITB	0.06	-0.50	-0.17	-	-0.32
N. Sols	-0.34	0.84	0.93	-0.32	-

In order to gain further valuable insights, we examine the correlation coefficients between the different columns presented in Table 3.9. The most significant insights from this analysis, which are independent of the instances themselves, are summarized in Table 3.10. Significant correlation indexes are highlighted in bold. We observe a strong positive correlation between the number of crowdshippers ($|K|$) and the number of improved solutions found (**N. Sols**). This correlation suggests that our ALNS algorithm can effectively use the larger pool of crowdshippers to its advantage. The observed correlations between the time to the best solution (**TTB[s]**) and the number of stations ($|N|$), as well as the correlation between **TTB[s]** and the number of crowdshippers ($|K|$), are also of interest. Generally, a larger pool of crowdshippers leads to a longer TTB, which is expected due to the increased number of solutions found. On the other hand, a larger set of stations results in shorter TTBs. To explain this further, it is important to consider the existing correlation between the number of stations ($|N|$) and the number of improved solutions found (**N. Sols**). Both of these correlations are negative, indicating that in a larger network, the ALNS algorithm has more possible alternatives for transferring parcels, enabling the heuristic to find solutions more easily. Finally, the nearly negligible correlation between $|N|$ and the average number of ISA iterations to the best solution (**ITB**) suggests that larger networks are not significantly more computationally demanding than smaller ones for our ALNS algorithm.

In Figure 3.4, we compare the solution found by our ALNS (Figure 3.4a) with the one found by CPLEX (Figure 3.4b) on the same instance, namely the third instance of the family with $|N| = 44$, $|K| = 75$, $|P| = 50$. The solution value remains the same, but the paths followed by the parcels differ slightly. CPLEX consumed the time limit of 1 hour with a final MIP gap equal to 0.58%, whereas our ALNS required the assigned 10 minutes. In each of the two graphs, the nodes represent the PT stations, with square labels indicating source stations. The double-headed arcs between the nodes represent direct connections between the two stations. Each arc represents a step in a parcel's path, with each step associated with the crowdshipper involved. Each parcel is associated with a different color, and the number on each colored arc indicates the crowdshipper responsible for the transfer. For clarity, each parcel's path is represented directly on the original PT network. Both in Figure 3.4a and Figure 3.4b, we observe that some parcels have been delivered by multiple crowdshippers,

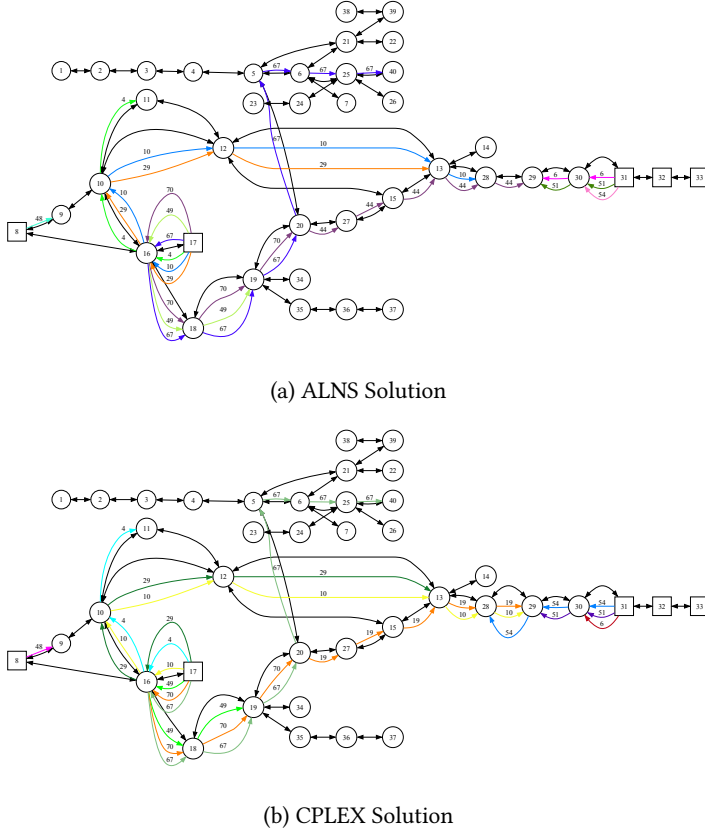


Figure 3.4: Visual comparison of the solutions obtained with (a) ALNS and (b) CPLEX for the same instance.

even though most parcels are handled by only one. Furthermore, in this specific instance, several parcels in both solutions have completed only one step in the network, while others have followed much longer paths. This behavior arises from the remuneration structure of crowdshippers. Delivering a parcel with a single crowdshipper is always cheaper, and paths with too many crowdshippers can exceed the backup cost.

3.7.4 Tuning of ALNS Parameters

This section presents a comprehensive analysis of the main ALNS parameters, namely the cooling rate and the time limit (TL), to evaluate the algorithm's robustness under different settings, providing guidance for decision makers. First, we analyze the impact of their values on the solution quality (effectiveness), then on the TTb (efficiency). Finally, we rank the various parameter values to summarize the findings and provide decision makers with a practical rule.

In our computational experiments, we run the ALNS algorithm with all possible combinations of cooling rates in $[0.75, 0.80, 0.85, 0.90]$ and time limits (in minutes) in $[1, 5, 10]$. Table 3.11 shows, for each instance class and across all the configurations, the mean best,

Table 3.11: ALNS solutions statistics

$ N $	$ K $	$ P $	Best	Worst	Avg.	Q_{25}	Q_{50}	Q_{75}	Gap%
24	75	50	611.83	618.17	614.94	614.17	615.00	615.83	1.06%
36	75	50	634.50	635.00	634.54	634.50	634.50	634.50	0.08%
40	75	50	672.67	673.67	672.85	672.67	672.67	672.88	0.15%
44	75	50	701.33	702.00	701.56	701.50	701.50	701.67	0.10%
24	100	75	938.50	946.33	942.41	941.33	942.17	943.54	0.84%
36	100	75	933.17	934.33	933.53	933.33	933.50	933.67	0.13%
40	100	75	995.33	999.00	997.04	996.33	996.83	997.33	0.38%
44	100	75	1054.00	1056.50	1054.96	1054.67	1055.17	1055.17	0.24%
24	150	120	1447.83	1463.00	1454.63	1452.29	1454.67	1456.42	1.05%
36	150	120	1481.50	1489.17	1484.16	1483.00	1483.67	1485.00	0.53%
40	150	120	1634.17	1642.17	1637.53	1636.17	1637.08	1638.54	0.50%
44	150	120	1724.17	1728.33	1726.11	1725.63	1726.00	1726.83	0.25%
24	200	160	1981.50	2005.83	1993.61	1990.00	1993.08	1996.63	1.23%
36	200	160	1866.83	1888.33	1875.52	1872.29	1874.50	1878.25	1.16%
40	200	160	2236.83	2248.50	2242.45	2240.50	2242.17	2243.92	0.52%
44	200	160	2205.83	2216.33	2211.43	2209.96	2211.33	2212.88	0.48%
24	250	225	2782.33	2808.83	2795.77	2791.29	2795.58	2799.67	0.96%
36	250	225	2909.17	2932.33	2917.52	2913.25	2916.83	2920.88	0.80%
40	250	225	3108.83	3123.67	3115.45	3113.46	3115.33	3117.21	0.48%
44	250	225	3216.33	3229.17	3222.19	3220.33	3222.00	3223.88	0.40%
24	300	250	3122.50	3160.67	3139.40	3134.17	3138.42	3144.88	1.22%
36	300	250	2886.33	2927.67	2903.95	2897.29	2903.25	2909.58	1.43%
40	300	250	3414.00	3433.17	3421.88	3419.13	3421.42	3424.33	0.57%
44	300	250	3517.67	3530.67	3523.75	3522.13	3523.42	3525.04	0.37%

worst, average, and the quartiles of objective values over all the instances in the class. The last column shows the percentage gap between the best and the worst. These statistics are computed over all the T_{max} and all the cooling rates. In fact, the maximal gap is below 1.5%, with an average of 0.62%, and a minimal at 0.08%. Moreover, it is possible to notice that the gap between the Q_{25} and the worst case is always below 1.10%, with an average gap at 0.43%. Analyzing the gap between the best case and Q_{75} , one can see that the maximal gap is 0.81%, the average is 0.35%, and the minimal is 0.00%. This table illustrate the general stability of the algorithm with respect to the configuration values across different instance sizes.

		Cooling Rate			
		0.75	0.80	0.85	0.90
TL	1	1929.33	1928.56	1927.73	1926.87
	5	1925.91	1925.40	1924.91	1923.99
	10	1924.81	1924.33	1923.82	1922.91

Table 3.12: Average objective value obtained from ALNS with various values for the cooling rate and the time limit.

Table 3.12 shows the average best objective function value found by our ALNS, computed

over all instances without class distinction. In the table, the columns represent the outcomes for the different cooling rates, while the rows show the results for each time limit. As seen in Table 3.12, the average best solution value returned by ALNS appears to be independent of the values taken by the two parameters.

Since the ALNS relies heavily on randomization, it is important to determine whether the differences in outcomes are due to chance alone or if the parameters play a pivotal role in the algorithm's performance. To investigate this behavior in depth, we conducted the Wilcoxon signed-rank paired test on the two parameters (for more information about the Wilcoxon test, see Tiit (2000)). The p -values from the tests comparing the cooling rate outcomes are all below 10^{-10} , indicating that, even if the differences are minimal, the null hypothesis can always be rejected, meaning the differences cannot be attributed to chance alone. Similarly, when comparing the outcomes based on time limits, the p -values are always below 10^{-10} . In this case as well, all tests allow us to reject the null hypothesis, indicating that even a minimal but significant difference exists between the outcomes for different time limit values.

Considering the average heuristic results with varying cooling rates and time limits, we observe a limited impact on the solution value. However, Wilcoxon's test results indicate otherwise. Although unexpected, this outcome can be attributed to key aspects of our algorithm's design. First, in our ALNS, the ISA runs repeatedly until the time limit is reached. Unlike standard implementations that use a smooth cooling rate (close to 1) for gradual intensification, we employ multiple restarts within a single run. An aggressive cooling rate (close to 0) shortens each diversification-intensification cycle, increasing the number of runs. Conversely, a smoother cooling rate extends each run, reducing the number of iterations but allowing a deeper search per iteration, often yielding better solutions.

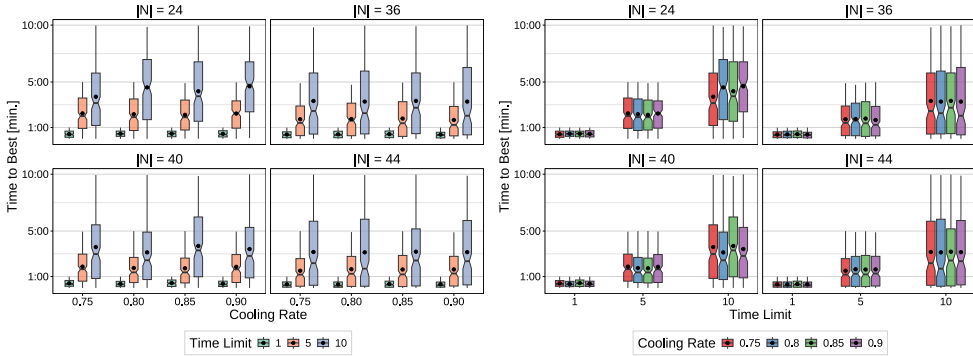
Second, the reheating policy prevents premature convergence by increasing the temperature whenever a new best solution is found, effectively restarting diversification. This balances the effects of the cooling rate, ensuring a thorough exploration of the search space and explaining the consistent performance of the algorithm in different settings of parameters.

Now, let us focus our analysis on the impact of the cooling rate and time limit on the time to the best solution found (TTB). The goal of this analysis is to assess the impact of the configuration parameters on the efficiency, i.e., the amount of time our ALNS takes to find the best solution provided as output. In Table 3.13 we present the TTBS quartiles and standard deviation with respect to the number of nodes in the network and the T_{max} . From

Table 3.13: ALNS time to best statistics

$ N $	T_{max}	Q_{25}	Q_{50}	Q_{75}	Std. Dev.
24	1	11.00	25.35	41.01	17.51
36	1	5.90	20.80	39.35	18.52
40	1	7.65	20.09	38.29	17.47
44	1	3.60	14.27	31.89	17.69
24	5	46.93	120.52	207.21	90.01
36	5	15.08	83.35	179.31	93.57
40	5	29.97	85.06	171.22	88.61
44	5	7.94	73.63	166.25	91.12
24	10	96.89	238.83	401.39	174.14
36	10	23.94	138.37	357.57	187.73
40	10	48.86	170.27	333.51	174.72
44	10	11.43	129.12	349.14	185.44

Table 3.13 it is possible to notice several interesting trends. In this table and in the following, Q_x , with $x \in [0, 100]$, identifies the x -th percentile of population under analysis. First, it can be observed that in all cases Q_{25} is smaller than a quarter of T_{max} . Similarly, Q_{50} is always less than $\frac{1}{2}T_{max}$ and Q_{75} is always less than $\frac{3}{4}T_{max}$. This shows that, in general, the algorithm finds the incumbent result well within the available time limit. As a consequence, an eventually too short runtime is not too decremental for the solution quality. On the other hand, these time to best show that the algorithm is capable of exploiting the extra available time to further improve the solution. Moreover, it is possible to notice that in networks $|N| = 36, 40, 44$ the value of Q_{50} for $T_{max} = 10$ is smaller than Q_{75} for $T_{max} = 5$. On the other hand, with $|N| = 24$ the gap between Q_{50} for $T_{max} = 10$ and Q_{75} for $T_{max} = 5$ is 15.26%. This is further evidence of the proposed ALNS's efficacy in finding good quality solutions in a short amount of time.

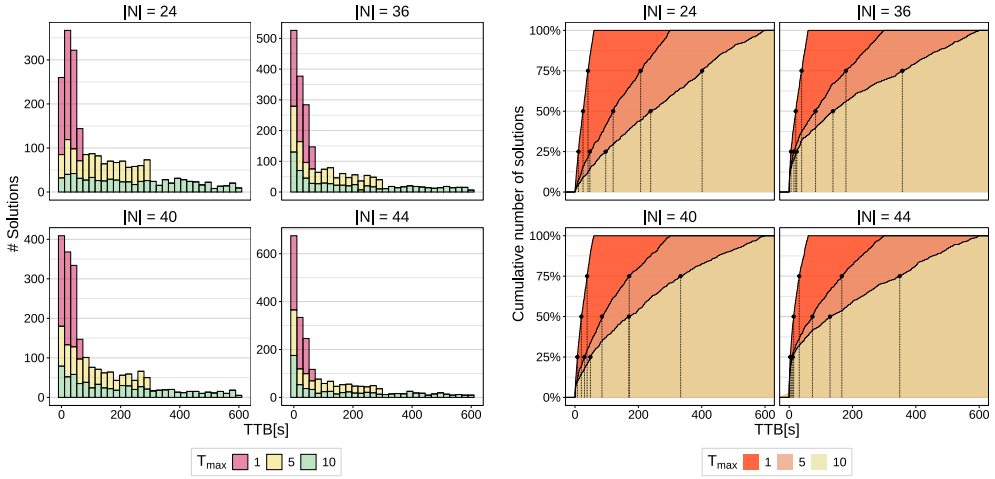


(a) Time-to-Best value for different ISA cooling rates (b) Time-to-Best value when changing the ALNS's time limit

Figure 3.5: Comparison of Time-to-Best metrics: (a) by cooling rate, (b) by time limit.

Figure 3.5a displays the TTB distribution across the different considered networks, grouped by time limit, under varying cooling rates. In this figure, the boxplots represent the quantiles of the TTB distributions, while the dots indicate the average TTB for each combination of cooling rate, time limit, and network. Unsurprisingly, a lower time limit corresponds to a lower TTB. Figure 3.5b shows the TTB distribution for each network, grouped by cooling rate, under varying time limits. Naturally, this figure presents the same information as Figure 3.5a but from a different perspective, serving two purposes. First, it confirms previous observations, such as the fact that longer time limits correspond to larger TTBs. Second, it clearly shows that there is no trend, i.e., no positive or negative correlation between cooling rate and TTB. From the analysis of Figure 3.5, it is clear that the time limit significantly impacts the time to the best solution found, demonstrating our algorithm's ability to continuously find new and improved solutions.

On the left, Figure 3.6a represents histograms, for each network and time limit, representing the number for solution for a specific time to best. From this figure it is possible to notice that a good number of incumbent solutions is found in the early stages of the optimization, independently from the actual T_{max} . Then, for $T_{max} = 5, 10$ the distribution tends to flatten, showing the algorithm's ability to find improving solutions even in later optimization stages. Such a flattening of the distribution is likely connected to the stochastic nature of the algorithm. In fact, improving on an already high quality solution is hard and mostly

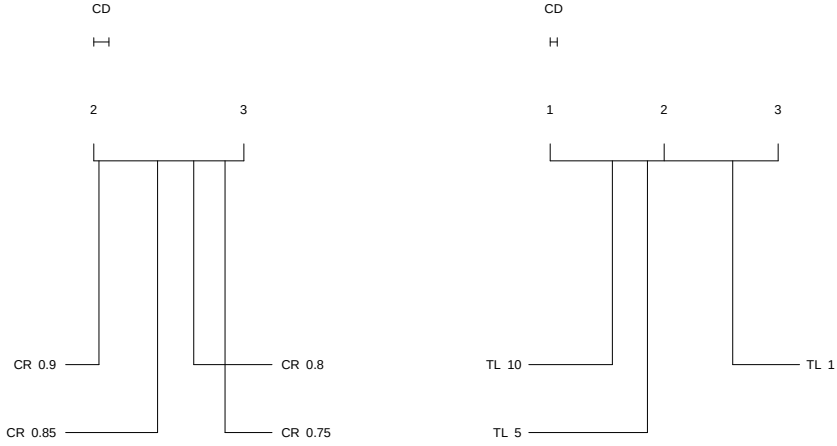


(a) Time to best histogram for each network and T_{max} (b) Time to best cumulative distribution for each network and T_{max}

Figure 3.6: Comparison of Time to Best statistics: (a) by Histogram, (b) by Empirical Cumulative Density

dependent on the right sequence of destroy and repair operators. On the right, Figure 3.6b shows, for each network, the empirical cumulative distribution function of the time to best with respect to each network and time limit. To improve the readability, the figure shows dots on the ECDFs lines representing, for each T_{max} , the value of the associated quantile. Several interesting patterns emerge from this figure. This figure gives a visual aid to understand the pattern emerging from Table 3.13. In fact, in the figure it is possible to notice that only in a small portion of the instances the incumbent value is found near the time limit. Moreover, it can be observed how close to each other the ECDFs are below the 50% line, and how they become almost indistinguishable below the 25% line.

To conclude, we present two critical distance plots (Demšar, 2006) to rank the different parameter values according to their overall performance. In these plots, a lower ranking value indicates better performance of a given configuration. The size of the Critical Distance (CD), computed using the Nemenyi test, determines when the test results can be considered significant (for more information on the Nemenyi test, see Tiit (2000)).



(a) Critical Distance (CD) by Cooling Rate (CR) (b) Critical Distance (CD) by Time Limit (TL)

Figure 3.7: Comparison of Critical Distance: (a) by Cooling Rate, (b) by Time Limit.

Figure 3.7a shows the CD ranking of the solution values under varying cooling rates. In this plot, the ranking is clear, with the distance between the scores consistently exceeding the critical distance. This image distinctly ranks a cooling rate of 0.9 as the best value among those tested. The second is 0.85, the third is 0.8, and the fourth is 0.75. Even if differences are minimal, a smoother cooling rate enhances the overall performance of our ALNS. A similar result is shown in Figure 3.7b, where the CD plot compares the solution quality of the ALNS under varying time limits. In this figure, all the scores exceed the critical distance, indicating that the ranking is significant. The best performing time limit is 10 minutes, followed by 5 minutes in second place, and 1 minute in third. This indicates that, in general, allowing the heuristic to run longer leads to better solutions. It is noteworthy that there is a significant gap between the one-minute and five-minute scores. However, the gap between the 5-minute and 10-minute scores is smaller. Increasing the time limit from 1 to 5 minutes multiplies the run time by five, while increasing it from 5 to 10 minutes only doubles the available time for the algorithm. Nonetheless, as seen in Table 3.9, the average TTB is below five minutes for each 10-minute ALNS execution. Furthermore, Figure 3.5b confirms this, showing that 75% of the executions find the incumbent well before the time limit.

Based on the performed CD ranking, a useful recommendation for decision-makers running this algorithm in an application is to configure the ALNS with a smooth cooling rate (≥ 0.85) and allow it to run for a relatively long time, more than 5 minutes. However, if a solution is needed very quickly, it is possible to run the algorithm with a short time limit and expect high quality results.

3.7.5 Sensitivity Analysis of System Efficiency

In this section, we perform a sensitivity analysis to evaluate how different values of the reward ρ granted to the crowdshippers (here referred to as *cs-cost*) and the backup cost γ_p affect the overall cost and efficiency of the delivery system. The *cs-cost* is the amount paid to the crowdshipper for delivering a parcel, while the backup cost is the expense incurred by the LSP when a parcel is delivered via the backup service. We assume that the *cs-cost* is

always lower than the backup cost, making crowdsipping generally preferable from a cost perspective.

We consider nine sets of instances, each representing a different combination of cs-cost and backup cost ranges. The cs-cost varies between the values [1; 2; 3], while the backup cost ranges are [4-8; 6-12; 8-16]. The backup cost for each parcel depends on its destination and is chosen from the corresponding range. We also vary the number of crowdshippers and parcels to be delivered for each set of instances. The number of crowdshippers takes the values [15; 30; 45], while the number of parcels varies between [10; 20; 30]. These are small instances to ensure that optimality can be reached and the obtained results are not biased. Each set of instances consists of nine subsets, each with a different combination of crowdshippers and parcels.

We can observe how these parameters influence the trade-off between cost and efficiency by varying the cs-cost and backup cost. Lower cs-cost and higher backup cost will encourage crowdsipping, as employing crowdshippers becomes more attractive than opting for the backup delivery alternative. The reverse is true when higher cs-cost and lower backup cost are imposed. We also expect that different public transportation lines will have varying levels of crowdsipping potential, depending on their network structure, frequency, and coverage.

Table 3.14 and Table 3.15 summarize the results of our sensitivity analysis. The average values are given for each set of instances relative to the percentage of parcels delivered by crowdshippers (in Table 3.14) and the objective function value (in Table 3.15). Columns are grouped by cs-cost ($\rho = 1, 2, 3$), with subcolumns representing backup cost ranges ($\gamma_p \in [4, 8]$, $\gamma_p \in [6, 12]$, $\gamma_p \in [8, 16]$), where the backup cost for each parcel is drawn uniformly from the specified range. Rows are labeled with two integers separated by a comma (e.g., **(10,15)**), where the first number indicates the total number of parcels to be delivered, and the second denotes the number of available crowdshippers. For example, the column group $\rho = 2$ with sub-column $\gamma_p \in [8, 16]$ corresponds to a cs-cost of 2 and backup cost ranging between 8 and 16, while the row **(30,45)** represents instances with 30 parcels to be delivered and 45 available crowdshippers.

Table 3.14: Sensitivity analysis – percentage of parcels delivered by crowdshippers across different cost configurations

(P , K)	$\rho = 1$			$\rho = 2$			$\rho = 3$		
	$\gamma_p \in [4, 8]$	$\gamma_p \in [6, 12]$	$\gamma_p \in [8, 16]$	$\gamma_p \in [4, 8]$	$\gamma_p \in [6, 12]$	$\gamma_p \in [8, 16]$	$\gamma_p \in [4, 8]$	$\gamma_p \in [6, 12]$	$\gamma_p \in [8, 16]$
(10,15)	2.50	26.66	30.00	0.00	15.55	28.33	0.00	10.00	26.67
(10,30)	8.75	56.11	60.43	5.83	42.22	62.77	0.00	20.55	48.23
(10,45)	14.54	70.00	81.81	7.22	53.33	82.77	0.00	41.66	63.89
(20,15)	0.94	14.72	18.00	0.83	14.16	17.77	0.00	7.78	15.83
(20,30)	7.50	32.77	33.33	2.22	25.00	33.88	1.38	20.55	28.88
(20,45)	24.25	50.27	59.44	18.30	43.05	51.38	3.61	32.22	39.79
(30,15)	1.67	11.66	13.51	3.15	10.37	12.40	1.11	9.07	11.85
(30,30)	4.63	21.37	22.74	8.33	16.27	21.76	0.92	16.66	18.82
(30,45)	19.44	40.55	43.51	12.96	35.92	40.74	3.89	32.04	36.85
Average	9.36	36.01	40.31	6.58	28.43	39.09	1.21	21.17	33.14

Table 3.15: Sensitivity analysis – objective function value across different cost configurations

(P , K)	$\rho = 1$			$\rho = 2$			$\rho = 3$		
	$\gamma_p \in [4, 8]$	$\gamma_p \in [6, 12]$	$\gamma_p \in [8, 16]$	$\gamma_p \in [4, 8]$	$\gamma_p \in [6, 12]$	$\gamma_p \in [8, 16]$	$\gamma_p \in [4, 8]$	$\gamma_p \in [6, 12]$	$\gamma_p \in [8, 16]$
(10,15)	57.13	86.88	109.87	61.72	87.27	110.11	59.83	92.00	111.94
(10,30)	57.25	76.94	98.00	59.58	82.11	95.17	58.28	86.88	101.94
(10,45)	57.09	71.00	87.00	58.72	79.11	92.39	57.83	84.38	96.27
(20,15)	111.62	171.27	220.40	120.10	177.88	226.11	120.66	174.44	224.94
(20,30)	111.16	162.72	209.91	116.30	173.22	212.22	118.61	173.05	217.72
(20,45)	107.75	147.72	179.44	117.50	155.55	191.38	121.33	164.88	204.92
(30,15)	170.88	257.66	336.88	178.30	262.61	339.33	176.38	266.22	350.44
(30,30)	174.77	245.94	320.17	178.00	254.35	327.35	180.61	258.55	334.52
(30,45)	162.96	229.05	293.14	178.89	240.44	300.56	181.34	256.15	306.17
Average	112.29	161.02	206.09	118.79	168.06	210.51	119.43	172.95	216.54

The results displayed in Table 3.14 and Table 3.15 confirm our expectations. The percentage of parcels delivered through crowdsipping and the total cost displayed by the objective function value are strongly influenced by the cs-cost and the backup cost. For example, when the cs-cost is set to 1 and the backup cost ranges between $[8, 16]$, we observe in Table 3.14 that over 80% of parcels are delivered by crowdshippers in the best-case scenario, with an average of 40.30% across the nine instance sets under investigation. On the other hand, when the cs-cost is set to 3 and the backup cost range lies within $[4, 8]$, we observe that crowdshippers do not deliver any parcels, as the cost of crowdsipping becomes comparable to the backup delivery cost, making it much less attractive overall. The average percentage of parcels delivered by crowdshippers in this cost setting is only 1.21%. An additional aspect worth noting from the results of our sensitivity analysis is linked to the ratio of crowdshippers available to the number of parcels to be delivered. We observe that an increasing number of crowdshippers relative to the number of parcels leads to a higher percentage of parcels delivered by crowdshippers. For example, considering the first column where cs-cost = 1 and the backup cost ranges between $[8, 16]$, we observe that the percentage of parcels delivered by crowdshippers increases from 30% to 81.81% as the number of crowdshippers increases from 15 to 45, while delivering the same number of parcels (i.e., 10). This is an important consideration when designing a crowdsipping delivery system, as ensuring a sufficient number of crowdshippers is available to deliver the parcels is crucial. Table 3.15 shows similar impacts in the trends for the objective function value. We can conclude that changes in the cs-cost and backup cost directly impact the delivery system, influencing both the total cost and the number of crowdshippers that can be employed. Therefore, it is crucial to carefully select the appropriate values for these costs.

To further enhance our sensitivity analysis, we also explore the impact of varying time-slot configurations on our crowdsipping delivery model. Our main testing framework is based on a full delivery day divided into $|T| = 85$ time-slots, each lasting $\Delta t = 5$ minutes. To challenge our approach and broaden our understanding, we conducted additional tests using different $(|T|, \Delta t)$ configurations, while still keeping the cs-cost and backup cost as defined previously (i.e., cs-cost = 2, and backup cost ranging between 12 and 20). Specifically, we examined the following configurations:

- (85,5): The day is divided into 85 time-slots, each lasting 5 minutes;
- (42,10): The day is divided into 42 time-slots, each lasting 10 minutes;
- (28,15): The day is divided into 28 time-slots, each lasting 15 minutes;
- (21,20): The day is divided into 21 time-slots, each lasting 20 minutes.

Upon analyzing the results shown in Table 3.16, we observe that the objective function value for the first three configurations (85,5), (42,10), and (28,15) shows only minor variations. More specifically, a modest increase in the objective function value was observed when comparing (85,5) with (42,15) across all instance groups. However, a more pronounced difference emerged when comparing the standard (85,5) and the (21,20) configurations. In the instance set (30,45), the objective function value increased by 9.11% moving from the (85,5) to the (21,20) configuration, highlighting that reducing the number of time-slots significantly impacts overall performance, even for small instances like those under investigation. Ad-

Table 3.16: Sensitivity analysis – objective function value across different time-slot configurations

(P , K)	$(T , \Delta t)$			
	(85,5)	(42,10)	(28,15)	(21,20)
(10,15)	138.58	142.66	141.50	150.33
(10,30)	122.21	127.13	133.33	138.73
(10,45)	109.47	114.60	116.69	135.16
(20,15)	284.91	287.12	293.33	299.21
(20,30)	269.41	277.34	275.87	289.37
(20,45)	235.77	243.18	256.29	279.65
(30,15)	446.00	459.33	451.25	452.91
(30,30)	422.86	413.41	433.47	443.39
(30,45)	389.52	392.05	410.25	425.00
Average	285.02	272.98	296.31	290.42

ditionally, as seen in Table 3.17, it becomes even more evident that reducing the number of available time-slots significantly affects the percentage of parcels delivered via crowdsipping. This effect is most pronounced in the (21,20) configuration, as shown by the data in the last column.

Managerial Insights

Based on the results obtained through our sensitivity analysis, we can draw some key managerial insights that should be taken into consideration while designing a PT-based crowdsipping delivery system:

- **Cost structure:** The relationship between the cs-cost and the backup delivery cost is crucial. Our results show that when the cs-cost is low (i.e., 1) and the backup cost is high (ranging from 8 to 16), an average of 40.31% of parcels are delivered by crowdsippers. However, this value drops dramatically to just 1.21% when the cs-cost increases to 3 and the backup cost decreases to the [4, 8] range. Therefore, to maximize crowdsipping utilization and minimize total delivery costs, maintaining a favorable cost ratio is essential.

Table 3.17: Sensitivity analysis – percentage of parcels delivered by crowdshippers across different time-slot configurations

(P , K)	$(T , \Delta t)$			
	(85,5)	(42,10)	(28,15)	(21,20)
(10,15)	2.76	3.16	2.25	1.16
(10,30)	6.52	5.00	6.20	3.13
(10,45)	8.76	9.08	9.34	5.00
(20,15)	3.08	4.08	3.00	2.08
(20,30)	5.91	5.95	6.12	4.08
(20,45)	11.90	10.80	10.90	6.39
(30,15)	3.54	2.95	3.08	1.34
(30,30)	5.63	7.12	5.78	4.52
(30,45)	11.60	11.80	11.20	7.09
Average	7.12	7.10	7.06	4.20

- **Crowdshippers availability:** Not surprisingly, increasing the number of available crowdshippers relative to the number of parcels to be delivered significantly improves the system efficiency. For example, with 10 parcels to be delivered, increasing the number of crowdshippers from 15 to 45 raises crowdshipping utilization from 30% to 81.81% in the most favorable cost scenario. This underscores the importance of incentivizing and maintaining a large, active crowdshipper network.
- **Scalability:** As the delivery volume increases, crowdshipping efficiency tends to decrease. When the cs-cost is set to 1 and the backup cost ranges in $[8,16]$, crowdshipping utilization drops from 81.81% for 10 parcels to 43.51% for 30 parcels (with 45 available crowdshippers). This provides useful information on the necessity of defining different distribution strategies based on the volume of parcels to be delivered.
- **Pricing:** The significant impact of the cs-cost on system performance (e.g., crowdshipping utilization dropping from 40.31% to 1.21% as the cs-cost increases from 1 to 3) suggests that pricing is a decisive factor for the ultimate success of crowdshipping delivery.
- **Network effect:** Larger systems with more crowdshippers and parcels to be delivered tend to achieve higher crowdshipping percentages. For example, when the cs-cost is set to 1 and the backup cost ranges in $[8,16]$, the 30-45 (parcels–crowdshippers) configuration achieves 43.51% crowdshipping utilization, compared to 30% for the 10-15 configuration. This implies that there may be a “critical mass” that needs to be achieved to make the system have a positive impact.
- **Backup delivery:** Even in the most favorable conditions (when the cs-cost is set to 1 and the backup cost ranges in $[8,16]$, 10 parcels to be delivered, 45 available crowdshippers), 18.19% of parcels still rely on backup delivery. This highlights the necessary role of the backup system in ensuring service reliability. When implementing such crowdshipping delivery frameworks, managers should not neglect the optimization of this backup system.

- **Route-specific strategies:** Although not explicitly quantified in our analysis, different PT lines have varying crowdsipping potentials. This implies that while implementing a PT-based crowdsipping system, one should consider developing and implementing route-specific strategies to maximize efficiency across different parts of the network.

By considering such factors, stakeholders can potentially make more informed decisions. The complex relationship between different factors in the success of a PT-based crowdsipping delivery system suggests the need for a more holistic approach, where changes in one area are carefully evaluated for their impact on the performance of the entire system.

3.7.6 Sensitivity to Travel Delays and Path Length Variability

While our problem is formulated within a deterministic environment, assuming PT adheres to predefined timetables, it is essential to consider system performance in real-world scenarios, where delays and disruptions may impact travel times, potentially altering parcel pickup schedules and rendering some delivery plans infeasible. To assess this aspect, we first examine the distribution of parcel path lengths (i.e., the number of crowdshippers required to transfer a parcel) in the solutions obtained by our ALNS. Next, we refine the analysis by focusing on parcel paths involving multiple crowdshippers, specifically analyzing the temporal gap between parcel drop-off and subsequent pickup at exchange stations. These metrics offer valuable insights into the robustness of the obtained solutions when faced with real-world timing variations. Finally, the analysis focuses on the impact of delays introduced along a predefined PT line, simulating disruptions to assess their effect on optimal solutions and the system's resilience.

Table 3.18: Percentage of parcel paths for each length, grouped by network size $|N|$ and number of available crowdshippers $|K|$

$ N $	Path Length	$ K $					
		75	100	150	200	250	300
24	1	89.22%	79.93%	81.15%	78.62%	84.40%	78.98%
	2	10.76%	19.84%	18.35%	20.94%	14.99%	20.63%
	3	0.02%	0.23%	0.50%	0.43%	0.60%	0.38%
	4	0.00%	0.00%	0.00%	0.00%	0.01%	0.01%
36	1	93.33%	91.59%	94.51%	93.65%	93.96%	93.83%
	2	6.67%	8.41%	5.35%	6.19%	5.87%	5.99%
	3	0.00%	0.00%	0.14%	0.16%	0.17%	0.16%
	4	0.00%	0.00%	0.00%	0.00%	0.00%	0.02%
40	1	94.45%	96.30%	93.92%	89.06%	90.38%	85.64%
	2	5.55%	3.66%	5.95%	10.39%	9.21%	13.73%
	3	0.00%	0.04%	0.14%	0.56%	0.42%	0.63%
	4	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
44	1	84.02%	86.20%	86.92%	81.16%	81.70%	70.15%
	2	14.58%	13.32%	12.95%	16.20%	17.15%	25.37%
	3	1.40%	0.48%	0.13%	2.60%	1.15%	4.24%
	4	0.00%	0.00%	0.00%	0.04%	0.00%	0.24%

Table 3.18 presents the distribution of parcel path lengths across the various network

configurations. Specifically, for each network size (denoted by $|N|$ in the first column) and number of available crowdshippers (indicated by $|K|$ in columns 3 to 8), the table shows the percentage of paths requiring from 1 to 4 parcel transfers (**Path Length** column). Table 3.18 shows a consistent pattern across all network configurations: paths involving a single crowdshipper dominate. Their share is lowest at 70.15% for $|N| = 44$, $|K| = 300$ and highest at 96.30% for $|N| = 40$, $|K| = 100$. These single-crowdshipper paths remain unaffected by typical service disruptions, as they do not require intermediate transfers and synchronization.

The high share of single-crowdshipper deliveries results from the objective function's structure, which inherently minimizes the number of involved crowdshippers to reduce costs. Since using fewer crowdshippers is the most cost-effective option under the reward scheme, our method assigns multiple crowdshippers only when necessary. This is evident in the $|N| = 44$, $|K| = 300$ case, where paths involving three (resp. four) crowdshippers reach just 4.24% (resp. 0.24%). The effectiveness of this mechanism depends on the network topology, as seen in Table 3.18, where instances with $|N| = 44$ display a higher share of multiple-crowdshipper parcel paths.

Moreover, the cost structure also influences parcel path length, particularly the crowdshipper reward ρ and backup cost γ_p . Ignoring σ_i and v_i for simplicity, a parcel $p \in P$ cannot be assigned to a path longer than $\left\lfloor \frac{\gamma_p}{\rho} \right\rfloor$, as such paths would exceed the backup delivery cost. This underscores that selecting ρ in relation to γ_p and network topology, while considering business constraints such as minimum crowdshipper involvement and commuter patterns, can also promote shorter paths, reducing exposure to travel disruptions. While single-crowdshipper parcel paths remain unaffected by typical travel disruptions, those requiring synchronization (ranging from 3.70% to 29.85% of cases, depending on the network configuration; Table 3.18) may be impacted. To evaluate this risk, we analyze the waiting time between parcel drop-off at exchange stations and pick-up by the next crowdshipper, assessing the likelihood of failed transfers due to upstream delays.

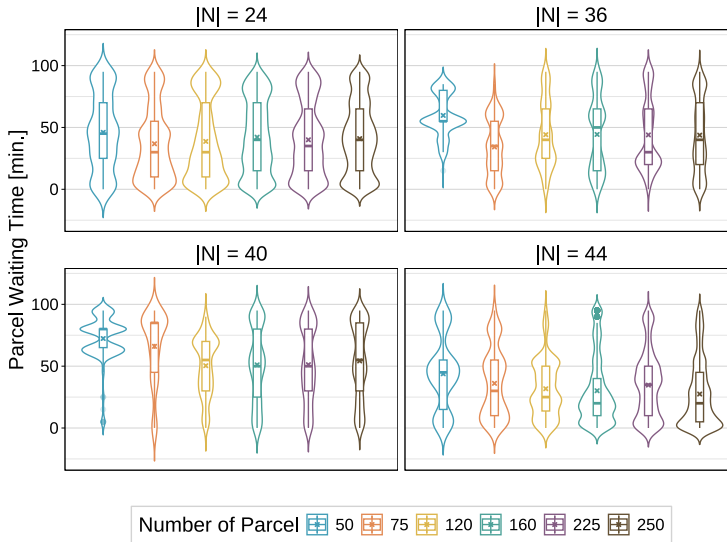


Figure 3.8: Distribution of parcel waiting times at exchange stations for paths involving multiple crowdshippers.

Figure 3.8 shows violin plots depicting the distribution of parcel waiting times at exchange stations, with each quadrant representing a different network size and further subdivided by the number of parcels per instance. Inner boxplots highlight the 25th, 50th, and 75th percentiles, while the central cross marks the average waiting time. Across all network sizes and parcel counts, the average waiting time remains around 45 minutes. The shortest average waiting time, just over 27 minutes, is observed for $|N| = 44$ and $|P| = 250$. Examining network-wide averages, we find approximately 40 minutes for $|N| = 24$, 43 minutes for $|N| = 36$, 53 minutes for $|N| = 40$, and 31 minutes for $|N| = 44$.

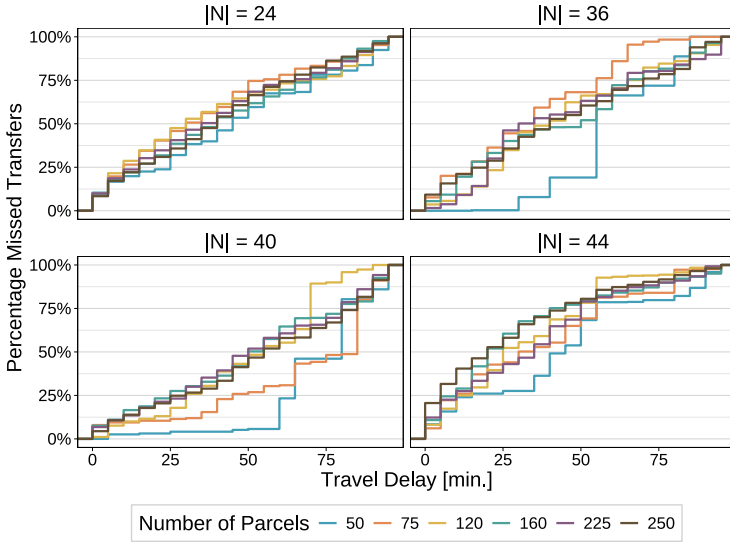


Figure 3.9: Percentage of parcels missing planned transfers at exchange stations as a function of the incurred travel delay.

Overall, the distributions in Figure 3.8 indicate that, for the parcel deliveries requiring synchronization between multiple crowdshippers, the waiting time at exchange stations is generally sufficient to prevent missed transfers. Figure 3.9 further illustrates this by showing the percentage of parcels involving two or more crowdshippers that fail to transfer at an exchange station (y-axis) as a function of the incurred travel delay before reaching that station (x-axis). The developed crowdshipping framework demonstrates strong resilience to typical travel delays, particularly in reliable transport systems such as trains, metros, and dedicated-lane buses. The results indicate that networks with $|N| = 36$ (resp. $|N| = 40$) are the most robust, with failure rates below 25% for delays under 20 (resp. 30) minutes and below 50% for delays up to 40 (resp. 55) minutes. The failure rate would increase to 50% in the case of travel delays larger than 35 (resp. 25) minutes.

Additional experiments were conducted to evaluate the resilience of the crowdshipping framework, considering an extreme configuration with $|P| = 30$ parcels, $|K| = 30$ crowdshippers, and travel delays ranging from 20% to 50% of the scheduled travel time. The network used for these tests has $|N| = 44$ nodes and $|L| = 7$ PT lines.

Figure 3.10 displays the distribution of the number of parcels delivered by crowdshippers under different delay conditions applied to each network line separately. Due to the reduced number of available crowdshippers, only a portion of the parcels can be assigned to crowd-

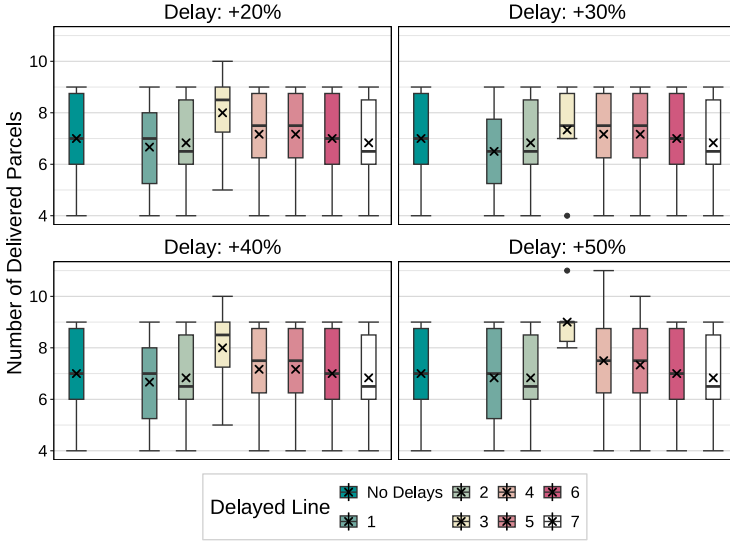


Figure 3.10: Number of parcels delivered in presence of travel delays for instances involving $|P| = 30$ parcels, $|K| = 30$ available crowdshippers, $|N| = 44$ stations and $|L| = 7$ public transportation lines.

shipping in this setup. The cross of each box plot (each line is associated with a different color) represents the average number of parcels delivered in each scenario. The black box plot in the leftmost part of each quadrant corresponds to the base case without any travel delays, serving as a benchmark for comparison. The impact of travel delays depends on which PT line is affected. Interestingly, delays on certain lines can lead to an improvement in the number of parcels ultimately delivered. For example, when line 3 is delayed, the number of delivered parcels is never reduced by more than 4.76%, but it can, in the best case, increase by 28.57%. A similar improvement, though smaller, is also observed for lines 4 and 5. This suggests that delays in certain parts of the network can create new synchronization opportunities, ultimately leading to more crowdshipping possibilities. However, these new opportunities come at a cost, as alternative crowdshippers must be compensated with a reward. In contrast, for line 1, the presence of delays consistently decreases the number of parcels delivered, as they cause missed transfers between crowdshippers. The impact is minimal, with a reduction of 7.14% (resp. 2.38%) in parcels delivered for 30% (resp. 50%) delays. For lines 2 and 7, the number of delivered parcels remains constant despite the incurred delays, indicating that these lines do not serve as primary bottlenecks limiting the system's delivery performance.

These additional experiments confirm that the system's resilience to travel delays is mainly influenced by two factors: the timing of crowdshipper movements and the network topology. The timing factor is especially evident for line 3, where the number of parcels delivered varies with increasing delays, suggesting that certain delivery routes become feasible or infeasible depending on the delay conditions. Regarding the network topology, the length and connectivity (number of intersections) of PT lines impact the effect of travel disruptions, as shown by the variations in Figure 3.10 across the different lines.

In summary, the analysis highlights several key points. First, the system is inherently

resilient to delays, primarily due to the majority of parcels being delivered by a single crowdshipper, reducing the need for precise synchronization. This is a result of the objective function, which minimizes the delivery costs and, indirectly, the number of crowdshippers involved. Second, for parcel paths requiring multiple crowdshippers, resilience to synchronization issues depends on the network topology and commuter patterns, with most deliveries unaffected by limited delays. Third, when a parcel transfer is missed, the application of dynamic rerouting procedures could ensure delivery by reassigning the parcel to a different crowdshipper. The development of such dynamic algorithms, which would also help handle situations where crowdshipper availability changes on short notice, requires a deeper understanding of the realistic time frames within which commuters can respond to crowdshipping requests. As such, we leave it as a promising avenue for future research. To conclude, travel delays would only significantly impact the system in rare cases involving multiple crowdshippers, missed transfers, and no alternative crowdshipping options, with the need for backup services remaining limited.

3.8 Environmental Impact Comparison: Crowdshipping versus Traditional Delivery Systems

This section analyzes how a crowdshipping system can reduce environmental impact compared to a scenario in which an LSP vehicle handles all deliveries. Specifically, we compare the total travel effort of the LSP in two scenarios: (i) the one derived from the optimal crowdshipping solution, where the LSP operates in the morning to deliver parcels to source stations and in the evening to perform backup deliveries for parcels not served by the crowdshippers, and (ii) a hypothetical case in which the LSP delivers all parcels directly to the destination stations. For scenario (i), we estimate the LSP's travel effort by solving two Traveling Salesman Problems (TSPs): one in the morning, covering the source stations (identified in our model's optimal solution), and another in the evening visiting the destination stations for parcels not handled by crowdshippers. In contrast, for scenario (ii), a single TSP is solved over the destination stations for all parcels.

We apply this analysis to a representative instance ($|N| = 24$, $|K| = 75$, $|P| = 50$, $Id = 3$), solved to optimality, where each parcel has a fixed, predefined destination station. Starting from this instance, we generate three additional ones by incrementally increasing the number of available crowdshippers, adding 75 randomly generated ones at each step. In this way, we analyze how the environmental footprint of the crowdshipping system evolves as the number of participants increases, thereby expanding the potential volume of parcels delivered via crowdshipping.

Table 3.19 presents the results and comparative analysis between the crowdshipping system (i) and the traditional delivery scenario (ii). All instances were solved to optimality using our mathematical model: column **Obj.** reports the optimal objective function value (i.e., optimal crowdshipping system cost), while column **#** indicates the number of parcels delivered by crowdshippers in the optimal solution. As the number of crowdshippers increases, a larger share of parcels is delivered through the public transport network, significantly reducing the dependence on LSP backup intervention and the associated global cost. This trend is captured by **Obj.** value which decreases from 629 to 340 as the number of crowdshippers grows from 75 to 300, and by the **#** value which shows a corresponding increase in the number of parcels delivered by crowdshippers from 15 to 50 (all parcels delivered by the crowdshippers). These results confirm the ability of our mathematical model to improve

cost-efficiency as more crowdsipping options become available.

The remaining columns in Table 3.19 report the estimated travel effort required by the LSP under both delivery systems, expressed as total travel times (in seconds). Under the **Crowdsipping System**, the morning, the evening and the total travel times are reported for each instance. While morning travel times remain constant across instances, yielding 3354 seconds and involving the same 3 source stations, as the number of available crowdsippers progressively increases, the travel time required by the backup service (**Evening CS**) decreases accordingly, as fewer parcels remain unassigned to crowdsippers. This leads to a reduced travel effort for the LSP, from 5439 seconds when 35 parcels require LSP delivery with 75 crowdsippers, down to 0 seconds with 300 crowdsippers, when the crowdsipping system can handle all parcel deliveries. In the **Traditional Delivery System**, where all parcels are delivered solely by the LSP, a single TSP is solved on the full set of destination stations, resulting in a constant travel time of 7043 seconds across all instances. Finally, column **Impr.** shows the percentage improvement of the total LSP travel time in the crowdsipping system over the traditional one, highlighting the reduction in vehicle routing effort due to the involvement of crowdsippers. Interestingly, when the number of available crowdsippers is low, as in the first instance, the traditional delivery system actually outperforms the crowdsipping-based approach, with column **Impr.** showing a negative value of -24.5%. However, as the number of crowdsippers increases, the trend reverses: with 150 crowdsippers, the crowdsipping system already achieves a positive improvement of 4.88%, which further rises to 52.38% when 300 crowdsippers are available.

Table 3.19: Travel time comparison between the Traditional Delivery System and the Crowdsipping System under varying numbers of crowdsippers

K	Optimal solution		Crowdsipping System			Traditional Delivery System	
	Obj.	#	Morning CS	Evening CS	Total CS	TSP	Impr.
75	629.0	15	3354	5439	8793	7043	-24.85%
150	474.0	33	3354	3345	6699	7043	4.88%
225	377.0	48	3354	1674	5028	7043	28.61%
300	340.0	50	3354	0	3354	7043	52.38%

While the initial comparison is based on travel times, the analysis becomes more impactful when directly considering CO₂ emissions, offering a clearer assessment of the crowdsipping system's effectiveness in reducing the LSP's environmental footprint. Assuming an average driving speed of 30 km/h for the LSP vehicle, travel times can be converted into distances. Considering a conservative fuel consumption of 8.5 liters of diesel per 100 km, typical for a modern urban delivery van, we estimate that traditional delivery systems that require an average daily travel time of 7043 seconds (approximately 58,69 km) would produce approximately 23 kg of CO₂ per day. In contrast, in the best crowdsipping scenario, with $|K| = 300$, the total travel time (corresponding only to the morning parcel transfer) is equal to 3354 seconds (about 27,5 km) resulting in about 11 kg of CO₂ emissions. If we assume the service operates 300 days per year and these figures are taken as representative averages, the crowdsipping system would prevent the emission of approximately 3.6 tons of CO₂ annually.

3.9 Conclusions

In this chapter, we investigated optimization strategies for the Public Transportation-based Crowdsipping Problem (PTCP), integrating public transportation (PT) networks into last-mile delivery systems. The proposed framework employs PT stations equipped with automated parcel lockers (APLs), whose location problem was dealt with in Chapter 2, as collection and transfer points, while commuters act as crowdshippers to transfer parcels along their regular journeys. A backup delivery service is included to ensure the completion of deliveries for parcels not successfully transferred through crowdsipping.

We formulated the PTCP as a Mixed-Integer Linear Programming (MILP) model, explicitly incorporating synchronization constraints to capture the operational characteristics of the system. The model minimizes total delivery costs by jointly optimizing the use of crowdshippers and the backup service. To address the computational complexity of the problem, we developed two complementary solution methods: an Adaptive Large Neighborhood Search (ALNS) metaheuristic and a Branch-and-Cut algorithm strengthened with multiple valid inequalities.

A broad set of synthetic instances, generated from randomized PT networks of different sizes and topologies, was employed to evaluate the effectiveness of the proposed approaches. Computational experiments revealed the limitations of a commercial solver in handling large-scale instances of PTCP, while the ALNS algorithm demonstrated efficiency in producing high-quality solutions within short computation times. In small instances, CPLEX occasionally outperformed the heuristic, although the ALNS consistently achieved near-optimal solutions with marginal optimality gaps.

An extensive sensitivity analysis highlighted the decisive influence of cost structures, particularly the relationship between crowdsipping remuneration and backup delivery costs, on overall system performance. The results also emphasized the importance of crowdshipper availability and system scale in ensuring efficient operations. Further analyses investigated the effects of time-slot discretization, travel delays, and path length variability, confirming the robustness of the framework under realistic conditions.

The proposed deterministic model establishes a balance between tractability and practical relevance, offering a foundation for the study of sustainable, PT-based last-mile delivery. Future research could enrich the framework by incorporating stochastic elements such as uncertain travel times or dynamic commuter availability. Promising directions include the integration of real-time coordination mechanisms, the design of refined pricing and incentive schemes, and the possibility of limited route deviations to enhance delivery flexibility. Finally, collaboration between public transport operators and logistics providers may lead to dedicated infrastructure and coordination protocols, facilitating the adoption of sustainable, PT-integrated crowdsipping systems.

Chapter 4

A Column Generation Algorithm for PTCP

This chapter present an exact algorithm to tackle the problem proposed in Chapter 3. This work has been presented at International Conference on Optimization and Decision Science (ODS) 2025, in Milan, Italy.

Contents

4.1	Problem Definition	76
4.1.1	Set Covering Formulation for PTCP	78
4.2	Introduction to Column Generation and Branch-and-Price	79
4.2.1	Why Branch-and-Price?	79
4.2.2	Column Generation	80
4.2.3	Branch-and-Price	81
4.3	Branch-and-Price Algorithm	82
4.3.1	Derivation of the Pricing Problem	82
4.3.2	Pricing Problem	84
4.3.3	Feasibility Cuts	88
4.3.4	Branching Rules	88
4.4	Experimental Results	90
4.4.1	Instance Generation	90
4.4.2	Analysis of Total Solution Time	90
4.4.3	Impact of the Percentage of Backup Parcels	94
4.4.4	Analysis of the Branch-and-Bound Tree	95
4.4.5	Analysis on the Number of Generated Columns	96
4.4.6	Comparison of the Solution Methods for PTCP	98
4.5	Conclusions	100

In Chapter 3 PTCP has been solved using a compact MILP, having a polynomial number of variables and constraints with respect to the instance size, formulation and an ALNS meta-heuristic. The first method proved itself useful to tackle only small-sized instances. On the other hand, the ALNS has been able to solve large-sized instances with ease. However, relying on a heuristic has a key limitation: solutions are generally not guaranteed to be optimal,

and even when they are, optimality cannot be certified. This may cause higher operating costs and, consequently, reduced profitability. BnP is an exact optimization framework that merges Branch-and-Bound with Column Generation. Branch-and-Bound systematically explores a search tree of subproblems to enforce integrality, using bounds to prune regions that cannot contain an optimal integer solution. Within BnP, the linear programming (LP) relaxation at each node is not solved over the full (huge) variable set; instead it is handled by Column Generation (CG), a decomposition technique designed for LPs with an extremely large number of variables. A Column Generation procedure starts from a restricted master problem containing only a small subset of variables (columns), solves its LP relaxation, and then checks whether additional columns could improve the solution. This check is performed by solving an auxiliary optimization subproblem known as the pricing problem (often called the *pricer*). The pricer uses information from the current LP solution (the dual prices) to search for a new column with negative reduced cost (i.e., a variable that would improve the objective if added to the master problem). If such a column is found, it is appended to the master model and the process repeats. If none exists, the current solution is optimal for the full LP relaxation, even though most variables were never explicitly generated. Specialized branching rules are usually in need to tighten integrality and keep a balanced search tree.

BnP algorithms are among the leading methods to exactly solve a plethora of combinatorial optimization problems including vehicle routing, packing, and scheduling.

4.1 Problem Definition

In this section, we first provide a formalization of the problem, extending the one proposed in Section 3.3 with sets and parameters required by the subsequently introduced set covering formulation.

We formulate the PTCP on a public transport network whose nodes are the stations equipped with parcel lockers, denoted by the set N . The locker capacity at station $i \in N$ is C_i . A subset $S \subset N$ of the stations are a priori selected by the LSP to act as *source stations*. Parcels at these source station are loaded, in the early morning, by a truck starting from a central depot. Each source station $i \in S$ is associated with two cost parameters: v_i , representing the truck driving cost of reaching station i , and σ_i , denoting the handling cost for loading a single parcel into the locker at i . All remaining stations, collected in the set $D = N \setminus S$, may be chosen by recipients as pickup locations and can also serve as intermediate nodes for parcel transfers.

P is the set of parcel that have to be delivered. Each parcel has a fixed destination station, decided by the its recipient, $i \in D$. Notice that we assume that parcel destinations belong to the set of non-source stations $D = N \setminus S$. Indeed, if a parcel were destined to a source station $i \in S$, it could be deposited directly by the logistics service provider when replenishing station i , and no crowdshipping transfer would be required. Therefore, such deliveries are not modeled within the PTCP and are excluded without loss of generality.

K represents the set of available crowdshippers. Each crowdshipper $k \in K$ has a start and destination station of their commute. We assume that a crowdshipper always takes the shortest path in the network. A crowdshipper can transfer parcels only over this path, from any station (but the last one) to any subsequent one. Moreover, each crowdshipper is associated with an arrival time at their origin station. If, given this start time, the crowdshipper would reach their final destination after the end of the planning horizon, the route is truncated at the last station that can be reached within the horizon. Each crowdshipper receives for their service a fixed remuneration ρ , independent from the length of the path taken with

the parcel.

To ensure proper synchronization between parcels and crowdshippers and to enforce locker-capacity constraints, the proposed model adopts a discrete-time representation. We identify with T the set of the equally sized discrete time slots. We assume that the size of a time slot is such that for a crowdshipper, it is possible to pick up a parcel and to board the public transport vehicle within it. Moreover, we assume that within any given time slot a parcel can be picked up or dropped off at most once, i.e., it cannot undergo multiple handling operations in the same time frame.

A backup service is in place for all the parcel that cannot be delivered by crowdshippers. Each station $i \in D$ is associated with a backup γ_i . Parcels that cannot be delivered through crowdshipping, either because no suitable crowdshipper is available or because the cheapest feasible crowdshipping route is more expensive than the fallback option, are delivered directly to their destination station.

In Table 4.1 it is possible to get a summary of the sets used in the mathematical formulation

Table 4.1: Sets

Name	Description
P	Set of all parcels to be delivered
K	Set of the available crowdshippers
N	Set of stations with lockers
T	Set of discrete time slots
S	Set of source stations
D	Set of destination stations
R_p	Set of paths for parcel $p \in P$

The novel formulation proposed in this chapter selects, for each parcel, the most efficient parcel path, i.e. the one that will allow to minimize the overall delivery cost. A parcel path can either be a sequence of transfers, done in sequence by distinct crowdshippers, from any station in S to the destination station, or the backup transfer. We denote with R_p the set of all feasible paths for parcel $p \in P$. Each path has associated three classes of parameters. First, binary parameter a_{pk}^r takes value 1 if parcel $p \in P$ is transferred by crowdshipper $k \in K$ over route $r \in R_p$ and 0 otherwise. Second, binary parameter s_{pi}^r takes value 1 if parcel $p \in P$ uses station $i \in N$ as loading station when transferred using route $r \in R_p$ and 0 otherwise. Finally, binary parameter o_{it}^{pi} takes value 1 if parcel $p \in P$, transferred along route $r \in R_p$, occupies station $i \in N$ at time $t \in T$ and 0 otherwise. In the first case, where parcel $p \in P$ is transferred by crowdshippers, c_p^r is proportional to the number of crowdshippers employed in the transfer plus the handling cost to load the parcel in the locker at the source station. Moreover, the a_{pk}^r associated with the employed crowdshippers, the s_{pi}^r associated with the starting station, and the o_{it}^{pi} associated with the intermediate stations are all set to 1. The other parameters take value 0. In the second case, where the parcel is handled by the backup service, the cost is equal to the backup cost for the destination station. In such paths, all associated parameters a_{pk}^r , s_{pi}^r , and o_{it}^{pr} take value 0. The parameters used in the model are summarized in Table 4.2.

The selection of the parcel path is done by the binary variable x_p^r , which takes value 1 if parcel $p \in P$ is transferred with path $r \in R_p$. The activation of the cost v_i for the source station $i \in S$ is managed by the variable z_i , taking value 1 if the source station is used, zero

Table 4.2: Parameters

Name	Description
c_p^r	cost to move parcel $p \in P$ over route $r \in R_p$
a_{pk}^r	Binary parameter takes value 1 if parcel $p \in P$ is transferred by crowdshipper $k \in K$ over route $r \in R_p$; 0 otherwise
s_{pi}^r	Binary parameter takes value 1 if parcel $p \in P$ uses station $i \in N$ as loading station when transferred using route $r \in R_p$; 0 otherwise
o_{it}^{pr}	Binary parameter takes value 1 if parcel $p \in P$, transferred along route $r \in R_p$, occupies station $i \in N$ at time $t \in T$; 0 otherwise

otherwise.

4.1.1 Set Covering Formulation for PTCP

The required variables are presented in Table 4.3.

Table 4.3: Variables

Name	Description
x_p^r	1 if parcel $p \in P$ is transferred via route $r \in R_p$, 0 otherwise
z_i	1 if station $i \in N$ is used as loading station, 0 otherwise

The Set Partitioning model for PTCP is as follows:

$$\min \sum_{p \in P} \sum_{r \in R_p} c_p^r x_p^r + \sum_{i \in S} v_i z_i \quad (4.1)$$

s.t

$$\sum_{r \in R_p} x_p^r \geq 1 \quad \forall p \in P \quad (4.2)$$

$$\sum_{p \in P} \sum_{r \in R_p} a_{pk}^r x_p^r \leq 1 \quad \forall k \in K \quad (4.3)$$

$$z_i \geq \sum_{r \in R_p} s_{pi}^r x_p^r \quad \forall i \in S, p \in P \quad (4.4)$$

$$\sum_{p \in P} \sum_{r \in R_p} o_{it}^{pr} x_p^r \leq C_i \quad \forall i \in N, t \in T \quad (4.5)$$

$$x_p^r \in \{0, 1\} \quad \forall p \in P, r \in R_p \quad (4.6)$$

$$z_i \in \{0, 1\} \quad \forall i \in S \quad (4.7)$$

Objective function (4.1) minimizes the overall delivery costs, made of the costs for deliver each parcel, either by crowdshipping or backup service, and shipment costs from the depot to the source stations. Constraints (4.2) impose that each parcel has to be delivered. Constraints (4.3) allow each crowdshipper to handle at most one parcel during their commute. Constraints (4.4) force activation of the shipment cost to station i , if it is used as source station. Constraints (4.5) impose to respect each station's locker capacity, at each time step. Finally conditions (4.6) and (4.7) define the variables and restrict their domains.

4.2 Introduction to Column Generation and Branch-and-Price

Column Generation (CG) can be seen as a decomposition variant of the simplex method for linear programs with an extremely large number of variables, typically far exceeding the number of constraints. In such methods, the identification of a variable candidate to enter the basic solution, and therefore the identification of an optimal solution, is done by an optimization problem rather than complete enumeration of the out-of-base variables. Branch-and-Price (BnP) refers to a CG algorithm embedded in a Branch-and-Bound framework to enforce integrality. The reader interested in a deeper discussion about the subject is referred to the excellent papers Feillet (2010); Costa et al. (2019); Petris et al. (2025) and books from Desaulniers et al. (2006); Uchoa et al. (2024); Desrosiers et al. (2024).

4.2.1 Why Branch-and-Price?

The vast majority of problems can be expressed using a compact, therefore with a polynomial number of variables and constraints with respect to the instance size, MILP. Once devised, such formulations are simple to implement using any black-box solver, which, given enough time and memory, will return an optimal solution. The solver uses a variation of the Branch-and-Cut algorithm described in Algorithm 3.4. Such algorithms, first solve the linear relaxation, i.e. integrality constraints are removed, and then enforce integrality with valid inequalities and with branching. This procedure, however, is as effective as the linear relaxation is close to the integer optimal. So, to solve exactly complex large-scale integer problem, one needs a formulation with a better linear relaxation.

Such reformulations, either hand-made or using machinery like Dantzig–Wolfe decomposition (Dantzig and Wolfe, 1961), take a set partitioning or set covering form. There, a reformulated variable correspond to, ideally, more than one original variable, therefore representing *a piece* of the original formulation. Set partitioning, or covering, constraints ensure that all required pieces are used, and the objective function minimizes the cost. For instance, take formulation (4.1)–(4.7) and compare it with formulation (3.1)–(3.23). (4.1)–(4.7) has a variable to represent an entire parcel path, which consists of several low-level decisions taken by (3.1)–(3.23). In the compact flow-based formulation, a parcel’s route must be described through two types of variables: flow variables, indicating where the parcel moves, and synchronization (time-indexed) variables, indicating when the parcel is at each location. All these variable are merged into one unique variable in the set covering formulation. Set covering objective is to select the cheapest parcel paths among the available ones in order to deliver all possible parcels. Naturally, since there an exponentially large number of parcel paths, the set covering formulation requires an exponential number of variables, one for each possible way to transfer a parcel through the network.

Such reformulations provide a tighter linear relaxation than the original counter part, making it easier to enforce integrality. Intuitively, it can be noticed by the fact that one reformulated variable corresponds to several original ones. By construction, it is possible to map any integer solution from one formulation to the other. However, this does not hold for continuous solutions. It can be noticed that a reformulated linear relaxation can always be mapped into an original’s formulation relaxed solution: All original variables associated with a reformulated one must take the same value. On the other hand, if the original variables take distinct values, the mapping from an original solution into a reformulated one is not possible. Therefore there are valid continuous solution of the original formulation that are

invalid for the reformulated formulation, making the latter's linear relaxation tighter.

Such reformulation has, as already discussed, a major drawback: they require an exponential number of variables. Therefore, it is not only computationally intractable, but it is also pointless to run a MILP after all possible solutions have been enumerated. In the next sections we introduce the two key steps to handle such non-compact formulations. First, in Section 4.2.2 we provide an introduction to Column Generation, and later, in Section 4.2.3 we provide a description of the Branch-and-Price method.

4.2.2 Column Generation

Let (4.8)–(4.10) be generic minimization linear program defined over variable set I and constraint set J :

$$\min \sum_{i \in I} c_i x_i \quad (4.8)$$

s.t.

$$\sum_{i \in I} a_{ij} x_i \geq b_j \quad j \in J \quad (4.9)$$

$$x_i \geq 0 \quad i \in I. \quad (4.10)$$

Saying that n is a number specifying the size of the instance, $|I|$ has an exponential size in terms of n , whereas $|J|$ is polynomial. Therefore, I size is such to make the whole problem intractable. In CG terminology, (4.8)–(4.10) is called the *Master Problem* (MP).

Let $\hat{I} \subseteq I$ be a subset of variables such that (4.8)–(4.10) defined over $\hat{I}$ is feasible, tractable, and bounded. Since this problem is defined on a reduced set of variables, in CG terminology, it is called the *Reduced Master Problem*. Naturally, it holds that $z_{MP}^* \leq z_{RMP}^*$. This fact can be observed by taking into account the dual problem of RMP:

$$\max \sum_{j \in J} b_j \lambda_j \quad (4.11)$$

s.t.

$$\sum_{j \in J} a_{ij} \lambda_j \leq c_i \quad i \in \hat{I} \quad (4.12)$$

$$\lambda_j \geq 0 \quad j \in J \quad (4.13)$$

Since (4.11)–(4.13) does not contain all constraints $I \setminus \hat{I}$, the global dual problem is potentially infeasible. If a dual problem is infeasible, its associated primal problem is suboptimal. Therefore, to identify the optimal global solution, z_{RMP}^* has to be inspected for violated constraints (4.12). Such an inspection procedure has to be able to either identify at least one violated (4.12) or prove that none exists, confirming dual feasibility and global optimality. If a violated constraint is identified, it has to be included in (4.8)–(4.10) as a new variable and the problem has to be reoptimized. Such inspection-reoptimization loop continues until all constraints are satisfied.

A violated (4.12) takes the following form:

$$c_i < \sum_{j \in J} a_{ij} \lambda_j \rightarrow \quad (4.14)$$

$$\rightarrow c_i - \sum_{j \in J} a_{ij} \lambda_j < 0 \quad (4.15)$$

Notice that the left hand side of (4.15) is the reduced cost of the out-of-base variable x_i , and the whole expression states that identifying a violated (4.12) is equivalent to find a new negative reduced cost variable, or to prove that none exists. The reduced cost column interpretation suggests that the suited variable to include in the RMP is the one with the most negative reduced cost. The optimization procedure used to identify the most negative reduced cost column is called *pricer*. Notice that such a procedure also determines the termination criteria: when the smallest values reduced cost is non-negative, the pricer has also to prove dual feasibility and global optimality.

With that in mind, it is possible to formulate Algorithm 4.1, a basic CG algorithm.

Algorithm 4.1 Column Generation Algorithm

- 1: Construct initial set $\hat{I}$
 - 2: **repeat**
 - 3: Optimize RMP and get dual variables
 - 4: Run pricer and add new negative reduced cost variables to RMP
 - 5: **until**(There are negative reduced cost variables)
-

Algorithm 4.1 starts by constructing the set of initial variables $\hat{I}$. Then, it runs final condition loop as follows. First, it optimizes the current RMP and retrieves the optimal dual variables. Then, it inspects for new negative reduced cost variables and if any are found, they are added to RMP. The loop terminates when the inspection procedure confirms that there aren't new negative reduced cost variables.

4.2.3 Branch-and-Price

CG by itself is a method to solve linear program, therefore, in general, an integral solution cannot be guaranteed. To enforce integrality, the most common method is to embed CG inside a Branch-and-Bound. Such an algorithm is called a Branch-and-Price (BnP).

BnPs are among the most efficient exact algorithms for a plethora of combinatorial problems, including vehicle routing (Şahin and Yaman, 2022; Lera-Romero et al., 2024; Duman et al., 2024), packing problems (Wahlen and Gschwind, 2025; Dell'Amico et al., 2020; Zhang et al., 2020), and scheduling (Speckenmeyer et al., 2025; Baretto et al., 2023; Fei et al., 2025), among others.

Algorithm 4.2 presents a pseudocode detailing the key steps of a BnP algorithm.

The algorithm starts by initializing a queue to hold the unexplored nodes, and the initial problem. At this point, it enters a loop that continues until there are nodes to explore. First, the node to explore is retrieved from the queue, and it is optimized with CG. Once its optimality is proved, the solution is checked for integrality. If it is integer, then the current incumbent is updated and the loop continued to the next node. Otherwise, a branching condition is identified, child nodes are initialized and added pushed into *queue*. For sake of simplicity, Algorithm 4.2, does not detail the handling of new integer solution nor node bounding. For information about these topics, refer to Algorithm 3.4.

In Line 9, branching is not well described, and for a valid reason. Unlike in a regular branch and bound, in BnP direct branching on the variables can be very ineffective. For instance, let's take into account a BnP for a VRP. In such a formulation, a variable represents

Algorithm 4.2 Generic Branch-and-Price Algorithm

```

1: queue  $\leftarrow$  new node queue
2: Create initial problem and add it to queue
3: while queue is not empty do
4:   prob  $\leftarrow$  pop queue
5:   Solve prob with Column Generation
6:   if prob is integer then
7:     Update current incumbent solution
8:   else
9:     Identify a branch condition, create child nodes, and add them to queue
10:  end if
11: end while

```

a feasible route from the depot, to some customers, and back to the depot. Variables are binary, taking value one when the path is selected, and zero otherwise. Suppose that one of the variable, in the optimal solution, takes value 0.5. The intuitive action would be to create to branches, one where such variable is imposed to take value at least one, and another where it is imposed to take value at most 0. The up-branch, the one imposing the variable to take at least value 1, is indeed quite effective: the route is selected, and all the customers associated to it are served. However, the down-branch, the one imposing the variable to take at most value 0, is very ineffective. Imposing such a variable to zero, is equivalent to create a new problem where that variable has never been created. Therefore it is possible for it to be generated again, and likely branched on in future iterations. This leads to unbalanced search trees, where the number of nodes in the down-branches is larger than the one in the up-branches. This issue is a consequence of the variables being somewhat artificial, since they represent high-level pieces of a solution, instead of low-level decision that can be effective into cutting large portions of the solution space.

4.3 Branch-and-Price Algorithm

It is possible to notice that model (4.1)–(4.7) has an exponential number of x_p^r variables, namely one for every possible way to delivery every parcel. As such, it is intractable by a regular Branch-and-Cut algorithm. Therefore, we propose a Branch-and-Price algorithm, where each node's relaxation is solved by Column Generation. In the following of this section, first we show the derivation of the pricer problem, then we illustrate a dynamic-programming (DP) algorithm to solve it.

4.3.1 Derivation of the Pricing Problem

Given the linear relaxation of Model (4.1)–(4.7), we define its dual variable as follow. Let $\pi_p \geq 0$, for $p \in P$, be the dual variable associated with constraint (4.2). Let $\lambda_k \leq 0$, for $k \in K$, be the dual variable associated with constraint (4.3). Let $\mu_{ip} \geq 0$, for $i \in S, p \in P$, be the dual variable associated with constraint (4.4). Let $\xi_{it} \leq 0$, for $i \in N, t \in T$, be the dual variable associated with constraint (4.5). It is possible to notice that in an optimal solution, neither x_p^r nor z_i can take a value larger than 1. In fact, a solution with larger value would be either more expensive than necessary, or simply infeasible. For instance $x_p^r = 2$ would violate constraint (4.3). Therefore, it is possible for both variables x_p^r and z_i to relax their

domain from $\{0, 1\}$ to $\mathbb{Z}_+$. In this way, we also remove the dual variables associated with the upper bound constraint.

Model (4.1)–(4.7) dual's is formulated as follow:

$$\max \sum_{p \in P} \pi_p + \sum_{k \in K} \lambda_k + \sum_{i \in N} \sum_{t \in T} c_i \xi_{it} \quad (4.16)$$

$$\pi_p + \sum_{k \in K} a_{pk}^r \lambda_k - \sum_{i \in N} s_{ip}^r \mu_{ip} + \sum_{i \in N} \sum_{t \in T} o_{it}^{pr} \xi_{it} \leq c_p^r \quad \forall p \in P, \forall r \in R_p \quad (4.17)$$

$$\sum_{p \in P} \mu_{ip} \leq v_i \quad \forall i \in N \quad (4.18)$$

$$\pi_p \geq 0 \quad \forall p \in P \quad (4.19)$$

$$\lambda_k \leq 0 \quad \forall k \in K \quad (4.20)$$

$$\mu_{ip} \geq 0 \quad \forall i \in N, p \in P \quad (4.21)$$

$$\xi_{it} \leq 0 \quad \forall i \in P, \forall t \in T \quad (4.22)$$

From (4.16)–(4.22) it is possible to observe that the reduced cost of an out of basis x_p^r variable is:

$$c_p^r - \pi_p^* - \sum_{k \in K} a_{pk}^r \lambda_k^* + \sum_{i \in N} s_{pi}^r \mu_{ip}^* - \sum_{i \in N} \sum_{t \in T} o_{it}^{pr} \xi_{it}^* \quad (4.23)$$

where π_p^* , λ_k^* , μ_{ip}^* , and ξ_{it}^* are the optimal value of dual variables. Taking into account that cost of a path starting from node $i \in S$ is

$$c_p^r = \sigma_i + \rho \sum_{k \in K} a_{pk}^r \quad (4.24)$$

which is the handling cost at its source station, σ_i , plus ρ time the number of crowdshippers, it is possible to substitute (4.24) into (4.23):

$$\sigma_i + \sum_{k \in K} (\rho - \lambda_k^*) a_{pk}^r - \pi_p^* + \sum_{i \in N} s_{pi}^r \mu_{ip}^* - \sum_{i \in N} \sum_{t \in T} o_{it}^{pr} \xi_{it}^*. \quad (4.25)$$

Since the objective of a column generation algorithm is to identify a negative reduced cost column, we need to identify a path such that (4.25) is strictly smaller than zero. This leads to the negative reduced cost formulation:

$$\sum_{k \in K} (\rho - \lambda_k^*) a_{pk}^r - \sum_{i \in N} \sum_{t \in T} o_{it}^{pr} \xi_{it}^* < \pi_p^* - \sigma_i - \sum_{i \in N} s_{pi}^r \mu_{ip}^* \quad (4.26)$$

For convenience the components of (4.25) that are not dependent on the actual path on the network are placed on the right-hand side of the inequality. In Equation (4.26) the left hand side of the inequality is strictly dependent only on the path the parcel follows from the source station to its destination. Specifically the first term represents the cost associated with the crowdshippers transferring the parcels, whereas the second represents the cost associated with a parcel waiting at a station occupying a spot in the parcel locker. It is possible to interpret the crowdshipper dual λ_k^* as the extra cost in which one incurs when reusing a crowdshipper already selected by the RMP for another parcel path. Similarly, the maximal capacity duals ξ_{it}^* represent the extra cost incurred when a parcel is stored in an overloaded station and time step. On the other hand, the right hand side of the inequality depends on

static, i.e. path independent. Therefore the final pricer formulation is the following shortest path with cumulative costs:

$$\min \sum_{k \in K} (\rho - \lambda_k^*) a_{pk}^r - \sum_{i \in N} \sum_{t \in T} o_{it}^{pr} \xi_{it}^*. \quad (4.27)$$

Notice, the right hand side of Equation (4.26) can be used to computed a valid upper bound on the cost of such a path, allowing to create an effective acceleration technique for dynamic program. The upper bound can be computed as $UB = \min \{b_p, \pi_p^* - c_0 - \sum_{i \in N} s_{pi}^r \mu_{ip}^*\}$.

4.3.2 Pricing Problem

In this section, we describe the procedures used to solve the pricing problem. First, we start with a high-level overview of the method, then we delve into the details of the procedure used to compute each parcel path.

Algorithm 4.3 Pricer

```

duals ← getDuals (rmp)
new-columns ← []
for  $p \in P$  do
  for  $src \in S$  do
    if  $src$  is not blocked then
      upper-bound ← computer-upper-bound(duals, p, src) ▷ See (4.26)
      dst ← get-destination(p)
      (cost, path) ← compute-path(src, dst, duals, upper-bound) ▷ See Algorithm 4.4
      and Algorithm 4.5
      if cost < upper-bound then
        new-column ← make-column (path)
        append new-column to new-columns
      end if
    end if
  end for
end for

```

Algorithm 4.3 computes, for each parcel and for each source station, the shortest path from the source station to the parcel's destination. The algorithm starts by getting the dual values from an optimal RMP solution, then it initializes an empty list of new columns. At this point two nested for loop, the outer iterating through all the parcels, the inner iterating through all the source stations S , are executed. In the body of the inner loop, the algorithm first computes the upper bound for the current parcel and source station, as described in the right hand side of Equation (4.26). Then, the algorithm retrieves the destination for the current parcel. Now, using the procedures described in Algorithm 4.4 and Algorithm 4.5, the algorithm computes the shortest path from the source to the destination station. The first value returned is the cost of the path, whereas the second is the path itself. If there is no path from the source to the destination that cost lower than the upper-bound the cost is set to $+\infty$. On the other side, if the cost is less than the current upper-bound, so it has a negative reduced cost, the path is converted into a new column for the RMP, and the column is inserted into the list of columns. Later these columns will be inserted in batch in the RMP.

The shortest path with cumulative costs (4.27) can be computed with the recursion presented in the following paragraph. Its practical implementation is presented in Algorithm 4.3. Let $\mathcal{S} = \{(i_1, j_1, k_1, t_1), \dots, (i_{|\mathcal{S}|}, j_{|\mathcal{S}|}, k_{|\mathcal{S}|}, t_{|\mathcal{S}|})\}$ be the set of all possible transfers done by crowdshippers. An entry $(i, j, k, t) \in \mathcal{S}$ represent a crowdshipper $k \in K$ that can transfer a parcel from $i \in N$ to $j \in N$ leaving i at time $t \in T$. Moreover, let M be an $|N| \times |T|$ matrix of labels. Label M_{it} represents the cheapest path to reach node $i \in N$ at time $t \in T$. Each label has fields presented in Table 4.4. The parcel path can be computed as a shortest path with cumulative

Table 4.4: Label Fields

Field	Domain	Descr.
Cost	$\mathbb{R}^+$	Cost to reach node $i \in N$ at time $t \in T$
Crowdshipper	K	Crowdshipper who did the last transfer
Prev. Location	N	Previous node in this path
Prev. Time	T	Departure time from the previous node

costs using the following recursion:

$$M(i, t) = \min \{M(i, t); M(i, t-1) + \xi_{it}; \min \{M(i', t') + \rho + \lambda_k + \xi_{it} \mid (i', j, k, t') \in \mathcal{S} \mid i = j \text{ and } t = t' + \delta_{i', j}\} \}. \quad (4.28)$$

Recursion in (4.28) states that the cost label in M_{it} is equal to the minimal of three elements:

1. The label itself, to represent a node $i \in N$ that cannot be reached at time $t \in T$ with a pass cheaper than the upper bound.
2. The label at the previous time step in this node. This propagates the previous solution to the current label. Cost has to increase by dual variable ξ_{it} associated with locker being used.
3. The label representing the cheapest transfer from another node arriving in i at time t . Such label is searches among all labels expanded by previous a node $i' \in N$ such that a crowdshipper can transfer a parcel from i' and arrive in i at time $t \in T$. The cost is updated by adding to it the crowdshipper remuneration ρ , the dual variable associated to this crowdshipper λ_k , and the cost st associated with locker being used ξ_{it} .

To simplify the notation, in (4.28) we intentionally ignore to update fields other than cost. Moreover, adding a number to a label should be read as *create a new label whose cost is the cost of the previous label plus this number*.

Algorithm 4.4 presents a forward implementation of (4.28), and it is used to compute the path for the current parcel from the given source station. First, the algorithm initializes an $|N| \times |T|$ label matrix, in which label $M(i, t)$ indicates the cheapest cost to reach node i at time step t . Additionally such labels also contain all the additional information to rebuild the path later on, namely the crowdshipper who drops the parcel in node i at time t , the previous node, and the time step, indicating where and when the crowdshipper pickup the parcel. The initial cost of each label is set to a valid upper bound for the cost of any feasible path. This value corresponds to the minimum between the backup cost for the parcel destination and right-hand side of (4.26). The first case represents a path whose cost is larger than the backup cost. The second case represents a path whose cost is such that the newly generated column cannot have a negative reduced cost. In either cases, a column whose value is larger than the

Algorithm 4.4 Compute Path

Require: *src*: Source station for the path, *duals*: Values of the dual variables, *upper-bound*: Maximal cost of any feasible path.

```

1:  $M \leftarrow |N| \times |T|$  matrix of labels of cost upper-bound
2:  $\text{cost}[M(\text{src}, 1)] \leftarrow -\xi_{\text{src},1}$ 
3: for  $t = 1$  to  $|T|$  do
4:   for  $i = 0$  to  $|N|$  do
5:      $\text{label} \leftarrow M(i, t)$ 
6:     if  $\text{cost}[\text{label}] < \text{upper-bound}$  then
7:       if  $t \neq |T|$  then
8:          $\text{new-label} \leftarrow \text{cost}$ 
9:          $\text{cost}[\text{new-label}] \leftarrow \text{cost}[\text{new-label}] - \xi_{i(t+1)}$ 
10:        if  $\text{cost}[\text{new-label}] < \text{cost}[M(i, t + 1)]$  then
11:           $M(i, t + 1) \leftarrow \text{new-label}$ 
12:        end if
13:      end if
14:      for arc in Successors(i, t) do
15:        if arc is not blocked and then
16:           $\text{new-label} \leftarrow \text{Empty Label}$ 
17:           $\text{cost}[\text{new-label}] \leftarrow \text{cost}[\text{label}] + \rho - \lambda_{\text{crs}[\text{arc}]} - \xi_{\text{dst}[\text{arc}], \text{time}[\text{arc}]}$ 
18:          if  $\text{cost}[\text{new-label}] < \text{cost}[M(\text{dst}[\text{arc}], \text{time}[\text{arc}])]$  then
19:             $\text{crowdshipper}[\text{new-label}] \leftarrow \text{crs}[\text{arc}]$ 
20:             $\text{prev-location}[\text{new-label}] \leftarrow i$ 
21:             $\text{prev-time}[\text{new-label}] \leftarrow t$ 
22:             $M(\text{dst}[\text{arc}], \text{time}[\text{arc}]) \leftarrow \text{new-label}$ 
23:          end if
24:        end if
25:      end for
26:    end if
27:  end for
28: end for
29: return M

```

upper bound cannot contribute positively to the optimization process and can therefore be ignored. Now, the algorithm updates the cost of label (*src*, 1) with the dual cost associated to the capacity constraint at that station during the first time step. The algorithm proceeds with two nested for loops: the outer loop iterates though all time steps, while the inner iterates though all nodes. The body of the inner for loop operates as follows. First, the algorithm checks if the cost of the current label at current node and time step is less than the known valid upper value. If this is not the case, then the current station and time have never, and will never, be used in any valid parcel path, therefore the current label doesn't need to be processed. If the cost is less the algorithm propagates the current label (Lines 7-13), then extends it (Lines 14-25). The propagation step represent a parcel that is not moved at time *t*, so in this case the current label is copied, the new label's cost is updated with the dual from the capacity constraints in this node at the next time step, finally if this new label is cheaper entry $M(i, t + 1)$ is updated with the new label. Now, the extension phase computes all possible transfers of the parcel from the current station during the current time step.

In this phase, a for loop iterates through all crowdshippers leaving the current node at the current time step. Each successor of the current node contains the information about the crowdshippers doing the transfer, the destination node of the crowdshipper, and the arrival time at the destination. For each successor, first it is checked if the arc is not blocked by a branching rule. If the arc is free, a new empty label is initialized, its cost is calculated from the previous cost, the crowdshipper remuneration, and the duals associated with the crowdshipper and the destination's load constraint. If the cost of this new label is smaller than the cost current one, all the parameters to reconstruct the solution are initialized, and the entry in the matrix M is updated. At the end, the algorithm returns the matrix M .

Algorithm 4.5 Build Path

Require: M : Filled dynamic matrix from Algorithm 4.4, src: Source station, dst: Destination station, upper-bound: Valid upper bound on the path cost.

```

1: cost =  $M(\text{dst}, |T|)$ 
2: if cost = upper-bound then
3:   return  $(+\infty, [])$ 
4: end if
5: path  $\leftarrow []$ 
6: t =  $|T|$ 
7: while dst  $\neq$  src do
8:   label  $\leftarrow M(\text{dst}, t)$ 
9:   t-begin  $\leftarrow \text{prev-time}[\text{label}]$ 
10:  arc  $\leftarrow (\text{crowdshipper}[\text{label}], \text{prev-location}[\text{label}], \text{dst}, t\text{-begin}, t)$ 
11:  append arc to path
12:  t  $\leftarrow t\text{-begin}$ 
13:  dst  $\leftarrow \text{prev-location}[\text{label}]$ 
14: end while
15: reverse path
16: return (cost, path)

```

Given an output from Algorithm 4.4, Algorithm 4.5 is used to reconstruct the path, which later can be used to determine the coefficients of the new column. The algorithm starts by getting the cost associated with the label at the destination station for the current parcel at the final time step. If this cost is equal to the upper bound, then no feasible path has been found and the algorithm terminates early by returning an infinity cost and an empty path. Otherwise, an empty list to store the path is initialized and the current time step is set to the last one. Now, the algorithm enters a while loop, that will terminate when the node to process is equal to the source node of the path. At each loop iteration the label at the current time step is retrieved from the matrix, a new arc of the path is created, the arc is appended to the path list, and finally the current time and node are reset to the start time and source node of the current label. Each arc is a tuple that indicates the crowdshipper doing the transfer, the source and destination station of the transfer, and the start and end time step of the transfer. After the while loop, the path is reversed. Finally, the cost and the path can be returned.

Column Selection Rule Algorithm 4.3 looks for, for each parcel, a new candidate (columns with a negative reduced cost) path for each station $i \in S$. So, it is possible to generate none, one, or more such paths. On a CG, one should always find a good balance between inserting a small number of column, to keep the RMP fast, and a large number of columns, which help

the algorithm to converge faster. Taking these things into account, Algorithm 4.3 can be configured to select and insert columns in the RMP according to the following rules:

- **All**: insert all the generated columns with a negative reduced cost in the master problem;
- **Best**: insert the column with the most negative reduced cost among generated ones;
- **First**: insert the first column with negative reduced cost that is generated.

The rationale behind **All** is to provide a more complete view of the instance to the master problem, in the hope to reduce the overall number of pricing problems solved. The next two rules, on the other hand, aim at keeping the master problem as small as possible. Rule **Best** does it while inserting only very promising columns, at the expense of longer pricing problems, whereas, rule **First**, accept to provide less promising columns, at the advantage of a faster pricing phase. Notice that **First** could, and should, be implemented in a short-circuited way, so as a negative reduced cost column is found the algorithm can stop.

4.3.3 Feasibility Cuts

To improve computational performance, constraints (4.4) and (4.5) are separated dynamically.

Identifying violated constraints (4.4) is relatively simple. Indeed, it is enough to iterate through all columns, and if the associated $x_p^r \geq \epsilon$ the appropriate constraint (4.4) is inserted into the RMP, given that this constraint doesn't already exist.

On the other hand, identifying violated constraints (4.5) is slightly more involved. First, a *load matrix*, of size $|N| \times |T|$, is constructed to specify the number of parcels at node $i \in N$ at time $t \in T$. Finally, the load matrix is inspected for capacity violation, and the associated constraints (4.5) are inserted into the RMP.

4.3.4 Branching Rules

In this paragraph we present the branching rules used to ensure the integrality of the final solution.

First, let us define the scoring function $g_\alpha(x) = \min\left(\frac{1}{\alpha}x; \frac{1}{\alpha-1}x - \frac{1}{\alpha-1}\right)$, where $\alpha \in]0, 1[$ is a user defined parameter, that will be used to select the variable to branch on. Notice that $g_\alpha(x)$ takes value 0 if $x = 0$ or $x = 1$, and it takes value 1 if $x = \alpha$. Function $g_\alpha(x)$ provides a *non-integrality* score, taking minimal, namely 0, value if its argument is either 0 or 1 (so it's integer) and taking maximal value, namely 1, when its argument takes value α . This function allows for a clean implementation of the branching selection of the variable closer to a certain fractional value. It is possible that the numerical approximation carried out by the CPU leads to values of $g_\alpha(\cdot)$ that are never exactly zero, we introduce parameter ϵ , a small number such that any value less than ϵ can be safely considered 0. In the implementation $\epsilon = 10^{-8}$.

Open/Close Source Station This rule forces one variable z_i to take an integer value. We denote with z_i^* the objective value of variable z_i in the current BnB node. The rule follows the following steps:

- Compute the non-integrality score $\delta_i = g_\alpha(z_i^*) \forall i \in N$;

- If $\max \delta_i \leq \epsilon$, then continue to the next rule;
- Select z_i with the maximal δ_i and impose:
 - $z_i \geq 1$
 - $z_i \leq 0$

To effectively impose $z_i \leq 0$, it is required for the pricer to *block* the source station i . This can be achieved by not creating paths starting from a close source station, see Line 5 of Algorithm 4.3.

Allow/Block Multi-Crowdshipper Path A parcel path is made of one, or more, crowdshipper transfers. For a given route $r' \in R$, delivering parcel $p' \in P$, having $a_{p'k'}^{r'} = 1$ where crowdshipper $k' \in K$ is moving a parcel from $i' \in N$ to $j' \in N$, is equivalent to have variable $x_{i'j'}^{p'k'} = 1$ in model (3.1)–(3.23).

Let $\hat{x}_p^r$ the optimal value of variable x_p^r in a branch and bound node. Let $R_{ij}^{pk} \subseteq R$ be the subset of routes where crowdshipper $k \in K$ transfers parcel $p \in P$ from station i to j . It holds that

$$\hat{x}_{ij}^{kp} = \sum_{r \in R_{ij}^{kp}} a_{pk}^r \hat{x}_p^r \quad \forall (i, j) \in A, k \in K, p \in P. \quad (4.29)$$

In a non-integer solution $\hat{x}_{ij}^{kp}$ can take values in $]0, 1[$, whereas in an integer solution $\hat{x}_{ij}^{kp}$ takes value that is either 0 or 1. Therefore, the branching rule is as follows:

- Compute value $\hat{x}_{ij}^{kp}$ as in Equation (4.29);
- Compute non-integrality score $\delta_{ij}^{kp} = g_\alpha(\hat{x}_{ij}^{kp}) \forall (i, j) \in A, k \in K, p \in P$;
- If $\max \delta_{ij}^{kp} \leq \epsilon$, then continue to the next rule;
- Select variable x_{ij}^{kp} with maximal δ_{ij}^{kp} and impose:
 - $x_{ij}^{kp} \geq 1$
 - $x_{ij}^{kp} \leq 0$

The down-branch $x_{ij}^{kp} \leq 0$ decision is rather easy to impose. Indeed, one has to modify function *Successors*, Line 14 in Algorithm 4.4, to not propose transfer of parcel p from i to j done by k . Since this transfer is not going to be proposed, it is impossible for variable x_{ij}^{kp} to take any value other than 0. The up-branch $x_{ij}^{kp} \geq 1$ is slightly more involved. In fact, the algorithm needs to select transfer for parcel p to j with crowdshipper k only if node i has been visited but not otherwise. Therefore, *Successors* has to be modified by removing all transfers from i but the one towards j made by crowdshipper k . In this way, the pricer is free to transfer p for i to j , but if it decides to do so, it must do it by employing crowdshipper k .

4.4 Experimental Results

In this section we first introduce the instance generation procedure, then we analyze the computation experiments run on the instances. The algorithm is implemented in C++, using CPLEX22.11 as back end LP solver. Experiments are run on a computer equipped with an Apple M3 Max CPU and 64Gb of RAM running macOS 26.1, with a one-hour time limit.

4.4.1 Instance Generation

The instances are constructed over four distinct random metro networks, with 24, 36, 40, 44 stations respectively. These are the same synthetic public transport network used in Section 3.7.1. For each network we generate a traffic pattern, used to determine the source-destination of each crowdshipper and the destination of each parcel, and a time pattern, used to determine when crowdshippers will start their commute. From these pattern, we generate a list of 4000 crowdshippers and a list of 2000 parcels for each network. From each crowdshipper list, we extract a subset of 2000 crowdshippers. Moreover, from each parcel list, we sample a list of 1000 parcels. Then, from the crowdshipper list we extract three sublists, with 500, 1000, and 1500 items respectively. Similarly, from the parcel list we sample three sub list, with 250, 500, and 750 items respectively. Notice that in all the cases the larger list is a super set of the smaller one. For all the combinations of number of crowdshippers, number of parcels we generate one instance. Such a procedure is repeated 4 times. This leads to a total of $4 \text{ (number of networks)} * 4 \text{ (number of crowdshippers set)} * 4 \text{ (number of parcel set)} * 4 \text{ (number of instances)} = 256$ instances. Notice that all the instances for a given network are generated from the same global population of crowdshippers and parcels. Each network has four source stations, each with $C_i = 250$, all the other stations have $C_i = 50$. The four source stations are chosen to be the one that maximize that average distance between them. This rule allows for the selection of the four station *most* on the border. Each network has a fixed and known depot set in a random location reasonably far away from the network.

We decided to generate a new set of larger instances with respect to the ones used in Section 3.7. There the maximal size of an instance is $|K| = 300$ and $|P| = 250$. During preliminary tests, this instances were solved to optimality in the matter of seconds, so it was impossible to use them as a reliable and complete way to test this algorithm.

4.4.2 Analysis of Total Solution Time

Preliminary analysis, non presented here for brevity, have highlighted that the value of α for $g_\alpha(\cdot)$, i.e. the non-integrality score, is not crucial for the optimal performance of the algorithm. Therefore, all results presented in the following are obtained with $\alpha = 0.5$ since it the value that tends to provide the most consistent results for the overall instance set.

All instances are solved to optimality. The average solve time, independently from the column selection rule or the instance class, is 15.28s. The maximal one is 126.08s and the minimal is 0.68s.

Table 4.5 reports, for each network (Col. $|N|$) and each number of crowdshippers (Col. $|K|$), the average (Avg.), median (Q_{50}), 75th (Q_{75}) and 90th (Q_{90}) quantile of the total run time for each column selection rule.

A quick look at the table shows that it is impossible to identify a clear winner among the configurations. However, it is possible to observe some interesting patterns. First, average columns, gap between the best and worst performing formulation tends to be not small.

Table 4.5: Total run time statistics for the three configurations by number of crowdshippers

$ N $	$ K $	Avg.			Q_{50}			Q_{75}			Q_{90}		
		All	Best	First	All	Best	First	All	Best	First	All	Best	First
24	500	8.53	9.91	9.36	6.64	7.59	6.11	12.14	16.36	13.93	16.04	19.04	19.18
24	1000	15.35	16.32	15.46	12.71	13.88	15.67	19.37	20.16	21.73	25.91	28.97	25.82
24	1500	22.44	23.11	26.00	19.70	21.37	20.85	33.35	30.60	38.73	41.14	40.62	45.71
24	2000	32.55	34.93	50.14	33.60	37.98	48.51	40.73	50.33	70.63	59.28	59.15	77.45
36	500	10.11	12.13	9.60	8.32	9.93	8.44	12.88	18.58	13.76	20.47	25.35	20.25
36	1000	9.79	12.11	9.97	8.07	9.47	8.61	13.46	14.46	13.42	20.83	27.17	18.93
36	1500	11.16	13.83	12.06	10.59	11.91	10.22	15.16	21.18	17.25	19.80	26.14	22.13
36	2000	15.56	17.57	17.52	13.97	16.14	16.86	21.10	25.53	24.79	25.84	30.03	26.55
40	500	16.23	6.13	17.36	11.52	4.86	12.91	25.51	9.60	24.42	37.26	13.13	43.82
40	1000	16.48	9.39	16.54	13.50	7.16	12.60	26.70	14.21	24.85	30.95	17.94	35.98
40	1500	17.20	11.55	16.49	12.61	8.97	13.35	26.19	16.01	27.62	37.26	28.48	35.47
40	2000	17.91	16.18	18.69	14.19	12.92	14.60	27.08	24.20	26.95	38.27	32.80	41.50
44	500	10.74	10.76	9.87	8.80	7.61	9.47	15.40	15.66	13.77	22.18	25.28	18.52
44	1000	8.65	10.66	8.75	7.42	8.25	6.95	13.38	17.04	11.85	15.17	21.40	16.79
44	1500	10.63	12.55	10.30	9.84	12.69	10.34	17.34	17.36	13.30	18.64	22.40	16.42
44	2000	13.07	15.95	13.35	13.15	16.47	13.17	16.12	21.61	16.69	22.47	25.48	22.79

It is possible to observe that the average gap is 36.23%, with a minimal value of 6.3% (for $|N| = 24, |K| = 1000$) and a maximal of 183.43% ($|N| = 40, |K| = 500$). However, it should be noticed that the second largest gap, found for $|N| = 40, |K| = 1000$, takes value 76.28%, showing that the largest value is more an outlier than the norm. Similar gaps, can be found for the median case. Moreover, this pattern is not related with the number of crowdshippers nor the with the network. It should be noticed that this irregular behavior does not come unexpected. In general, it is very hard to understand the correct strategy. In some cases *All*, *Best*, and *First* might have similar performance, for instance where there are few, maybe even just one, reduced cost columns for each parcels. In other, investing more time to find the column with the smallest reduced cost might prove to be the best strategy, whereas in other, for instance when all columns have comparable reduced costs, just selecting the first one is the best strategy. Finally, in other cases, a larger number of columns added at each CG iteration might be the best one.

It is possible to notice a mild positive correlation between the number of crowdshippers and the total runtime. In fact, between the number of crowdshippers ($|K|$) and the total run time in the average case there is a Pearson' correlation of 0.53. When selecting only solution from the *All* configuration the correlation takes value 0.52, for configuration *Best* the correlation is 0.64, and correlation for configuration *First* takes value 0.50. The reason for such a behavior is two fold. First of all, instances with a larger number crowdshippers make for a more computationally expensive pricer. More crowdshipper, in fact, corresponds to more successors in Algorithm 4.4, and therefore a longer run. Second, with more crowdshippers there's room to generate more parcel paths at negative reduced cost, resulting in a more complex whole optimization.

Table 4.6 reports, for each network (Col. $|N|$) and each number of parcels (Col. $|P|$), the average (Avg.), median (Q_{50}), 75th (Q_{75}) and 90th (Q_{90}) quantile of the total run time for each column selection rule. In this table and in the following, Q_x , with $x \in [0, 100]$, identifies the x -th percentile of population under analysis.

Similar to the previous analysis, also in this table it is not possible to identify a clear best column selection rule. Moreover, also in this case it is possible to notice the wide performance difference between the fastest and the slowest column selection rule. In the average

Table 4.6: Total run time statistics for the three configurations by number of parcels

$ N $	$ P $	Avg.			Q_{50}			Q_{75}			Q_{90}		
		All	Best	First	All	Best	First	All	Best	First	All	Best	First
24	250	4.86	4.91	8.03	4.97	5.66	7.96	6.45	6.81	10.70	7.80	7.18	12.01
24	500	15.19	17.98	25.43	11.14	14.09	10.00	18.49	22.66	24.40	33.56	36.59	62.52
24	750	22.84	23.38	26.13	23.83	21.25	20.38	30.12	29.77	37.58	35.47	37.98	48.51
24	1000	35.98	37.98	41.37	36.56	35.91	36.02	49.18	50.57	65.86	59.28	59.15	72.13
36	250	2.99	3.17	3.78	2.21	2.64	3.14	3.31	3.77	4.73	5.31	5.87	7.90
36	500	7.85	7.98	8.16	7.19	8.12	7.19	9.62	9.67	9.91	11.17	12.15	13.52
36	750	13.13	16.96	15.14	11.68	16.05	12.48	14.72	19.88	18.15	18.09	20.99	22.42
36	1000	22.65	27.54	22.07	22.58	27.19	22.13	25.44	29.63	24.79	28.56	36.41	26.63
40	250	1.95	1.64	2.17	1.95	1.65	2.10	2.09	2.22	2.44	2.42	2.40	2.53
40	500	7.99	6.08	8.06	8.13	5.92	7.97	8.84	7.96	8.89	9.37	9.72	9.64
40	750	21.58	12.52	20.10	22.52	12.04	19.54	24.24	15.58	22.56	25.05	18.89	23.73
40	1000	36.30	23.00	38.75	36.53	22.01	39.53	40.44	29.46	44.25	42.44	32.80	47.30
44	250	2.92	2.95	3.11	3.18	2.65	3.10	3.80	4.22	3.87	4.00	4.79	4.95
44	500	7.80	8.27	8.12	6.79	7.33	7.24	8.94	11.33	10.86	12.12	12.81	12.32
44	750	12.67	14.56	11.78	12.81	14.12	11.66	13.45	17.13	12.80	15.12	19.00	13.92
44	1000	19.70	24.14	19.26	18.50	23.76	18.52	22.43	26.25	20.89	26.99	32.15	24.22

case, the average gap between the best and worst performing rule is 32.10%, with a minimal of 3.91%, and a maximal of 72.39%. Similarly large gaps can be found for the other statistics.

It is possible to notice a strong correlation between the number of parcels to deliver and the overall time required by the algorithm. Pearson's coefficient between these two figures takes value 0.89 for *All*, 0.90 for *Best*, and 0.80 for *First*. The overall coefficient is 0.86. These results are far from being unexpected. A larger parcel set, corresponds to more pricer problems to solve, and more columns to add to RMP.

Table 4.7: Column selection rule ranking by $|K|$ and $|P|$

$ K $	$ P $	All	Best	First
500	250	1.94	1.56	2.50
500	500	2.44	1.50	2.06
500	750	2.06	1.88	2.06
500	1000	1.81	2.12	2.06
1000	250	1.94	1.75	2.31
1000	500	2.31	2.00	1.69
1000	750	2.12	2.06	1.81
1000	1000	1.62	2.12	2.25
1500	250	1.62	1.62	2.75
1500	500	1.81	2.38	1.81
1500	750	2.06	2.12	1.81
1500	1000	1.81	2.19	2.00
2000	250	1.44	2.06	2.50
2000	500	1.44	2.06	2.50
2000	750	1.50	2.25	2.25
2000	1000	1.88	2.12	2.00
# Best		7	4	3

Table 4.7 shows the statistical ranking, in terms of computing time, of the three column selection rules, by the size of the crowdshippers and parcels sets. Statistical ranking is com-

puted as follows. For each instance, the column selection rule is assigned to its position in the leader board. So the fastest formulation takes score 1, the second 2, and the third 3. In case of a tie, the equivalent rules are assigned the same score, which is the average of their position in the leader board. For instance, if the two fastest rules take the same value, each gets score 1.5. The final score, reported in Table 4.7, is the average among all the scores obtained by each rule. The last row identifies how many times the column selection rule has been the best performing one for the specific $|K|$ - $|P|$ couple.

From this table, it is possible to observe several interesting patterns. First, in small sized instances, both in $|K|$ and in $|P|$, rule Best tends to be the best performing, whereas in larger instances All is the best one. In medium sized instances, it is harder to find a real winner.

Table 4.8: Comparison of column selection rules by $|K|$ and $|P|$

$ K $	$ P $	Rule	First			Best		
			Wins	Losses	p -value	Wins	Losses	p -value
500	250	All	12	4	0.08	5	11	0.21
500	250	Best	12	4	0.08	-	-	-
500	500	All	6	10	0.45	3	13	0.02
500	500	Best	11	5	0.21	-	-	-
500	750	All	7	9	0.80	8	8	1.00
500	750	Best	10	6	0.45	-	-	-
500	1000	All	10	6	0.45	9	7	0.80
500	1000	Best	7	9	0.80	-	-	-
1000	250	All	10	6	0.45	7	9	0.80
1000	250	Best	11	5	0.21	-	-	-
1000	500	All	4	12	0.08	7	9	0.80
1000	500	Best	7	9	0.80	-	-	-
1000	750	All	6	10	0.45	8	8	1.00
1000	750	Best	7	9	0.80	-	-	-
1000	1000	All	12	4	0.08	10	6	0.45
1000	1000	Best	8	8	1.00	-	-	-
1500	250	All	13	3	0.02	9	7	0.80
1500	250	Best	15	1	0.00	-	-	-
1500	500	All	8	8	1.00	11	5	0.21
1500	500	Best	5	11	0.21	-	-	-
1500	750	All	6	10	0.45	9	7	0.80
1500	750	Best	7	9	0.80	-	-	-
1500	1000	All	9	7	0.80	10	6	0.45
1500	1000	Best	7	9	0.80	-	-	-
2000	250	All	13	3	0.02	12	4	0.08
2000	250	Best	11	5	0.21	-	-	-
2000	500	All	14	2	0.00	11	5	0.21
2000	500	Best	10	6	0.45	-	-	-
2000	750	All	12	4	0.08	12	4	0.08
2000	750	Best	8	8	1.00	-	-	-
2000	1000	All	9	7	0.80	9	7	0.80
2000	1000	Best	7	9	0.80	-	-	-

To better understand the algorithm's performance in relation to the instance size, in Table 4.8 we propose an instance-wise comparison detailing how many times one selection rule outperforms the others. Columns Wins specifies the number of times the rule on the row outperforms the one in the column. On the other side, columns Losses tell how often rule on the row is outperformed by the one in the column. Finally, column p -value reports the p -value from the Binomial test. We use 0.05 as a threshold value to discard the null hypothesis. Notice that each row describes 16 instances, namely 4 instances for each network.

The most interesting thing to observe is that, with the exception of a few cases, a better performing selection rule never outperforms another one in a statistically significant way. In fact, it is possible to find a significant result only in 5 comparisons out of 48. In four of these cases, the losing rule is First, whereas in the other the losing one is All against Best. Best is never outperformed in a statistically significant way. Moreover, it should be noticed that if we choose a tighter threshold values, for instance 0.01, only two cases would be significant.

By taking the absolute difference between the number of Wins and Losses over all the formulations, it is possible to observe a maximal difference of 14, with a median of 4, Q_{75} is 6.5, and Q_{90} is 8.6. This shows that, in general, the difference between two column selection rules rather small, and deeply connected with the property of a specific instance.

4.4.3 Impact of the Percentage of Backup Parcels

Table 4.9: Impact of backup parcels on algorithmic performance

$ N $	$ K $	$corr(B\%, rt)$	$corr(B\%, np)$	$corr(B\%, nn)$
24	500	0.78	0.82	-0.06
24	1000	0.67	0.77	-0.35
24	1500	0.82	0.84	0.02
24	2000	0.72	0.82	-0.08
36	500	0.76	0.77	-0.02
36	1000	0.76	0.77	0.17
36	1500	0.86	0.86	0.24
36	2000	0.86	0.88	-0.30
40	500	0.67	0.72	0.10
40	1000	0.74	0.79	-0.14
40	1500	0.78	0.81	-0.42
40	2000	0.82	0.84	-0.30
44	500	0.77	0.76	0.11
44	1000	0.81	0.83	-0.30
44	1500	0.84	0.85	-0.36
44	2000	0.87	0.86	0.09

Table 4.9 shows, for each network (Col. $|N|$) and number of crowdshippers (Col. $|K|$), the Pearson' correlation coefficients between the fraction of backup parcels ($B\%$) and the total run time (rt), the number of pricing problems solved (np), and the number of branch-and-bound nodes (nn).

With a minimum of 0.67 and a maximum of 0.87, Table 4.9 shows a general strong correlation between the percentage of backup parcels and the total run time. Similarly, there's a strong correlation between the fraction of backup parcels and the number of pricing problems solved. Such behavior is easy to explain. Backup is, by construction, the most expensive

delivery option. Therefore, the algorithm has all the reason to prefer using crowdshippers whenever possible. If available crowdshippers are either not enough or simply don't follow the right patterns, CG has to continue until it can prove all delivery options are taken into account. This naturally leads to a larger number of pricing problems solved, which in turns increases the total run time of the algorithm. However, it should be noticed that a larger fraction of backup parcels does not alter the formulation strength. Such an observation is confirmed by checking the correlation index between the fraction of backup parcels and the number of BnB nodes generated. These correlations range from weak negative to weak positive, with many taking values small enough to exclude any correlation whatsoever. Since correlation is never strong, and it oscillates between positive and negative values, it is safe to assume that a larger fraction of backup parcel in the optimal solution is not a direct contributor to poorer primal bounds.

4.4.4 Analysis of the Branch-and-Bound Tree

Table 4.10: Statistics about the number of BnB nodes for the three configurations

N	K	Avg.			Q ₅₀			Q ₇₅			Q ₉₀		
		All	Best	First	All	Best	First	All	Best	First	All	Best	First
24	500	16.62	16.25	17.50	15.00	15.00	17.00	15.50	17.00	17.50	18.00	19.00	20.00
24	1000	23.75	34.62	98.12	19.00	18.00	17.00	22.00	27.50	31.50	29.00	55.00	111.00
24	1500	21.88	17.75	31.12	20.00	15.00	21.00	24.50	17.00	29.00	32.00	26.00	34.00
24	2000	44.88	43.00	169.50	20.00	17.00	18.00	55.50	26.50	52.00	121.00	101.00	391.00
36	500	15.00	15.88	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	19.00	15.00
36	1000	15.00	14.75	15.12	15.00	15.00	15.00	15.00	15.00	15.00	16.00	15.00	16.00
36	1500	16.12	17.25	14.50	15.00	16.00	15.00	17.00	17.50	15.00	17.00	21.00	15.00
36	2000	27.50	23.88	21.00	15.00	15.00	17.00	20.00	19.50	21.50	23.00	29.00	31.00
40	500	15.25	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00
40	1000	15.50	15.00	15.00	15.00	15.00	15.00	15.00	15.00	15.00	16.00	15.00	15.00
40	1500	16.12	15.38	16.50	15.00	15.00	15.00	15.50	15.00	17.00	19.00	16.00	20.00
40	2000	16.62	17.25	15.00	15.00	15.00	15.00	15.50	15.00	15.00	17.00	15.00	15.00
44	500	15.25	15.00	15.12	15.00	15.00	15.00	15.00	15.00	15.00	16.00	15.00	15.00
44	1000	23.88	15.50	16.62	15.00	15.00	15.00	15.00	15.00	17.00	15.00	16.00	21.00
44	1500	18.12	20.50	21.38	15.00	15.00	15.00	21.50	17.00	17.00	25.00	19.00	22.00
44	2000	18.50	17.12	15.38	15.00	16.00	15.00	15.50	19.00	15.00	31.00	20.00	18.00

In Table 4.10 we present the average, median, 75th, and 90th percentile of the total number of generated nodes, grouped for network size ($|N|$) and number of crowdshippers ($|K|$). The global minimum number of nodes explored is 11, the maximum is 1613, and the global average is 23.83. In all instances, it was required to branch on variables z_i multiple times before start to branch on variables x_p^r . This is a natural behavior easy to explain. Variable z_i is associated with cost v_i in the objective function. Therefore, the cheapest solution is the one using the *least amount* of v_i as possible. This can be achieved by taking combinations of paths starting from multiple stations at the same time, in such a way that constraint (4.2) is satisfied. This is the reason why we first check for the integrality of z_i and then of x_p^r . Until z_i are forced to be integer, it is easy for the solver to find a combination of fractional x_p^r such the constraints are satisfied and the v_i does not have to be paid as a whole. Only once all required z_i are set to their optimal value in an integer solution, the algorithm can start to branch on x_p^r variables.

In Table 4.10 it is possible to observe that, in general, the average total number of nodes explored is quite consistent along the various instance classes for the different column selection rules. Ignoring the outliers $|N| = 24$ with $|K| \geq 1000$, it is possible to observe that

the percentage gap between the largest and smallest number of nodes in the BnB tree takes a minimal value of 1.67%, a maximal of 54%, and an average of 14.4%. A similar pattern can be observed in the median case, where in only 4 out of 16 cases the median number of nodes differs among the column selection rules. For Q_{75} this happens 8 times, and for Q_{90} it happens 12 times. The outliers are likely due to an unfortunate interaction between the instance properties and the selection rule *First*. Likely this produced a large number of column whose source node is always the same, forcing BnB to exhaust such columns before it was possible to use other source stations and solve instance to integrality.

It is also possible to notice that there is no trend connecting the number of crowdshippers and the number of nodes. In the overall solution set it is possible to notice that the Pearson's correlation coefficient between $|K|$ and the number of BnB nodes takes value 0.08. When filtering for the number of stations, we observe that for $|N| = 24$ the coefficient take value 0.13, for $|N| = 36$ it takes value 0.21, for $|N| = 40$ we obtain 0.16, finally for $|N| = 44$ we get 0.06. These are all values that allow to assume that there is no clear linear relationship between the number of nodes and the number of crowdshippers.

4.4.5 Analysis on the Number of Generated Columns

Table 4.11: Statistics on the number of generated columns for the three configurations

$ N $	$ K $	Avg.			Q_{50}			Q_{75}			Q_{90}		
		All	Best	First	All	Best	First	All	Best	First	All	Best	First
24	500	59089.25	49244.25	47731.94	59839.00	45950.50	36931.00	80394.00	67966.00	71373.25	106627.50	83143.00	93669.50
24	1000	108826.19	80672.12	95546.94	92203.00	61646.00	68493.00	141283.75	108884.00	157515.25	180687.50	149205.00	178748.00
24	1500	132646.00	94215.69	128839.81	135534.50	71324.50	118443.50	175781.75	155922.25	189363.75	250739.50	186127.00	230806.50
24	2000	163652.81	113211.69	188717.56	148950.50	95796.50	175346.00	235704.50	168531.00	272690.75	335036.00	215198.00	353817.00
36	500	42594.06	43442.00	10778.31	20827.00	13284.00	7903.00	45596.25	51935.00	10875.50	63937.00	105551.00	13557.50
36	1000	41770.94	41969.38	28078.94	35985.50	28805.00	16494.50	60637.50	51111.75	40974.25	78791.50	93789.00	63543.00
36	1500	45499.06	47552.25	46950.44	42379.00	38588.50	39598.50	56636.00	66055.50	58136.00	71794.00	104238.50	93680.50
36	2000	53214.88	44916.94	51388.00	54830.00	37183.00	44322.50	70341.00	55118.50	68995.50	81249.00	76117.00	90445.50
40	500	4587.56	3475.38	5549.38	4500.00	3356.50	5137.00	5611.75	3960.75	6701.50	6590.50	6035.50	8865.00
40	1000	8912.81	5370.25	8734.88	7510.50	5482.50	7910.00	9696.75	6695.00	10948.75	12270.00	8108.50	14540.50
40	1500	12153.06	8298.94	13195.75	12625.50	8169.00	13063.50	16111.00	9500.00	19157.00	18926.50	13418.50	20854.00
40	2000	19525.88	14126.75	14328.38	15896.50	10558.50	14330.50	22524.50	16486.00	20082.50	27630.50	28366.00	21303.00
44	500	12662.88	6707.81	9996.69	9334.00	5850.00	8520.00	12343.00	7891.50	10774.00	29563.00	11761.00	17031.50
44	1000	18062.06	20737.88	25144.62	16943.50	11785.00	17741.00	24224.50	23193.00	30857.25	28978.00	43580.00	46656.00
44	1500	35961.62	35984.62	32719.69	30450.50	23110.50	31413.50	50634.50	54241.25	44791.25	60370.00	75350.00	55458.50
44	2000	50167.31	47438.75	49065.81	43475.50	45044.50	47231.00	61322.75	65914.00	64920.50	83630.00	73434.50	84373.00

In Table 4.11 we present the average, median, 75th, and 90th percentile of the total number of generated columns with negative reduced cost, groped for network size ($|N|$) and number of crowdshippers ($|K|$).

There are several interesting observations about this statistics. First of all, we focus on the average number of columns generated by each column selection rule. There it is possible to notice that column selection *All* is the one who generated the most columns in 8 out of 16 cases. On the other hand, column selection *Best* does it 4 times, and *First* does it 4 times as well. This result does not come unexpected: *All* can generate, with these instance, up-to 4 times more columns per iteration, so it is natural the in many cases this column selection rule ends up with a larger number of columns. It is interesting to observe that when *All* is the column selection rule with the largest number of columns, the gap between its number of columns and the minimal one is on average of 39.6%, with a minimal of 5.75%, and a maximal of 88.8%. This illustrates that, in general, the total number of columns with negative reduced cost is not particularly large. Such a comment, is confirmed by checking the same gaps for Q_{90} . It is possible to observe that the average gap in this statistic is 58.9%, the maximal is 151.0%, with a minimal of 21.1%, confirming that, in general, most pricer iterations do not

produce a negative reduced column. Finally, it is possible to observe that, in the average case, the total number of generated columns tends to be quite consistent among the column selection rules. In fact, only in 40% of the cases the gap between the largest and smallest number of columns for each selection rule is larger than 50%. Similar figures can be observed also for Q_{75} , where only 38% of the cases have a gap larger than 50%. For Q_{90} , we observe practically the same figures as in Q_{75} .

In Table 4.11, it is possible to observe that, in general, there is a mild correlation between the number of columns and the number of crowshippers. In fact, for $|N| = 24$ Pearson's correlation takes value 0.43, for $|N| = 36$ it takes value 0.19, for $|N| = 40$ the correlation coefficient is 0.51, and for $|N| = 44$ it is 0.59. All mild correlation coefficients are intuitive, and don't need any explanation. On the contrary it is interesting to observe the small correlation coefficient for $|N| = 36$. It is possible to observe in Table 4.11 that in the average case for $|N| = 36$ the algorithm generated fewer columns for $|K| = 1000$ than for $|K| = 500$. This is possible to observe that in these instances with $|K| = 500$ 86.8% of the parcels are transferred with the backup service, whereas with $|K| = 1000$ with figure takes value 73.1%. As already observed in Table 4.9, the larger the fraction of backup parcel, the harder the problem. Therefore, such a decrease in the number of generated column is just related to the instance properties.

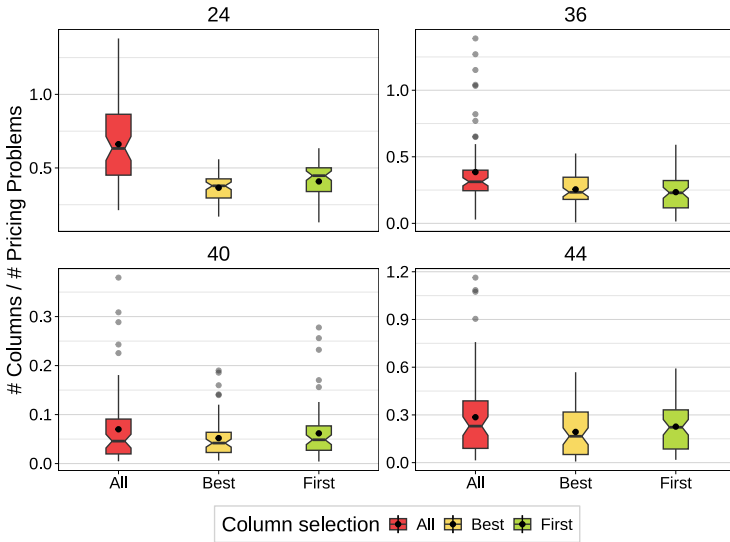


Figure 4.1: Distribution of the ratios between the number of generated columns and number of pricing problems

To conclude this analysis, Figure 4.1 presents, for each column selection rule and each network, the distribution of the ratio between the number of generated columns and the number of pricing problems. The dot in inside the boxplots represent the average value. First of all, notice that for *All* it is normal to generate more than one column for each pricing problem, therefore it is normal to have ratios larger than one. Moreover, it is possible to observe that, with the exception of $|N| = 24$, the distribution of these ratios are quite consistent between the column selection rules. Not only this further confirms that the general behavior of the column selection rules is relatively consistent, but it helps to explain why

the column selection rules have practically indistinguishable performance. In most cases, less than 50% of the pricing problems generate a column. Moreover, in extreme some cases, like for $|N| = 40$, this percentage goes below 10%. This suggests that in many cases, all the selection rules propose the same column. When this happens over a large number of pricing problems, the differences between the column selections averages out, leading to relatively consistent performances.

Now, we focus on the outlier represented by *All* in $|N| = 24$. It is possible to observe that average ratio is 0.66, the largest among all configurations and networks, while the average ratio for *Best* is 0.36, and for *First* it is 0.40. This behavior is easy to explain. All networks are used to construct instances with crowdshipper sets of the same cardinality. It is easy to imagine that in a small network a crowdshipper of a given size is more likely to provide redundant paths than in a larger network. And with more redundant paths it is easy to generate more columns. This has an even larger impact with column selection rule *All*, that proposes multiple column for each parcel.

4.4.6 Comparison of the Solution Methods for PTCP

In this section, we provide a comparison of the three solution method proposed for PTCP. We generated a set of 128 instances by sampling larger scale instances introduced in Section 4.4.1. For each network, we selected 8 instances, and each was sampled to have $|P| = 50, 100, 150, 200$ and $|K| = 50, 100, 150, 200$. All combinations of $|P|$ and $|K|$ are tested. Such small sizes are chosen to ensure that all methods can tackle these instances effectively. To provide a meaningful comparison of the methods, we choose to only represent the 29 instances where both CPLEX and the proposed column generation found the optimal solution. Table 4.12 summarizes the results.

The first three column represents the network (N), the number of crowdshippers ($|K|$), and number of parcels ($|P|$). Then, the next two column, with the common label Obj. Val., report the objective value from the heuristic (ALNS) and from the exact algorithms (Opt.). Finally, the last three columns, with the common label Time, show the computational time from CPLEX, ALNS, and the column generation. For the ALNS, we report the time to best, whereas for the other methods the total computational time is shown.

ALNS was configured with the best performing parameter setting, found in Section 3.7.3. The column generation was configured to include all negative reduced cost columns in the RMP.

First of all, we discuss the objective values. In only 10 out of 29 instances ALNS did not find the optimal solution. Moreover, in these 10 cases the average gap between the optimal and worst solution is 0.35%, showing the effectiveness of the ALNS in this small-sized instance set.

Now, we analyze the computational time. CPLEX scores an average time of 238.21s, with a minimum of 10.6s, and a maximum of 1506.62s (approx. 25 minutes). ALNS scores an average, minimum, and maximum time of 61s, 0s, and 563.73s respectively. Finally, BnP takes 0.45s in the average case, with a minimum of 0.02s and a maximum of 1.44s. These results are quite unsurprising.

CPLEX presents a large variance in its run time values, with a standard deviation 336.24. In general, CPLEX seems to be more negatively affected by a larger number of crowdshipper than a by a larger number of parcels. This can be notices by observing that the Pearson's coefficient between CPLEX solve time and $|K|$ is 0.68, a value showing a strong correlation, while the same statistic takes between solve time and $|P|$ takes value 0.36, showing a weak

Table 4.12: Comparison of solution methods for PTCP

N	K	P	Obj. Val.		Time [s]		
			ALNS	Opt.	MILP	ALNS	BnP
36	50	200	4296.00	4296.00	68.61	0.26	0.66
36	50	150	3138.00	3138.00	43.24	0.86	0.37
40	50	200	4228.00	4228.00	94.71	0.01	1.02
36	50	50	1019.00	1019.00	13.02	0.01	0.05
40	50	150	3211.00	3211.00	25.13	0.00	0.13
36	100	150	3112.00	3112.00	250.91	40.60	0.49
40	50	150	3216.00	3216.00	33.61	0.02	0.39
40	100	50	983.00	976.00	23.38	2.56	0.02
44	50	200	4135.00	4135.00	36.51	0.09	0.50
36	150	100	1829.00	1805.00	400.67	109.72	0.35
24	50	150	3109.00	3109.00	48.57	0.01	0.16
40	150	50	1020.00	1020.00	90.72	3.88	0.10
24	50	200	4257.00	4257.00	127.04	0.02	0.88
40	150	200	4301.00	4301.00	1506.62	0.00	0.66
40	100	100	2064.00	2064.00	128.07	33.96	0.33
44	50	50	1001.00	1001.00	12.58	0.00	0.05
40	150	100	1981.00	1979.00	275.27	140.71	0.46
44	50	50	965.00	964.00	10.60	1.98	0.06
36	50	100	2103.00	2103.00	26.44	2.04	0.23
36	100	100	1927.00	1925.00	242.94	175.30	0.40
36	150	150	2983.00	2980.00	597.02	253.94	0.74
36	150	200	3990.00	3982.00	987.18	37.43	1.44
44	100	200	4107.00	4107.00	311.67	11.46	0.88
36	200	100	1608.00	1598.00	478.02	209.90	0.38
40	150	150	3025.00	3021.00	598.78	563.73	0.73
36	150	100	1890.00	1888.00	296.86	158.13	0.32
24	50	150	3161.00	3161.00	120.42	0.13	0.87
36	50	100	2078.00	2078.00	28.53	5.68	0.22
44	50	150	3036.00	3036.00	30.94	16.34	0.32

correlation. Moreover, Pearson's correlation between solve time and the product $|P| * |K|$ is 0.91, a very strong correlation. Such coefficients show the large impact the scale of the problem has on CPLEX ability to solve it efficiently, therefore explaining the large variance in its solve time.

It is interesting to notice that in 3 cases, ALNS found the optimal solution just by using its constructive heuristic. It is possible to identify these cases by observing the ALNS time column with entries 0.0. In general, ALNS is quite fast at identifying high quality solution. This can be noticed by looking at the 75th quantile of the run times, taking value 40.6s. However, in the remaining 25% of the instances ALNS struggle to find the best solution quick. In fact, the average solve time in the long running 25th percentile is 230.20s.

Finally, BnP is clearly the best performing method. This is clear not only in terms of run time but also by taking into account their consistency. Indeed, solve time standard deviation is just 0.34. Interestingly, unlike CPLEX, BnP has a no evident correlation, taking value 0.21,

between $|K|$ and its solve time, while it has a strong correlation, 0.78, between the solve time and $|P|$. This is natural, since more parcels means more pricing problems.

In summary, these analysis show that all methods can be effective in solving PTCP, and each comes with its strengths and weaknesses. However, considering the scalability issues presented by CPLEX, BnP is the best method to solve large scale instances at optimality.

4.5 Conclusions

In this Chapter we formulated a set covering formulation of PTCP introduced in Chapter 3. To solve such a formulation efficiently, we propose and implement a Branch-and-Price algorithm, where the pricing problem is solved using a tailored dynamic programming algorithm. The proposed algorithm outperforms by several orders of magnitude the performance of the compact MILP model proposed in Chapter 3. We've conducted a large number of computational experiments, highlighting the impact over the total computational time of parameters like the number of crowdshippers and parcels, and algorithmic configurations. These computational experiments show that the harder instances are the ones with a larger share of backup parcels in the optimal solution. However the latter figure does not affect the strength of the linear relaxation.

Thanks to its good computational performance, proposed formulation and algorithm can be extended to take into account the stochastic nature of the problem at hand. Such extensions include a stochastic or robust reformulations of the set covering model (4.1)–(4.7).

Another interesting avenue for future research is the design and implementation of a predictor to automatically switch between the available column selection rules. In this way, the algorithm could automatically adapt to the current state of the CG and the BnB tree and accelerate the search procedure.

Part II

Equitable Vehicle Routing Problems

Chapter 5

An Off-Peak Hours Delivery Policy to Mitigate Last-Mile Delivery Social Impact

This chapter extends Renata Mansini, Filippo Ranza, and Roberto Zanolli. Towards sustainable last-mile delivery: Introducing an off-peak urban policy to mitigate environmental and social impacts. IFAC-PapersOnLine, 58(2):186–191, 2024. ISSN 2405-8963. 3rd IFAC Workshop on Integrated Assessment Modeling for Environmental Systems IAMES 2024. The original paper was presented at the 3rd IFAC Workshop on Integrated Assessment Modeling for Environmental Systems.

Contents	
5.1	Introduction 103
5.2	Problem Description 104
5.3	Mathematical Formulation 105
5.4	Instance Generation 107
5.5	Experimental Analysis 108
5.5.1	Benchmark Instances 108
5.5.2	Managerial Insights and Sensitivity Analysis 109
5.6	Conclusions 112

Summary In recent years, last-mile delivery research has shifted from focusing solely on cost minimization to a broader consideration of sustainability and environmental impact. The following chapter explores the novel trend of evaluating the social impact of last-mile delivery, emphasizing factors such as road safety, congestion, and noise pollution. A key strategy that has garnered attention in addressing these multifaceted concerns is Off-Peak Hours Delivery (OPHD), which involves scheduling deliveries during non-peak hours, often in the evenings or at night. This chapter investigates implementing a city-wide policy inspired by OPHD, strategically encouraging or discouraging deliveries in specific urban areas and time slots. The objectives of such a policy span from environmental considerations to

nuanced social factors, potentially leading to dynamic shifts in time slot preferences. Adopting such a policy aligns with the growing citizen concern for environmental and social issues, leveraging changing attitudes toward sustainable practices. To evaluate this policy, we formalize it as a bi-objective mixed-integer programming model, aiming to strike a balance between the economic interests of retailers and the municipality's overarching goals. Through realistic instances, the paper offers managerial insights, providing valuable perspectives on the practical effects of the proposed policy on delivery operations. This facilitates a deeper understanding of the ramifications of integrating social considerations into the optimization framework.

5.1 Introduction

The evolution of last-mile delivery within the scientific community has witnessed a continual shift in focus, transitioning from initial considerations centered around cost minimization and profit maximization (Mansini and Zanotti, 2022; Ouyang et al., 2023; Côté et al., 2024a) towards sustainability goals. The surge in e-commerce, accentuated by the COVID-19 pandemic, has underscored the relevant impact of last-mile delivery companies not only on their direct customers but on the broader community. Notably, sustainability and environmental evaluation of last-mile delivery impact have garnered significant attention, prompting a thorough examination of various components and the proposal of different solutions. These include analyses of different-sized time windows (Manerba et al., 2018a), the integration of parcel lockers (Bonomi et al., 2022a), and the use of drones and autonomous robots (Lemardelé et al., 2021), among other aspects. An emerging trend involves assessing the social impact of last-mile delivery, encompassing considerations such as road safety, congestion, and noise pollution (Viu-Roig and Alvarez-Palau, 2020). Addressing the environmental, social, and cost challenges induced by last-mile delivery, the Off-Peak Hours Delivery (OPHD) strategy has gained attention (Sánchez-Díaz et al., 2017). OPHD entails scheduling deliveries during evening or overnight hours to bypass traffic congestion, reduce overall delivery time, and achieve concurrent benefits such as cost reduction (Fu and Jenelius, 2018), pollution mitigation (Mousavi et al., 2021; Saleh et al., 2022), and enhanced customer satisfaction (Holguín-Veras et al., 2011). Primarily adopted in freight transportation for the last mile, OPHD faces challenges such as recipient resistance and community rejection due to overnight noise pollution (Holguín-Veras et al., 2006a,b). An additional concern arises from the health and annoyance costs associated with road noise exposure, as reported by the European Environment Agency. These costs escalate from 17€ (noise between 50 and 54dB) to 72€ (over 75dB) per dB (European Environment Agency, 2019).

A municipality can employ several strategies to alleviate the impact of last-mile logistics in urban areas, including the promotion of sustainable mobility through bike rental programs, the establishment of safe paths for foot or bicycle couriers, the identification of restricted traffic zones in densely populated or historical areas, and encouragement of electric or low-emission vehicle use in designated zones. Advanced traffic monitoring systems and the introduction of logistic consolidation hubs further contribute to efficient traffic flow and the use of electric vehicles.

This study explores the potential impact of a city-wide policy for home deliveries that closely aligns with OPHD, investigating its effects on delivery companies (or retailers) and their operational costs. Instead of imposing night deliveries, which might face limited customer acceptance, we consider a scenario where a city administration seeks to encourage or discourage deliveries in specific time slots according to the urban areas. These time slots

could align with the conventional OPHD definition or be chosen for different reasons. For instance, a focus on the social impact of last-mile operations might lead to discouraging deliveries in a mid-afternoon time slot in a district with several schools, coinciding with children going home. Moreover, the flexibility of these time slots allows for dynamic adjustments, such as discouraging deliveries during specific public events that attract crowds. Adopting such a policy holds the potential for favorable reception by citizens, leveraging the increasing importance placed on environmental and social issues. Additionally, coupling this policy with promoting unattended delivery paradigms, like parcel lockers, can mitigate the slight loss in flexibility regarding delivery options. In freight transportation, delivery companies naturally lean towards an OPHD approach due to its inherent benefits (Sánchez-Díaz et al., 2017). However, customer reluctance often hinders its adoption, even with financial incentives (Holguín-Veras et al., 2006a). Navigating the complex landscape of multi-stakeholder optimization, where each stakeholder may have multiple conflicting goals, presents a significant challenge. Identifying which objectives to optimize and how to achieve a satisfactory balance becomes intricate, especially when dealing with a problem of this magnitude (Bonomi et al., 2024b). When the companies' goals align with those of the city administration, particularly in the joint pursuit of environmental goals through time window penalties, there is room to control traffic congestion during peak hours, leading to lower emissions and reduced average time-to-delivery. However, the distribution company's primary goal is to minimize transport costs while ensuring customer satisfaction. This objective may still conflict with the municipality's objectives. It is not uncommon for a delivery company, especially when customers pay a premium for its services, to incur additional costs to meet customers' expectations. These added costs may include expenses related to congestion control implemented by city administrations, typically involving variable urban tolls based on the time of day to discourage unnecessary vehicle access. Conversely, when social considerations dictate penalties for specific time windows, striking the right balance between citizens' benefits and the economic viability of last-mile operations becomes crucial for promoting a more sustainable last-mile logistics system.

To model such a potential trade-off, we propose a bi-objective vehicle routing problem optimizing the goals associated with the main decision-makers: the retailer, primarily interested in minimizing total travel time, and the municipality, which seeks to reduce social and environmental impact by mandating the retailer's minimization of penalties for accessing specific zones within defined time windows. We call this problem the Off-Peak Vehicle Routing Problem (OP-VRP).

5.2 Problem Description

Let D be the set of districts (areas, zones) into which a city can be partitioned. The OP-VRP can be formulated on a directed graph $G = (V, A)$ where the node set $V = \{0\} \cup N$ consists of the depot (node 0) and the set N of delivery points (customers) within the urban area, whereas arc set A represents the connections between nodes. Set N can be partitioned into $|D|$ disjoint subsets, i.e., $N = \cup_{d \in D} N_d$ with $N_d \cap N_f = \emptyset \forall d, f \in D, d \neq f$. Subset N_d represents the subset of delivery points located in district $d \in D$. The travel cost and time associated with each arc $(i, j) \in A$ are denoted as c_{ij} and t_{ij} , respectively. Both travel costs and times satisfy the triangular inequality and are proportional to the distances traveled. The service time required to serve the delivery point $i \in N$ is represented by s_i . We refer to W as the set of disjoint time slots into which the day (or specific periods like morning/afternoon) is partitioned. Penalty coefficient $p_w^d > 0$, used to incentivize or penalize a specific

time window of a defined district, serves as a multiplicative factor of the transportation cost incurred by the delivery company when serving delivery points belonging to district $d \in D$ during the time interval $[a_w, b_w]$, where a_w and b_w denote the start and end times of time slot $w \in W$. The fleet of vehicles involved in the service is denoted as F . Each vehicle must return to the depot within a working time limit t_{max} . The retailer's objective is to visit all delivery points, minimizing the total time required for the service and, as a secondary objective, minimizing the total penalties paid for visiting customers within the districts during the selected time slots.

5.3 Mathematical Formulation

To provide a mathematical formulation of the OP-VRP we introduce the following sets of variables.

For each district $d \in D$ and each time slot $w \in W$, we introduce the binary variable y_w^d , taking value 1 if district d is visited in time slot w , 0 otherwise. The second set of binary variables is defined for each arc $(i, j) \in A$, x_{ij} take value 1 if arc (i, j) is traversed, and 0 otherwise. The final set consists of continuous non-negative variables z_{ij} for all $(i, j) \in A$, defining the arrival time at node j when coming from node i . From now on, for any subset $S \subset V$, let $\delta^+(S) = \{(i, j) \in A : i \in S, j \notin S\}$ and $\delta^-(S) = \{(i, j) \in A : i \notin S, j \in S\}$ be the sets of arcs leaving from and entering set S , respectively. Moreover, for simplicity, we call $\delta^+(i) = \delta^+(\{i\})$, $i \in V$ and $\delta^-(i) = \delta^-(\{i\})$, $i \in V$.

The mathematical formulation of OP-VRP as a bi-objective problem is as follows:

$$\min f_1 := \sum_{(i,j) \in A} t_{ij} x_{ij} \quad (5.1)$$

$$\min f_2 := \sum_{d \in D} \sum_{w \in W} p_w^d y_w^d \sum_{i \in N_d} \sum_{j \in \delta^-(i)} c_{ji} x_{ji} + \sum_{(j,0) \in \delta^-(0)} c_{j0} x_{j0} \quad (5.2)$$

$$\sum_{(0,i) \in \delta^+(0)} x_{0i} = \sum_{(j,0) \in \delta^-(0)} x_{j0} \leq |F| \quad (5.3)$$

$$\sum_{(j,i) \in \delta^-(i)} x_{ji} = \sum_{(i,j) \in \delta^+(i)} x_{ij} = 1 \quad i \in N \quad (5.4)$$

$$\sum_{(i,j) \in \delta^+(i)} z_{ij} - \sum_{(j,i) \in \delta^-(i)} z_{ji} \geq \sum_{(i,j) \in \delta^+(i)} (t_{ij} + s_i) x_{ij} \quad i \in N \quad (5.5)$$

$$z_{0i} = t_{0i} x_{0i} \quad i \in \delta^+(0) \quad (5.6)$$

$$z_{ij} \geq (t_{0i} + t_{ij} + s_i) x_{ij} \quad i, j \in V, i \neq j \quad (5.7)$$

$$z_{ij} \leq (t_{max} - s_j - t_{j0}) x_{ij} \quad i, j \in V, i \neq j \quad (5.8)$$

$$\sum_{w \in W} a_w y_w^d \leq \sum_{(j,i) \in \delta^-(i)} z_{ji} \quad d \in D, i \in N_d \quad (5.9)$$

$$\sum_{(j,i) \in \delta^-(i)} z_{ji} \leq \sum_{w \in W} b_w y_w^d \quad d \in D, i \in N_d \quad (5.10)$$

$$\sum_{w \in W} y_w^d \leq 1 \quad d \in D \quad (5.11)$$

$$x_{ij} \in \{0, 1\} \quad (i, j) \in A \quad (5.12)$$

$$z_{ij} \geq 0 \quad (i, j) \in A \quad (5.13)$$

$$y_w^d \in \{0, 1\} \quad d \in D, w \in W \quad (5.14)$$

Objective function f_1 in (5.1) minimizes the total travel time of all vehicles. In contrast, Objective function f_2 in (5.2) minimizes the total penalty. For each district $d \in D$, the latter is computed as an approximation of the total traveling costs associated with selected arcs entering nodes of the district ($\sum_{i \in N_d} \sum_{j \in \delta^-(i)} c_{ji} x_{ji}$) multiplied by the penalty term $\sum_{w \in W} p_w^d y_w^d$ that depends on the time slot, among all the available ones, that will be selected for the district (corresponding variable y_w^d set to 1). To account for the cost of returning to the depot, we also add a term ($\sum_{(j,0) \in \delta^-(0)} c_{j0} x_{j0}$). Notice that objective function f_2 directly multiplies variables x and y and thus is non-linear. However, it can be linearized. First, we introduce the non-negative variable q_w^d , taking the value of the associated penalty if district d is traversed during time window w , and zero otherwise. The linear reformulation of (5.2) is as follows:

$$\min \sum_{d \in D} \sum_{w \in W} q_w^d + \sum_{(j,0) \in \delta^-(0)} c_{j0} x_{j0} \quad (5.15)$$

$$q_w^d \geq p_w^d \sum_{i \in N_d} \sum_{j \in \delta^-(i)} c_{ji} x_{ji} - M_d(1 - y_w^d) \quad w \in W, d \in D \quad (5.16)$$

$$q_w^d \geq 0 \quad w \in W, d \in D \quad (5.17)$$

where M_d is a large enough value that can be set to $\sum_{i \in N_d} \max_{j \in \delta^-(i)} c_{ji}$. Notice that, due to constraints (5.11), at most one variable y_w^d will take value 1 for each district $d \in D$, eliminating M_d and forcing the corresponding variable q_w^d to take the value $p_w^d \sum_{i \in N_d} \sum_{j \in \delta^-(i)} c_{ji} x_{ji}$. Constraints (5.3) indicate that no more than $|F|$ vehicles can be used and that the number of vehicles that leave the depot must also re-enter it. Constraints (5.4) guarantee that exactly one arc enters and leaves each node $i \in N$. Constraints (5.5) manage subtour elimination by establishing that if a vehicle visits node j right after node i (i.e., $x_{ij} = 1$), the time elapsed between the arrival times in the two nodes is at least equal to the service time s_i required to serve node i plus the travel time t_{ij} required to go from i to j (Gobbi et al., 2019a, 2023b). Constraints (5.6) set the minimum time required to reach the first node i after the depot, whereas Constraints (5.7)-(5.8) define lower and upper bounds on the values of each variable z_{ij} , $i, j \in N, i \neq j$. Constraints (5.9) and (5.10) force the activation of variable y_w^d when the arrival time at node $i \in N_d$ is included in the time slot $[a_w, b_w]$. Constraints (5.11) impose that, for each district $d \in D$, at most one time slot $w \in W$ can be activated. Finally, the remaining constraints are variable definitions and domain restrictions.

Model (5.1)–(5.14) guarantees optimality is the graph it's formulated on has costs t_{ij} respecting the triangular inequality. This property is connected to Constraints (5.7) and (5.8). Specifically, Constraints (5.7) tighten the range of values z_{ij} can assume, by taking into account the time it takes to reach node i from the depot 0. Such bounds are valid if and only if $t_{0j} < t_{0i} + s_i + t_{ij} \forall (i, j) \in A$, otherwise, if such an inequality is not satisfied, the associated bound will render arc $(0, j)$ unusable. Clearly, only graphs whose distances respect the triangular inequality will satisfy such an inequality for all values of s_i . The case for (5.8) is perfectly analogous, therefore will not be discussed further.

If the underling graph does not respect the triangular inequality, two courses of actions are possible.

1. It is possible, for instances when the triangular inequality is violated by a small margin, to *triangularize* the graph. Such a procedure replaces the cost of the arc (i, j) with the cost of the shortest path from i to j . Since a shortest path is made of sub shortest paths, this will ultimately guarantee that the new distances respect the triangular inequality.

2. If the former point is not applicable, Model (5.1)–(5.14) can be relaxed by removing (5.7) and by relaxing (5.8) to

$$z_{ij} \leq t_{\max} x_{ij} \quad (i, j) \in A. \quad (5.18)$$

Such a re-formulation is correct for graph not respecting the triangular inequalities. Moreover, it remains correct for graph respecting it, albeit with weaker bounds.

5.4 Instance Generation

The construction of realistic instances for the vehicle routing problem requires an accurate and coherent representation of the underlying street network. The street topology of the selected city was therefore extracted and processed from OpenStreetMap¹ data through OSMnx (Boeing, 2017). This approach ensures that the resulting network reproduces the real urban infrastructure while remaining fully reproducible and transparent.

OpenStreetMap was chosen as the data source because it provides open, community-curated geographic information covering a broad range of urban and transportation features. Its open-source nature grants full control over data acquisition and processing, facilitating replication and verification by other researchers. Although minor local inconsistencies may occur, the completeness and reliability of OpenStreetMap in most metropolitan areas make it particularly suitable for studies of urban mobility and last-mile logistics.

OSMNx retrieves street networks directly from OpenStreetMap by specifying both a geographical boundary and a network type. In this study, the administrative limits of the target urban area were used to define the region of interest. Only roads accessible to motorized vehicles were included to ensure that the resulting graph represents the infrastructure relevant to last-mile delivery operations. OSMNx automatically downloads all nodes and edges within the defined boundary, along with associated metadata such as street names, segment lengths, functional classifications, and access restrictions.

During graph construction, OSMnx performs several preprocessing operations to refine and standardize the data. The network is represented as a directed multigraph, which allows multiple edges between the same pair of nodes. This representation accommodates typical urban layouts, such as multi-lane highways or parallel one-way streets. OSMnx simplifies the topology by merging consecutive edges that share identical attributes and by collapsing intermediate nodes that do not represent actual intersections or terminal points. Edge attributes are standardized, self-loops and redundant edges are removed, and disconnected components are identified for possible exclusion. The network is then projected from geographic coordinates to a planar coordinate system to enable accurate metric distance calculations.

Despite the extensive preprocessing carried out by OSMnx, additional post-processing was required to ensure consistency and analytical robustness. Very small disconnected components, often representing cul-de-sacs, parking access roads, or mapping artifacts, were identified and removed to guarantee full network connectivity. The classification of road types was also standardized to obtain uniform and coherent labeling across the dataset. In particular, inconsistencies such as the coexistence of “residential” and “living street” categories, which refer to comparable functional types, were harmonized under a single classification scheme. This normalization ensures a consistent interpretation of street hierarchy and accessibility.

¹openstreetmap.org

After the construction and cleaning of the street network, a subset of nodes was identified to represent potential customer locations. Candidate nodes were selected among those tagged as commercial or service-oriented, including categories such as *commercial*, *amenity*, *office*, *library*, and *logistics*, among others. The procedure implemented to select suitable customer nodes proceeds as follows. First, a random node is chosen to serve as the depot. Then, the list of potential customer nodes is randomly shuffled. The algorithm iterates through the shuffled list and selects nodes according to the following conditions: a) a minimal distance $d_{\min}$ between any pair of selected nodes, b) a minimal distance $2d_{\min}$ between each selected node and the depot, c) an average distance between all selected nodes and the depot close to a target value d_{avg} , and d) compliance with the node type criteria described above. If the algorithm fails to select a sufficient number of customer nodes, the entire procedure restarts until the desired number of nodes is obtained.

The resulting graph provides a coherent and computationally efficient abstraction of the street infrastructure of the selected city, where intersections are represented as nodes and street segments as weighted edges. OSMnx returns the edges weights in terms of distance, we convert their traversal time in seconds, assuming an average speed of 30km/h. This graph, together with the selected customer nodes and depot, forms the spatial foundation for the generation of last-mile delivery instances and for the evaluation of routing algorithms under realistic urban conditions.

5.5 Experimental Analysis

In this section, we present the results obtained by solving the OP-VRP mathematical model on realistic instances and evaluate the interplay between the *company oriented* objective function (5.1) and the *municipality oriented* one (5.2). All tests have been conducted on a computer equipped with an Apple M3 Max with 64Gb of RAM running macOS 26.1, using Gurobi 12 as a Mixed Integer Programming Solver. All instances solved to optimality.

5.5.1 Benchmark Instances

We generated instances based on the road graph of San Francisco, CA, USA, using OpenStreetMap data. We generated instances with 14 and 19 customers nodes randomly selected on the map, with at least one node for each city district ($|D| = 11$) and an additional node outside the city serving as the depot. Hence, $|V| = 15$ or $|V| = 20$. Together with the rules described in Section 5.4, these points are selected ensuring that (i) there is a customer within the border of each district and (ii) the depot is outside the city border (as shown in Figure 6.21), so no district suffers an unfair disadvantage. Two values of $|W|$, corresponding to 1-hour and 30-minute time slots, are considered over an operational time window $t_{\max}$ of 4 hours. Results should be scalable based on an increase of $t_{\max}$ and the number of customers. For each district $d \in D$, we set p_w^d values such that the best time slot has $p_w^d = 1$ (no penalty), and the worst has $p_w^d = 2$ (twice the arc costs). Time slots in between have evenly spaced penalties between 1 and 2. Computed penalties are randomly assigned to each time slot. The number of vehicles $|F|$ is always set to 3. We generated 20 base graphs with different customer locations. Given $|W| \in \{4, 8\}$, the total number of instances is 80. Figure 6.21 provides the visual representation of an instance with dotted lines indicating district borders, different colors representing each district, and symbols denoting the depot and the customers.

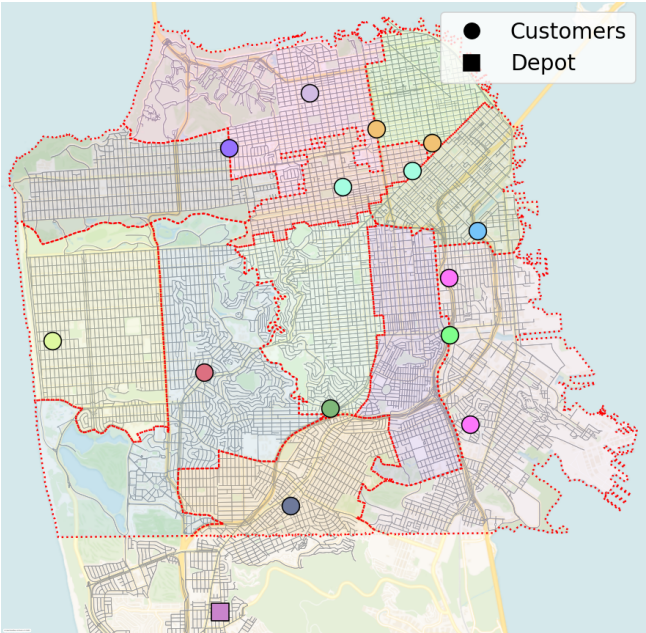


Figure 5.1: Example of an instance with 15 nodes.

5.5.2 Managerial Insights and Sensitivity Analysis

The initial analysis revolves around the impact of constraints (5.11) that impose entering each district within a single time slot.

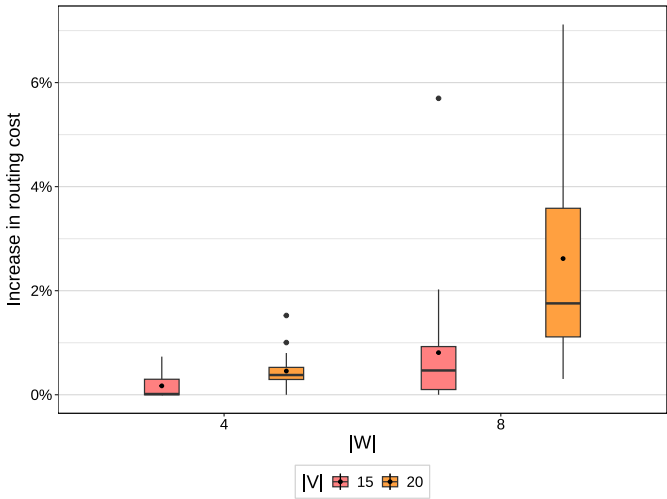


Figure 5.2: Worsening of f_1 with constraints (5.11).

Figure 5.2 illustrates the value deterioration of f_1 when such constraints are incorporated, contrasting the optimization outcomes with and without them. The color of the box plot

identifies the value $|V|$. As expected, the use of shorter time slots ($|W| = 8$) exhibits a more noticeable effect, resulting in an average increase of about 0.83% and 2.61% in the total time required for delivery operations for $|V| = 15$ and $|V| = 20$ respectively. Instances with negligible impact (approximately 0%) are observed, while the maximal increase for $|V| = 15$ is 5.70% and for $|V| = 20$ is 7.12%. When $|W| = 4$, the average impact stays below 0.5% for both instance sizes, with the maximum hovering around 0.73% and 1.52% for $|V| = 15$ and 20. A 1-hour time window appears reasonable for most practical scenarios, such as discouraging visits to commercial areas during rush hours or school districts during the start or end of the school day. This initial assessment shows how the one-slot policy can influence delivery operations. In this scenario, the delivery company can select its preferred time slot without considering how the city decision-makers value it. The more fine-grained the control over the access to a district is, the higher the increase in costs will be.

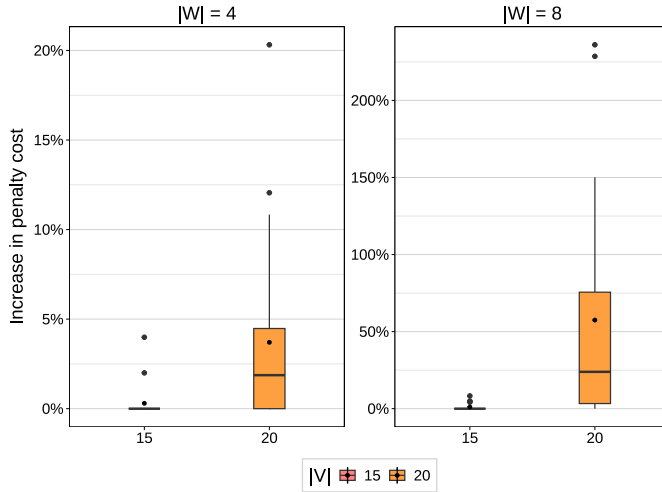


Figure 5.3: Worsening of f_2 with constraints (5.11).

Figure 5.3 shows the distribution in penalty costs (f_2) when constraints (5.11) are enforced. Notice that quadrants are on different scales. It is interesting to observe that with the exception of a couple of outliers, constraints (5.11) have no effect on the value of f_2 when $|V| = 15$. However, when $|V| = 20$ the outcomes are quite different. In fact, there is an average increase of 3.70% and 57.48% for $|W| = 4$, and 8 respectively. Naturally, instance with more customers require a vehicle to spend more time inside a district. So, enforcing constraint (5.11) might force the solver to identify paths that traverse exactly one district per time slot, instead of selecting the same time slot for multiple district, reducing the freedom it has to visit districts only during low penalties time windows. While 1 hour time windows might prove to be quite challenging to correctly navigate, the true problem rises with 30 minutes time windows.

Figure 5.4 provides a visual representation of how solutions evolve when optimizing f_1 without constraints (5.11), with them, and when optimizing f_2 , respectively. In this specific instance with $|W| = 4$, the amount of traveled kilometers varies significantly: from 47.9 in the first case to 56.8 in the second, reaching 64.3 in the third one. Notably, the first solution achieves all deliveries with a single vehicle, while the other two require the activation of two vehicles. Interestingly, the one-slot policy forces the company to use a second vehicle to

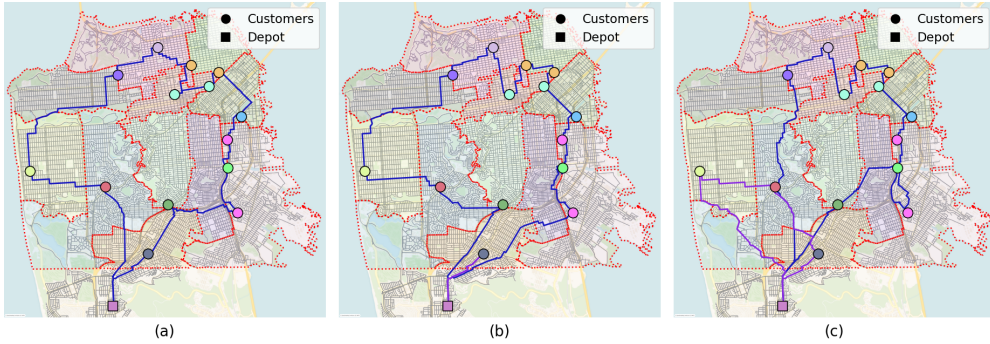
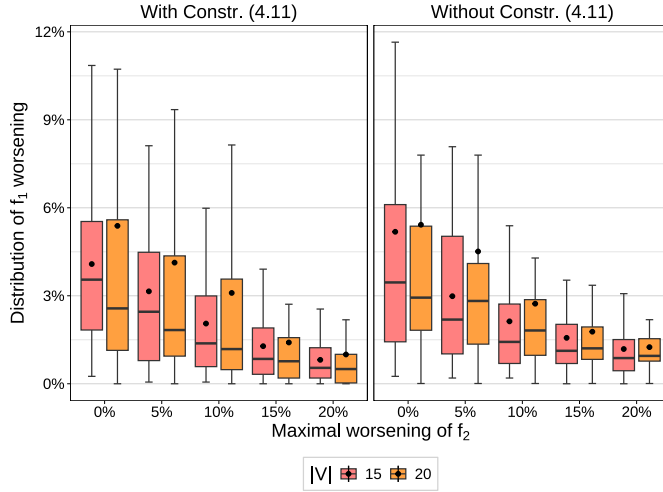


Figure 5.4: Example of a solution when optimizing (a) f_1 without constraints (5.11), (b) f_1 , and (c) f_2 .

service a single customer. Comparing the solutions in Figures 5.4(b) and 5.4(c) with the one in Figure 5.4(a), we can notice that the path followed to visit nodes located northeast of the city changes. Moreover, in both solutions, the driver exits and re-enters a district after visiting an adjacent one. This is possible when opting for the same time slot in both districts. A trade-off emerges: choosing the optimal slot for neighboring districts or finding a compromise with a *middle ground* slot. A preliminary insight, applicable to all instances, is evident: the introduction of a single-slot policy and the penalization of unwanted slots can result in a noticeable increase in total traveled distance and, consequently, environmental emissions. However, it is important to recognize that this model is more tactical than operational. For instance, if a city discourages a specific time slot due to heavy traffic, yet a company uses it based on lower predicted distance, this may not translate to lower emissions in practice. The company driver may get stuck in traffic in that rush hour slot with the engine of its vehicle still producing emissions. Nevertheless, in certain cases, a trade-off between distance traveled and added social value will emerge. Balancing social (e.g., avoiding a school-day ending) against environmental costs becomes a crucial consideration.

To comprehensively assess the interplay between the two objectives, we employ the ϵ -constraint method considering a specific lexicographic order. In this method, we prioritize and optimize objectives sequentially, ensuring that while enhancing lower-priority objectives, higher-priority ones remain within defined boundaries. Initially, we optimize f_1 , yielding an optimal value denoted as f_1^* . Subsequently, we solve five additional problems where f_2 is minimized, while bounding f_1 to stay below predefined thresholds: 0%, 5%, 10%, 15%, and 20%, both with and without constraints (5.11). The reverse process starts by optimizing f_2 . Therefore, each instance we conduct 24 optimizations. Across the 80 instances, a total of 1920 optimizations are performed. On average, each optimization required 17.90 seconds, with a maximal of 3229s. Solving f_1 first (without constraints on f_2) is slightly harder than solving f_2 first (without constraints on f_1). On average the first optimization takes 27.53s, while the latter takes 12.49s. In general optimizing with constraints (5.11) takes on average 26.67s vs 9.14s of optimization without it. The optimization of f_2 with $|W| = 8$ and constraints (5.11) enforced takes by far the longest, with an average time of 101.93s. In the second place there the optimization of f_1 in the same conditions with an average time of 78.13s. Figure 5.5 illustrates the degradation in f_1 when optimizing f_2 first and allowing a certain percentage deterioration in f_2 .

It is evident that f_2 significantly influences the value of f_1 . In general it is possible to


 Figure 5.5: worsening of f_1 for different degradation of f_2 .

notice that when constraints (5.11) are enforced the average worsening of $f_1|(f_2, \epsilon)$ when compared to f_1^* is smaller than without the constraint. Even if counter intuitive this behavior is natural. Adding a constraint worsens both f_1 and f_2 , and reduce their *free space* to worsen each other in order to improve.

5.6 Conclusions

Municipalities can implement different policies to reduce congestion during peak hours, including dynamic toll systems based on traffic density, which dissuade vehicles from entering the city unnecessarily during high-demand periods. In this study, we address a complex optimization problem for last-mile delivery that incorporates both environmental and social considerations to discourage retail companies from delivering in specific time slots. We name it the Off-Peak Vehicle Routing Problem (OP-VRP) and develop a bi-objective mixed-integer programming formulation optimizing both the traveling time and the total penalty costs incurred by a delivery company. We solve the problem on realistic instances, encompassing 80 customers configurations and a total of 1920 optimizations. Results reveal that restricting to one time slot per area leads to a modest median cost increase of approximately 3%, even in the challenging setting of 30-minute time slots. A noteworthy observation is that allowing a 5% cost increase could help delivery companies align their objectives more closely with the city administration's goals. This finding suggests a potential avenue for collaboration, fostering a compromise that benefits both stakeholders. As a future development, we aim to refine the model by incorporating time-dependent travel times and costs, reflecting real-world traffic patterns. This evolution will enhance the model's applicability to urban logistics scenarios, providing a more accurate representation of the challenges and opportunities in the last-mile delivery optimization landscape.

Chapter 6

On the Equitable Vehicle Routing Problem

This chapter is extends from a submission to the European Journal of Operations Research. Part of this work was presented at the International Conference of Optimization and Decision Science ODS 2024.

Contents

6.1	Introduction	114
6.2	Literature Review	116
6.2.1	Fair and Equitable Vehicle Routing Problems	116
6.2.2	On the k -sum Optimization Problems	119
6.2.3	Kernel Search Algorithms	119
6.3	Problem Definition	121
6.4	Minimization of the k Largest Variables in a Linear Program	122
6.5	Mathematical Formulation	124
6.6	Solution Methodology	126
6.7	Real-World Instances	126
6.8	Sensitivity Analysis	129
6.8.1	Sensitivity analysis on main parameters	129
6.8.2	Managerial insights	140
6.8.3	Statistical analysis of the results	142
6.9	A Kernel Search for k-EqVRP	144
6.9.1	An Introduction to Kernel Search	144
6.9.2	The Kernel Search Matheuristic	145
6.9.3	Initial VRP solution	146
6.9.4	Variable Ranking	149
6.9.5	Kernel Search Algorithm	149
6.10	Computational Results	153
6.10.1	Instances generation	154
6.10.2	Comparison of the k -sum formulations	155
6.10.3	TS-ALNS	157
6.10.4	Gurobi performance on the k -sum formulation	160

6.10.5 SAKS performance 161

6.11 Conclusions 165

Summary In recent decades, the exponential growth of e-commerce has accelerated the expansion of last-mile delivery (LMD), the most costly and inefficient segment of the logistics chain, contributing to traffic congestion, noise, and pollution. Traditional LMD solutions are typically based on Vehicle Routing Problems (VRPs), focusing on cost minimization, fairness in the treatment of customers and workers, and the mitigation of environmental impacts. However, these approaches often fail to account for the disproportionate disruption endured by residents living near heavily congested roads, such as freeways, main streets, and inter-sections. In this chapter, we introduce the k -Equitable Vehicle Routing Problem (k -EqVRP), a bi-objective extension of the classical VRP that minimizes both the overall routing time and the disruption experienced by residents on the k most densely populated streets, where k is a parameter chosen by the planner. The two objectives are addressed in lexicographic order by means of the ϵ -constraint method, where the routing time is minimized first, and the disruption is subsequently minimized while allowing a controlled increase in travel time. The problem is modeled on a multigraph representation, which permits to avoid densely populated streets through alternative routing options while preventing excessive increases in travel distances. We conducted an extensive sensitivity analysis on a large number of instances derived from street networks and population data of Charleston and San Francisco, USA. The results show that it is possible to substantially reduce the impact on the population without incurring significant additional routing costs. The study also provides practical insights for policymakers and practitioners engaged in the design of sustainable urban transportation strategies. Moreover, to tackle large scale instances we propose a novel Self Adapting Kernel Search (SAKS) matheuristic, designed to handle k -EqVRP. Such a matheuristic, designed to dynamically adapt its metaparameters according to the algorithmic performance, proves to be very effective, returning high quality results in a short computational time.

6.1 Introduction

E-commerce has become synonymous with economic convenience, speed, and easy access to products. Recent data show that customers increased from 1.2 billion in 2017 to 2.4 billion in 2023, with revenue growth of 2.21 trillion over the same period. Analysts predict an annual growth rate of 9.48% in the next four years (Statista, 2024). However, this convenience comes at an environmental cost, particularly in urban areas. Home deliveries contribute to increasing vehicular traffic, adding pressure on already congested roads. Each delivery potentially introduces another vehicle, leading to slower urban mobility, higher levels of air and noise pollution, negative public health impacts, and decreased road safety.

Sustainable solutions are necessary to mitigate these issues and reduce the overall impact of e-commerce. These include optimizing either delivery routes to minimize the number of vehicles (Nasir and Kuo, 2024) or time windows assignment to delivery zones (Côté et al., 2024a), investing in low-emission transport systems such as electric vehicles (Bruglieri et al., 2023), exploring new delivery paradigms such as crowd-shipping and off-peak deliveries (Gajda et al., 2025; Mansini et al., 2024), and adopting autonomous vehicles such as drones and droids, which offer greater flexibility, scalability, and cost savings (Corberán et al., 2025). Consumer education is also relevant to promote more responsible purchasing

habits and support sustainable delivery practices (Bonomi et al., 2022c; Helo and Thai, 2024). A significant operational challenge in sustainable last-mile delivery (LMD) is the efficient routing of vehicles. The criteria for evaluating this efficiency have evolved over time, particularly in urban settings, where routing problems need to balance various objectives. These involve minimizing travel time and costs, as traditionally done (Laporte, 1992; Bektaş, 2006) and also consider environmental and social impacts, and service quality (Bonomi et al., 2025). A particularly important research direction has focused on variants of the Vehicle Routing Problem (VRP), often referred to as Pollution Routing Problems (PRPs), that go beyond the traditional goal of minimizing travel distance, incorporating more comprehensive and environmentally conscious objectives. In particular, they account for factors such as greenhouse gas emissions, fuel consumption, travel times, and associated costs (Asghari et al., 2021). By integrating these considerations, PRPs aim to balance operational efficiency with sustainability, addressing the growing need for greener and more socially responsible logistics solutions.

The impacts on human health of vehicular pollutants are well documented (Knol et al., 2009; Lipfert and Wyzga, 2008; Di et al., 2017; Southerland et al., 2021). Pollutant concentrations near highways and major roads are higher than in surrounding areas (Lal et al., 2020). For example, carbon monoxide (CO) concentrations drop to background levels within 150 meters from highways, and pollutants such as nitrogen dioxide (NO₂), ozone (O₃), and particulate matter (PM₁₀) show similar decay within 600 meters (Karner et al., 2010). The footprint of transportation extends beyond fuel use and greenhouse gas emissions. Low-income populations in high-traffic neighborhoods often carry a disproportionate share of last-mile delivery externalities, such as traffic, accident risk, and health impacts, despite making fewer online purchases themselves (Fried et al., 2024; Yuan and Wang, 2021; Antonczak et al., 2023). Satellite data confirms that low-income neighborhood residents are often exposed to up to 37% more NO₂ than those in wealthier areas (Demetillo et al., 2020, 2021). These disparities highlight the need for new LMD paradigms that reduce costs and environmental impacts and mitigate social inequities caused by disproportionate exposure to traffic and pollutants.

To address these challenges, we propose the k -Equitable Vehicle Routing Problem (k -EqVRP). This variant of the traditional Vehicle Routing Problem (VRP) minimizes the total delivery time and the broader externalities that affect city inhabitants, especially those in low-income areas. In this model, we incorporate the environmental and social impacts of LMD by steering traffic away from densely populated and vulnerable neighborhoods whenever possible. The VRP is traditionally formulated on a directed graph, where each arc (i, j) represents the shortest path between locations i and j on a street network. However, exclusively optimizing metrics such as distance or travel time can overlook critical factors such as environmental and social impacts, particularly for residents living along access roads or main streets with heavy traffic flows. k -EqVRP balances traditional optimization objectives, measured as cost-efficiency for delivery companies, with the minimization of negative externalities, quantified as the number of residents affected by last-mile deliveries. More precisely, we define the impact on each street as the product of the number of residents living there and the number of times it is traversed by a delivery vehicle (population impact). To measure the impact of a set of routes on local communities, we introduce the concept of k -equity, which is defined as the average impact on the k most affected streets in a delivery solution, where k is a parameter set by the policymaker. While equity in optimization models is often addressed via a min-max formulation (for example, minimizing the maximum route length in classical VRP), this approach tends to disproportionately emphasize outliers, potentially leading to suboptimal trade-offs in practical applications. For this reason, we decided to introduce eq-

uity in our optimization model through a k -sum formulation (see Gupta and Punnen (1990)), which allows for a more flexible and robust approach.

k -EqVRP exploits a multigraph representation, introducing multiple arcs for each pair of nodes. Each arc represents a path (a sequence of streets in the real network). Different paths can share one or more streets. Each path is characterized by two features: delivery travel time and population impact, measured as the number of people living along the streets belonging to the path. The problem minimizes both the total traveling times of the vehicles involved in the delivery service than the impact on the k -most populated streets.

The paper offers several key contributions: (i) We introduce the k -Equitable Vehicle Routing Problem (k -EqVRP), a new last-mile delivery model based on a multigraph formulation that allows multiple paths between nodes. This structure enables flexible and socially-aware routing decisions by balancing total travel time and population exposed to delivery operations. (ii) We introduce a bi-objective mathematical formulation of the problem, using a k -sum equity metric that minimizes the cumulative impact on the k most affected streets, effectively quantifying distributional fairness. This is, to the best of our knowledge, the first paper to incorporate a k -sum-based equity measure into a routing problem formulation. (iii) We propose a lexicographic optimization approach that prioritizes total travel time minimization, followed by the optimization of population impact, ensuring cost-efficient yet equitable solutions. (iv) We conduct a comprehensive computational study on realistic instances built from detailed street networks and population data of two cities (Charleston and San Francisco), providing robust empirical validation and practical relevance. (v) We perform an extensive sensitivity analysis and derive relevant managerial insights to support practitioners and policymakers in designing last-mile delivery strategies that mitigate social and environmental externalities. Our results show that equitable routing can effectively combine operational efficiency with goals of social justice and environmental sustainability, making it suitable for real-world implementation in urban logistics systems. (vi) We propose a Self Adapting Kernel Search (SAKS) metaheuristic to tackle large scale instances. To this end, we first compare four well-known k -sum formulations applied to a VRP context to understand which is the most effective. Extensive computational experiments run on synthetic instances constructed over the Solomon instance set prove that SAKS can handle large scale instances effectively.

6.2 Literature Review

In this section we present a review of the literature related to this work. Considering the multiple aspects dealt with in this work, the following review is structured in three subsections. First, in Section 6.2.1 we present the related works concerning fair and equitable VRPs, an issue often approached with multi-objective formulations. Then, in Section 6.2.2 we introduce the relevant works on the k -sum optimization, detailing its applications and more philosophical considerations as well. Finally, Section 6.2.3 discusses Kernel Search algorithms applied to routing problems, k -sum optimization, and bi-objective problems.

6.2.1 Fair and Equitable Vehicle Routing Problems

In this section, we propose a review of the main literature on single and multi-objective routing problems focused on sustainability and equity, possibly making use of a multigraph.

In the context of the VRP, externalities refer to the social, environmental, and economic consequences of routing decisions. However, the specific notion of social impact as intended

in this work, namely the effect on the urban population, is rarely addressed in the literature. Therefore, in the following, we focus on studies that adopt a broader interpretation of social impact, as well as those that consider other types of externalities.

PRP, first introduced in Bektaş and Laporte (2011), extends the classical capacitated VRP with time windows by including fuel consumption and emissions minimization. Vehicle emissions depend on the slope of the traversed street and vehicle parameters such as load and speed. Speed is a decision variable and can be different for each arc. Demir et al. (2014) propose a bi-objective extension of the PRP. The first objective function minimizes the overall fuel consumption, whereas the second minimizes the total driver working time. Dabia et al. (2017) analyze a variant of the PRP where speed is decided for vehicles instead of arcs, and the start-time of a vehicle is a decision variable. In Raeesi and Zografos (2019), the authors extend the PRP to a graph structure that includes both required nodes, representing customers, and non-required nodes such as street junctions. This variant is referred to as the Steiner PRP. More recently, Gao et al. (2023) propose the Time-dependent PRP taking into account traffic to further mitigate the environmental impact. Finally, in Lai et al. (2024), the authors define a PRP with speed optimization. The proposed model takes into account the terrain topography, including varying road gradients across all arcs.

Multi-objective optimization is a common approach for developing green variants of routing problems, where the optimization of traditional costs (typically total travel cost or time) is combined with a second objective aimed at minimizing greenhouse gas emissions (Govindan et al., 2014; Mirzapour Al-e-Hashem et al., 2019; Xu et al., 2021; Zhou et al., 2022). In this field, most studies consider aspects related to estimating emissions based on distance traveled. The higher the fuel consumed, the higher the GHG emissions. Only a restricted set of works considers fuel consumption as directly dependent on vehicle load, traffic speed, and acceleration frequency as in PRP variants.

Behnke et al. (2021) propose the emission-oriented VRP that optimizes fuel emissions depending on the vehicle load. Their mathematical formulation is based on a multigraph where different emission-minimal arcs for traveling between two locations are introduced. The goal of the problem is to minimize the total emissions. Zhang et al. (2022c) aims at minimizing delivery costs, environmental impact, and workforce satisfaction. Similarly, Golmohammadi et al. (2024) include the maximization of job opportunities when opening distribution centers as the problem's goal alongside the delivery cost minimization and the minimization of the environmental impact.

As far as social impact is concerned, in Lin et al. (2011), the authors propose a three-objective problem in humanitarian logistics dealing with the supply of critical items to people in need. The three objective functions minimize, respectively, the total penalization due to items delivered outside the time window, the total routing time, and the difference in satisfaction between different groups of people.

Equity (fairness) in VRP is a critical aspect of sustainable logistics, especially when considering social externalities. Typically, equity is intended as the equitable distribution of resources among all the actors of a problem. This is particularly common in the healthcare domain. For instance, Decerle et al. (2019) propose a three-objective nurse routing problem, where one of the goals is to minimize the sum of maximal time difference between nurses working times. Bonomi et al. (2024a) propose a multi-objective multi-actor nurse routing problem. Each actor represents a different stakeholder in the problem, namely the operation center, the nurses, and the patients receiving care at home. The authors propose three different objective functions for every actor, each managing a different metric. For nurses and patients, the objective functions are equity-related, such as balancing the nurses' workload

or the overrun of the patient service deadline. Makboul et al. (2024) propose the Green Home Health Care problem, considering the minimization of the total travel costs, the total CO₂ emissions, and the balancing of the nurses' workload. Bonomi et al. (2022d) propose a Nurse Routing Problem where several fairness metrics are solved alongside the total travel time minimization.

In VRPs social impact is usually measured as workload balance between workers (Lee and Ueng, 1999; Matl et al., 2018; Kritikos and Ioannou, 2010; Golden et al., 1997), but in Dukkanci et al. (2022), where the authors define it as dependent on driver's and customer's welfare. Another equity metric is customer satisfaction. In Mansini et al. (2024), the authors propose an Off-Peak Hour VRP where penalties are imposed based on environmental and social impact considerations. Although in general equity is seen as a goal to reach, another approach imposes it as a requirement to be satisfied (Zehtabian, 2024). The interested reader can also refer to the review Xinying Chen and Hooker (2023).

Paper	Impact		Equity
	Environmental	Social	
Golden et al. (1997)		✓	
Lee and Ueng (1999)		✓	
Kritikos and Ioannou (2010)		✓	
Bektaş and Laporte (2011)	✓		
Lin et al. (2011)		✓	
Demir et al. (2014)	✓		✓
Govindan et al. (2014)	✓		
Dabia et al. (2017)	✓		
Matl et al. (2018)		✓	
Decerle et al. (2019)		✓	
Mirzapour Al-e-Hashem et al. (2019)	✓		
Raeesi and Zografos (2019)	✓	✓	
Behnke et al. (2021)	✓		
Xu et al. (2021)	✓		
Dukkanci et al. (2022)		✓	
Zhang et al. (2022c)	✓	✓	✓
Zhou et al. (2022)	✓		
Gao et al. (2023)	✓		
Bonomi et al. (2024a)		✓	✓
Golmohammadi et al. (2024)	✓	✓	✓
Lai et al. (2024)	✓		
Makboul et al. (2024)	✓		✓
Mansini et al. (2024)		✓	
This Work		✓	✓

Table 6.1: Comparison of the literature on Vehicle Routing Problems

In Table 6.1, we summarize the main literature contributions related to fairness-related VRPs. Papers analyzed are listed in the first column. The column labeled *Impact Environmental* indicates whether the routing problem, addressed in the paper, accounts for environmental impact. The *Impact Social* column specifies whether social considerations are integrated into the problem formulation. Finally, the *Equity* column identifies whether the paper explicitly incorporates equity-related aspects. To the best of our knowledge, we are

the first to apply the concept of equity, measured through the k -sum metric, to a vehicle routing problem.

6.2.2 On the k -sum Optimization Problems

The k -sum optimization problem, which consists of minimizing (maximizing) the sum of the k largest (smallest) values in a set S of functions generalizes the classical min(max)-sum case (when $k = |S|$) and the min-max (max-min) case (when $k = 1$). This metric is commonly used as a measure of inequality (Filippi et al., 2019). Frequently, k -sum is used to address the drawbacks of min/max strategies, which only reduce the unfairness for the worst case, without considering what happens to the remainder, and also leading to more expensive solutions when compared to the traditional minimization of the overall costs (Bertazzi et al., 2015). The k -sum optimization problem was first modeled by Gupta and Punnen (1990). This formulation requires an exponential number of constraints, one for each possible sum of k elements from S , making it computationally impractical. The first compact k -sum formulation, based on the dual of a knapsack problem, can be found in Ogryczak and Tamir (2003). Another compact formulation is presented in Blanco et al. (2014). Puerto et al. (2017) discuss several continuous, discrete, and combinatorial k -sum problems dealing with computational complexity issues. Aouad and Segev (2019) propose approximation algorithms for the k -median problem, a specific form of k -sum emerging from location theory. Blanco et al. (2024) propose discrete and continuous location problems with a new envy measure that minimizes the k worst locations without the need to know all the customer preferences in advance. León et al. (2020) generalize the concept of k -sum to multi-objective stochastic problems, showing an application to the multi-objective stochastic knapsack problem. In Ponce et al. (2018), different k -sum formulations are applied to the problem of electing a committee by approval voting (i.e., each voter can select more than one candidate). The winners are selected as those that minimize the k largest Hamming distances between the selected committee and the voters' selection.

6.2.3 Kernel Search Algorithms

Kernel Search matheuristic was first proposed by Angelelli et al. (2010) to tackle multi dimensional knapsack problems. Over the year, several extensions and improvements have been proposed in the literature (Angelelli et al., 2012; Guastaroba and Speranza, 2014; Carrabs et al., 2025). In the following we propose a review of kernel search algorithms designed to tackle routing problems, Bi-Objective formulations, and k -sum formulations.

Archetti et al. (2021a) propose a Kernel Search for the Inventory Routing Problem, where a set of customers have to be replenished over time at minimal transportation and inventory cost. KS uses the initial solution and the variable rankings are identified with the Tabu Search algorithm presented in Archetti et al. (2017). In Bruglieri et al. (2023) the authors introduce a Random Kernel Search matheuristic to solve the electric vehicle routing problem with time windows. Such an algorithm internally uses a Random k -Degree matheuristic to find the initial solution, that is later improved by the Random Kernel Search. Borčinová (2022) proposes a KS to optimize a VRP. The algorithm, instead of using the linear relaxation, uses the saving algorithm proposed by Clarke and Wright (1964) to construct the initial solution. Hanafi et al. (2020) propose a KS algorithm for a Team Orienteering Problem with precedence constraints. Gobbi et al. (2023a) study a Nurse Routing Problem where the priority of the satisfied requests is maximized. Large instances are solved with an hybrid matheuristic algorithm combining an Adaptive Large Neighborhood Search with a Kernel Search. Assunção

and Santos (2025) Robust Steiner Team Orienteering Problem with Decreasing Priorities, a robust variation of the Team Orienteering Problem emerging in disaster relief where location are known but demand cannot be foreseen. The authors propose, among others, a Kernel Search algorithm tailored to solve the problem at hand. Interestingly, such algorithm uses a dynamic time limit to solve the buckets, where to later (and potentially lower quality) buckets is given a shortest amount of time. The time limit decreases following a geometric series, down to its minimal allowed value. Their algorithm iterates the bucket only once. Zhang et al. (2019) analyze a closed-loop supply chain for perishable items, and propose a dedicated kernel search algorithm to tackle the problem. Their algorithm constructs an initial kernel of a random size, selecting the variables with largest value in the linear relaxation. Similarly the number of buckets depends on the size of the kernel set and size of other problem's set. Later, Zhang et al. (2022d) extended the previous proposing a bi-objective formulation, aiming at the profit maximization and the carbon emission minimization, for a closed-loop supply chain with perishable items. The authors propose a kernel search that, using the ϵ -constraint method (Ehrgott, 2005; Mansini et al., 2024; Gobbi et al., 2025; Manerba et al., 2025), finds an approximation of the Pareto frontier. Li et al. (2021) study a capacitated multimodal transportation planning problem, aiming at satisfying all customers' requests at the minimal cost. A kernel search and dynamic programming algorithm is proposed. This algorithm, at each iteration uses a dynamic program to select the most promising variables to be included in the kernel set, then the new restricted problem is optimized using a MILP solver. To enhance the formulation, at each KS iteration valid inequalities are included in the formulation. Guastaroba et al. (2017) propose an enhancement of the Kernel Search framework called Adaptive Kernel Search, aimed at solving general hard MILP problems. This algorithm using the root node relaxation instead of the linear relaxation to construct the kernel and the buckets. Moreover, the authors add functionality to handle general integer variables, functionalities to find an initial feasible solution, and a classification mechanism to identify harder instances. Carvalho et al. (2024) take into account a multi-item production and distribution system where the overall production, inventory, and transportation costs have to be minimized. To address this problem, the authors propose a Hybrid Kernel Search, extended from the one presented in Carvalho and Nascimento (2018), that uses a dedicated MIP based algorithm to find the initial solution to build the kernel set. Filippi et al. (2016) study a Bi-Objective index tracking problem, where the average excess return is maximized and the total tracking error in the re-balanced portfolio is minimized. The authors propose a Kernel Search, extended from the one proposed in Guastaroba and Speranza (2012), to tackle the problem at hand. The algorithm identifies heuristic solution for the two objective function independently. These values are used to find the range for the value ϵ . Finally, the algorithm run a mono-objective kernel search with an ϵ -constraint for a given number of steps, iteratively increasing the value of ϵ , to find an approximation of the Pareto frontier. Filippi et al. (2021) propose a Kernel Search algorithm fair facility location problem, a variation of a facility location where the weighted sum average distance traveled by all customers, and the average distance traveled by the $\beta\%$ most distance customers is minimized. This algorithm is designed to find, for a given linear combination of the two objective functions, the value of the objective function under different β values. Each step is optimized by an adaptive kernel search algorithm. Huerta-Muñoz et al. (2022) study a heterogeneous flexible periodic vehicle routing problem. The authors propose three formulations, with different constraints and variables, and compare their performance when used within a Kernel Search algorithm.

Table 6.2 provides a comparison of the various contributions in the literature. The first column identifies the source of the study. The second one specifies if the problem at hand is

a routing problem in a quite general sense, whereas the third column identifies algorithms that are explicitly designed to deal with bi-objective problems. Finally, the fourth column tells if the algorithm handles a k -sum formulation of sum sort. As it is possible to observe

Paper	Routing	Bi-Obj.	k -Sum
Archetti et al. (2021a)	✓		
Bruglieri et al. (2023)	✓		
Borčinová (2022)	✓		
Gobbi et al. (2023a)	✓		
Assunção and Santos (2025)	✓		
Zhang et al. (2019)	✓		
Hanafi et al. (2020)	✓		
Zhang et al. (2022d)	✓	✓	
Li et al. (2021)	✓		
Guastaroba et al. (2017) ¹	✓		✓
Carvalho et al. (2024)	✓		
Filippi et al. (2016)		✓	
Filippi et al. (2021)			✓
Huerta-Muñoz et al. (2022)	✓		
This Work	✓	✓	✓

Table 6.2: Comparison of Related Kernel Search Literature

from this table, to the best of our knowledge this is the first time a Kernel Search algorithm is used to tackle a Bi-Objective VRP with a k -sum formulation.

6.3 Problem Definition

The k -EqVRP is formulated on a directed multigraph $G = (V, A)$, where $V = \{0\} \cup N$ is the set of nodes (N represents the set of customers and the node 0 is the depot) and A is the arc set. Set A is partitioned into subsets $A_{ij}, i, j \in V, i \neq j$, each corresponding to the set of multiple arcs connecting each pair of nodes i and j , $A = \bigcup A_{ij}, i, j \in V, i \neq j$. We indicate as (i, j, p) the p -th alternative arc, $p = 1, \dots, |A_{ij}|$, that connects nodes i and j . For each arc (i, j, p) , we define t_{ij}^p as the corresponding traversal time.

Each arc (i, j, p) corresponds to a path that traverses zero, one or more residential streets. Let R be the set of residential streets of a real network, each one included in at least one path associated with arcs in A . Each street $r \in R$ is assigned a value w_r , indicating the number of people living there, and a subset $A_r \subseteq A$ representing the arcs in A that share the traversal of street r in the corresponding path. For each customer $i \in N$, the service time is denoted by s_i . The service is performed by a set F of homogeneous vehicles (delivery vans), each leaving the depot 0 at time 0. We assume that vehicles cannot wait at the depot or any other nodes of the graph (visited for a delivery). Each customer has to be visited exactly once. Furthermore, the total route duration of a vehicle, comprising travel and service times, cannot exceed a given threshold t_{max} .

The k -EqVRP is formulated as a bi-objective problem that looks for a set of routes, starting and ending at depot 0 and complying with all the described constraints, while minimizing

¹Works on general MILP problems, and so also on routing problems and k -sum optimization.

the sum of the travel time of all vehicles and the k -equity defined as the average impact on the k most affected streets in a delivery solution.

6.4 Minimization of the k Largest Variables in a Linear Program

In this section, we provide a quick introduction to the k -sum problems, namely the minimization, or maximization, of the k largest, smallest, variables in a linear program. This class of problems, often referred to as k -sum optimization problems (KSOP), arises in several application domains (including location analysis, risk minimization, and equity-oriented optimization) whenever decision makers are interested in controlling the impact on the k most critical or most affected elements rather than on the entire set. These formulations are instrumental to get a compact MILP formulation for k -EqVRP in Section 6.5.

Without loss of generality, in this section we focus on minimization problems.

First, let us define some notation used throughout this section. Let $\mathcal{X} := \{x \in \mathbb{R}^n | Ax \geq b\}$ be the feasible region of a generic linear program, where $A \in \mathbb{R}^{n \times m}$ is the constraint matrix, and $b \in \mathbb{R}^m$ is the vector of the known terms. We identify a feasible solution $x \in \mathcal{X}$, where $x_i, i = 1, \dots, n$ is the i -th term of x .

Given the aforementioned notation, it is straightforward to define a linear program that minimizes the sum of the variables each multiplied by an appropriate cost coefficient: $z^* = \min_{x \in \mathcal{X}} c^T x$, where $c \in \mathbb{R}^n$. Similarly, it is simple to define a linear model to minimize the largest component x_i multiplied by an appropriate cost coefficient:

$$z^* = \min\{t : t \geq c_i x_i \quad \forall i \in 1, \dots, n; x \in \mathcal{X}\} \quad (6.1)$$

However, it is not so simple to define a linear model that minimizes the values of the k largest components x_i multiplied by their cost coefficient. Indeed, the first linear formulation for such a problem is due to Gupta and Punnen (1990) and is as follows:

$$z^* = \min \left\{ t : t \geq \sum_{i \in S} c_i x_i \quad \forall S \subset 1, \dots, n; |S| = k; x \in \mathcal{X} \right\}. \quad (6.2)$$

Such formulation add a new constraint for each possible way to sum k items out of n . Being a non-compact formulation, requiring exactly $\binom{n}{k}$ additional constraints, it applicable only if these constraints are separated dynamically or if k is either small or very close to n .

The first compact formulation is due to Ogryczak and Tamir (2003). In the following we propose a derivation of such formulation based on duality theory. Let us define the knapsack program

$$\max \left\{ d^T y : \sum_{i=1}^{n'} y_i = k; y \in \{0, 1\}^{n'} \right\} \quad (6.3)$$

The vector $d \in \mathbb{R}_+^{n'}$ represent the profits, i.e. d_i is the profit of the i -th item. The vector y represents binary variables whose i -th term y_i is set to one if and only if the i -th item is selected in the optimal solution. This problem is subject to the only constraint that the number of selected numbers has to be equal to k . Naturally, the optimal solution of (6.3) selects the k largest values in d , and the objective values is their sum.

From here, is possible to observe that the constraint matrix is totally unimodular, so it is possible to relax the integrality constraints on y . Moreover, if we assume that $d_i > 0 \quad i = 1, \dots, n'$, it is possible to relax the equality constraint to a less or equal one.

$$\max \left\{ d^T y : \sum_{i=1}^{n'} y_i \leq k; 0 \leq y_i \leq 1 \quad i = 1, \dots, n' \right\} \quad (6.4)$$

Since program in (6.4) is feasible and bounded, it is possible to calculate its dual.

$$\min \left\{ kt + \sum_{i=1}^{n'} r_i : t + r_i \geq d_i, d_i \in \mathbb{R}_+, t \in \mathbb{R}_+ \right\} \quad (6.5)$$

Notice that, in (6.5), the profit coefficient d_i is alone on the right-hand side of a constraint, therefore it is possible to substitute it with any arbitrary complex linear expression. So the following program minimizes the k largest entries in $x \in \mathcal{X}$:

$$\min \left\{ kt + \sum_{i=1}^{n'} r_i : t + r_i \geq x_i, d_i \in \mathbb{R}_+, t \in \mathbb{R}_+, x \in \mathcal{X} \right\} \quad (6.6)$$

A similar formulation is proposed by Filippi et al. (2019), where the authors instead of minimizing the sum over the k largest values, minimize their average. This difference results in the following formulation:

$$\min \left\{ kt + \sum_{i=1}^{n'} r_i : kt + kr_i \geq d_i, d_i \in \mathbb{R}_+, t \in \mathbb{R}_+ \right\} \quad (6.7)$$

The full derivation of such a program is not presented here since it is equivalent to the one already proposed for the formulation in (6.5).

Another possible formulation is due to Blanco et al. (2014). Such formulation is based on the dual of an assignment problem.

$$\begin{aligned} \max \left\{ \sum_{i=1}^{n'} \sum_{j=1}^k d_i p_{ij} : \right. \\ \sum_{i=1}^{n'} p_{ij} = 1, \quad j \in 1, \dots, k; \\ \sum_{j=1}^k p_{ij} = 1, \quad i \in 1, \dots, n'; \\ \left. p_{ij} \in \{0, 1\} \quad i \in 1, \dots, n', j \in 1, \dots, k \right\} \quad (6.8) \end{aligned}$$

As above $d \in \mathbb{R}_+$ represents the profit of the items. The model works by selecting a permutation of the elements of d . The two constraints set exactly k p_{ij} variables to one. The first constraint imposes that for each position only one item is selected, whereas the second imposes that for i position only one item is selected. The objective function maximizes the sum over the k selected entries from d , thus selecting the k largest entries.

It is possible to observe two things about the program in (6.8):

- under the assumption that $q_r > 0$, it is possible to relax the equality constraints to less or equal;
- the constraints matrix is totally unimodular, so it is possible to relax the integrality constraint on p_{ik} .

This lead to the following equivalent formulation:

$$\begin{aligned} \max \left\{ \sum_{i=1}^{n'} \sum_{j=1}^k d_i p_{ij} : \right. \\ \sum_{i=1}^{n'} p_{ij} \leq 1, \quad j \in 1, \dots, k; \\ \sum_{j=1}^k p_{ij} \leq 1, \quad i \in 1, \dots, n'; \\ \left. 0 \leq p_{ij} \leq 1 \quad i \in 1, \dots, n', j \in 1, \dots, k \right\} \end{aligned} \quad (6.9)$$

This program is feasible and bounded, so it is possible to compute its valid dual formulation:

$$\min \left\{ \sum_{j=1}^k u_j + \sum_{i=1}^{n'} v_i : u_j + v_i \geq d_i, i \in 1, \dots, n', j \in 1, \dots, k; u_j \geq 0, v_i \geq 0 \right\} \quad (6.10)$$

In (6.10), it is possible to observe that the cost coefficient q_r is now on the right-hand side of a constraint, allowing to place arbitrarily complex linear equations in its place.

So to minimize the k largest components of $x \in \mathcal{X}$ using Blanco et al. (2014) formulation, the final program is:

$$\begin{aligned} \min \left\{ \sum_{j=1}^k u_j + \sum_{i=1}^{n'} v_i : \right. \\ u_j + v_i \geq x_i \quad i \in 1, \dots, n', j \in 1, \dots, k; \\ \left. u_j \geq 0, v_i \geq 0, x \in \mathcal{X} \right\} \end{aligned} \quad (6.11)$$

6.5 Mathematical Formulation

In this section, we first present the mathematical formulation of the problem's routing constraints, followed by a description of the mathematical modeling of the objective functions and of the additional constraints they require.

To model the constraints of the k -EqVRP, we introduce two sets of variables:

- Binary variable $x_{ij}^p, (i, j, p) \in A_{ij}, i, j \in V, i \neq j$: takes value 1 if the p -th arc (path) between i and j is traversed, and 0 otherwise;
- Continuous variable $z_{ij}, i, j \in V, i \neq j$: indicates the arrival time in node j when coming from node i , and 0 otherwise.

For each subset $S \subset V$ of nodes, let $\delta^+(S) = \{(i, j, p) \in A : i \in S, j \notin S, p = 1, \dots, |A_{ij}|\}$ and $\delta^-(S) = \{(i, j, p) \in A : i \notin S, j \in S, p = 1, \dots, |A_{ij}|\}$ be the sets of arcs leaving and entering set S , respectively. To simplify notation, the special case with $|S| = 1$ is indicated as $\delta^+(i)$ ($\delta^-(i)$) instead of $\delta^+(\{i\})$ ($\delta^-(\{i\})$).

The constraints for k -EqVRP are formulated as follows:

$$\min f_1 = \sum_{(i,j,p) \in A} t_{ij}^p x_{ij}^p \quad (6.12)$$

$$\min f_2 = ky + \sum_{r \in R} d_r \quad (6.13)$$

$$y + d_r \geq \sum_{i,j,p \in A_r} w_r x_{ij}^p \quad r \in R \quad (6.14)$$

$$\sum_{(0,i,p) \in \delta^+(0)} x_{0i}^p = \sum_{(j,0,p) \in \delta^-(0)} x_{j0}^p \leq |F| \quad (6.15)$$

$$\sum_{(j,i,p) \in \delta^-(i)} x_{ji}^p = \sum_{(i,j,p) \in \delta^+(i)} x_{ij}^p = 1 \quad i \in N \quad (6.16)$$

$$\sum_{j \in V, j \neq i} z_{ij} - \sum_{j \in V, j \neq i} z_{ji} = \sum_{(i,j,p) \in \delta^+(i)} (t_{ij}^p + s_i) x_{ij}^p \quad i \in N \quad (6.17)$$

$$z_{ij} \leq \sum_{(i,j,p) \in A_{ij}} (t_{max} - s_j - \min_{(j,0,u) \in A_{j0}} t_{j0}^u) x_{ij}^p \quad i, j \in V, i \neq j \quad (6.18)$$

$$z_{ij} \geq \sum_{(i,j,p) \in A_{ij}} (s_i + t_{ij}^p + \min_{(0,i,u) \in A_{0i}} t_{0i}^u) x_{ij}^p \quad i, j \in V, i \neq j \quad (6.19)$$

$$\sum_{(i,j,p) \in A_{ij}} x_{ij}^p \leq 1 \quad i, j \in V, i \neq j \quad (6.20)$$

$$z_{0i} = \sum_{(0,i,p) \in A_{0i}} t_{0i}^p x_{0i}^p \quad i \in N \quad (6.21)$$

$$y \geq 0 \quad (6.22)$$

$$d_r \geq 0 \quad r \in R \quad (6.23)$$

$$x_{ij}^p \in \{0, 1\} \quad (i, j, p) \in A \quad (6.24)$$

$$z_{ij} \geq 0 \quad i, j \in V, i \neq j \quad (6.25)$$

Objective function (6.12) minimizes the total travel time, whereas Objective function (6.13) minimizes the k largest impacts. Constraints (6.14) defines the impact as the population on street $r \in R$ multiplied by the number of times it is traversed by a vehicle. Objective function (6.13) and constraints (6.14) are defined according to formulation (6.5) from Ogryczak and Tamir (2003). Constraints (6.15) ensure that the number of vehicles leaving the depot is equal to the number of vehicles entering the depot and that this number is lower than or equal to the fleet size $|F|$. Constraints (6.16) ensure that each customer i in N is visited exactly once by guaranteeing the selection of exactly one incoming arc and one outgoing arc for each node. Constraints (6.17) impose that if a vehicle visits node j immediately after node i in a path p , then the time elapsed between the arrival times in the two nodes is equal to the service time s_i plus the travel time t_{ij}^p associated with the arc selected to travel from i to j (Gobbi et al., 2019b, 2023a). Constraints (6.18) and (6.19) impose upper and lower bounds on the arrival time at node i when coming from node j . In particular, natural upper bound t_{max} is strengthened by considering the time required to serve node j and return to the depot

(the minimum time in this case, considering the multiple available arcs). The lower bound is strengthened by imposing that the arrival time in j is at least equal to the time to serve node i plus the minimum time to go from the depot to node i and the minimum time to go from i to j . Notice that, together, constraints (6.17)–(6.19) also impose subtour elimination. Constraints (6.20) set that at most one arc for each pair of nodes in V is selected. Constraints (6.21) impose that the arrival time in node i when coming from the depot is exactly equal to the travel time t_{0i}^p associated with the p -th arc selected to travel from 0 to i . Finally, conditions (6.24)–(6.25) introduce integrality and non-negativity restrictions on the variables.

6.6 Solution Methodology

We solve the bi-objective k -EqVRP lexicographically by using the ϵ -constraint method (Section 2.4.2). In this application, we consider both optimization directions. We first consider the k -sum (function f_2) as the primary function to optimize and comply with the transportation cost function (f_1) in the constraint. To this aim, we first identify the optimal value f_1^* of the function f_1 (routing time) without considering any constraint on f_2 . We identify this optimization as $f_2|(f_1^*, \epsilon)$. For the sake of simplicity, we refer to the optimization problem that has f_1^* as the optimal solution with $f_1|(f_2^*, \infty)$.

Notice that there is a trade-off between the two objective functions, even though both are minimized: optimizing the k -sum steers the routing solution away from the most populated streets, potentially leading to higher transportation costs. Let f_2^* be the optimal solution value when f_2 is optimized as a single objective and no constraint on the other function is imposed. Similarly, we perform reverse optimization by considering f_1 as the primary function and f_2 in the constraints. We identify this optimization as $f_1|(f_2^*, \epsilon)$. As above, we refer to the optimization problem that has f_2^* as the optimal solution with $f_2|(f_1^*, \infty)$.

To enforce the current formulation, we also introduce General Connectivity Constraints (GCC) and Subtour Elimination Constraints (SEC):

$$\sum_{(i,j,p) \in \delta^+(S)} x_{ij}^p \geq 1 \quad S \subset N, |S| \geq 2. \quad (6.26)$$

$$\sum_{(i,j,p) \in A(S)} x_{ij}^p \leq |S| - 1 \quad S \subset N, |S| \geq 2. \quad (6.27)$$

where $A(S) = \{(i, j, p) : i, j \in S, p = 1, \dots, |A_{ij}|\}$ with S a subset of nodes strictly included in N . In the first case, GCCs are dynamically separated during the search by solving a Max Flow Problem through Boykov and Kolmogorov's algorithm (Boykov and Kolmogorov, 2004), whereas SECs are introduced from scratch by only adding the violated constraints with $|S| = 2$.

6.7 Real-World Instances

To create realistic scenarios, we base our instances on the real networks of two cities: San Francisco (CA, USA) and Charleston (SC, USA). The two cities have been selected to represent contrasting urban planning contexts: a large-size, densely populated city with 873,965 inhabitants and a density of 7,192.73 people per square kilometer (U.S. Census Bureau, b) characterized by a predominantly standard grid road network (San Francisco), and a medium-



Figure 6.1: Map of San Francisco

sized city with 150,227 inhabitants and a density of 505.45 per square kilometer (U.S. Census Bureau, a) characterized by a more unstructured road network (Charleston). San Francisco's road network offers many alternative paths with similar travel time between pairs of nodes, whereas Charleston's network often shows substantial routing time increases for alternative paths. As a result, the two cities are expected to yield different insights when solving the k -EqVRP. The road network is source and processed as described in Section 5.4. Population impact $w_r, r \in R$ is computed from population data derived from the 2010 US Census and processed using Cenpy (Cenpy, 2025). The census provides population counts for small city areas called *tracts*. Each tract contains between 1200 and 8000 people, with an average of 4000. Population is allocated only to streets labeled as *residential* in openstreetmap based on their length within the area associated with a tract, with random noise added to introduce variability. The allocation is then adjusted to ensure that the sum of the population of the streets matches the tract's population.

Besides the base road network, the instances are generated considering the parameters reported in Table 6.3. Parameter $\bar{t}$ indicates the average traveling time between all pairs of nodes in V . Parameter $\bar{p}$ defines the number of alternative paths for each pair of nodes. Alternative paths between each pair of locations are generated by first computing the shortest-time path on the road network, whose length serves as a reference point in the actual path generation. The paths are obtained by solving shortest-path problems where street costs are set to population weights w_r , subject to a time constraint that allows a controlled deterioration (up to γ) compared to the optimal travel-time path. For each successive value of γ ,



Figure 6.2: Map of Charleston

Parameter	Values
$\bar{t}$	$\{150s, 300s, 450s, 600s\}$
$\bar{p}$	$\{1, 3, 5\}$
t_{max}	$\{3h, 4h, 5h\}$
s_i	900s
$ N $	10
$ F $	3

Table 6.3: Instance generation parameters.

the algorithm ensures that the new path is different from each previous ones. This extra constrain is required since it is possible that the shortest path is already does not traverse residential streets or that to reduce the cost further requires an increase in the travel time that is beyond the allowed worsening γ . Notice such choice is coherent with the proposed optimization problem. Indeed, a different path traverses different residential streets, and as such gives room to the solver to mitigate the disturbance in the k worst cases. All paths are then reoptimized to minimize the travel time, ensuring that the population impact cannot increase. In our case, we use $\gamma = \{0.00, 0.25, 0.50, 0.75, 1.00\}$ to generate up to 5 alternative paths per pair. For each parameter combination, as shown in Table 6.3, we generate 25 different node selections, giving rise to a total number of instances equal to 1800, 900 for each city.

Regarding the lexicographic approach adopted and the measure of equity through the k -sum, we considered two different parameters in our experiments: $k = \{1, 5, 10, 15, 20\}$ and $\epsilon = \{0.00, 0.05, 0.10, 0.25, 0.50, 1.00\}$. Hence, the total number of solutions considered in this work is 54000.

6.8 Sensitivity Analysis

This section presents the computational results obtained from solving the proposed MILP formulations under different scenarios. We also highlight managerial insights to assist decision-makers in the last-mile delivery field. In Section 6.8.1 we provide a detailed analysis of the computational experiments. Section 6.8.2 summarizes the previous experimental analysis by providing managerial insights for practitioners and policymakers. Finally, in Section 6.8.3, we discuss the statistical relevance of the obtained results.

6.8.1 Sensitivity analysis on main parameters

In this section, we evaluate the impact of parameters k , ϵ , t_{max} , and $\bar{t}$ on the objective functions f_1 and f_2 .

Sensitivity analysis for parameter k

To analyze the impact of parameter k on objective function f_1 , we first optimize $f_2|(f_1^*, 0)$.

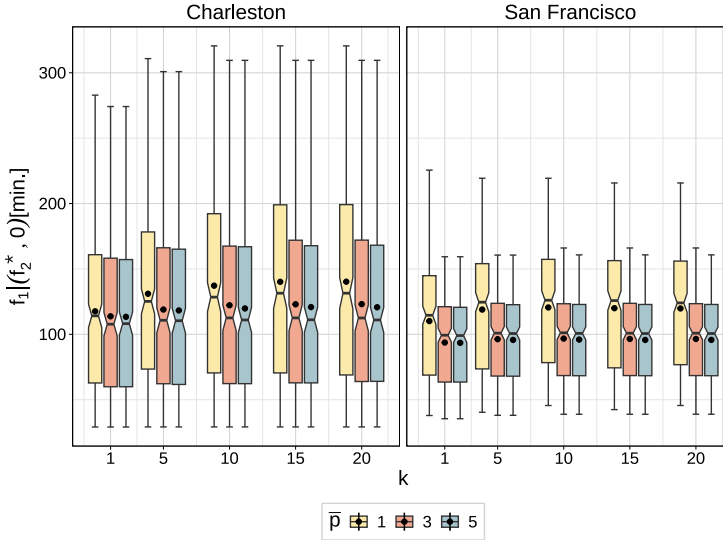


Figure 6.3: Distribution of optimal objective function value of $f_1|(f_2^*, 0)$, when changing the value of k , considering Charleston and San Francisco.

Figure 6.3 shows how the distribution values of f_1 change when modifying k , when considering both cities. The most noticeable trend is the general increase in the routing time as the parameter k increases. This behavior is expected: larger values of k determine greater changes to the original routing plan to reduce the overall population impact. Moreover, having more paths for each pair of nodes (higher $\bar{p}$ value) allows solutions with shorter routing times. The largest decrease in routing times occurs when $\bar{p}$ increases from 1 to 5, with an average decrease of about 12% for Charleston and about 24% for San Francisco. For larger values of k , routing times keep increasing up to $k = 15$ for Charleston. This is consistent with the network structure for this city, in which the alternative paths have very different costs.

On the contrary, in San Francisco, there is no significant routing increase when moving from $k = 5$ to $k = 10$, due to the similarity (in terms of cost) between the alternative paths. Interestingly enough, after a peak in $k = 10$ the value of f_1 slightly decreases in a counter intuitive way. We explain this situation by providing an example.

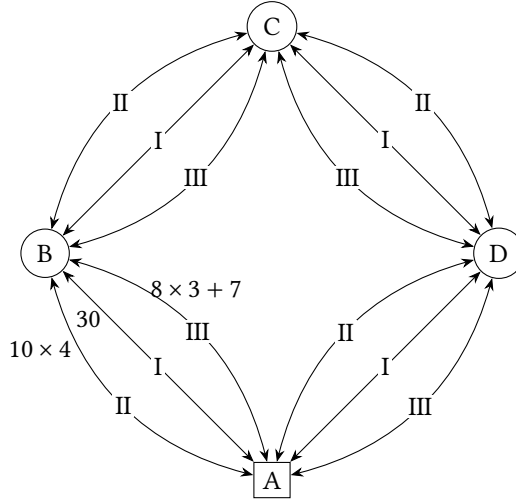


Figure 6.4: Example of a Multigraph

Figure 6.4 shows a small multigraph with four nodes, where multiple arcs (representing paths on the road network) can exist between the same pair of nodes. Node A, depicted as a square, represents the depot. Double-headed arrows indicate that both directions, from node i to node j and from j to i , are included. The arcs labeled I, II, and III have lengths of 1, 2, and 3, respectively. For simplicity, we assume that only the arcs between nodes A and B are associated with the population data. Specifically, arc I corresponds to a single street with a population of 30 people; arc II traverses 10 streets, each with 4 residents; and arc III includes 8 streets with 3 residents each, plus one street with a population of 7.

When optimizing $f_1|(f_2^*, \infty)$, it is straightforward to see that there are two equivalent optimal solutions: the cycle A-B-C-D-A and the cycle A-D-C-B-A. Both routes exclusively use arcs labeled I and have a total cost of 4. These two solutions are also Pareto efficient with respect to $f_2|(f_1^*, 0)$, for any value of k .

When optimizing $f_2|(f_1^*, \infty)$, objective function value depends on the chosen value of k . Table 6.4 shows the value of f_2 when traversing the three alternative arcs connecting A and B, for $k = 1, 5, 10$, and 15. The optimal objective values for $f_1|(f_2^*, 0)$ are shown in Table 6.5. All the values represented in the table are Pareto-efficient.

k	Arc		
	I	II	III
1	30	4	7
5	30	20	19
10	30	40	31
15	30	40	31

Table 6.4: Value of f_2 when traversing each option between A and B under the change of k

k	1	5	10	15
$f_1 (f_2^*, 0)$	5	6	4	4
f_2^*	4	19	30	30
Arc	II	III	I	I

Table 6.5: Optimal solutions for the optimization $f_1|(f_2^*, 0)$ for different values of k

When $k = 1$, only the most impacted street counts, so the solution selects the arc with the street with the least population, that is II. The other alternatives would have a cost of 30 and 7, for arcs I and III, respectively. With $k = 5$, the most convenient option becomes arc III, with $f_2^* = 19$ and the objective function value of $f_1|(f_2^*, 0)$ equal to 6. Finally, with $k = 10$ and 15, the most convenient option becomes arc I, with $f_2^* = 30$ in both cases. This example shows that, counter intuitively, the optimal objective function value of $f_1|(f_2^*, 0)$ may not grow monotonically with the increase of k .

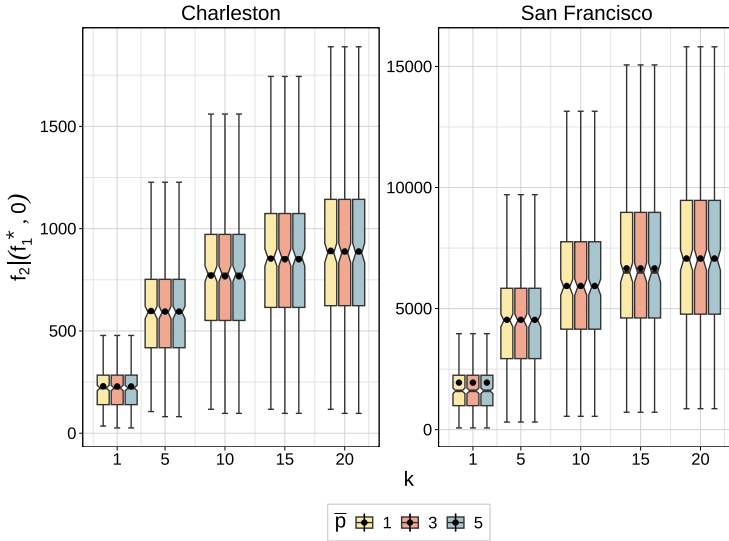
Figure 6.5: Distribution of the optimal objective function value of $f_2|(f_1^*, 0)$ when changing the value of k , considering Charleston and San Francisco.

Figure 6.5 illustrates the impact of the parameter k on $f_2|(f_1^*, 0)$ values for both cities. Two observations can be drawn from the figure. First, the optimal values of $f_2|(f_1^*, 0)$ consistently decrease as the value of k increases. This result aligns with our earlier analysis of Figure 6.3. When $\epsilon = 0$, there is no flexibility to improve f_2 values, and having alternative paths does not help. In this case, the impact of k is significant when increasing from 1 to 5 (with an average decrease of 38.2% in Charleston, and 29.9% in San Francisco), but diminishes for similar increments at higher values of k (e.g., from $k = 10$ to $k = 15$).

Sensitivity analysis on parameter ϵ

In this section, we analyze the impact of different values of the parameter ϵ when solving the optimization problem $f_2|(f_1^*, \epsilon)$.

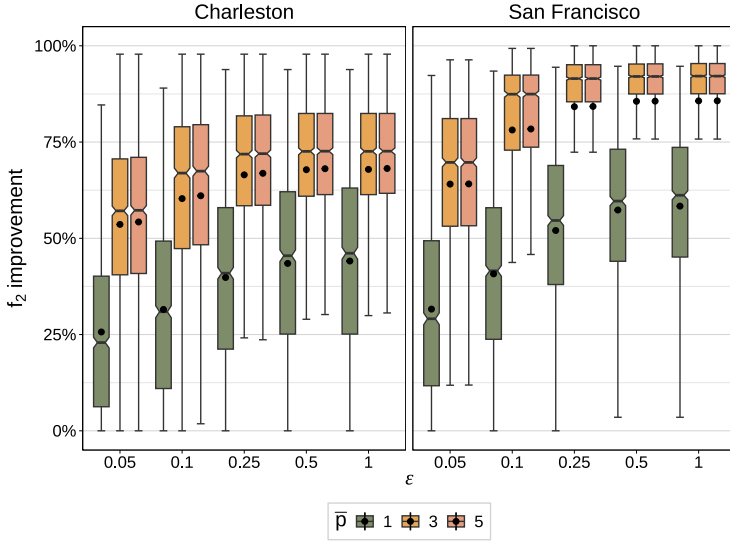


Figure 6.6: Distribution of equity value improvement, in the two cities, when comparing $f_2|(f_1^*, \epsilon)$ with $f_2|(f_1^*, 0)$

Figure 6.6 shows the average equity percentage improvement for different values of $\epsilon > 0$ with respect to the base case with $\epsilon = 0$. The data points for both cities are grouped by the values of the parameter $\bar{p}$.

It is evident that there are common patterns between the two cities. In both cities, independently of the number of parallel arcs, it is possible to observe an asymptotic improvement of the equity values. There is a sharp improvement between $\epsilon = 0$ to $\epsilon = 0.25$, then the values start to settle. In Charleston, averaging the three classes of instances corresponding to the three values of $\bar{p}$, there is an average decrease of 44.5 in the value of the equity when moving from $\epsilon = 0$ to $\epsilon = 0.05$. In San Francisco, this improvement has value 53.3. It should be noted that in both cities, the average improvement in equity value more than doubles when moving from instances with $\bar{p} = 1$ to instances with $\bar{p} > 1$. This fact shows the positive effect of considering alternative paths (multigraph) to mitigate the population impact. For both cities, there is a minimal difference when increasing ϵ from 0.5 to 1.0. In fact, such an average improvement is only 0.58% and 2.4% for Charleston and San Francisco, respectively. This suggests that a value of ϵ equal to 0.5 is almost enough to reach the optimal value of $f_2|(f_1^*, \infty)$, and no further improvements can be obtained by increasing it.

Similarly to what has been observed in Figure 6.5, we can notice that there is almost no difference (average percentage improvement below 0.75) between instances with $\bar{p} = 3$ and $\bar{p} = 5$.

It is easy to observe that, for both cities, the difference between instances with $\bar{p} = 1$ and $\bar{p} = 3$ is quite remarkable. In Charleston, the average gap stands at 26.3 percentage points, with a minimal gap at 23.8 when $\epsilon = 1$ and a maximum of 28.8 percentage points when $\epsilon = 0.1$. In San Francisco, the same average gap is larger, reaching 31.5 percentage points, with a minimum value of 27.3 percentage points for $\epsilon = 1.0$ and a maximum of 37.40 percentage points for $\epsilon = 0.05$. What is even more remarkable, showing the effectiveness of the introduction of alternative paths, is that, in both cities, the improvement observed for

instances with $\bar{p} = 3$ and $\bar{p} = 5$ and $\epsilon = 0.05$ is better than the one observed for $\bar{p} = 1$ and $\epsilon = 1$. This outcome comes as no surprise. In fact, with one path between any pair of nodes in the graph, the only available way to reduce the equity is by changing the order in which customers are visited. This order cannot be altered too much without violating the ϵ -constraint.

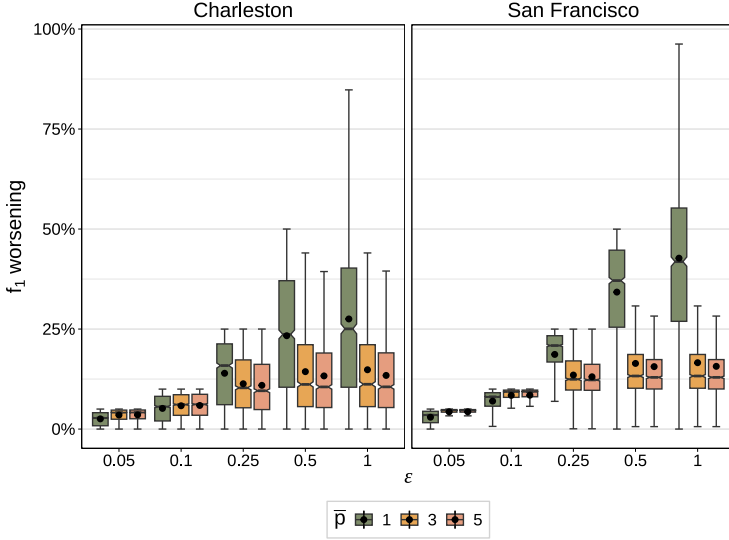


Figure 6.7: Distribution of traveling time worsening, in the two cities, when comparing the value of f_1 computed on the optimal solution of $f_2|(f_1^*, \epsilon)$ with $f_1|(f_2^*, \infty)$

Figure 6.7 shows the distribution of the traveling time worsening when comparing the value of f_1 computed on the optimal solution of $f_2|(f_1^*, \epsilon)$ with $f_1|(f_2^*, \infty)$. Results are shown grouped for $\bar{p}$ values by separating the two cities. These graphs show the actual cost of the equity, since, even if a certain ϵ worsening of f_1^* is allowed, the computed optimal solutions may not take full advantage of this flexibility, but have values of f_1 that are considerably better than $(1 + \epsilon)f_1^*$. This is especially true for larger values of ϵ , where in instances with $\bar{p} = 3$ and $\bar{p} = 5$, the worsening is consistently lower than half of the maximum allowed. This trend is also present for smaller ϵ , even if it is less pronounced.

In both cities, a similar pattern emerges. For small worsening values, below 10%, the behavior of the instances with $\bar{p} = 1, 3, 5$ is almost the same. For larger values, the worsening for instances with $\bar{p} = 1$ starts to diverge from those with $\bar{p} = 3, 5$. Let us consider the gap between $\bar{p} = 1$ and $\bar{p} = 3$. In Charleston, the gap settles at 0.95 percentage points for $\epsilon = 0.05$ (with minimum obtained for $\bar{p} = 1$), and reaches 12.7 points for $\epsilon = 1.0$ (minimum got for $\bar{p} = 3$). Similarly, in San Francisco the first difference is 1.35 percentage points, in favor of $\bar{p} = 1$, for $\epsilon = 0.05$, and with $\epsilon = 1.0$ the figure grows to 26.1 points, in favor of $\bar{p} = 5$. In both cities, the absolute value of this gap grows monotonically. Also interesting, after 0.25, the percentage worsening for $\bar{p} = 3$ and $\bar{p} = 5$ remains almost constant. Naturally, this is related to the behavior emerged in Figure 6.6, where the equity value does not show relevant improvements with those ϵ values. As expected from the other analysis, the three and five arcs instances in general worsen the f_1 objective value less than the other configuration. In both cities, for $\bar{p} = 1$, it is possible to observe a relevant worsening of the routing time to

achieve an almost indistinguishable improvement of the equity value.

An interesting fact to notice, by looking at Figure 6.7, is the difference between the average routing time worsening with $\epsilon = 0.05$ in $\bar{p} = 3$ instances and the one with $\epsilon = 1.0$ in the $\bar{p} = 1$ instances. In fact, in Charleston the worsening is more than 7 times larger, whereas in San Francisco, this figure is more than 9 times.

It is interesting to note that in Charleston, the actual average worsening for large ϵ values is smaller than in San Francisco. This difference is likely connected to the different population density and street topology in the two cities. Given this, it is reasonable to expect that achieving better equity in San Francisco requires a greater increase in routing time.

For illustrative purposes, in Figures 6.8-6.11, we showcase the comparison of two solutions obtained by setting $\epsilon = 0$ and 1, respectively, in Charleston and San Francisco. Both

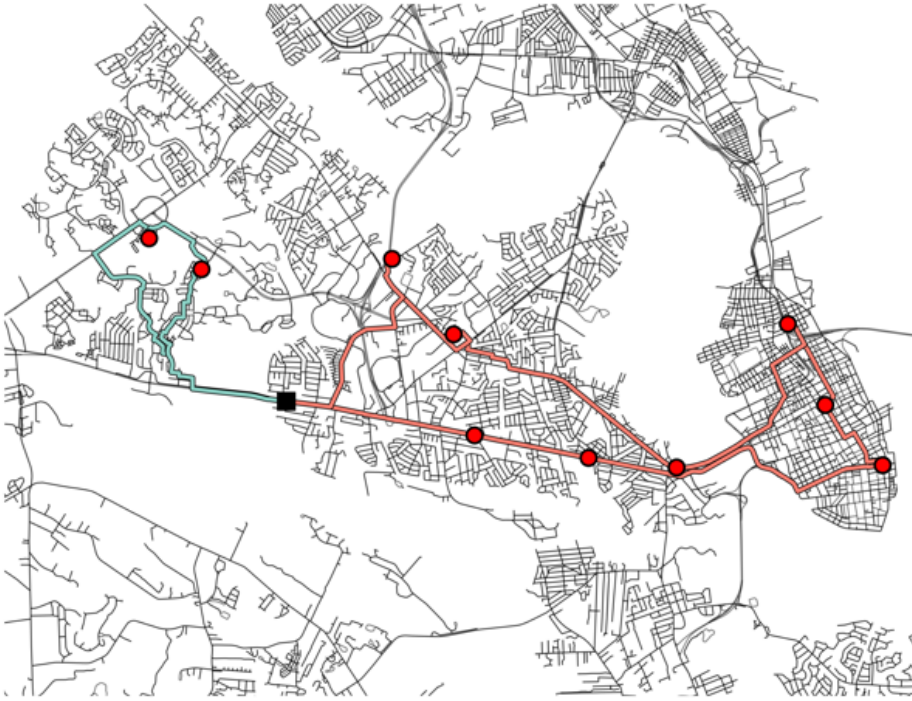
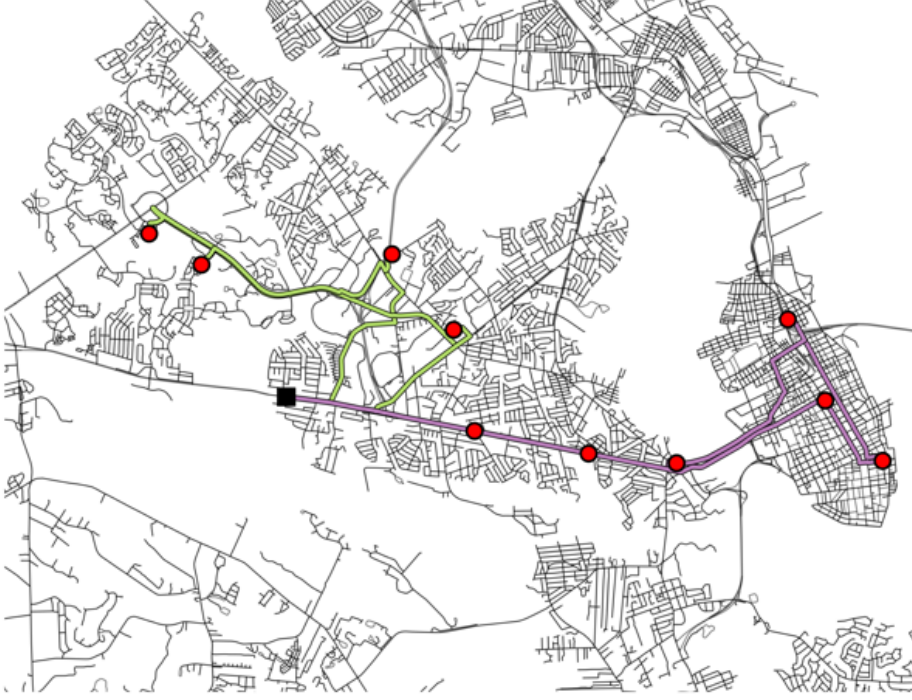


Figure 6.8: Solution in Charleston with $\epsilon = 0$

instances are characterized by $\bar{t} = 450s$, $k = 15$, and $t_{max} = 4h$. In each figure, the solution consists of two routes (two vehicles) highlighted in different colors. The black square represents the depot, whereas the red dots are the customers. By comparing the two solutions for the Charleston instance, it is evident that reducing the population exposure can alter the routing plan. With $\epsilon = 0$, one vehicle is assigned to serve only two customers, and the chosen routes follow shorter paths that traverse residential areas. In contrast, the solution obtained with $\epsilon = 1$ shows a more balanced load distribution between the two vehicles, selecting routes entering residential zones only when strictly necessary.

Turning to the San Francisco instance, both solutions highlight how minimizing population impact can lead to non-intuitive routing choices. In the solution with $\epsilon = 0$, one vehicle is used to serve a single customer. A similar pattern emerges with $\epsilon = 1$, where one vehicle

Figure 6.9: Solution in Charleston with $\epsilon = 1$

serves only two customers. In both solutions, the routes avoid residential areas whenever possible. This often results in longer paths, especially in the $\epsilon = 1$ case. Even in the solution with $\epsilon = 0$, a similar behavior can be observed on a smaller scale: for example, the sky-blue path crosses a park to bypass a residential zone.

Sensitivity analysis on parameters t_{max} and $\bar{t}$

We now analyze the impact of parameter t_{max} on the routing time and equity.

Figure 6.12 displays boxplots showing the distribution of routing times in each city, grouped by t_{max} values. San Francisco's values are more concentrated, while Charleston's are more spread out, due to San Francisco's more compact topology. In both cities, higher t_{max} values lead to lower routing times. This is because a larger time limit allows the use of fewer vehicles, thereby reducing total routing time. In Charleston, the average number of vehicles used decreases from 2.25 ($t_{max} = 3h$) to 1.64 ($4h$) and 1.17 ($5h$). Similarly, in San Francisco, the average number of vehicles is 2.02 ($3h$), 1.57 ($4h$), and 1.02 ($5h$).

Figure 6.13 illustrates the distribution of the equity values in each city, grouped by t_{max} . In Charleston, as t_{max} increases, the equity function value slightly decreases. This is because allowing more time for each vehicle enables solutions with longer paths that, in turn, have better equity values. In San Francisco, $t_{max} = 5h$ yields the best equity values, as expected. However, the worst-performance configuration is $t_{max} = 4h$, showing a larger difference between upper and lower quantiles and higher average values. This is due to the sharp reduction in the routing time when moving from a $3h$ to a $4h$ time limit. Generally, a lower number of vehicles (due to longer time limits) leads to lower routing times but also less

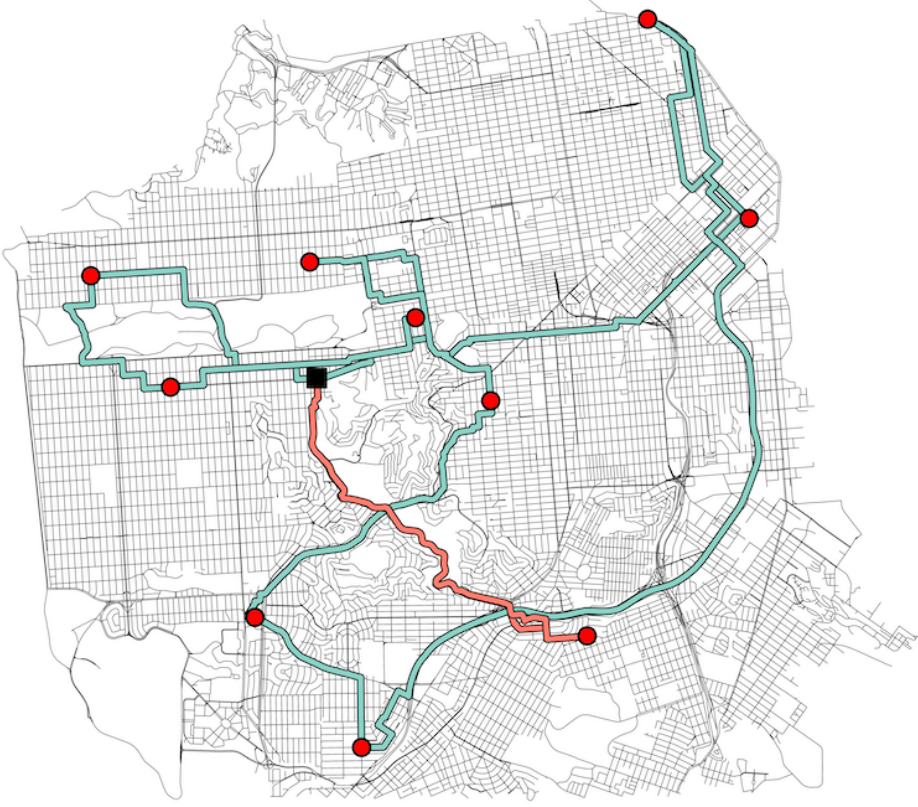


Figure 6.10: Solution in San Francisco with $\epsilon = 0$

flexibility to improve equity. While a larger time limit often leads to better routing times, it does not guarantee a lower impact on the population in this context.

Figure 6.14 shows the evolution of the equity values in both cities as ϵ changes. Values are grouped by the parameter t_{max} . The behavior is consistent with observations in Figure 6.6. In Charleston, average equity values for the $3h$ case are slightly larger than the $4h$ ones, which are in turn larger than the $5h$ instances. This aligns with Figure 6.13.

In San Francisco, for $0.05 \leq \epsilon \leq 0.25$, equity values for $4h$ instances are larger than for $3h$ ones. To understand this, we can consider the average *remaining delivery time*, measured as t_{max} minus the total travel time for the vehicle minus the customers' service time. In the 10% worst cases, with $0.05 \leq \epsilon \leq 0.25$, for $t_{max} = 4h$, the average remaining delivery time is equal to 930s, compared to over 2032s for $3h$ and 2326s for $5h$. Since the ϵ -constraint likely prevents the use of additional vehicles, reducing equity depends on increasing current route lengths. With limited remaining delivery time, altering the routing plan is difficult, leading to worse performance for the $4h$ time limit in San Francisco.

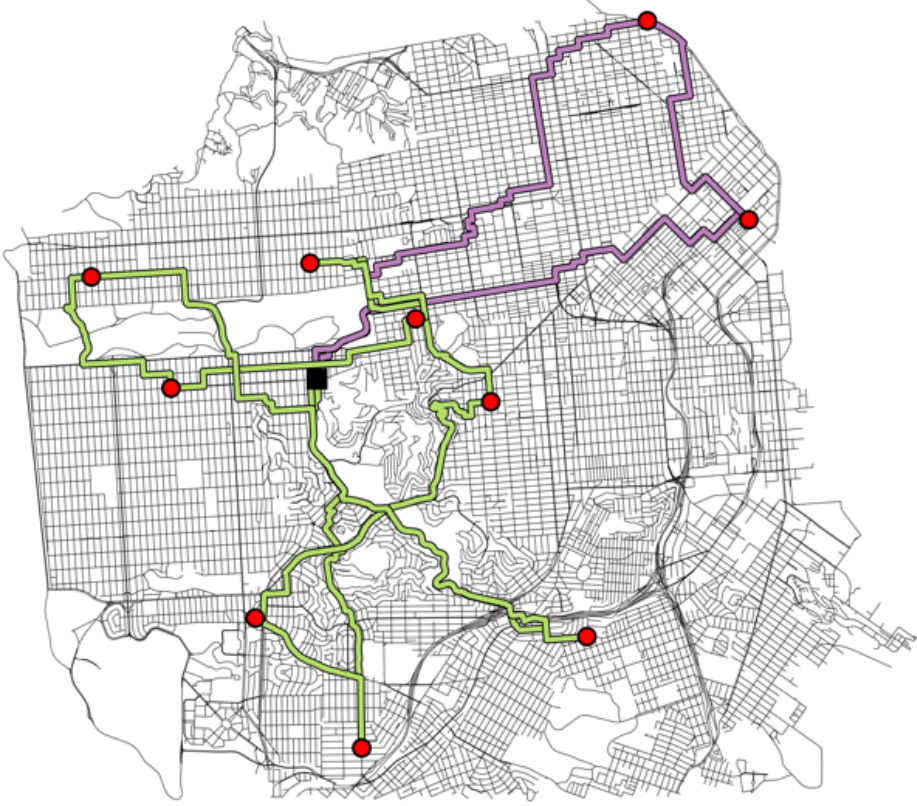
Figure 6.11: Solution in San Francisco with $\epsilon = 1$

Figure 6.15 depicts the distribution of routing times based on the average distance between nodes, grouped by maximum route duration. As expected, routing times increase with $\bar{t}$. In Charleston, the routing time more than doubles when passing from $\bar{t} = 150s$ to $\bar{t} = 300s$, increases 1.53 times when passing to $\bar{t} = 450s$. Finally when passing to $\bar{t} = 600s$ the routing time increases 1.36 times. Similarly, in San Francisco the first switch increases the routing time by 1.91 times, the second switch increases the cost by 1.30 times, finally the last switch increases the cost by 1.12 times. It is interesting to note that in both cities a similar pattern emerges. At the beginning the increase in routing time is fairly large, then the value tends to become less and less relevant.

Lastly, the inverse relationship between the routing time and the time limit remains valid, also when taking into account $\bar{t}$. In fact, as can be easily seen, with $t_{max} = 3h$ the average routing time is maximal for each average distance. We observe that in Charleston, the routing time with $t_{max} = 3h$ is 1.15 times larger than the average, whereas this figure for San Francisco takes the value 1.12.

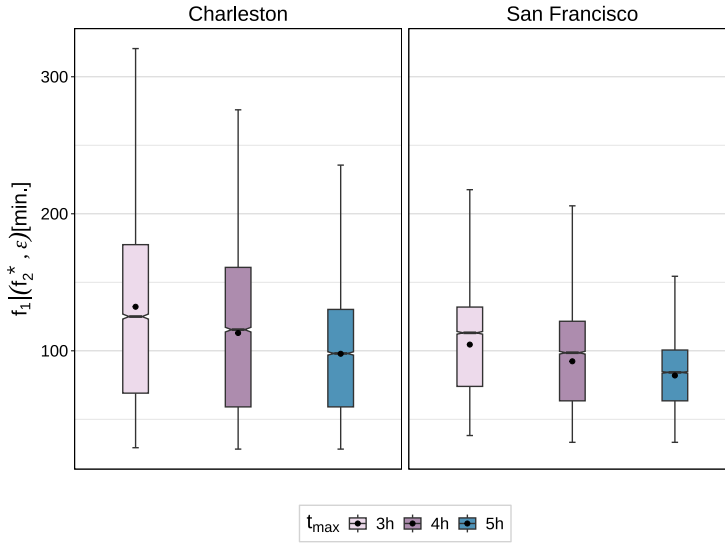


Figure 6.12: Distribution of the value of f_1 in the solutions of $f_2|(f_1^*, \epsilon)$ with respect to t_{max} , in the two cities

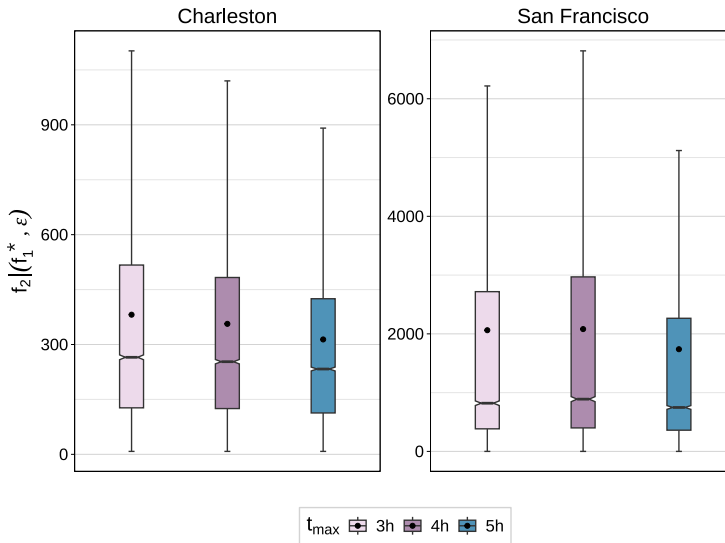


Figure 6.13: Distribution of the value of f_2 in the solutions of $f_2|(f_1^*, \epsilon)$ with respect to t_{max} , in the two cities

In Figure 6.16, statistics of the equity value by the average distance between nodes are shown. Values are grouped by the maximum route duration. Equity tends to grow with the average distance between nodes, whereas it is inversely proportional to t_{max} .

In both cities, when $\bar{t} = 150s$, the differences between the various values of t_{max} are almost negligible. For example, in Charleston, the gap between the maximum and minimum

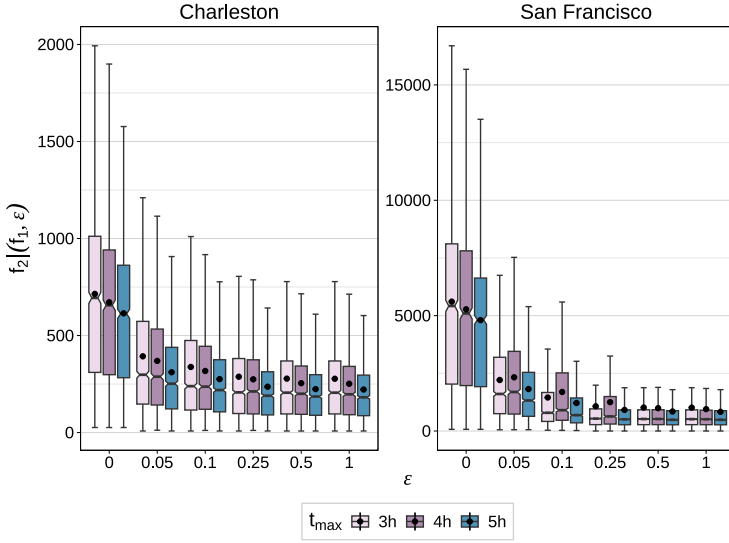


Figure 6.14: Distribution of the value of f_2 in the solutions of $f_2|(f_1^*, \epsilon)$ with respect to t_{max}

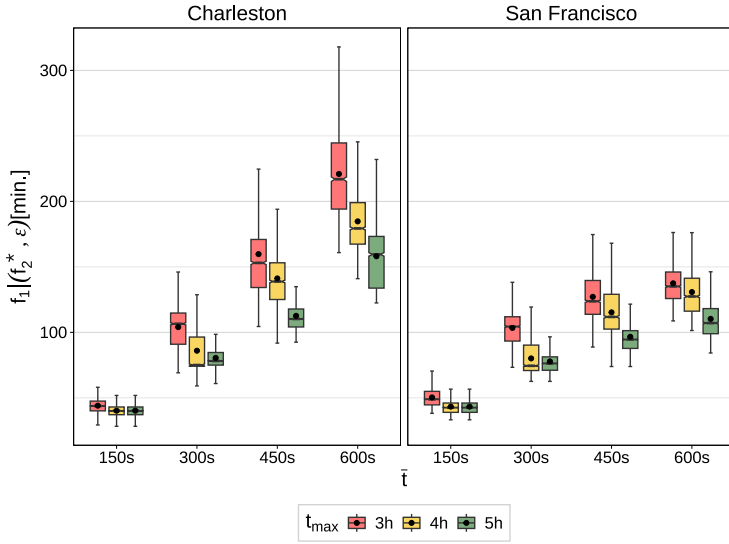


Figure 6.15: Distribution of the routing time by the average distance between nodes

average equity is only 1.56%, while in San Francisco it is 8.27%. These differences are particularly small when compared to those observed at other values of $\bar{t}$: on average, the difference is 28.73% in Charleston and 21.71% in San Francisco. The fact that, regardless of the value of t_{max} , the average equity is almost always the same for $\bar{t} = 150s$ can be attributed to two concurrent factors. First, since with such a small average distance between nodes, it is possible to complete the routing schedule a lot in advance. It can be observed that in Charleston, on average, 27.10% of the available routing time is not used. Similarly, in San Francisco,

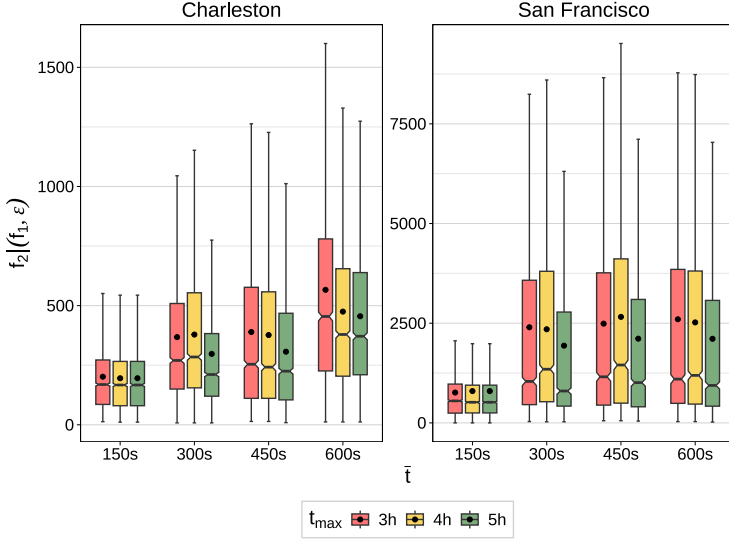


Figure 6.16: Distribution of the equity value by the average distance between nodes

this figure has the value 26.23%. This shows that t_{max} is not a constraining factor for the equity function in $\bar{t} = 150s$ instances. Second, with such a small average distance between two nodes, it is quite likely that the equity values, especially with a larger ϵ , are indeed the optimal ones.

Looking at San Francisco, in Figure 6.16, it is possible to observe that in 300 and 450 s instances the equity values tend to be higher with $t_{max} = 4h$ than with $3h$. This confirms what has been discussed in Figure 6.15.

6.8.2 Managerial insights

Our sensitivity analysis of key parameters reveals several critical insights for practitioners and policymakers who aim to optimize last-mile deliveries while mitigating their impact on urban residents. We briefly summarize the main results for each parameter and how they translate into practical strategies:

1. Parameter k : increasing k leads, as expected, to a gradual improvement of the equity. However, after a certain point, the marginal gains diminish while the routing time, although increasing, remains within reasonable limits. This suggests that managers should consider increasing k up to an optimal threshold while evaluating routing efficiency. Particular attention in the selection of the value of k should be placed especially for cities like Charleston where alternative paths have diverse costs. Having at least three alternative paths for any pairs of nodes mitigates the negative impact of k on routing costs. Moreover, it is desirable that k value should be related to the number of customers and scale accordingly.
2. Parameter t_{max} : on one side, as t_{max} increases, the routing time decreases. Allowing more time per route reduces the number of vehicles required, optimizing routing efficiency. Managers should carefully adjust t_{max} to balance between reducing the number

of vehicles and maintaining operational flexibility also considering city network. For cities with more compact networks (e.g., San Francisco), the reduction in vehicles may be less pronounced compared to cities with more spread population distribution (e.g., Charleston). On the other side, increasing t_{max} often leads to better equity values, as longer routes can be balanced more effectively. However, a larger t_{max} does not necessarily guarantee significant improvements in equity in all scenarios. For example, in San Francisco, increasing the t_{max} from 3 to 4 hours did not lead to an improvement in equity. Managers should be cautious about setting excessively long time limits, as they may reduce flexibility in equity improvements without substantial gains in routing efficiency.

3. Parameter $\bar{t}$: as the average distance between nodes increases, routing times naturally rise, though the incremental increase may become less significant with higher $\bar{t}$ values (e.g., for San Francisco). Similarly, equity tends to increase with larger distances but is inversely related to the t_{max} value. Jointly fine-tuning both t_{max} and $\bar{t}$ is essential for balancing routing efficiency and equity. Managers should adjust these parameters based on the network's characteristics and their operational priorities.
4. Parameter ϵ : when dealing with two different objective functions, it becomes relevant for managers to analyze Pareto frontiers of the bi-objective problem to understand trade-offs and support informed and efficient decision-making. In fact, there is no single optimal solution for a multi-objective problem, and the identification of the best trade-off for a company depends on the strategic vision of its management.

In the following, we show, as an example, the Pareto frontiers of two instances, one related to Charleston and one to San Francisco. The Pareto frontiers have been generated for each value of $k = \{1, 5, 10, 15, 20\}$.

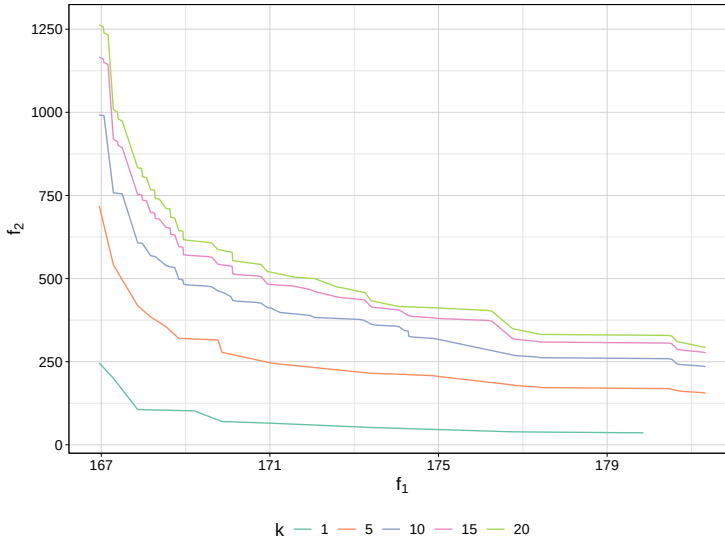


Figure 6.17: Pareto frontier for each k value - city of Charleston ($\bar{t} = 600s$, $t_{max} = 4h$, and $\bar{p} = 3$)

Figure 6.17 and Figure 6.18 depict the five Pareto frontiers for Charleston and San

Francisco, respectively. In both cases, the gap between the frontiers is large when comparing $k = 1$ and $k = 5$, but it gradually decreases as k increases. In all the Pareto frontiers, the decrease in equity values is very sharp at the beginning (for lower values of f_1) and becomes almost flat at the end (for higher values of f_1).

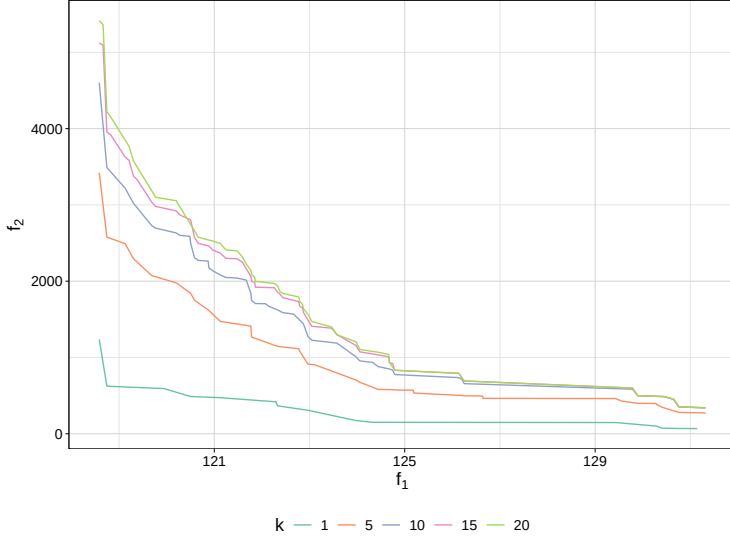


Figure 6.18: Pareto Frontier by k in San Francisco ($\bar{t} = 600s$, $t_{max} = 4h$, and $\bar{p} = 3$)

6.8.3 Statistical analysis of the results

In this section, we first analyze the Pearson correlation index between the objective functions and the most relevant parameters. Using a dimensionless statistic gives us the possibility of comparing emerging behaviors in the two cities, ignoring differences in population densities and street topologies. At the end, we provide the statistical relevance of the results obtained for different values of $\bar{p}$ and ϵ from an equity point of view.

In Table 6.6, results are grouped by city (first column), objective function type (second column), number of parallel arcs (third column). The RS column indicates the correlation between the objective functions and the number of residential streets traversed in the solution. The NV column indicates the correlation between the number of vehicles used by the solutions and the objective functions. The other columns take the names of the parameters introduced in Section 6.3. First, we focus on Charleston. The total routing time is strongly correlated with parameter $\bar{t}$: with any number of paths, the correlation is always above 0.88. With respect to NV, the correlation is significant, although less pronounced. As already noticed in the previous analysis, there is a weak negative correlation between t_{max} and the routing time. Notably, there is no correlation between the routing time and parameters k and ϵ . It is also possible to notice a weak negative correlation between the number of residential streets traversed and the routing time. In fact, in Charleston many shortest paths are likely to be on residential streets, meaning that traversing more residential streets might return cheaper routing times. As expected, there is a moderate negative correlation between equity and k . A milder effect is also present between the equity and ϵ . It is possible

City	Obj	$\bar{p}$	$\bar{t}$	t_{max}	k	ϵ	RS	NV
Charleston	f_1	1	0.88	-0.22	0.05	0.16	-0.13	0.64
		3	0.90	-0.23	0.02	0.08	-0.10	0.66
		5	0.90	-0.23	0.02	0.07	-0.10	0.67
	f_2	1	0.36	-0.09	-0.50	-0.14	0.08	0.16
		3	0.23	-0.06	-0.40	-0.27	0.16	0.11
		5	0.23	-0.06	-0.40	-0.27	0.16	0.11
San Francisco	f_1	1	0.82	-0.23	0.03	0.32	-0.41	0.54
		3	0.86	-0.28	0.01	0.12	-0.36	0.53
		5	0.87	-0.28	0.01	0.11	-0.35	0.53
	f_2	1	0.26	-0.05	-0.39	-0.25	-0.17	0.06
		3	0.13	-0.02	-0.25	-0.29	0.18	0.03
		5	0.13	-0.02	-0.25	-0.29	0.18	0.03

Table 6.6: Pearson correlation index between parameters and objective function values

to notice that the correlation between f_2 and ϵ becomes larger when the number of parallel arcs increases, whereas it gets smaller with respect to k . A counter intuitive outcome is the negligible impact of the number of residential streets traversed on the equity value. Due to Charleston's street layout and population distribution, the number of residential streets traversed is irrelevant for the equity value. The correlation of $\bar{t}$ with f_2 is rather weak. It is not hard to imagine that larger distances mean a larger geographical area to cover, which implies a larger impact on the population. This effect, however, is partially mitigated by the number of parallel arcs, as it can be observed by the decrease in the correlation index when the number of paths increases.

San Francisco shows similar correlation results. The correlation of parameter ϵ with f_1 is higher with respect to Charleston but remains weak. Compared to Charleston, it is possible to see a moderate negative correlation between RS and the routing time.

To further investigate the results, we check whether the equity values produced using different ϵ and $\bar{p}$ values are statistically different in a pairwise manner. In order to do that, we apply the Bonferroni-adjusted Wilcoxon test. The method ranks the equity values for each instance. If the average difference between the ranks of two parameters (computed out of all instances) is greater than a Critical Difference (CD), then the two parameter values have a significantly different performance. CD is a constant value computed according to a chosen significance level α , equal to 0.01 in this case. In Demšar (2006), the author introduces a way to visually represent CD and the difference between parameters. We use this representation in Figure 6.19. In the upper line, each parameter is connected to the corresponding average rank. CD value is also represented. If the difference between the ranks is less than CD, a bold line is drawn across the lines belonging to the two parameter values, indicating that their results are not statistically different. Figure 6.19a and Figure 6.19b show the statistical impact of the parameter ϵ on the equity value in Charleston and San Francisco, respectively, whereas Figure 6.19c and Figure 6.19d show the statistical impact of the number of parallel arcs on the equity value in the two cities. In Figure 6.19a and Figure 6.19b, we note that the difference between $\epsilon = 0.50$ and $\epsilon = 1.00$ is not statistically significant. It is worth noting that $\epsilon = 0.25$ obtains an average ranking below two, confirming previous analysis that showed limited improvements beyond this value of ϵ . In Figure 6.19c and Figure 6.19d, we can notice that the difference between $\bar{p} = 3$ and $\bar{p} = 5$ is not statistically significant for San Francisco, but in the Charleston case, it is barely above the Critical Distance threshold.

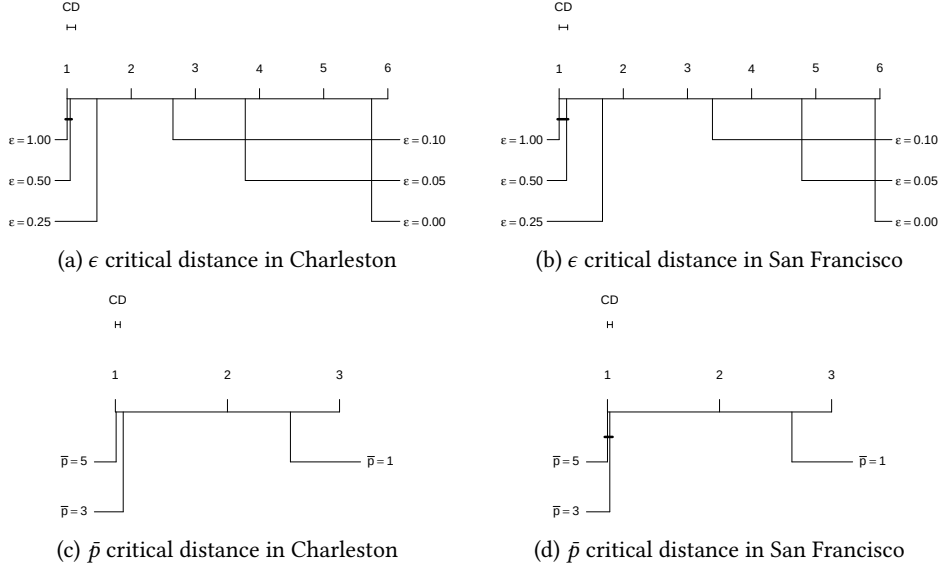


Figure 6.19: Critical Distance plots on the equity value

6.9 A Kernel Search for k -EqVRP

In this section, we introduce the Kernel Search algorithms designed to tackle k -EqVRP. The following of this section is structured as follows. First in Section 6.9.1 we provide a general introduction of the Kernel Search Matheuristic. Then, Section 6.9.2 presents the algorithm in detail.

6.9.1 An Introduction to Kernel Search

Kernel Search (KS) is a general purpose matheuristic framework proposed by Angelelli et al. (2010). KS strategically controls a MILP solver to identify high-quality solutions within a shorter time with respect to a full optimization.

Before continuing with the introduction of the pseudo code, let us define some notation. Let I be a set of binary variables, and let J be a set linear constraints. Let P be a generic minimization problem, defined as follows:

$$P := \min \left\{ \sum_{i \in I} c_i x_i : \sum_{i \in I} a_{ij} x_i \geq b_j \forall j \in J, x_i \in \{0, 1\} \forall i \in I \right\}. \quad (6.28)$$

Now, let $\hat{I} \subseteq I$ be a subset of variables, and $P(\hat{I})$ be the problem restricted to variables in $\hat{I}$, as follows:

$$P(\hat{I}) := \min \left\{ \sum_{i \in I} c_i x_i : \sum_{i \in I} a_{ij} x_i \geq b_j \forall j \in J, x_i \leq 0 \forall i \in I \setminus \hat{I}, x_i \in \{0, 1\} \forall i \in \hat{I} \right\}. \quad (6.29)$$

Notice that $P(I) \equiv P$. Moreover, we define with x_i an optimization variable of P , with $\bar{x}_i^{lp}$ we identify its optimal value in a linear relaxation, and with $\bar{x}_i^{rc}$ we identify its reduced cost.

We this notation, we can proceed with the introduction of the KS pseudo code in Algorithm 6.1.

Algorithm 6.1 Kernel Search Matheuristic

```

1: Solve  $P$  linear relaxation.
2: Construct Kernel:  $\Lambda := \{i : \bar{x}_i^{LP} \geq 1 \forall i \in I\}$ 
3: Sort variables in  $I \setminus \Lambda$ , and split them in NB equally size buckets.
4: Solve  $P(\Lambda)$  as MILP, and let  $z^*$  be the incumbent value
5: for  $b \in 1..NB$  do
6:   Add constraints:
       1.  $\sum_{i \in b_b} x_i \geq 1$ 
       2.  $\sum_{i \in I} c_i x_i \leq z^*$ 
7:   Solve  $P(\Lambda \cup b_b)$  and update  $z^*$ 
8:   Remove additional constraints
9:   Construct  $\Lambda_b = \{i : x_i^* = 1 \forall i \in b_b\}$ 
10:   $\Lambda = \Lambda \cup \Lambda_b$ 
11: end for
  
```

Its main steps are as follows. First, the linear relaxation is solved to identify the set of promising variables. These are called the Kernel Set. The remaining variables are sorted and partitioned into subsets, called buckets, according to some criterion. Then, KS solves a sequence of MILPs, one for each bucket, where only variables in the union of the kernel set and the current bucket are allowed to take values other than zero. Each iteration serves two purposes. First and foremost, it looks for a new improving feasible solution; second, it identifies variables candidate to enter the kernel set. Kernel set is non-decreasing, i.e. when a variable enters the kernel set it cannot leave it. The idea is to iteratively identify only the most relevant variables, and to have, at the last iteration, a kernel set containing almost all the variables selected in an optimal solution.

A common way to construct the initial kernel set is to select variables set to one in the linear relaxation. Moreover, variables outside the kernel set are sorted in non-decreasing order with respect to their value in the relaxation; whereas zero-valued variables are sorted in non-increasing order of their reduced costs. Such a sorting hopes to group together similarly promising variables in the same bucket, allowing the solver to try to use them effectively.

6.9.2 The Kernel Search Matheuristic

The core assumption of a KS as the one described above is that it is possible to gather useful information about the problem from its linear relaxation. When such an assumption does not hold, the performance of the algorithm can be quite mediocre, therefore it is required to identify new mechanisms to construct the kernel set, and new methods to sort variables outside.

As an example in Figure 6.20 we present the relaxed solution of the same instance. On the left when optimizing f_1 (Figure 6.20(a)), on the right when optimizing f_1 (Figure 6.20(b)), both subject to (6.14)–(6.25). The color of the arc, ranging from red to green, indicates the value taken by the associated x_{ij}^P variable, where colors closer to red indicates a value closer to 0, and colors closer to green indicates values closer to 1. In both graphs, it is possible to observe similar patterns. First, many times if arc (i, j) is selected so is arc (j, i) . In both

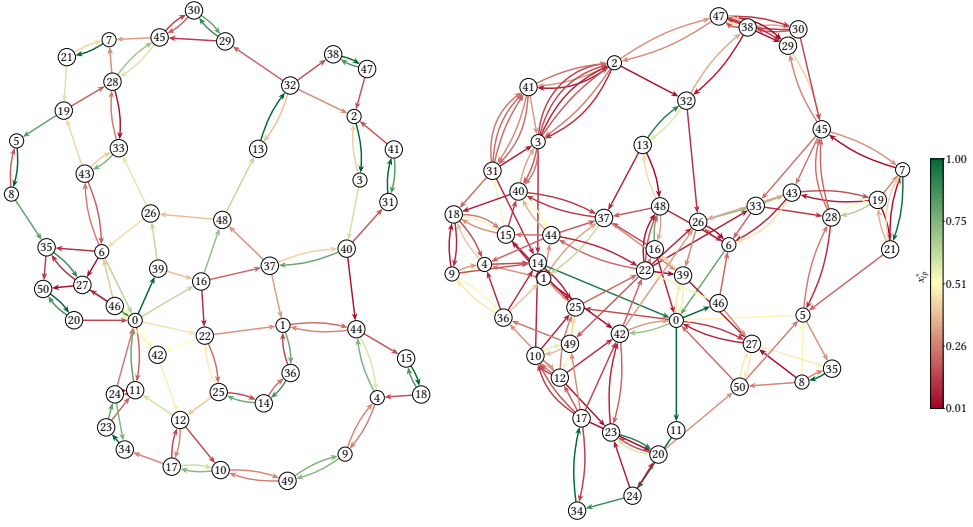
Figure 6.20(a): f_1 LP solutionFigure 6.20(b): f_2 LP solution

Figure 6.20: Example of an LP relaxation of an instance with 51 nodes

graphs, it is possible to observe several subtours. Paying attention at Figure 6.20(b), it is possible to observe that almost all non-null x_{ij}^p have a value that is less than 0.25. Moreover, in this solution a large number of parallel arcs have been selected to very similar values.

This simple analysis shows that the quality of information contained in the linear relaxation is quite low. In fact, a very small number of variables takes values close to 1, therefore creating a very small, and likely infeasible, initial kernel. Moreover, the sorting variables outside the kernel set according to their objective value can prove to be ineffective, since many of them take similar values.

To address this issue, we propose a novel way to construct the initial kernel, and an innovative mechanism to sort variables outside it. The general outline of the algorithm is as follows:

1. A high-quality, according to f_1 , solution is found with a dedicated Tabu Search/ALNS heuristic. This procedure is described in Section 6.9.3.
2. The solution from Step 1 is used to compute the variable rank in an iterative process where multiple LPs are optimized. The rationale and the ranking routine are introduced in Section 6.9.4.
3. A so called Self Adapting Kernel Search is run with the initial kernel set found at Step 1, and bucket variables sorted as in Step 2. Buckets are iteratively until time limit is reached, at each iteration number of buckets and their time limit are dynamically adjusted according to the previous iteration. The algorithm is detailed in Section 6.9.5.

6.9.3 Initial VRP solution

The VRP is found using a custom Tabu Search (TS) (Glover, 1986, 1989) running an embedded Adaptive Large Neighborhood Search (ALNS) (Ropke and Pisinger, 2006; Pisinger and Ropke,

2007). We call this algorithm TS-ALNS. Besides the fact the k -EqVRP is formulated over a multigraph, TS-ALNS considers only the shortest one in terms of travel time. The pseudo code is presented in Algorithm 6.2.

Algorithm 6.2 TS-ALNS for VRP

Require: instance data, parameter It_{max} (outer iterations)

```

1:  $x \leftarrow InitialSolution()$  ▷ Parallel Cheapest Insertion
2:  $x_{best} \leftarrow x$  ▷ Set best solution so far
3: Initialize operator weights and tabu list
4: for  $1..It_{max}$  do
5:   for  $i \in 1..|N|$  do
6:      $best \leftarrow \text{false}$ 
7:      $TT \leftarrow SelectTabuTenure()$  via roulette wheel
8:      $R_{max} \leftarrow SelectMaxRemoval()$  via roulette wheel
9:     for  $j \in 1..|N|$  do
10:       $UpdateTabuList()$  ▷ decrease tabu counters
11:       $(d, r) \leftarrow SelectDestroyRepair()$  via roulette wheel
12:       $v \leftarrow RandomInteger(1, R_{max})$ 
13:       $S_{rem} \leftarrow SelectRemovableNodes(x, v)$  avoiding tabu nodes
14:       $x' \leftarrow r(d(x, S_{rem}))$  ▷ destroy & repair
15:       $MarkNodesAsTabu(S_{rem}, TT)$ 
16:      if  $cost(x') < cost(x)$  then
17:         $UpdateOperatorWeightsImprovement(d, r)$ 
18:      end if
19:      if  $cost(x') < cost(x_{best})$  then
20:         $x_{best} \leftarrow x'$ 
21:         $UpdateOperatorWeightsBest(d, r)$ 
22:         $best \leftarrow \text{true}$ 
23:      end if
24:       $x \leftarrow x'$  ▷ move to new solution
25:    end for
26:    if not best then
27:       $x \leftarrow x_{best}$  ▷ restart from best
28:    else
29:       $UpdateScores(TT, R_{max})$  ▷ reward good param choices
30:    end if
31:  end for
32:   $ResetOperatorWeights()$ 
33:   $ResetTabuList()$ 
34: end for return  $x_{best}$ 

```

First, we present a general outline of the algorithm, then we provide a detailed description of all the operation it performs. The ALNS part of the algorithm select, using a roulette wheel, a *destroy* operator that removes nodes from the solution and a *repair* operator that reinserts the removed nodes in a (hopefully) cheaper manner. The TS part of the algorithm is carried out by a tabu-list, specifying for each node for how many iterations which nodes cannot be removed from the solution, prohibiting ALNS from removing them with the destroy operator. Nodes that have been removed and reinserted are set as tabu for a certain number of iteration.

In detail, the algorithm works as follows. First, an initial solution is identified using a parallel cheapest insertion heuristic. This initial solution is then copied into the incumbent solution. At this point, three nested loops are executed. The outer one (lines 4–34) runs I , a parameter specified by the user, times. Inside it, the median loop (5–31) runs an intensification procedure, aiming at improving the current best. At the beginning of this loop the size of the tabu-tenure and the maximal number of items removable by each destroy operator. Then, the inner loop (9–25) run the local search iterations. In each iteration, first the tabu-list is updated by decreasing by one the number of iterations tabu nodes cannot be removed from the current solution. Then, using a roulette wheel, the algorithm selects a destroy and a repair operators. At this point the algorithm selects a random number of nodes to remove, and proceeds to remove this number of non-tabu nodes from the current solution. Notice that if this number is larger than the pool of non-tabu nodes, then all non-tabu nodes are removed from the solution. The algorithm reinserts the removed nodes, according to the logic defined by the repair operator, then set the new re-inserted nodes as tabu for TT iterations. If the cost of the new solution x' is better than the incumbent, then the weights for the selected operators are updated. Moreover, if the solution is the new best, the weights are updated and the best solution are updated. After the inner loop, if a new best solution has not been found, the incumbent is reset to be best solution, otherwise the score for the selected tabu tenure and max remove size are updated. At the end of the outer loop, the roulette wheels and the tabu list are reset to the initial values, to avoid an over fitting of their parameters.

Given a valid VRP route $\Pi = \{0, \dots, i, j, k, \dots, 0\}$, we define the removal discount node j as $c_{ij} + c_{jk} - c_{ik}$, that is the reduction in cost of Π when j is removed from it. The ALNS algorithm uses the following remove operators.

- Global Random: remove randomly selected nodes from all the available routes.
- Random Route Max Cost: remove nodes with the largest discount from a randomly selected route.
- Longest Route Max Cost: remove nodes with the largest discount from the longest route.
- Global Max Cost: remove nodes with the largest discount from all the available routes.
- Global Min Cost: remove nodes with the smallest discount from all the available routes.

We propose the following repair operators.

- Best Cheapest Insertion: insert nodes sequentially into the cheapest possible feasible position.
- Regret- q : insert nodes using the regret rule, with $q = 2, 3, 4$. Regret-based insertions (Potvin and Rousseau, 1993) mitigate the myopic behavior of pure greedy heuristics as the cheapest insertion. Let δ_i^h denote the cost of inserting node i into its h -th cheapest feasible position. The Regret- q score is:

$$i^* := \arg \max_{i \in N} \left(\sum_{h=2}^q \delta_i^h - \delta_i^1 \right). \quad (6.30)$$

which selects the node whose deferred insertion would be most detrimental. Nodes are inserted iteratively until none remain.

- First Cheapest Insertion: insert nodes in the cheapest position on the first feasible path.
- Worst Cheapest Insertion: insert nodes in the most expensive cheapest position among all paths.

6.9.4 Variable Ranking

In this section we introduce our variable ranking algorithm.

Given an initial VRP solution, Algorithm 6.3 computes two distinct scores for each variable by repeatedly solving LP models, one for each arc (i, j) active in the initial solution. In each LP model, the variable associated with the currently processed arc is set to 1, while the others are left unconstrained. Such a process allows to inspect the impact each variable has on the global solution. Moreover, by inspecting the objective value of each variable, it allows to understand which variables are most likely to be selected in high-quality or optimal solution by inspecting the following metrics. The first score **transition** counts how many times the variable has changed value with respect to the previous iteration. Variables with a higher **transition** score are the ones about which the solver is less *sure*. The rationale is that variables taking value 0, or anything close to it, are most likely variables that will never be used in a high-quality solution. On the other hand, variables taking value 1, or anything close to it, are very likely to be selected in a high-quality solution. This leaves out the variables that *oscillates* between 0 and 1 in different iterations, for which it is harder to predict if they will or won't be selected in a high-quality solution. The second score **set-high** counts how many times the variable has an objective value that is at least equal to threshold quantile of the values of the non-zero variables. Variables with a higher **set-high** score are the ones that are most often used in an LP relaxation, independently from the state of the other variables. We propose to lexicographically sort variables according to the **transition** score first, and in case of ties by their **set-high** score. In this way, a kernel search algorithm more hard to classify variables first, allowing to quickly identify good quality solution.

Assuming that the input is a solution, who's a collection of paths, and a route is the sequence of arcs representing a vehicle's cycle, the algorithm works as follows. The algorithm first initializes the score of each arc to zero. Then, for each route in the solution, and for each arc in the route the algorithm computes the cumulative score for each arc as follows. First, a new constraint forcing the x_{ij}^0 variable associated with the currently processed arc to one is added to the problem. The new model is then optimized as an LP, the solution is retrieved, and a set containing all variable whose value is larger than ϵ_0 , a small number to filter numerical noise, is constructed. At this point, the algorithm iterates through each arc in A . For each arc, the algorithm retrieves the value of the associated x variable, and if this value is above the q_h the associated **set-high** score is increased. Moreover, if the value changed from the previous solution, then the associated **transition** score is increased by one. Once all the arcs have been processed, the algorithm removes the constraint and proceeds to process the next arc.

6.9.5 Kernel Search Algorithm

Before presenting the proposed Kernel Search (KS) algorithm, let us introduce some notation. Let $\bar{x}$ be a feasible solution with respect to (6.15)-(6.25). We call $f_i(\bar{x})$ the value of objective function $f_i, i = 1, 2$ of solution $\bar{x}$. With $f_2^e(\bar{A})|f_1(\bar{x})$, with $\bar{A} \subseteq A$, we denote the following

Algorithm 6.3 *RankScore*(*sol*, ξ)**Require:** *sol*: initial VRP solution; parameter $\xi \in (0, 1]$

```

1: Initialize score array  $S[(i, j, p)]$  with fields set-high and transition equal to zero
2: Initialize array  $prev[(i, j, p)] \leftarrow \text{undefined}$  ▷ Stores value from previous LP
3: for each route in sol do
4:   for each arc  $(i, j, p)$  in route do
5:     Add constraint:  $x_{ij}^p = 1$ 
6:     Solve LP relaxation and obtain solution  $\bar{x}$ 
7:      $\mathcal{A} \leftarrow \{\bar{x}_{i'j'}^{p'} : (i', j', p') \in A, \bar{x}_{i'j'}^{p'} \geq \epsilon_0\}$ 
8:      $\tau \leftarrow Q_\xi(\mathcal{A})$  ▷ Threshold quantile
9:     for each variable  $(i', j', p') \in A$  do
10:        $v \leftarrow \bar{x}_{i'j'}^{p'}$ 
11:       if  $v \geq \tau$  then
12:          $S[(i', j', p')].\text{set-high} += 1$ 
13:       end if
14:       if  $v \neq prev[(i', j', p')]$  then
15:          $S[(i', j', p')].\text{transition} += 1$ 
16:       end if
17:        $prev[(i', j', p')] \leftarrow v$ 
18:     end for
19:     Remove constraint  $x_{ij}^p = 1$ 
20:   end for
21: end for
22: return  $S$ 

```

optimization problem:

$$\begin{aligned}
 \min_{x, z} \{ & f_2 : \\
 & f_1 \leq (1 + \epsilon)f_1(\bar{x}) \\
 & x_{ij}^p = 0 \quad \forall (i, j, p) \in A \setminus \bar{A}, \\
 & (6.15) - (6.25) \}
 \end{aligned} \tag{6.31}$$

The optimization model (6.31) minimizes the objective function f_2 , subject to (6.15)–(6.25), a maximal worsening of the incumbent of f_1 , and a restriction of the available arcs. Notice that such a problem can be infeasible even if the original unrestricted problem is feasible. Moreover, in the following, we identify with $K \subseteq A$ the kernel set and with $B = \{b_1, \dots, |B|\}$ the set of non-overlapping buckets, with $b_i \subseteq A$.

The proposed KS algorithm extends the usual iterative algorithmic framework by including an adaptive system to dynamically update the algorithm metaparameters according to the optimization process. At the end of each bucket set iteration the algorithm controls various statistics, described in detail later, about the previous optimizations and decides if the number of buckets and the time limit have to increase, decrease, or be kept unchanged.

In Algorithm 6.4 we introduce the pseudocode of the proposed KS algorithm.

The algorithm works as follows. First, a VRP solution is identified using Algorithm 6.2. Such a solution will be used a reference point, specifying the maximal routing cost that any solution is allowed to take. Variables are ranked with Algorithm 6.3, using the TS-ALNS

Algorithm 6.4 Self-Adjusting Kernel Search (SAKS)

```

1: Find VRP solution  $\bar{x}^{vrp}$  using Algorithm 6.2
2: call  $RankScore(\bar{x}^{vrp}, \xi)$  ▷ call Algorithm 6.3
3:  $\bar{x}^* \leftarrow \bar{x}^{vrp}$ 
4:  $nb \leftarrow nb_{min}$ 
5:  $tl \leftarrow tl_{min}$ 
6: noImproveCount  $\leftarrow$  0
7: while  $t < T_{max}$  do
8:   betterSolutionFound  $\leftarrow$  false
9:   Initialize kernel set  $K$  with the selected arcs in  $\bar{x}^*$ 
10:  Initialize  $nb$  buckets with remaining arcs
11:  for  $B_i \in B$  do
12:     $\bar{x} \leftarrow$  Solve  $f_2^\epsilon(K \cup B_i) | f_1(\bar{x}^{vrp})$  with  $tl$  time limit
13:    if  $f_2(\bar{x}) < f_2(\bar{x}^*)$  then
14:       $\bar{B}_i = \{(i, j, p) \in B_i : \bar{x}_{ij}^p = 1\}$ 
15:       $K \leftarrow K \cup \bar{B}_i$ 
16:       $\bar{x}^* \leftarrow \bar{x}$ 
17:      betterSolutionFound  $\leftarrow$  true
18:      call  $RankScore(\bar{x}^*, \xi)$ 
19:    end if
20:    call CollectStatistics() ▷ current bucket
21:  end for
22:   $s_{nb} \leftarrow$  ComputeBucketNumberScore()
23:  if  $s_{nb} > 0$  then
24:     $nb \leftarrow \min(nb_{max}, nb + \Delta_{nb})$ 
25:  else if  $s_{nb} < 0$  then
26:     $nb \leftarrow \max(nb_{min}, nb - \Delta_{nb})$ 
27:  end if
28:   $s_{tl} \leftarrow$  ComputeTimeLimitScore()
29:  if  $s_{tl} > 0$  then
30:     $tl \leftarrow \min(tl_{max}, tl + \Delta_{tl})$ 
31:  else if  $s_{tl} < 0$  then
32:     $tl \leftarrow \max(tl_{min}, tl - \Delta_{tl})$ 
33:  end if
34:  if not betterSolutionFound then
35:    noImproveCount  $\leftarrow$  noImproveCount + 1
36:  else
37:    noImproveCount  $\leftarrow$  0
38:  end if
39:  if noImproveCount ==  $MAX_{imp}$  then
40:     $nb \leftarrow nb_{min}$ 
41:     $tl \leftarrow tl_{min}$ 
42:  end if
43: end while
44: return  $\bar{x}^*$ 

```

solution as input. The loop on lines 7–43 runs the KS algorithm until the time limit allows. In this loop, first initialize the kernel set using the x_{ij}^p variables set in the incumbent. All the other variables are sorted in non-decreasing order based on the current ranking. The algorithm creates nb equally sized buckets, with the last potentially smaller if the number of variables outside K is not divisible by nb . At this point, for each bucket $b_i \in B$, model (6.31) is solved restricted to the variables in $K \cup b_i$. If the best solution returned by the solver, either optimal or the best found within the time limit, is better than the incumbent, then the incumbent is updated, the kernel set is extended with the arcs from b_i selected in $\bar{x}$, and the variables ranking are updated. Once all the buckets have been processed, the number of buckets nb and the iteration time limit tl are updated. For both parameters an update score is computed. If such a value is larger than zero, then the parameter is increased. If the score is smaller than zero, then the parameter is decreased. Otherwise, the value is left unchanged. After that, the algorithm checks if the incumbent hasn't changed in the last MAX_{imp} iterations, and if it is the case the metaparameters are reset to their initial values.

During the evaluation of each bucket, the function `CollectStatistics()` records the behavior of the restricted MIP solved on that bucket. In particular, it keeps track of the minimum, average, and maximum MIP gap (Gurobi parameter) observed across all buckets (`MipGap_min`, `MipGap_avg`, `MipGap_max`), together with the number of subproblems solved to optimality (`nOpt`), those that improved the incumbent solution (`nImp`), and those terminated by the time limit with or without a feasible solution (`nTimeLimitFeasible`, `nTimeLimitInfeasible`). These statistics provide a global picture of how difficult the current set of restricted MIPs is to solve and allow the algorithm to adjust its metaparameters adaptively.

The update score associated with the number of buckets, denoted by s_{nb} , is computed by evaluating a set of conditions, each controlled by a threshold parameter. For every difficulty condition that holds, namely:

$$MipGap_max > \gamma_1$$

$$MipGap_max - MipGap_min > \gamma_2$$

$$MipGap_avg > \gamma_3$$

$$nTimeLimitFeasible < nTimeLimitInfeasible$$

the score s_{nb} is increased by one. These conditions signal that the restricted MIP problems tend to be hard, typically because buckets are too large; increasing nb reduces bucket size and decreases the complexity of the associated subproblems. Conversely, for every ease condition that holds

$$MipGap_max < \rho_1$$

$$MipGap_max - MipGap_min < \rho_2$$

$$MipGap_avg < \rho_3$$

$$nOpt < nImp$$

$$nTimeLimitFeasible > nTimeLimitInfeasible$$

the score s_{nb} is decreased by one. These cases suggest that the restricted problems might be too easy, often because buckets are too small, and reducing nb enlarges the neighborhood explored at each iteration.

The update score associated with the time limit, denoted by s_{tl} , is computed analogously. In particular, the first three difficulty conditions correspond exactly to the first three difficulty conditions used for computing s_{nb} , as they reflect the intrinsic hardness of the restricted MIP problems. Whenever one of these conditions holds, s_{tl} is increased by one. The score is also increased when the solver frequently reaches the time limit, either with or without a feasible solution:

$$\text{nTimeLimitFeasible} > \frac{nb}{2}$$

$$\text{nTimeLimitInfeasible} > \frac{nb}{2}$$

These cases indicate that the allocated time is insufficient for the solver to converge reliably.

Symmetrically, the first three easy conditions correspond exactly to the first three easy conditions used in the update of s_{nb} and indicate that the restricted problems are consistently easy. Each of these contributes a -1 to the score. The score is also decreased when few bucket optimizations hit the time limit:

$$\text{nTimeLimitFeasible} < \frac{nb}{4},$$

or when the majority of them are solved to optimality:

$$\text{nOpt} > \frac{nb}{2},$$

indicating that the solver has more time than necessary.

In summary, both update rules rely on the same core measures of problem difficulty captured by the MIP gap thresholds, while the additional conditions for each parameter (nb and tl) reflect the specific role they play in controlling bucket size and allotted computational time.

Taken together, the dynamic adjustment of the metaparameters and the specialized variable-ranking mechanism described in Section 6.9.4 make the proposed procedure fundamentally different from traditional Kernel Search schemes. For these reasons, we refer to the method as the *Self-Adjusting Kernel Search* (SAKS), emphasizing its ability to refine both the search structure and the exploration priorities through continuous, data-driven feedback.

6.10 Computational Results

In this section, we present the computational results of the proposed algorithms. We first describe the two instance classes used in our experiments. We then provide a preliminary comparison of the different k -sum formulations, identifying the most effective one to be adopted in the subsequent analysis. All models are solved with Gurobi 12. Notice that formulation $\mathcal{F}_1$ is not compact, and constraints are handled dynamically as lazy constraints by the solver.

In Section 6.10.3, we examine the performance of TS-ALNS as a standalone heuristic and compare it with the results obtained by Gurobi, which is run for one hour. Next, in Section 6.10.4, we discuss the outcomes of the bi-objective problem, focusing on the optimization of the k -sum impact under an ϵ -constraint on the travel time. Finally, in Section 6.10.5 we present the performance of SAKS on medium- and large-scale instances.

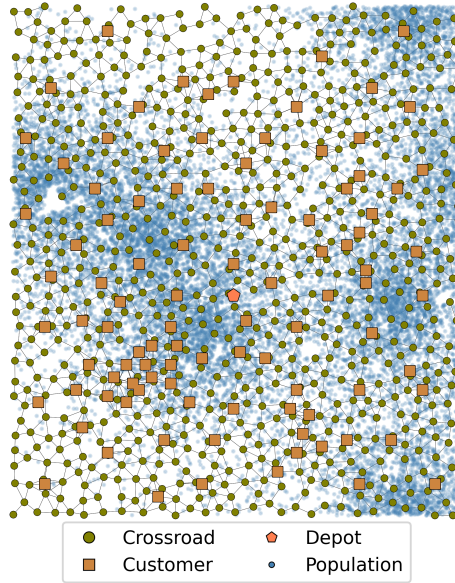


Figure 6.21: Example of an instance made from random (R) points

6.10.1 Instances generation

Set A consists of 30 instances with 50 customers and one depot. We employ this set to evaluate the four k -sum formulations and to tune the SAKS metaparameters. Set B consists of 75 instances with 100 customers and one depot. We use it to assess the computational performance of SAKS.

In both instance sets, the road network is generated by randomly placing a set of intersection points on a plane and connecting nearby intersections to form the street graph. The population is then constructed by placing a number of attraction points (e.g., schools, parks) and repulsion points (e.g., industrial areas) and generating household locations that lie sufficiently close to attractions and sufficiently far from repulsion sources (see Figure 6.21). Each household is assigned to its nearest street segment. Customers and the depot are then placed on the map and linked to their nearest intersection. In Set A, customer locations are generated randomly while enforcing a minimum separation distance. In Set B, customer coordinates are taken from the Solomon VRP instances (Solomon, 1987), which include three spatial distributions: uniformly random (R), clustered (C), and clustered–uniformly random (RC).

For every pair of customers and for each customer–depot pair, we first compute the shortest path. We then generate five additional paths that minimize population exposure while allowing deviations of 0, 25, 50, 75, and 100% from the shortest travel distance. Each candidate path is subsequently refined to reduce travel time without increasing its population impact. On the same plane, we generate five alternative street networks and five population distributions. For every combination of customer set, street map, and population distribution, we construct one EqVRP instance, yielding a total of 75 instances.

Figure 6.21 presents an example of a synthetic instance. The brown squares represent customers and the orange pentagon represents the depot. These points are from the uniform

random Solomon points. Olive circle represents crossroads and junctions in the virtual street networks. Lines connecting them and Solomon points show street segments. Finally, blue dots represent the population. A deeper blue color corresponds to a densely populated area.

All algorithms were implemented in C, with Gurobi 12 used as the MILP backend. All experiments were executed on a machine equipped with an AMD Ryzen 9 3950X (16 cores), 32 GB of RAM, and running Ubuntu 22.

6.10.2 Comparison of the k -sum formulations

In this section, we present the experimental results conducted to identify the most suitable formulation among the ones presented in Section 6.4.

Each instance of Set A is optimized with $k = 5, 10$, and 20 . This leads to a total of $3 \times 30 = 90$ instances. For each instance, first objective function f_1 (travel time) is optimized for 15 minutes, obtaining a VRP solution. Then each k -sum formulation is solved with a time limit of 1 hour, by allowing a 10% worsening of the VRP solution value. Given the four formulations, this leads to a total of $90 \times 4 = 360$ solutions. The VRP solution is used as MIP start for each k -sum formulation.

Table 6.7: k -worst value statistics for the four formulations

k		Q_{05}	Q_{25}	Avg.	Q_{50}	Q_{75}	Q_{95}
5	$\mathcal{F}_1$	985.85	1087.25	1391.43	1260.50	1545.25	2293.40
5	$\mathcal{F}_2$	699.00	760.00	1027.77	887.50	1031.00	1988.20
5	$\mathcal{F}_3$	699.90	859.50	1077.77	923.00	1184.00	2012.95
5	$\mathcal{F}_4$	641.50	812.75	1031.10	918.50	1056.25	2008.15
10	$\mathcal{F}_1$	2032.25	2382.25	3125.53	2798.00	3338.50	4127.25
10	$\mathcal{F}_2$	1231.00	1428.50	1729.07	1654.00	1785.75	2732.80
10	$\mathcal{F}_3$	1361.40	1558.25	1833.87	1752.00	1995.75	2791.40
10	$\mathcal{F}_4$	1261.00	1470.50	1738.97	1677.00	1853.50	2689.20
20	$\mathcal{F}_1$	3811.95	4805.25	5833.20	5243.00	6371.25	9514.05
20	$\mathcal{F}_2$	2091.65	2719.50	2977.67	2948.50	3314.75	3990.45
20	$\mathcal{F}_3$	2208.80	2788.75	3151.33	3171.50	3486.00	4114.30
20	$\mathcal{F}_4$	2219.90	2478.50	2949.63	2807.00	3298.25	3957.00

In Table 6.7, for each value of k and for each k -sum formulation (from $\mathcal{F}_1$ to $\mathcal{F}_4$), in this table and in the following, Q_x , with $x \in [0, 100]$, identifies the x -th percentile of population under analysis. we report the distribution of the best objective values found by Gurobi within the 1-hour time limit. Specifically, the table shows the empirical 5th and 25th percentiles (Q_{05} and Q_{25}), the average and the median, and the 75th and 95th percentiles (Q_{75} and Q_{95}) computed over all benchmark instances.

All four k -sum formulations model the same optimization problem and would therefore return identical objective values if solved to optimality. The differences observed in the table arise exclusively from the limited time threshold assigned to the solver. The reported statistics thus measure how effectively each formulation guides Gurobi toward high-quality solutions within one hour, capturing both solution quality and robustness across instances.

Several trends emerge from the table. First, formulation $\mathcal{F}_1$ is consistently the worst-performing across all values of k and across all statistics. On average over all values of k , the percentage gap between $\mathcal{F}_1$ and the best-performing formulation is 67.9%, 66.0%, and 76.7% for Q_{25} , Q_{50} , and Q_{75} , respectively. The smallest gap observed amounts to 15.4% for

$k = 5$ under the Q_{95} statistic, whereas the largest gap reaches 140% for $k = 20$ under the same statistic. These values indicate that $\mathcal{F}_1$ is substantially harder for the solver to explore effectively, even when considering the upper quantiles of performance.

Across all values of k and all statistics, formulation $\mathcal{F}_1$ is by far the weakest performer, returning the largest objective values and exhibiting the lowest robustness. The formulations $\mathcal{F}_4$ and $\mathcal{F}_2$ consistently produce the best results, with $\mathcal{F}_3$ typically ranking between them and $\mathcal{F}_1$. Although $\mathcal{F}_2$ and $\mathcal{F}_4$ are compact and appear structurally similar, their computational behavior seems to differ. The $\mathcal{F}_2$ model provides a sharper and more informative LP relaxation when k is small ($k = 5, 10$), yielding faster progress in the branch-and-bound tree. Conversely, as k grows ($k = 20$), the $\mathcal{F}_4$ formulation produces a more stable and less degenerate relaxation, which results in better solver performance.

The performance of formulation $\mathcal{F}_3$ is consistently inferior to both $\mathcal{F}_2$ and $\mathcal{F}_4$. On average across all quantiles and all values of k , $\mathcal{F}_3$ is 6.45% worse than $\mathcal{F}_2$, with gaps of 6.6%, 7.9%, and 4.8% for $k = 5, 10$, and 20, respectively. This trend is particularly evident in the higher quantiles, where the average gap between $\mathcal{F}_2$ and $\mathcal{F}_3$ reduces to 2.16%, but remains positive for all k , taking values of 1.24%, 2.14%, and 3.10%. Across all quantiles and all values of k , the average percentage gap between formulations $\mathcal{F}_2$ and $\mathcal{F}_4$ is only 0.38%, confirming their overall similarity in solver performance. Examining each value of k separately, the average gap across all quantiles is 1.13% for $k = 5$, 1.79% for $k = 10$, and -1.77% for $k = 20$, the latter indicating that $\mathcal{F}_4$ slightly outperforms $\mathcal{F}_2$ when k becomes large.

To assess the statistical significance of these observations, we conducted several complementary statistical analyses. We first computed an average ranking to identify, for each value of k , which formulation performs best on average and how the others compare. We then applied a Binomial test to evaluate how often one formulation outperforms another and whether these pairwise differences are statistically significant. Finally, to validate the robustness of these findings, we performed paired t -tests and Wilcoxon tests for all formulation pairs and all values of k , using the Holm correction to account for multiple comparisons. In all tests, the null hypothesis was rejected at the 0.05 significance level.

Table 6.8: Formulation Rankings

k	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$
5	3.92	1.58	2.52	1.98
10	4.00	1.72	2.42	1.87
20	4.00	1.90	2.33	1.77
All	3.97	1.73	2.42	1.87

Table 6.8 reports the average ranking of the four formulations for each value of k and for the entire test set. The ranking is computed by assigning score 1 to the best formulation on each instance, score 2 to the second best, and so on; in the case of ties, the average score is assigned. Smaller values therefore indicate better overall performance. The results show a consistent ordering across all values of k . Formulation $\mathcal{F}_2$ achieves the best average rank for both $k = 5$ and $k = 10$, while $\mathcal{F}_4$ becomes the best-performing formulation for $k = 20$ confirming previous results. Formulation $\mathcal{F}_3$ systematically ranks third, whereas $\mathcal{F}_1$ is always the poorest performer. As k increases, the gap between the formulations narrows, indicating that the relative differences among models become less pronounced for larger values of k . The last row of the table, aggregating all instances, confirms this global pattern: overall, $\mathcal{F}_2$ obtains the best average ranking, followed by $\mathcal{F}_4$, $\mathcal{F}_3$, and finally $\mathcal{F}_1$.

Table 6.9: Pairwise formulation comparisons: number of wins, losses, and ties

Form.	$\mathcal{F}_4$				$\mathcal{F}_3$				$\mathcal{F}_1$			
	Wins	Losses	Ties	p -value	Wins	Losses	Ties	p -value	Wins	Losses	Ties	p -value
$\mathcal{F}_2$	48	35	7	0.09	61	26	3	$1.12 \cdot 10^{-4}$	90	0	0	$8.08 \cdot 10^{-28}$
$\mathcal{F}_4$	-	-	-	-	63	24	3	$1.74 \cdot 10^{-5}$	88	1	1	$1.45 \cdot 10^{-25}$
$\mathcal{F}_3$	-	-	-	-	-	-	-	-	89	1	0	$7.35 \cdot 10^{-26}$

Table 6.9 provides a pairwise comparison between formulations, reporting wins, losses, ties, and the corresponding p -values from the Binomial test. These results reinforce the trends observed in the ranking analysis. Formulation $\mathcal{F}_2$ outperforms all others most frequently and is the only one that always dominates $\mathcal{F}_1$. Its superiority over $\mathcal{F}_3$ and $\mathcal{F}_4$ is further confirmed by extremely small p -values, whereas the comparison with $\mathcal{F}_4$, as expected, does not yield statistical evidence of a significant difference. The second row shows that $\mathcal{F}_4$ clearly dominates $\mathcal{F}_3$ and $\mathcal{F}_1$, despite tying or losing only rarely. The third row indicates that $\mathcal{F}_3$ also significantly outperforms $\mathcal{F}_1$, which wins only once.

Table 6.10: p -values for t -test and Wilcoxon tests

Form.	k	$\mathcal{F}_4$		$\mathcal{F}_1$		$\mathcal{F}_3$	
		t -test	Wilcoxon	t -test	Wilcoxon	t -test	Wilcoxon
$\mathcal{F}_1$	5	$1.52 \cdot 10^{-8}$	$1.20 \cdot 10^{-5}$	-	-	-	-
$\mathcal{F}_3$	5	0.02	0.03	$4.12 \cdot 10^{-7}$	$2.79 \cdot 10^{-8}$	-	-
$\mathcal{F}_2$	5	0.80	0.80	$2.66 \cdot 10^{-8}$	$1.12 \cdot 10^{-8}$	$4.09 \cdot 10^{-3}$	$1.69 \cdot 10^{-3}$
$\mathcal{F}_1$	10	$2.72 \cdot 10^{-4}$	$1.12 \cdot 10^{-8}$	-	-	-	-
$\mathcal{F}_3$	10	0.03	0.02	$5.31 \cdot 10^{-4}$	$1.12 \cdot 10^{-8}$	-	-
$\mathcal{F}_2$	10	0.68	0.51	$2.72 \cdot 10^{-4}$	$1.12 \cdot 10^{-8}$	0.01	$9.67 \cdot 10^{-3}$
$\mathcal{F}_1$	20	$1.43 \cdot 10^{-8}$	$1.12 \cdot 10^{-8}$	-	-	-	-
$\mathcal{F}_3$	20	$3.08 \cdot 10^{-3}$	$3.39 \cdot 10^{-3}$	$3.68 \cdot 10^{-8}$	$1.12 \cdot 10^{-8}$	-	-
$\mathcal{F}_2$	20	0.66	0.73	$2.09 \cdot 10^{-8}$	$1.12 \cdot 10^{-8}$	0.03	0.05
$\mathcal{F}_1$	All	$4.59 \cdot 10^{-12}$	$1.07 \cdot 10^{-15}$	-	-	-	-
$\mathcal{F}_3$	All	$1.36 \cdot 10^{-5}$	$7.03 \cdot 10^{-6}$	$2.52 \cdot 10^{-11}$	$1.06 \cdot 10^{-15}$	-	-
$\mathcal{F}_2$	All	0.83	0.72	$4.70 \cdot 10^{-12}$	$1.06 \cdot 10^{-15}$	$1.24 \cdot 10^{-4}$	$1.34 \cdot 10^{-5}$

Table 6.10 reports the p -values from paired t -tests and Wilcoxon tests for each value of k and for the full dataset. For $k = 5$ and 10, the only non-significant comparison is between $\mathcal{F}_2$ and $\mathcal{F}_4$, confirming that these two formulations behave similarly when k is small. For $k = 20$, $\mathcal{F}_2$ and $\mathcal{F}_4$ remain statistically indistinguishable, while the gaps between $\mathcal{F}_3$ and the two best formulations narrow. Over the entire test set, the only pair that remains not statistically different is again $\mathcal{F}_2 - \mathcal{F}_4$.

Overall, these statistical tests confirm that $\mathcal{F}_2$ and $\mathcal{F}_4$ form the top tier of k -sum formulations, while $\mathcal{F}_3$ and $\mathcal{F}_1$ are consistently inferior. Since $\mathcal{F}_2$ outperforms $\mathcal{F}_4$ slightly more often, especially for smaller values of k , we adopt $\mathcal{F}_2$ as the default formulation in the following experiments.

6.10.3 TS-ALNS

In this section, we present the outcome of the computational experiments conducted to evaluate TS-ALNS. These tests are conducted over Set B, using only the subset of instances with

a different underlying VRP graph. TS-ALNS random number generator is configured with a fixed seed, in this way different runs always produce the same output value.

Table 6.11: TS-ALNS results and computational time with $I = 10, 25, 50, 100$

Name	Map	I 10		I 25		I 50		I 100	
		Obj.	Time	Obj.	Time	Obj.	Time	Obj.	Time
C101	1	856.03	1.19	854.72	3.06	854.25	6.25	852.55	12.54
C101	2	918.89	1.37	918.89	3.45	912.97	6.50	912.97	12.97
C101	3	884.50	1.12	883.67	2.82	868.38	5.83	866.95	11.67
C101	4	868.59	1.24	868.43	2.90	868.43	5.77	868.43	11.60
C101	5	864.44	1.18	861.24	3.11	861.24	6.11	861.24	11.99
R101	1	952.08	1.43	950.93	3.21	944.09	6.53	944.09	12.99
R101	2	1002.78	0.98	1001.31	2.58	996.17	5.25	969.52	9.98
R101	3	1005.30	1.11	1005.30	3.10	1005.29	6.03	997.64	12.24
R101	4	1015.48	1.37	1010.91	3.53	1010.91	6.92	1010.91	13.64
R101	5	1011.43	1.46	1011.43	3.52	1011.42	7.06	976.78	13.71
RC101	1	1109.97	1.03	1094.70	2.63	1064.58	5.36	1064.58	11.13
RC101	2	1142.55	1.31	1142.54	3.49	1141.30	6.90	1141.30	13.52
RC101	3	1137.65	1.25	1137.65	3.31	1121.90	6.85	1113.87	13.11
RC101	4	1155.21	1.33	1155.21	3.36	1143.48	6.56	1142.47	12.95
RC101	5	1108.47	1.14	1108.47	2.77	1102.75	5.51	1102.75	11.23

Table 6.12: TS-ALNS results and computational time with $I = 200, 300, 400$

Name	Map	I 200		I 300		I 400	
		Obj.	Time	Obj.	Time	Obj.	Time
C101	1	852.55	25.18	852.55	32.85	852.55	44.38
C101	2	885.42	25.58	883.74	38.44	883.74	51.38
C101	3	866.95	23.01	866.95	35.16	866.95	46.31
C101	4	868.43	23.03	865.93	34.81	865.93	46.75
C101	5	860.00	23.85	857.33	35.73	857.33	48.04
R101	1	944.09	25.47	944.09	38.35	944.09	49.33
R101	2	969.52	19.48	968.94	28.67	968.52	37.91
R101	3	993.05	25.05	993.05	38.13	993.05	51.00
R101	4	1010.91	27.50	1010.91	41.47	1010.91	55.94
R101	5	975.63	27.42	975.63	40.23	975.54	53.11
RC101	1	1064.58	22.06	1064.58	33.34	1064.58	43.09
RC101	2	1112.75	26.92	1097.52	40.37	1085.87	54.37
RC101	3	1089.95	25.77	1089.95	38.74	1082.25	52.48
RC101	4	1142.47	25.03	1142.47	32.66	1142.47	37.13
RC101	5	1102.75	22.69	1102.72	33.89	1102.72	45.04

Table 6.11 and Table 6.12 report the performance of TS-ALNS for different values of the number of outer iterations (I). The first table considers short runs ($I = 10, 25, 50, 100$), whereas the second table reports longer runs ($I = 200, 300, 400$). Both tables use the same layout: for each instance and each value of I , we report the objective value obtained (column *Obj.*) and

the corresponding computational time in seconds (column *Time*).

Overall, TS-ALNS reaches its best average solution at $I = 210$ for instances C101 (average time 26.6 seconds), at $I = 215$ for R101 (25.2 seconds), and at $I = 250$ for RC101 (31.8 seconds). Averaged over all instances, the best solution typically appears around $I = 228$ with a runtime of 27.9 seconds. However, the improvements obtained by increasing I are relatively small: from $I = 10$ to $I = 400$, the average improvement is only 1.47% for C101, 1.89% for R101, and 3.11% for RC101, with an overall average of 2.16%. These results indicate that TS-ALNS is able to produce high-quality solutions even with a modest number of outer iterations.

Since the time limit of Algorithm 6.4 is global and must include the TS-ALNS runtime, it is essential to select a value of I that balances solution quality and running time. To select a suitable value of I , we analyze the marginal improvement obtained by increasing the number of outer iterations. For each instance and each value of I , we compute the normalized improvement $(obj_{t-1} - obj_t)/(t - t_{-1})$, which measures the reduction in routing cost per unit of additional computation time. Figure 6.22 reports these values for all 15 VRP instances.

Across all instance types (C101, R101, and RC101), the curves show that the largest improvements occur within the first 100 outer iterations. In several cases the curves flatten even earlier, around 50 iterations, while in others they decline more gradually and stabilize only after approximately 200 iterations. Overall, the common trend is that marginal gains drop sharply after $I = 100$, with subsequent iterations contributing little relative to their computational cost. Given that substantial progress is consistently achieved before $I = 100$ and that both earlier and later flattening behavior point to diminishing returns beyond this threshold, we adopt $I = 100$ as an effective compromise between solution quality and computational efficiency, leaving more time available for SAKS in the subsequent optimization phase.

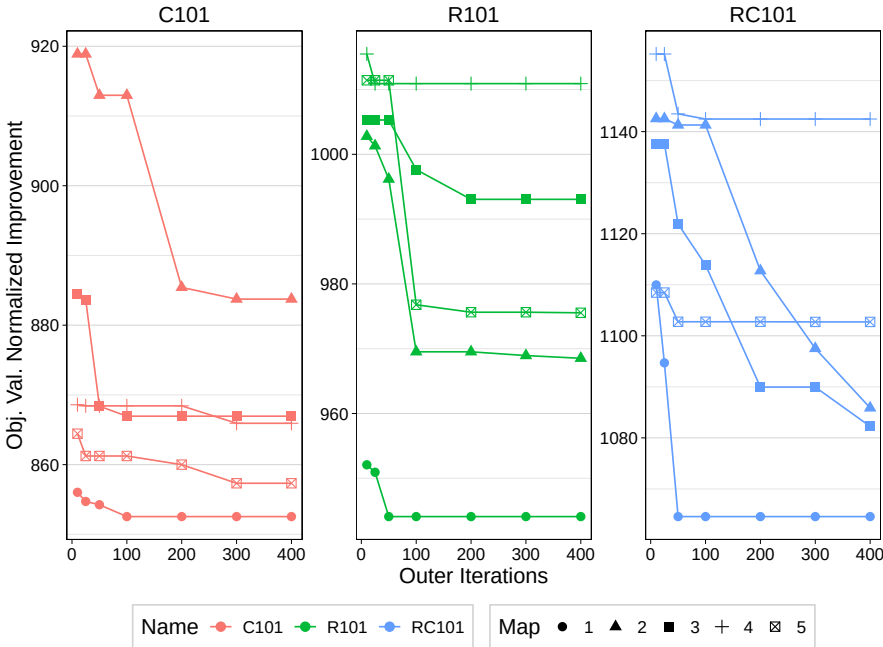


Figure 6.22: Regression of the normalized improvements

Table 6.13 reports on the comparison between TS-ALNS (configured to run for 100 outer

iterations) and Gurobi (GRB) solving each VRP instance with a time limit of 1 hour. Columns

Table 6.13: Gurobi results in 1 hour compared to TS-ALNS

Name	Map	TS	GRB	Diff.%	Gap%
C101	1	852.55	864.19	-1.37%	10.87%
C101	2	912.97	874.68	4.19%	10.79%
C101	3	866.95	865.97	0.11%	10.31%
C101	4	868.43	853.30	1.74%	9.46%
C101	5	861.24	848.29	1.50%	10.21%
R101	1	944.09	940.92	0.34%	4.30%
R101	2	969.52	963.88	0.58%	5.77%
R101	3	997.64	957.22	4.05%	5.58%
R101	4	1010.91	1010.88	0.00%	8.85%
R101	5	976.78	971.75	0.51%	5.15%
RC101	1	1064.58	1064.54	0.00%	6.55%
RC101	2	1141.30	1086.53	4.80%	8.89%
RC101	3	1113.87	1081.04	2.95%	8.52%
RC101	4	1142.47	1104.78	3.30%	10.14%
RC101	5	1102.75	1074.73	2.54%	8.58%

TS and **GRB** report the objective values obtained by TS-ALNS and Gurobi, respectively, while **Diff.%** gives their percentage difference. The last column, **Gap%**, shows Gurobi's MIP gap after one hour of computation.

Overall, Gurobi produces slightly better solutions, but the difference is small: the average gap between TS-ALNS and Gurobi is below 2%, with median values even smaller (1.50%) and always under 5% across all instance types. Since TS-ALNS finds these solutions in only a few seconds compared with Gurobi's one-hour limit, its performance can be considered quite strong. Gurobi's MIP gaps remain nonzero for all instances, ranging from about 4% to over 12%, confirming that even one hour of computation is insufficient to fully close the search tree. This further highlights the value of TS-ALNS: it delivers near-optimal solutions very quickly, while Gurobi struggles to significantly improve them within the available time.

6.10.4 Gurobi performance on the k -sum formulation

In this section we analyze Gurobi's performance on the 75 instances when optimizing f_2 with $k \in \{5, 10, 20\}$ and imposing an ϵ -constraint on f_1 (allowing at most a 10% worsening from the value returned by TS-ALNS with 100 outer iterations).

Table 6.14: Gurobi k-sum results

Name	k	Q_{25}			Avg.			Q_{50}			Q_{75}			Q_{90}		
		Obj.	Gap	Impr.	Obj.	Gap	Impr.	Obj.	Gap	Impr.	Obj.	Gap	Impr.	Obj.	Gap	Impr.
C101	5	195.00	70.65%	0.00%	223.88	70.92%	10.21%	219.00	75.44%	0.00%	236.00	81.95%	0.00%	303.20	84.54%	56.56%
C101	10	251.00	71.47%	0.00%	371.68	71.96%	12.80%	366.00	77.49%	0.00%	401.00	81.84%	20.00%	523.80	84.43%	46.38%
C101	20	554.00	75.28%	0.00%	627.84	74.24%	8.10%	620.00	76.11%	0.00%	707.00	81.32%	0.00%	880.20	84.52%	40.18%
R101	5	198.00	53.04%	0.00%	251.76	59.99%	6.62%	241.00	62.17%	0.00%	300.00	69.44%	0.00%	318.00	75.36%	24.00%
R101	10	334.00	55.94%	0.00%	427.00	59.43%	7.49%	417.00	63.39%	0.00%	497.00	67.37%	0.00%	554.80	73.39%	36.81%
R101	20	527.00	46.37%	0.00%	672.44	55.30%	11.29%	640.00	61.56%	0.00%	802.00	66.75%	30.66%	900.80	71.18%	39.62%
RC101	5	224.00	66.76%	0.00%	251.96	70.72%	6.65%	252.00	73.25%	0.00%	286.00	76.48%	0.00%	311.20	79.99%	32.22%
RC101	10	353.00	60.09%	0.00%	394.68	67.48%	13.00%	416.00	71.71%	0.00%	458.00	75.32%	30.24%	505.00	79.28%	48.54%
RC101	20	642.00	65.48%	0.00%	699.52	68.99%	7.03%	717.00	70.43%	0.00%	780.00	76.23%	0.00%	816.80	80.44%	30.90%

Table 6.14 reports aggregated statistics for each instance group and each value of k . For each block of columns, we show the objective value, the MIP gap, and the improvement of the incumbent with respect to the initial solution, evaluated at the 25th percentile, mean, median, 75th percentile, and 90th percentile. Overall, Gurobi exhibits slow convergence and remains far from optimality on these models. The MIP gaps remain very large across all configurations: the minimum gaps are already high (20.3% for C101, 15.4% for R101, and 35.8% for RC101), while the average gaps rise to 72.4%, 58.2%, and 69.1%, respectively. In all groups, the mean gap is significantly lower than the median, indicating that only a few instances have relatively small gaps, while the majority exhibit gaps close to or above 70%. The solver rarely improves the initial solution provided by TS-ALNS: only 14, 15, and 18 improved incumbents are observed for R101, C101, and RC101, respectively (i.e., fewer than 21% of all cases). Improvements do not correlate with the value of k (11, 19, and 17 improvements for $k = 5, 10, 20$). When improvements do occur, they are typically large: average improvements exceed 6.5% in all groups, and the top 10% exceed 24%.

These results indicate that improvements are almost exclusively found by Gurobi’s heuristics, since the branch-and-cut process rarely closes the gap. At the same time, the heuristics are not sufficiently effective to reliably improve the initial solution. Overall, the combination of large MIP gaps and few incumbent improvements confirms the need for a dedicated heuristic approach to solve these instances effectively.

6.10.5 SAKS performance

In this section we evaluate the performance of the proposed SAKS algorithm and compare its results with those obtained by Gurobi. To assess the robustness of SAKS with respect to its main design parameters, we tested the algorithm under two configurations, summarized in Table 6.15. These configurations reflect the outcome of a parameter-tuning phase aimed at identifying suitable ranges for the key KS hyperparameters. Configuration 1 was manually designed after several preliminary experiments, whereas Configuration 2 was generated through automatic hyperparameter tuning using Optuna (Akiba et al., 2019), a widely adopted framework for algorithmic parameter optimization. The tuning phase was conducted on the instances of Set A, whereas SAKS was evaluated on instances of Set B to avoid overfitting.

Table 6.15: Configuration Parameters

Conf.	nb_{min}	nb^{max}	tl_{min}	tl^{max}
1	5	30	15	40
2	9	26	15	42

For each configuration, we then solved all 75 benchmark instances in Set B with $k = 5, 10$, and 20, using a time limit of 10 minutes per run. In total, this resulted in 450 SAKS executions.

Preliminary tests, not presented here for brevity, allows us to identify good threshold values for `ComputeBucketNumberScore` and `ComputeTimeLimitScore` functions used in SAKS. Their values are reported in Table 6.16. Keep in mind that γ_x thresholds identify if the problem is too hard, whereas ρ_x identify if the problem is too easy.

Table 6.17 reports aggregated performance statistics for the two SAKS configurations. The first column identifies the instance class, the second the value of k , and the remaining columns present the 25th percentile, average, median, 75th percentile, and 90th percentile for each configuration. The results indicate that the two configurations perform very similarly.

Table 6.16: SAKS Threshold values

Param.	Value	Formula
γ_1	0.3	$\text{MipGap_max} > \gamma_1$
γ_2	0.2	$\text{MipGap_max} - \text{MipGap_min} > \gamma_2$
γ_3	0.1	$\text{MipGap_avg} > \gamma_3$
ρ_1	0.001	$\text{MipGap_max} < \rho_1$
ρ_2	0.1	$\text{MipGap_max} - \text{MipGap_min} < \rho_2$
ρ_3	0.001	$\text{MipGap_avg} < \rho_3$

Table 6.17: KS results with different configurations

Name	k	Q_{25}		Avg.		Q_{50}		Q_{75}		Q_{90}	
		Conf. 1	Conf. 2	Conf. 1	Conf. 2	Conf. 1	Conf. 2	Conf. 1	Conf. 2	Conf. 1	Conf. 2
C101	5	91.00	98.00	109.60	110.88	112.00	109.00	126.00	126.00	135.40	144.40
C101	10	165.00	174.00	195.68	200.28	198.00	199.00	218.00	225.00	243.20	245.60
C101	20	299.00	311.00	348.16	347.24	342.00	347.00	380.00	380.00	437.00	421.60
R101	5	146.00	135.00	158.76	160.08	156.00	149.00	171.00	181.00	187.40	214.00
R101	10	254.00	256.00	289.40	295.76	287.00	295.00	326.00	325.00	353.40	349.20
R101	20	422.00	433.00	493.48	489.84	504.00	498.00	536.00	533.00	620.40	580.40
RC101	5	132.00	141.00	157.84	163.32	153.00	148.00	168.00	170.00	192.20	221.00
RC101	10	252.00	246.00	287.60	294.36	273.00	295.00	311.00	323.00	361.00	357.60
RC101	20	452.00	468.00	500.48	512.00	490.00	497.00	539.00	552.00	599.40	603.40

Only in one case, RC101 with $k = 5$, the percentage gap between the best and worst configuration exceeds 3%. On average, however, the gap remains even smaller: 1.26% for C101, 1.25% for R101, and 2.7% for RC101, with minimum differences consistently below 1%. A further observation is that the performance difference between configurations decreases as k increases. Such an observation is supported by the Pearson's correlation coefficient between the gap and the value of k taking value -0.37, indicating a weak negative correlation between them. It is also easy to rationalize this: Since higher values of k lead to more challenging instances, the specific configuration appears to have a decreasing influence on the quality of the solution.

Table 6.18: Configuration ranking

Inst.	k	Conf. 1	Conf. 2	GRB
C101	5	1.58	1.74	2.68
R101	5	1.64	1.60	2.76
RC101	5	1.38	1.70	2.92
C101	10	1.48	1.76	2.76
R101	10	1.66	1.66	2.68
RC101	10	1.52	1.96	2.52
C101	20	1.60	1.64	2.76
R101	20	1.84	1.68	2.48
RC101	20	1.32	2.00	2.68
C101	-	1.55	1.71	2.73
R101	-	1.71	1.65	2.64
RC101	-	1.41	1.89	2.71
All	-	1.56	1.75	2.69

To compare the two SAKS configurations, Table 6.18 reports their statistical ranking, computed following the same procedure used in Table 6.8. Recall that lower ranking values correspond to better performance. The first block shows, for each instance group and each value of k , the average rank of Configuration 1, Configuration 2, and Gurobi (GRB). The second block aggregates the ranks over all k within each instance group, while the last row reports the ranking aggregated over all instances and all k . The results show that both SAKS configurations consistently outperform Gurobi, which always attains the highest (worst) average rank. Between the two configurations, performance differences are small, but Configuration 1 tends to achieve the best overall ranking (1.56), followed closely by Configuration 2 (1.75). Examining the instance- k combinations, each configuration is best at least once, and neither configuration is consistently superior across all cases. The narrow spread of ranking values confirms that SAKS performance is largely robust to parameter choices.

Table 6.19: Binomial tests

Conf.	2				GRB			
	Wins	Losses	Tot.	p -val.	Wins	Losses	Tot.	p -val.
1	132	88	220	1.82×10^{-3}	190	35	225	2.85×10^{-27}
2	-	-	-	-	191	34	225	5.19×10^{-28}

Table 6.19 reports pairwise Binomial tests comparing the two SAKS configurations and Gurobi (GRB). For each comparison, the table shows how many times the configuration on the row outperforms the one on the column (Wins), how many times it is outperformed (Losses), the total number of comparisons, and the corresponding p -value. The Binomial test assesses whether the probability of winning differs significantly from 0.5; small p -values therefore indicate that one method consistently outperforms the other. The comparison between Configurations 1 and 2 shows that Configuration 1 wins more often (132 vs. 88), and the very small p -value confirms that this advantage is statistically significant. When comparing SAKS to Gurobi, both configurations clearly outperform the solver: Configuration 1 wins 190 times against 35 losses, and Configuration 2 wins 191 times against 34 losses, with both comparisons yielding p -values equal to zero up to numerical precision.

Table 6.20 reports the p -values of paired Wilcoxon and paired t -tests comparing Configuration 1 with Configuration 2 and with Gurobi (GRB) for each instance group. These tests evaluate whether the distributions of objective values produced by two methods differ significantly. The comparison between Configurations 1 and 2 shows a heterogeneous behavior across instance groups. For C101 and R101, the p -values are generally above 0.05, indicating no statistically significant difference between the two settings. In contrast, for the RC101 instances, especially for $k = 10$ and $k = 20$, the tests return p -values below 0.05, showing that Configuration 1 performs significantly better than Configuration 2 on these harder instances. The comparisons with Gurobi are unambiguous: every test returns $p=0.00$ (up to numerical precision), confirming that both SAKS configurations consistently and significantly outperform the MILP solver across all instance classes. Overall, these results indicate that while Configurations 1 and 2 perform similarly on most instances, their differences become statistically relevant on more challenging RC101 cases. Both configurations, however, dominate Gurobi by a wide statistical margin.

In Table 6.21 we present the gap between KS and Gurobi. These statistics, cumulative over KS configuration, are presented for each instance group, first column, and value of k , second column. The third column presents the 25th percentile, the fourth presents the

Table 6.20: Wilcoxon tests

Conf.	Name	Conf. 2		GRB	
		W	T	W	T
1	C101	0.50	0.44	7.69×10^{-6}	2.48×10^{-6}
2	C101	-	-	7.69×10^{-6}	2.48×10^{-6}
1	C101	0.06	0.06	2.50×10^{-6}	1.05×10^{-6}
2	C101	-	-	2.50×10^{-6}	1.12×10^{-6}
1	C101	0.57	0.88	2.50×10^{-6}	1.75×10^{-8}
2	C101	-	-	2.50×10^{-6}	1.75×10^{-8}
1	R101	0.84	0.75	1.15×10^{-4}	2.01×10^{-5}
2	R101	-	-	1.15×10^{-4}	1.86×10^{-5}
1	R101	0.56	0.28	7.69×10^{-6}	1.59×10^{-5}
2	R101	-	-	7.69×10^{-6}	1.59×10^{-5}
1	R101	0.77	0.60	6.23×10^{-4}	1.38×10^{-4}
2	R101	-	-	1.91×10^{-4}	1.38×10^{-4}
1	RC101	0.09	0.07	3.58×10^{-7}	3.30×10^{-9}
2	RC101	-	-	3.33×10^{-5}	4.23×10^{-8}
1	RC101	0.02	0.04	1.55×10^{-3}	7.33×10^{-6}
2	RC101	-	-	1.55×10^{-3}	8.94×10^{-6}
1	RC101	1.50×10^{-3}	6.90×10^{-4}	1.97×10^{-5}	3.70×10^{-7}
2	RC101	-	-	2.47×10^{-5}	6.77×10^{-7}

Table 6.21: Gaps between KS and Gurobi

Name	k	Q_{25}	Avg.	Q_{50}	Q_{75}	Q_{90}
C101	5	-56.66%	-43.83%	-51.50%	-42.13%	2.37%
C101	10	-54.16%	-40.41%	-45.69%	-36.44%	1.32%
C101	20	-51.36%	-40.39%	-44.59%	-38.72%	1.20%
R101	5	-42.72%	-31.86%	-38.17%	-28.52%	7.17%
R101	10	-40.52%	-28.07%	-32.69%	-23.04%	3.31%
R101	20	-35.21%	-22.04%	-28.46%	1.19%	11.12%
RC101	5	-47.35%	-34.20%	-39.09%	-23.25%	-9.23%
RC101	10	-38.74%	-23.65%	-30.77%	-1.41%	4.19%
RC101	20	-39.00%	-25.25%	-30.85%	-13.33%	3.10%

average, and the fifth shows the median case. Finally the last two column present 75th and 90th percentile respectively.

This table show that in the average and median case, KS outperforms Gurobi by a large margin in all instance groups and all values of k . On the other side, watching column Q_{90} it is possible to observe that in the best cases for Gurobi, it can outperform KS.

Table 6.22 reports the gap between KS and Gurobi when Gurobi at least as good as one of the configurations. The table layout is the same as in Table 6.21.

From this table it is possible to observe that the average and median columns, when Gurobi outperforms KS, this happens by a small margin. Ignoring the outlier of R101 $k = 5$, the gap stay consistently under 10% both in the median case and in the average one as well. Moreover it is possible to notice that these gaps are dependent on instance group. In fact,

Table 6.22: Gaps between KS and Gurobi Best Solutions

Name	k	Q_{25}	Avg.	Q_{50}	Q_{75}	Q_{90}
C101	5	2.08%	8.58%	4.59%	12.62%	24.76%
C101	10	2.75%	6.12%	4.96%	8.51%	11.80%
C101	20	3.00%	5.26%	3.78%	7.01%	10.20%
R101	5	9.04%	15.93%	13.73%	21.35%	25.84%
R101	10	1.59%	9.67%	8.86%	15.35%	20.77%
R101	20	4.38%	8.10%	8.59%	12.77%	14.53%
RC101	5	1.85%	2.22%	2.22%	2.59%	2.81%
RC101	10	0.36%	2.45%	2.62%	5.60%	6.76%
RC101	20	2.58%	5.00%	5.56%	7.09%	8.34%

the largest gaps for RC101 are comparable with the average ones for the other groups.

6.11 Conclusions

In this chapter, we introduced and analyzed a novel bi-objective extension of the Vehicle Routing Problem in the context of last-mile delivery. The proposed formulation simultaneously minimizes the routing time and the congestion on the most densely populated streets. The second objective is defined through the concept of k -equity, measured as the average impact on the k most affected streets in a delivery solution. The impact of a street is quantified as the product of the number of residents living there and the number of times a vehicle traverses it. The model relies on a multigraph representation that allows multiple alternative paths between pairs of nodes, thus enabling different trade-offs between routing duration and population exposure. An extensive sensitivity analysis was performed on a large set of instances derived from real-world data from the cities of Charleston and San Francisco, USA. The computational results demonstrate that significant reductions in population impact can be obtained with only marginal increases in routing time. The study provides several rules of thumb that can assist decision makers in designing policies for last-mile delivery that are both efficient and equitable. As a direction for future research, we plan to refine the estimation of population impact by incorporating temporal variations, allowing routing strategies to adapt to fluctuations in exposure throughout the day.

Chapter 7

Conclusions

In this thesis, we have studied two classes of Last Mile Delivery (LMD) Problems concerning alternative multi-modal delivery methods and Fair Vehicle Routing Problem (VRP). Using compact Mixed Integer Linear Programming (MILP) formulation we run comprehensive sensitivity analysis on realistic instances to draw useful insights for decision makers in public and private sector. Precisely, we formulate single-objective MILPs to address complex decision making process, and bi-objective problems to address conflicting or intertwined decision. Then, we propose, and validate, exact and heuristic methods to tackle large scale instances efficiently. Proposed algorithms prove themselves efficient and effective in solving hard instances, outperforming state-of-the-art solver terms of solution quality and computational time.

The computational analysis presented throughout Chapters 2 to 6 provides a comprehensive assessment of both modeling and algorithmic choices. Chapter 2 shows that when installing lockers in public transport station, in order to facilitate a public transport crowdshipping system, under fixed and variable budget constraint, it is possible to efficiently locate them in such a way to not only maximize the total customers satisfaction, but also network coverage, to provide service to customers in sparsely populated areas. In Chapter 3, we shows that given a large enough number of crowdshippers, it is possible to prove a high-quality delivery service that is both cheaper and more environmentally friendly than a traditional one. We also show that a compact MILP model is incapable of handling realistically sized instances. Therefore, we propose a tailored Adaptive Large Neighborhood Search metaheuristic to handle these instances efficiently. Chapter 4 reports the results of a high-performance Branch-and-Price algorithm for the problem introduced in Chapter 3, which illustrate that it is possible to solve to optimality very large scale instance in short computational time. Moreover, results show that a major contributing factor in the instance complexity is the number of backup parcels in the optimal solution. In Chapter 5, we provide computational results, run on real-world base instances, showing that it is possible to re-route traffic away from certain city districts during peak-hours without increasing too much the total routing cost. Finally, Chapter 6 shows that it is possible to steer heavy-vehicular traffic away densely populated streets without increasing to much the total routing cost. This is especially true when more than one path is allowed between two customers. Moreover, the proposed Kernel Search algorithm outperforms Gurobi in large scale instances, allowing to apply such a system in a real-world applications.

In conclusion, this thesis show, in Part I, that it is possible to reduce environmental im-

pacts with out a sacrifice in delivery efficiency, and, in Part II, that it is possible to reduce disturbance imposed to residents without resorting to excessively expensive delivery plans. Future avenues of research, besides the ones highlighted at the end of each chapter, include the integration of the two proposed paradigms into one, where disturbance, costs, and environmental impacts are all taken into account by one method combining alternative delivery methods with disturbance minimization.

Bibliography

- Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- Enrico Angelelli, Renata Mansini, and M Grazia Speranza. Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers & Operations Research*, 37(11): 2017–2026, 2010.
- Enrico Angelelli, Renata Mansini, and M Grazia Speranza. Kernel search: A new heuristic framework for portfolio selection. *Computational Optimization and Applications*, 51(1): 345–361, 2012.
- Brittany Antonczak, Tammy M Thompson, Mindi W DePaola, and Gregory Rowangould. 2020 near-roadway population census, traffic exposure and equity in the united states. *Transportation Research Part D: Transport and Environment*, 125:103965, 2023.
- Ali Aouad and Danny Segev. The ordered k-median problem: surrogate models and approximation algorithms. *Mathematical Programming*, 177(1):55–83, 2019.
- Claudia Archetti, Martin Savelsbergh, and M Grazia Speranza. The vehicle routing problem with occasional drivers. *European Journal of Operational Research*, 254(2):472–480, 2016.
- Claudia Archetti, Natashaia Boland, and M Grazia Speranza. A matheuristic for the multivehicle inventory routing problem. *INFORMS Journal on Computing*, 29(3):377–387, 2017.
- Claudia Archetti, Gianfranco Guastaroba, Diana L Huerta-Muñoz, and M Grazia Speranza. A kernel search heuristic for the multivehicle inventory routing problem. *International Transactions in Operational Research*, 28(6):2984–3013, 2021a.
- Claudia Archetti, Francesca Guerriero, and Giusy Macrina. The online vehicle routing problem with occasional drivers. *Computers & Operations Research*, 127:105144, 2021b. ISSN 0305-0548.
- Alp M. Arslan, Niels Agatz, Leo Kroon, and Rob Zuidwijk. Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers. *Transportation Science*, (1): 222–235, 2019. Publisher: INFORMS.
- Mohammad Asghari, S Mohammad J Mirzapour Al-e, et al. Green vehicle routing problem: A state-of-the-art review. *International Journal of Production Economics*, 231:107899, 2021.

- Lucas Assunção and Andréa Cynthia Santos. The robust steiner team orienteering problem with decreasing priorities under budgeted uncertainty. *Operations Research Perspectives*, page 100344, 2025.
- Roberto Baretto, Thierry Garaix, and Xiaolan Xie. A branch-and-price-and-cut algorithm for operating room scheduling under human resource constraints. *Computers & Operations Research*, 152:106136, 2023.
- Martin Behnke, Thomas Kirschstein, and Christian Bierwirth. A column generation approach for an emission-oriented vehicle routing problem on a multigraph. *European Journal of Operational Research*, 288(3):794–809, 2021.
- Tolga Bektaş. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219, 2006. ISSN 0305-0483.
- Tolga Bektaş and Gilbert Laporte. The pollution-routing problem. *Transportation Research Part B: Methodological*, 45(8):1232–1250, 2011.
- Luca Bertazzi, Bruce Golden, and Xingyin Wang. Min–max vs. min–sum vehicle routing: A worst-case analysis. *European Journal of Operational Research*, 240(2):372–381, 2015. ISSN 0377-2217.
- Victor Blanco, Justo Puerto, and Safae El Haj Ben Ali. Revisiting several problems and algorithms in continuous location with ℓ_r norms. *Computational Optimization and Applications*, 58(3):563–595, 2014.
- Víctor Blanco, Alfredo Marín, and Justo Puerto. Intra-facility equity in discrete and continuous p-facility location problems. *Computers & Operations Research*, 162:106487, 2024.
- Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017. ISSN 0198-9715.
- V. Bonomi, R. Mansini, and R. Zanotti. Last mile delivery with parcel lockers: evaluating the environmental impact of eco-conscious consumer behavior. *IFAC-PapersOnLine*, 55(5):72–77, 2022a.
- V. Bonomi, R. Mansini, and R. Zanotti. Last Mile Delivery with Parcel Lockers: evaluating the environmental impact of eco-conscious consumer behavior. *IFAC-PapersOnLine*, 55(5):72–77, 2022b.
- V Bonomi, R Mansini, and R Zanotti. Last mile delivery with parcel lockers: Evaluating the environmental impact of eco-conscious consumer behavior. *IFAC-PapersOnLine*, 55(5):72–77, 2022c.
- Valentina Bonomi, Renata Mansini, and Roberto Zanotti. Fairness in home healthcare: can patient-centered and nurse-centered measures concur to the same goals? *IFAC-PapersOnLine*, 55(10):3136–3141, 2022d.
- Valentina Bonomi, Renata Mansini, and Roberto Zanotti. Mediating governance goals with patients and nurses satisfaction: a multi-actor multi-objective problem including fairness. *International Journal of Production Research*, 62(10):3753–3780, 2024a.

- Valentina Bonomi, Renata Mansini, and Roberto Zanotti. Mediating governance goals with patients and nurses satisfaction: a multi-actor multi-objective problem including fairness. *International Journal of Production Research*, 62(10):3753–3780, 2024b.
- Valentina Bonomi, Daniele Manerba, Renata Mansini, and Roberto Zanotti. Optimizing attended home delivery: Multiple recovery options and customer availability profiles to face synchronization failures. *International Journal of Production Economics*, 279:109463, 2025. ISSN 0925-5273.
- Zuzana Borčinová. Kernel search for the capacitated vehicle routing problem. *Applied Sciences*, 12(22):11421, 2022.
- Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004.
- Nils Boysen, Stefan Schwerdfeger, and Felix Weidinger. Scheduling last-mile deliveries with truck-based autonomous robots. *European Journal of Operational Research*, 271(3): 1085–1099, 2018. ISSN 0377-2217.
- Nils Boysen, Stefan Fedtke, and Stefan Schwerdfeger. Last-mile delivery concepts: a survey from an operational research perspective. *Or Spectrum*, 43(1):1–58, 2021.
- Maurizio Bruglieri, Massimo Paolucci, and Ornella Pisacane. A matheuristic for the electric vehicle routing problem with time windows and a realistic energy consumption model. *Computers & Operations Research*, 157:106261, 2023.
- Giuseppe Bruno, Antonio Diglio, Carmela Piccolo, and Eduardo Picicelli. Solutions for sustainable last-mile delivery: Pick-up points location with customers' choice. *Research in Transportation Economics*, 113:101612, 2025.
- Heleen Buldeo Rai, Sara Verlinde, and Cathy Macharis. Shipping outside the box. Environmental impact and stakeholder analysis of a crowd logistics platform in Belgium. *Journal of Cleaner Production*, 202:806–816, 2018.
- Valentina Carbone, Aurélien Rouquet, and Christine Roussat. The rise of crowd logistics: a new way to co-create logistics value. *Journal of Business Logistics*, 38(4):238–252, 2017.
- F Carrabs, R Cerulli, R Mansini, D Serra, and C Sorigente. Hybridizing carousel greedy and kernel search: A new approach for the maximum flow problem with conflict constraints. *European Journal of Operational Research*, 324(2):414–435, 2025.
- Desiree M Carvalho and Maria CV Nascimento. A kernel search to the multi-plant capacitated lot sizing problem with setup carry-over. *Computers & Operations Research*, 100: 43–53, 2018.
- Desiree M Carvalho, Raf Jans, Silvio A de Araujo, and Diego J Fiorotto. A heuristic approach for the integrated production–transportation problem with process flexibility. *Computers & Operations Research*, 163:106502, 2024.
- Merve Seher Cebeci, Rodrigo Javier Tapia, Ali Nadi, Michiel de Bok, and Lóránt Tavasszy. Does Crowdshipping of Parcels Generate New Passenger Trips? Evidence from the Netherlands. *Transportation Research Record*, 2678(6):360–375, 2024.

- Cenpy. Cenpy, 2025. URL <https://cenpy-devs.github.io/cenpy/>.
- Rong Cheng, Andreas Fessler, Otto Nielsen, Allan Larsen, and Yu Jiang. Assessing the potential impacts of public transport-based crowdshipping: A case study in a central district of Copenhagen. *Frontiers of Engineering Management*, 07 2024.
- Stephanie Chevalier. Retail e-commerce sales worldwide from 2014 to 2027. <https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/>, 2024. [Online; accessed March 6, 2024].
- Geoff Clarke and John W Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.
- Teresa Corberán, Isaac Plana, José M Sanchis, and Paula Segura. The multidepot drone general routing problem with duration and capacity constraints. *International Transactions in Operational Research*, 32(6):3756–3779, 2025.
- Luciano Costa, Claudio Contardo, and Guy Desaulniers. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4):946–985, 2019.
- Jean-François Côté, Renata Mansini, and Alice Raffaele. Multi-period time window assignment for attended home delivery. *European Journal of Operational Research*, 316(1): 295–309, 2024a.
- Jean-François Côté, Renata Mansini, and Alice Raffaele. Multi-period time window assignment for attended home delivery. *European Journal of Operational Research*, 316(1): 295–309, 2024b.
- Said Dabia, Emrah Demir, and Tom Van Woensel. An exact approach for a variant of the pollution-routing problem. *Transportation Science*, 51(2):607–628, 2017.
- George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.
- George B Dantzig and Philip Wolfe. The decomposition algorithm for linear programs. *Econometrica: Journal of the Econometric Society*, pages 767–778, 1961.
- Iman Dayarian and Martin Savelsbergh. Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders. *Production and Operations Management*, 29(9): 2153–2174, 2020.
- Jérémy Decerle, Olivier Grunder, Amir Hajjam El Hassani, and Oussama Barakat. A memetic algorithm for multi-objective optimization of the home health care problem. *Swarm and evolutionary computation*, 44:712–727, 2019.
- Mauro Dell’Amico, Fabio Furini, and Manuel Iori. A branch-and-price algorithm for the temporal bin packing problem. *Computers & operations research*, 114:104825, 2020.
- Mary Angelique G Demetillo, Aracely Navarro, Katherine K Knowles, Kimberly P Fields, Jeffrey A Geddes, Caroline R Nowlan, Scott J Janz, Laura M Judd, Jassim Al-Saadi, Kang Sun, et al. Observing nitrogen dioxide air pollution inequality using high-spatial-resolution remote sensing measurements in Houston, Texas. *Environmental Science & Technology*, 54 (16):9882–9895, 2020.

- Mary Angelique G Demetillo, Colin Harkins, Brian C McDonald, Philip S Chodrow, Kang Sun, and Sally E Pusede. Space-based observational constraints on NO₂ air pollution inequality from diesel traffic in major us cities. *Geophysical Research Letters*, 48(17): e2021GL094333, 2021.
- Emrah Demir, Tolga Bektaş, and Gilbert Laporte. The bi-objective pollution-routing problem. *European journal of operational research*, 232(3):464–478, 2014.
- Emrah Demir, Yuan Huang, Sebastiaan Scholts, and Tom Van Woensel. A selected review on the negative externalities of the freight transportation: Modeling and pricing. *Transportation Research Part E: Logistics and Transportation Review*, 77, 05 2015.
- Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, 7:1–30, 2006.
- NYC Department of Transportation. Improving the efficiency of truck deliveries in nyc, 2019.
- Onur Derse and Tom Van Woensel. Integrated people and freight transportation: A literature review. *Future Transportation*, 4(4):1142–1160, 2024. ISSN 2673-7590.
- Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.
- Jacques Desrosiers, Marco Lübbecke, Guy Desaulniers, and Jean Bertrand Gauthier. *Branch-and-price*. GERAD, HÉC Montréal, 2024.
- Yael Deutsch and Boaz Golany. A parcel locker network as a solution to the logistics last mile problem. *International Journal of Production Research*, 56(1-2):251–261, 2018.
- DHL. Delivery innovations to reduce last-mile costs, 2023. URL <https://www.dhl.com/discover/en-nz/logistics-advice/logistics-insights/last-mile-delivery>.
- Qian Di, Yan Wang, Antonella Zanobetti, Yun Wang, Petros Koutrakis, Christine Choirat, Francesca Dominici, and Joel D Schwartz. Air pollution and mortality in the medicare population. *New England Journal of Medicine*, 376(26):2513–2522, 2017.
- Luigi Di Puglia Pugliese, Daniele Ferone, Giusy Macrina, Paola Festa, and Francesca Guerriero. The crowd-shipping with penalty cost function and uncertain travel times. *Omega*, 115:102776, 2023.
- Okan Dukkanci, Özlem Karsu, and Bahar Y Kara. Planning sustainable routes: Economic, environmental and welfare concerns. *European Journal of Operational Research*, 301(1): 110–123, 2022.
- Ece Naz Duman, Duygu Taş, and Bülent Çatay. A bidirectional branch-and-price algorithm with pulse procedure for the electric vehicle routing problem with flexible deliveries. *Transportation Research Part C: Emerging Technologies*, 165:104699, 2024.
- Matthias Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.
- Ralf Elbert and Johannes Rentschler. Freight on urban public transportation: A systematic literature review. *Research in Transportation Business & Management*, 45:100679, 2022. ISSN 2210-5395. doi: <https://doi.org/10.1016/j.rtbm.2021.100679>. Urban logistics: From research to implementation.

- European Commission. Sustainable and Smart Mobility Strategy – putting European transport on track for the future. Technical report, European Commission, 2020.
- European Environment Agency. The first and last mile — the key to sustainable urban transport, 2019.
- Aiyue Fei, Qian Hu, and Ying Liu. A branch-and-price algorithm for the single machine scheduling problem with periodic maintenance to minimize makespan. *Computers & Operations Research*, page 107214, 2025.
- Dominique Feillet. A tutorial on column generation and branch-and-price for vehicle routing problems. *4or*, 8(4):407–424, 2010.
- Andreas Fessler, Philip Cash, Mikkel Thorhauge, and Sonja Haustein. A public transport based crowdshipping concept: Results of a field test in denmark. *Transport Policy*, 134: 106–118, 2022. ISSN 0967-070X.
- Carlo Filippi and Francesca Plebani. Metro stations as crowd-shipping catalysts: An empirical and computational study. 2021.
- Carlo Filippi, Gianfranco Guastaroba, and Maria Grazia Speranza. A heuristic framework for the bi-objective enhanced index tracking problem. *Omega*, 65:122–137, 2016.
- Carlo Filippi, Włodzimierz Ogryczak, and M. Grazia Speranza. Bridging k-sum and CVaR optimization in MILP. *Computers & Operations Research*, 105:156–166, 2019. ISSN 0305-0548.
- Carlo Filippi, Gianfranco Guastaroba, Diana L Huerta-Muñoz, and Maria Grazia Speranza. A kernel search heuristic for a fair facility location problem. *Computers & Operations Research*, 132:105292, 2021.
- Travis Fried, Anne Victoria Goodchild, Ivan Sanchez-Diaz, and Michael Browne. Evaluating spatial inequity in last-mile delivery: a national analysis. *International Journal of Physical Distribution & Logistics Management*, 54(5):501–522, 2024.
- Jiali Fu and Erik Jenelius. Transport efficiency of off-peak urban goods deliveries: A Stockholm pilot study. *Case Studies on Transport Policy*, 6(1):156–166, 2018.
- Mikele Gajda, Nils Boysen, and Olivier Gallay. Impact of bot return policies in van-and-bot delivery systems. *International Journal of Production Research*, 62(17):6360–6379, 2024.
- Mikele Gajda, Olivier Gallay, Renata Mansini, and Filippo Ranza. Optimizing last-mile delivery through crowdshipping on public transportation networks. *Transportation Research Part C: Emerging Technologies*, 179:105250, 2025.
- Wenqi Gao, Zhixing Luo, and Houcai Shen. A branch-and-price-and-cut algorithm for time-dependent pollution routing problem. *Transportation Research Part C: Emerging Technologies*, 156:104339, 2023.
- Valerio Gatta, Edoardo Marcucci, Marialisa Nigro, Sergio Maria Patella, and Simone Serafini. Public transport-based crowdshipping for sustainable city logistics: Assessing economic and environmental impacts. *Sustainability*, 11(1):145, 2018.

- Valerio Gatta, Edoardo Marcucci, Marialisa Nigro, Sergio Maria Patella, and Simone Serafini. Public Transport-Based Crowdshipping for Sustainable City Logistics: Assessing Economic and Environmental Impacts. *Sustainability*, 11(1), 2019a.
- Valerio Gatta, Edoardo Marcucci, Marialisa Nigro, and Simone Serafini. Sustainable urban freight transport adopting public transport-based crowdshipping for b2c deliveries. *European Transport Research Review*, 11(1):1–14, 2019b.
- Valerio Gatta, Marialisa Nigro, and Simone Serafini. Sustainable urban freight transport adopting public transport-based crowdshipping for B2c deliveries. *European Transport Research Review*, 11, 12 2019c.
- Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.
- Fred Glover. Tabu search i. *ORSA J. Comput.*, 1:190–206, 1989.
- A. Gobbi, D. Manerba, R. Mansini, and R. Zanotti. A kernel search for a patient satisfaction-oriented nurse routing problem with time-windows. *IFAC-PapersOnLine*, 52(13):1669–1674, 2019a.
- Alessandro Gobbi, Daniele Manerba, Renata Mansini, and Roberto Zanotti. A kernel search for a patient satisfaction-oriented nurse routing problem with time-windows. *IFAC-PapersOnLine*, 52(13):1669–1674, 2019b.
- Alessandro Gobbi, Daniele Manerba, Renata Mansini, and Roberto Zanotti. Hybridizing adaptive large neighborhood search with kernel search: a new solution approach for the nurse routing problem with incompatible services and minimum demand. *International Transactions in Operational Research*, 30(1):8–38, 2023a.
- Alessandro Gobbi, Daniele Manerba, Renata Mansini, and Roberto Zanotti. Hybridizing adaptive large neighborhood search with kernel search: a new solution approach for the nurse routing problem with incompatible services and minimum demand. *International Transactions in Operational Research*, 30(1):8–38, 2023b.
- Alessandro Gobbi, Renata Mansini, Lorenzo Moreschini, and Filippo Ranza. Optimal locker placement for budget-constrained crowdshipping in public transport networks. *IFAC-PapersOnLine*, 59(10):3041–3046, 2025.
- Bruce L. Golden, Gilbert Laporte, and Éric D. Taillard. An adaptive memory heuristic for a class of vehicle routing problems with minmax objective. *Computers & Operations Research*, 24(5):445–452, 1997. ISSN 0305-0548.
- Amir-Mohammad Golmohammadi, Hamidreza Abedsoltan, Alireza Goli, and Irfan Ali. Multi-objective dragonfly algorithm for optimizing a sustainable supply chain under resource sharing conditions. *Computers & Industrial Engineering*, 187:109837, 2024.
- Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958. doi: 10.1090/S0002-9904-1958-10224-4.

- Kannan Govindan, Ahmad Jafarian, Roohollah Khodaverdi, and Kannan Devika. Two-echelon multiple-vehicle location-routing problem with time windows for optimization of sustainable supply chain network of perishable food. *International Journal of Production Economics*, 152:9–28, 2014.
- Philippe Grangier, Michel Gendreau, Fabien Lehuédé, and Louis-Martin Rousseau. An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, 254(1):80–91, 2016. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2016.03.040>.
- Gianfranco Guastaroba and Maria Grazia Speranza. Kernel search: An application to the index tracking problem. *European Journal of Operational Research*, 217(1):54–68, 2012.
- Gianfranco Guastaroba and Maria Grazia Speranza. A heuristic for bilp problems: the single source capacitated facility location problem. *European Journal of Operational Research*, 238(2):438–450, 2014.
- Gianfranco Guastaroba, M Savelsbergh, and Maria Grazia Speranza. Adaptive kernel search: A heuristic for solving mixed integer linear programs. *European Journal of Operational Research*, 263(3):789–804, 2017.
- SK Gupta and AP Punnen. k-Sum optimization problems. *Operations research letters*, 9(2): 121–126, 1990.
- Saïd Hanafi, Renata Mansini, and Roberto Zanotti. The multi-visit team orienteering problem with precedence constraints. *European journal of operational research*, 282(2):515–529, 2020.
- Jonas Hatzenbühler, Erik Jenelius, Gyöző Gidófalvi, and Oded Cats. Modular vehicle routing for combined passenger and freight transport. *Transportation Research Part A: Policy and Practice*, 173:103688, 2023.
- Petri Helo and Vinh V Thai. Logistics 4.0—digital transformation with smart connected tracking and tracing devices. *International Journal of Production Economics*, 275:109336, 2024.
- Vera C. Hemmelmayr, Jean-François Cordeau, and Teodor Gabriel Crainic. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12):3215–3228, 2012. ISSN 0305-0548. doi: <https://doi.org/10.1016/j.cor.2012.04.007>.
- José Holguín-Veras, Noel Pérez, Brenda Cruz, and John Polimeni. Effectiveness of financial incentives for off-peak deliveries to restaurants in Manhattan, New York. *Transportation Research Record*, 1966(1):51–59, 2006a.
- José Holguín-Veras, Qian Wang, Ning Xu, Kaan Ozbay, Mecit Cetin, and John Polimeni. The impacts of time of day pricing on the behavior of freight carriers in a congested urban area: Implications to road pricing. *Transportation Research Part A: Policy and Practice*, 40(9):744–766, 2006b.
- José Holguín-Veras, Kaan Ozbay, Alain Kornhauser, Matthew A. Brom, Shrisan Iyer, Wilfredo F. Yushimito, Satish Ukkusuri, Brandon Allen, and Michael A. Silas. Overall impacts of off-hour delivery programs in New York City Metropolitan Area. *Transportation Research Record*, 2238(1):68–76, 2011.

- Diana L Huerta-Muñoz, Claudia Archetti, Elena Fernández, and Federico Perea. The heterogeneous flexible periodic vehicle routing problem: mathematical formulations and solution algorithms. *Computers & Operations Research*, 141:105662, 2022.
- Wanchen Jie, Jun Yang, Min Zhang, and Yongxi Huang. The two-echelon capacitated electric vehicle routing problem with battery swapping stations: Formulation and efficient methodology. *European Journal of Operational Research*, 272(3):879–904, 2019. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2018.07.002>.
- Martin Joerss, Jürgen Schröder, Florian Neuhaus, Christoph Klink, and Florian Mann. Parcel delivery: the future of last mile. *Travel, Transport & Logistics*, 32, 2016.
- Michael Kahr. Determining locations and layouts for parcel lockers to support supply chain viability at the last mile. *Omega*, 113:102721, 2022.
- Halil Karlı and Rukiye Gizem Öztaş Karlı. Consumer resistance to parcel lockers: An innovation resistance theory perspective. *Cities*, 168:106501, 2026.
- Alex A Karner, Douglas S Eisinger, and Deb A Niemeier. Near-roadway air quality: synthesizing the findings from real-world data. *Environmental Science & Technology*, 44(14):5334–5344, 2010.
- Kerim U Kızıl and Barış Yıldız. Public transport-based crowd-shipping with backup transfers. *Transportation Science*, 57(1):174–196, 2023.
- Anne B Knol, Jeroen J de Hartog, Hanna Boogaard, Pauline Slottje, Jeroen P van der Sluijs, Erik Lebret, Flemming R Cassee, J Arjan Wardekker, Jon G Ayres, Paul J Borm, et al. Expert elicitation on ultrafine particles: likelihood of health effects and causal pathways. *Particle and Fibre Toxicology*, 6:1–16, 2009.
- Manolis N Kritikos and George Ioannou. The balanced cargo vehicle routing problem with time windows. *International Journal of Production Economics*, 123(1):42–51, 2010.
- David Lai, Yasel Costa, Emrah Demir, Alexandre M Florio, and Tom Van Woensel. The pollution-routing problem with speed optimization and uneven topography. *Computers & Operations Research*, 164:106557, 2024.
- Raj M Lal, Anu Ramaswami, and Armistead G Russell. Assessment of the Near-Road (monitoring) network including comparison with nearby monitors within US cities. *Environmental Research Letters*, 15(11):114026, 2020.
- Ailsa H. Land and Alison G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960. doi: 10.2307/1910129.
- Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992. ISSN 0377-2217.
- Tzong-Ru Lee and Ji-Hwa Ueng. A study of vehicle routing problems with load-balancing. *International Journal of Physical Distribution & Logistics Management*, 29(10):646–657, 1999.
- Clément Lemardelé, Miquel Estrada, Laia Pagès, and Mònika Bachofner. Potentialities of drones and ground autonomous delivery devices for last-mile logistics. *Transportation Research Part E: Logistics and Transportation Review*, 149:102325, 2021.

- Javier León, Justo Puerto, and Begoña Vitoriano. A risk-aversion approach for the multiobjective stochastic programming problem. *Mathematics*, 8(11):2026, 2020.
- Gonzalo Lera-Romero, Juan Jose Miranda Bront, and Francisco J Soullignac. A branch-cut-and-price algorithm for the time-dependent electric vehicle routing problem with time windows. *European Journal of Operational Research*, 312(3):978–995, 2024.
- Zhanxuan Li and Baicheng Li. The data-driven optimization of parcel locker locations in a transit co-modal system with ride-pooling last-mile delivery. *Applied Sciences*, 15(9):5217, 2025.
- Zhaojin Li, Ya Liu, and Zhen Yang. An effective kernel search and dynamic programming hybrid heuristic for a multimodal transportation planning problem with order consolidation. *Transportation Research Part E: Logistics and Transportation Review*, 152:102408, 2021.
- Yen-Hung Lin, Rajan Batta, Peter A Rogerson, Alan Blatt, and Marie Flanigan. A logistics model for emergency supply of critical items in the aftermath of a disaster. *Socio-Economic Planning Sciences*, 45(4):132–145, 2011.
- Yunhui Lin, Yuan Wang, Loo Hay Lee, and Ek Peng Chew. Profit-maximizing parcel locker location problem under threshold Luce model. *Transportation Research Part E: Logistics and Transportation Review*, 157:102541, 2022.
- Frederick W Lipfert and Ronald E Wyzga. On exposure and response relationships for health effects associated with exposure to vehicular traffic. *Journal of Exposure Science & Environmental Epidemiology*, 18(6):588–599, 2008.
- Yan Liu, Bin Guo, Chao Chen, He Du, Zhiwen Yu, Daqing Zhang, and Huadong Ma. Foodnet: Toward an optimized food delivery network based on spatial crowdsourcing. *IEEE Transactions on Mobile Computing*, 18(6):1288–1301, 2019. doi: 10.1109/TMC.2018.2861864.
- Bohao Ma, Chee-Chong Teo, and Yiik Diew Wong. Location analysis of parcel locker network: Effects of spatial characteristics on operational performance. *Transportation Research Part E: Logistics and Transportation Review*, 192:103776, 2024.
- Salma Makboul, Said Kharraja, Abderrahman Abbassi, and Ahmed El Hilali Alaoui. A multiobjective approach for weekly Green Home Health Care routing and scheduling problem with care continuity and synchronized services. *Operations Research Perspectives*, 12: 100302, 2024.
- Daniele Manerba, Renata Mansini, and Roberto Zanotti. Attended home delivery: reducing last-mile environmental impact by changing customer habits. *IFAC-PapersOnLine*, 51(5): 55–60, 2018a.
- Daniele Manerba, Renata Mansini, and Roberto Zanotti. Attended home delivery: Reducing last-mile environmental impact by changing customer habits. *IFAC-PapersOnLine*, 51(5): 55–60, 2018b.
- Daniele Manerba, Renata Mansini, Lorenzo Tomasetti, and Roberto Zanotti. Bi-objective optimization in steelmaking: Balancing scrap costs, energy consumption, and steel quality. *IFAC-PapersOnLine*, 59(10):1706–1711, 2025.

- Renata Mansini and Roberto Zanotti. Two-phase kernel search: An application to facility location problems with incompatibilities. In *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - Volume 1: ICORES*, pages 105–111, 2022.
- Renata Mansini, Filippo Ranza, and Roberto Zanotti. Towards sustainable last-mile delivery: Introducing an off-peak urban policy to mitigate environmental and social impacts. *IFAC-PapersOnLine*, 58(2):186–191, 2024. ISSN 2405-8963. 3rd IFAC Workshop on Integrated Assessment Modeling for Environmental Systems IAMES 2024.
- Piotr Matl, Richard F Hartl, and Thibaut Vidal. Workload equity in vehicle routing problems: A survey and analysis. *Transportation Science*, 52(2):239–260, 2018.
- C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer Programming Formulation of Traveling Salesman Problems. *J. ACM*, 7(4):326–329, October 1960. ISSN 0004-5411. doi: 10.1145/321043.321046.
- Seyed MJ Mirzapour Al-e-Hashem, Yacine Rekik, and Ebrahim Mohammadi Hoseinhajlou. A hybrid L-shaped method to solve a bi-objective stochastic transshipment-enabled inventory routing problem. *International Journal of Production Economics*, 209:381–398, 2019.
- Seyed Sina Mohri, Neema Nassir, Russell G Thompson, Patricia Sauri Lavieri, and Hadi Ghaderi. Crowd-shipping systems with public transport passengers: Operational planning. *Transportation Research Part E: Logistics and Transportation Review*, 194:103916, 2025.
- Sarbast Moslem and Francesco Pilla. A hybrid decision making support method for parcel lockers location selection. *Research in Transportation Economics*, 100:101320, 2023.
- Kianoush Mousavi, Sabrina Khan, Sabbir Saiyed, Glareh Amirjamshidi, and Matthew J. Roorda. Pilot off-peak delivery program in the region of Peel. *Sustainability*, 13(1):246, 2021.
- Jamal Abdul Nasir and Yong-Hong Kuo. Stochastic home care transportation with dynamically prioritized patients: An integrated facility location, fleet sizing, and routing approach. *Transportation Research Part B: Methodological*, 184:102949, 2024.
- Włodzimierz Ogryczak and Arie Tamir. Minimizing the sum of the k largest functions in linear time. *Information Processing Letters*, 85(3):117–122, 2003.
- Filippo Maria Ottaviani, Giovanni Zenezini, Alberto De Marco, and Antonio Carlin. Locating Automated Parcel Lockers (Apl) with known customers’ demand: a mixed approach proposal. *European Journal of Transport and Infrastructure Research*, 23(2):24–45, 2023.
- Zhiyuan Ouyang, Eric K.H. Leung, and George Q. Huang. Community logistics and dynamic community partitioning: A new approach for solving e-commerce last mile delivery. *European Journal of Operational Research*, 307(1):140–156, 2023.
- Pinar Ozyavas, Paul Buijs, Evrim Ursavas, and Ruud Teunter. Designing a sustainable delivery network with parcel locker systems as collection and transfer points. *Omega*, 131: 103199, 09 2024.
- Matthias W. Padberg and Giovanni Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987. doi: 10.1016/0167-6377(87)90002-2.

- Shichang Pan, Lele Zhang, Russell Thompson, and Hadi Ghaderi. A parcel network flow approach for joint delivery networks using parcel lockers. *International Journal of Production Research*, 59:1–26, 12 2020.
- Matteo Petris, Claudia Archetti, Diego Cattaruzza, Maxime Ogier, and Frédéric Semet. A tutorial on branch-price-and-cut algorithms. *4OR*, pages 1–52, 2025.
- David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.
- Diego Ponce, Justo Puerto, Federica Ricca, and Andrea Scozzari. Mathematical programming formulations for the efficient solution of the k-sum approval voting problem. *Computers & Operations Research*, 98:127–136, 2018.
- Jean-Yves Potvin and Jean-Marc Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.
- Sarah Powell, Ann Melissa Campbell, and Mojtaba Hosseini. Underground freight transportation for package delivery in urban environments. *arXiv preprint arXiv:2405.04618*, 2024.
- Justo Puerto, Antonio M Rodríguez-Chía, and Arie Tamir. Revisiting k-sum optimization. *Mathematical Programming*, 165:579–604, 2017.
- Ramin Raeesi and Konstantinos G Zografos. The multi-objective Steiner pollution-routing problem on congested urban road networks. *Transportation Research Part B: Methodological*, 122:457–485, 2019.
- Luigi Ranieri, Salvatore Digiesi, Bartolomeo Silvestri, and Michele Roccotelli. A review of last mile logistics innovations in an externalities cost reduction vision. *Sustainability*, 10(3):782, 2018.
- Grand View Research. Last-mile delivery market size, share & trends analysis report, 2023–2030, 2023. URL <https://www.grandviewresearch.com/horizon/outlook/last-mile-delivery-market-size/global>.
- Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.
- Munise Kübra Şahin and Hande Yaman. A branch and price algorithm for the heterogeneous fleet multi-depot multi-trip vehicle routing problem with time windows. *Transportation Science*, 56(6):1636–1657, 2022.
- Marc Saleh, Tufayel Chowdhury, James Vaughan, An Wang, Kianoush Mousavi, Matthew J. Roorda, and Marianne Hatzopoulou. Climate and air quality impacts of off-peak commercial deliveries. *Transportation Research Part D: Transport and Environment*, 109:103360, 2022.
- Martin Savelsbergh and Tom Van Woensel. 50th anniversary invited article—city logistics: Challenges and opportunities. *Transportation Science*, 50, 03 2016.

- Arianna Seghezzi, Chiara Siragusa, and Riccardo Mangiaracina. Parcel lockers vs. home delivery: a model to compare last-mile delivery cost in urban and rural areas. *International Journal of Physical Distribution & Logistics Management*, 52(3):213–237, 2022. ISSN 0960-0035. Publisher: Emerald Publishing Limited.
- Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.
- Veronica A Southerland, Susan C Anenberg, Maria Harris, Joshua Apte, Perry Hystad, Aaron van Donkelaar, Randall V Martin, Matt Beyers, and Ananya Roy. Assessing the distribution of air pollution health risks within cities: a neighborhood-scale analysis leveraging high-resolution data sets in the Bay Area, California. *Environmental health perspectives*, 129(3): 037006, 2021.
- Philipp Speckenmeyer, Constanze Hilmer, Gerhard Rauchecker, and Guido Schryen. Parallel branch-and-price algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *Computers & Operations Research*, 173: 106804, 2025.
- Statista. ecommerce - worldwide. <https://www.statista.com/outlook/emo/ecommerce/worldwide?currency=USD>, 2024. Accessed: April 22, 2024.
- Iván Sánchez-Díaz, Peter Georén, and Märta Brolinson. Shifting urban freight deliveries to the off-peak hours: a review of theory and practice. *Transport Reviews*, 37(4):521–543, 2017.
- Ene-Margit Tiit. Nonparametric statistical methods. myles hollander and douglas a. wolfe, wiley, chichester, 1999. no. of pages: xiii+ 779. price:£ 39.95. isbn 0-471-19045-4, 2000.
- Eduardo Uchoa, Artur Pessoa, and Lorenza Moreno. Optimizing with column generation: Advanced branch-cut-and-price algorithms (part i). 2024.
- U.S. Census Bureau. Quickfacts: Charleston city, south carolina, a. URL <https://www.census.gov/quickfacts/fact/table/charlestoncitysouthcarolina/POP010220>. Accessed: 2025-07-14.
- U.S. Census Bureau. Quickfacts: San francisco city, california, b. URL <https://www.census.gov/quickfacts/fact/table/sanfranciscocitycalifornia/POP010220>. Accessed: 2025-07-14.
- Yuliya Vakulenko, Daniel Hellström, and Klas Hjort. What’s in the parcel locker? exploring customer value in e-commerce last mile delivery. *Journal of Business Research*, 88:421–427, 2018.
- Marta Viu-Roig and Eduard J. Alvarez-Palau. The impact of e-commerce-related last-mile logistics on cities: A systematic literature review. *Sustainability*, 12(16):6492, 2020.
- Julia Wahlen and Timo Gschwind. Branch-and-price for the set-union bin packing problem. *INFORMS Journal on Computing*, 2025.
- Yang Wang, Mengyu Bi, Jianhui Lai, Chenxi Wang, Yanyan Chen, and José Holguín-Veras. Recourse strategy for the routing problem of mobile parcel lockers with time windows under uncertain demands. *European Journal of Operational Research*, 316(3):942–957, 2024.

- Setyo Tri Windras Mara, Rachmadi Norcahyo, Panca Jodiawan, Luluk Lusiantoro, and Achmad Pratama Rifai. A survey of adaptive large neighborhood search algorithms and applications. *Computers & Operations Research*, 146:105903, 2022. ISSN 0305-0548. doi: <https://doi.org/10.1016/j.cor.2022.105903>.
- Alexander Wyrowski, Nils Boysen, Dirk Briskorn, and Stefan Schwerdfeger. Public transport crowdshipping: moving shipments among parcel lockers located at public transport stations. *OR Spectrum*, 2024.
- Violet Xinying Chen and John N Hooker. A guide to formulating fairness in an optimization model. *Annals of Operations Research*, 326(1):581–619, 2023.
- Xiaofeng Xu, Chenglong Wang, and Peng Zhou. GVRP considered oil-gas recovery in refined oil distribution: From an environmental perspective. *International Journal of Production Economics*, 235:108078, 2021.
- Gege Yang, Yin Huang, Ying Fu, Biao Huang, Sishi Sheng, Lisha Mao, Shumin Huang, Yan Xu, Jia Le, Yun Ouyang, et al. Parcel locker location based on a bilevel programming model. *Mathematical problems in engineering*, 2020(1):5263689, 2020.
- Quan Yuan and Jueyu Wang. Goods movement, road safety, and spatial inequity: Evaluating freight-related crashes in low-income or minority neighborhoods. *Journal of Transport Geography*, 96:103186, 2021.
- Shohre Zehetabian. A fair multi-commodity two-echelon distribution problem. *EURO Journal on Transportation and Logistics*, 13:100126, 2024.
- Han Zhang, Yongbo Lv, and Jianwei Guo. New development direction of underground logistics from the perspective of public transport: A systematic review based on scientometrics. *Sustainability*, 14(6), 2022a. ISSN 2071-1050.
- Meijing Zhang and Lynette Cheah. A framework to organize public transport-based crowdshipping. *Journal of Public Transportation*, 27:100133, 2025.
- Meijing Zhang, Lynette Cheah, and Costas Courcoubetis. Exploring the potential impact of crowdshipping using public transport in singapore. *Transportation Research Record: Journal of the Transportation Research Board*, page 036119812211232, 2022b. ISSN 0361-1981, 2169-4052.
- Shang Zhang, Tho Le, and Matthew Roorda. Optimizing Parcel Locker Locations in a City Crowd Logistics Network. *Transportation Research Record: Journal of the Transportation Research Board*, 2677:036119812311674, 05 2023.
- Shuai Zhang, Siliang Liu, Weibo Xu, and Wanru Wang. A novel multi-objective optimization model for the vehicle routing problem with drone delivery and dynamic flight endurance. *Computers & Industrial Engineering*, 173:108679, 2022c.
- Yipei Zhang, Feng Chu, Ada Che, Yugang Yu, and Xin Feng. Novel model and kernel search heuristic for multi-period closed-loop food supply chain planning with returnable transport items. *International journal of production research*, 57(23):7439–7456, 2019.
- Yipei Zhang, Ada Che, and Feng Chu. Improved model and efficient method for bi-objective closed-loop food supply chain problem with returnable transport items. *International Journal of Production Research*, 60(3):1051–1068, 2022d.

- Zheng Zhang, Brian T Denton, and Xiaolan Xie. Branch and price for chance-constrained bin packing. *INFORMS Journal on Computing*, 32(3):547–564, 2020.
- Yizi Zhou, Rupal Mandania, and Jiyin Liu. Green vehicle routing and dynamic pricing for scheduling on-site services. *International Journal of Production Economics*, 254:108602, 2022.
- Fernando Zingler and Navneet Vidyarthi. A conceptual framework for integrating urban freight deliveries with public transportation and crowdsourcing. *Transportation Research Procedia*, 82:1293–1312, 2025.
- Özhan Zurel, Laurent Van Hoyweghen, Stijn Braes, and Aurelie Seghers. Parcel lockers, an answer to the pressure on the last mile delivery? In *New Business and Regulatory Strategies in the Postal Sector*, Topics in Regulatory Economics and Policy. Springer International Publishing, 2018. ISBN 978-3-030-02937-1.