UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Crowdsensing-driven route optimisation algorithms for smart urban mobility

Petar Mrazović

Co-funded by the
Erasmus+ Programme
of the European Union

Crowdsensing-driven Route Optimisation Algorithms for Smart Urban Mobility

PETAR MRAZOVIĆ

School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden 2018
◇
Computer Architecture Department
UPC Polytechnic University of Catalonia
Barcelona, Spain 2018

KTH School of Electrical Engineering
and Computer Science
SE-164 40 Kista
SWEDEN

TRITA-EECS-AVL-2018:65
ISBN 978-91-7729-947-9

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av doktorsexamen i informationsoch kommunikationsteknik måndagen den 22 oktober 2018 kl. 11:00 i Sal C, Electrum, Kungliga Tekniska högskolan, Kistagången 16, Kista.

Tryck: Universitetsservice US AB

Abstract

Urban mobility is often considered as one of the main facilitators for greener and more sustainable urban development. However, nowadays it requires a significant shift towards cleaner and more efficient urban transport which would support for increased social and economic concentration of resources in cities. A high priority for cities around the world is to support residents' mobility within the urban environments while at the same time reducing congestions, accidents, and pollution. However, developing a more efficient and greener (or in one word, smarter) urban mobility is one of the most difficult topics to face in large metropolitan areas. In this thesis, we approach this problem from the perspective of rapidly evolving ICT landscape which allow us to build mobility solutions without the need for large investments or sophisticated sensor technologies.

In particular, we propose to leverage Mobile Crowdsensing (MCS) paradigm in which citizens use their mobile communication and/or sensing devices to collect, locally process and analyse, as well as voluntary distribute geo-referenced information. The mobility data crowdsensed from volunteer residents (e.g., events, traffic intensity, noise and air pollution, etc.) can provide valuable information about the current mobility conditions in the city, which can, with the adequate data processing algorithms, be used to route and manage people flows in urban environments.

Therefore, in this thesis we combine two very promising Smart Mobility enablers – MCS and journey/route planning, and thus bring together to some extent distinct research challenges. We separate our research objectives into two parts, i.e., research stages: (1) *architectural challenges in designing MCS systems* and (2) *algorithmic challenges in MCS-driven route planning applications*. We aim to demonstrate a logical research progression over time, starting from fundamentals of human-in-the-loop sensing systems such as MCS, to route optimisation algorithms tailored for specific MCS applications. While we mainly focus on algorithms and heuristics to solve NP-hard routing problems, we use real-world application examples to showcase the advantages of the proposed algorithms and infrastructures.

Keywords: smart cities, smart mobility, urban mobility, mobile crowdsensing, route/journey planning, route optimisation, heuristic algorithms

Sammanfattning

Urban rörlighet anses ofta vara en av de främsta möjliggörarna för en hållbar statsutveckling. Idag skulle det dock kräva ett betydande skifte mot renare och effektivare stadstransporter vilket skulle stödja ökad social och ekonomisk koncentration av resurser i städerna. En viktig prioritet för städer runt om i världen är att stödja medborgarnas rörlighet inom stadsmiljöer medan samtidigt minska trafikstockningar, olyckor och föroreningar. Att utveckla en effektivare och grönare (eller med ett ord; smartare) stadsrörlighet är en av de svåraste problemen att bemöta för stora metropoler. I denna avhandling närmar vi oss problemet från det snabba utvecklingsperspektivet av IT landskapet i städer vilket möjliggör byggandet av rörlighetslösningar utan stora stora investeringar eller sofistikerad sensortenkik.

I synnerhet föreslår vi utnyttjandet av den mobila rörlighetsavkännings, eng. Mobile Crowdsensing (MCS), paradigmen i vilken befolkningen exploaterar sin mobilkommunikation och/eller mobilasensorer med syftet att frivilligt samla, distribuera, lokalt processera och analysera geospecifik information. Rörlighetavkänningssdata (t.ex. händelser, trafikintensitet, buller och luftföroreningar etc.) inhämtad från frivilliga i befolkningen kan ge värdefull information om aktuella rörelsesförhållanden i stad vilka, med adekvata databehandlingsalgoriter, kan användas för att planera människors rörelseflöden inom stadsmiljön.

Såtillvida kombineras i denna avhandling två mycket lovande smarta rörlighetsmöjliggörare, eng. Smart Mobility Enablers, nämligen MCS och rese/ruttplanering. Vi kan därmed till viss utsträckning sammanföra forskningsutmaningar från dessa två delar. Vi väljer att separera våra forskningsmål i två delar, dvs forskningssteg: (1) arkitektoniska utmaningar vid design av MCS-system och (2) algoritmiska utmaningar för tillämpningar av MCS-driven ruttplanering.

Vi ämnar att visa en logisk forskningsprogression över tiden, med avstamp i mänskligt dirigerade rörelseavkänningssystem som MCS och ett avslut i automatiserade ruttoptimeringsalgoritmer skräddarsydda för specifika MCS-applikationer. Även om vi förlitar oss på heuristiska lösningar och algoritmer för NP-svåra ruttproblem förlitar vi oss på äkta applikationer med syftet att visa på fördelarna med algoritm- och infrastrukturförslagen.

Nyckelord: smarta städer, smart rörlighet, rörlighetsavkänning, mobil publikmätning, ruttplanering, ruttoptimering, heuristiska algoritmer

Resumen

La movilidad urbana es considerada una de las principales desencadenantes de un desarrollo urbano sostenible. Sin embargo, hoy en día se requiere una transición hacia un transporte urbano más limpio y más eficiente que soporte una concentración de recursos sociales y económicos cada vez mayor en las ciudades. Una de las principales prioridades para las ciudades de todo el mundo es facilitar la movilidad de los ciudadanos dentro de los entornos urbanos, al mismo tiempo que se reduce la congestión, los accidentes y la contaminación. Sin embargo, desarrollar una movilidad urbana más eficiente y más verde (o en una palabra, más inteligente) es uno de los temas más difíciles de afrontar para las grandes áreas metropolitanas. En esta tesis, abordamos este problema desde la perspectiva de un panorama TIC en rápida evolución que nos permite construir movilidad sin la necesidad de grandes inversiones ni sofisticadas tecnologías de sensores.

En particular, proponemos aprovechar el paradigma Mobile Crowdsensing (MCS) en el que los ciudadanos utilizan sus teléfonos móviles y dispositivos, para nosotros recopilar, procesar y analizar localmente información georreferenciada, distribuida voluntariamente. Los datos de movilidad recopilados de ciudadanos que voluntariamente quieren compartirlos (por ejemplo, eventos, intensidad del tráfico, ruido y contaminación del aire, etc.) pueden proporcionar información valiosa sobre las condiciones de movilidad actuales en la ciudad, que con el algoritmo de procesamiento de datos adecuado, pueden utilizarse para enrutar y gestionar el flujo de gente en entornos urbanos.

Por lo tanto, en esta tesis combinamos dos prometedoras fuentes de movilidad inteligente: MCS y la planificación de viajes/rutas, uniendo en cierta medida los distintos desafíos de investigación. Hemos dividido nuestros objetivos de investigación en dos etapas: (1) Desafíos arquitectónicos en el diseño de sistemas MCS y (2) Desafíos algorítmicos en la planificación de rutas aprovechando la información del MCS. Nuestro objetivo es demostrar una progresión lógica de la investigación a lo largo del tiempo, comenzando desde los fundamentos de los sistemas de detección centrados en personas, como el MCS, hasta los algoritmos de optimización de rutas diseñados específicamente para la aplicación de estos. Si bien nos centramos en algoritmos y heurísticas para resolver problemas de enrutamiento de clase NP-hard, utilizamos ejemplos de aplicaciones en el mundo real para mostrar las ventajas de los algoritmos e infraestructuras propuestas.

Palabras clave: Ciudades inteligentes, movilidad inteligente, mobile crowdsensing, planificación de viajes/rutas, optimización de rutas, algoritmos heurísticos

Acknowledgements

This work has been supported by the Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC) funded by the Education, Audiovisual and Culture Executive Agency (EACEA) of the European Commission under the FPA 2012-0030.

I am deeply thankful to the following people, without the help and support of whom, I would not have managed to complete this thesis:

My supervisors Mihhail Matskin and Josep-Lluís Larriba-Pey for their guidance, constant feedback and encouragement throughout this work.

EMJD-DC programme coordinators, Vladimir Vlassov and Leandro Navarro, for their guidance through this joint doctorate journey.

My co-authors, Bahaeddin Eravci, Burcu Kolbay, Elif Eser, Hakan Ferhatosmanoğlu, and Nima Dokoohaki. Working with them has been a great pleasure and invaluable learning experience.

My friends and colleagues, Edward Tjörnhammar and Pedro Herruzo, for translating the summary of this thesis in Swedish and Spanish.

My friends, colleagues, and fellow PhD students at UPC and KTH, especially Ariel Duarte, Arnau Prat-Pérez, Burcu Kolbay, Edward Tjörnhammar, Erol Kazancli, Joan Guisado-Gámez, Pedro Herruzo, and Shatha Jaradat, for being supportive and for the good times we spent hanging out together.

My EMJD-DC colleagues for sharing this joint doctorate experience.

My friends and colleagues at Sparsity Technologies. Working with them on challenging real-world problems has been a great pleasure and learning experience.

My brother, Rudi Mrazović, for his love and encouragement.

My parents, Bosa and Slavko Mrazović, for their love, patience, continuous support, and immeasurable understanding.

My soon-to-be-wife, Barbara Jelčić, for changing her life path to be closer to me and help me finish my PhD.

Petar Mrazović

Petar Mrazović
Stockholm, September 13, 2018

Contents

List of Figures

List of Tables

List of Acronyms

ACF	Autocorrelation Function
AR(I)MA	Autoregressive (Integrated) Moving Average
BRP	Bicycle sharing Rebalancing/Repositioning Problem
BSS	Bicycle Sharing System
BnB	Branch and Bound Algorithm
BnC	Branch and Cut Algorithm
BnCnP	Branch and Cut and Price Algorithm
BnP	Branch and Price Algorithm
ETC	Electronic Toll Collection
EVP	Emergency Vehicle Preemption
GLS	Guided Local Search
GPS	Global Positioning System
GRASP	Greedy Randomized Adaptive Search Procedure
HAP	Historical Availability Profile
HDC	Highway Data Collection
HM	Historical Mean
HPP	Hamiltonian Path Problem
HPP	Hamiltonian Path Problem
ICT	Information and Communication Technology
ILS	Iterated Local Search
ITS	Intelligent Transport System
ITS	Intelligent Transportation System
IoT	Internet of Things
LR	Linear Regression
LSTM	Long Short-Term Memory
LV	Last Value
MAE	Mean Absolute Error
MCS	Mobile Crowdsensing
MILP	Mixed-integer Linear Programming Problem
MIMO	Multi-Input Multi-Output
MOP	Mixed Orienteering Problem
MSE	Mean Squared Error
MTG	Mobile Tourist Guide
MVRP	Multi-Vehicle Route Planner
OP	Orienteering Problem
PET	Personalized Electronic Tourist guide
PGI	Parking Guidance Information System
PNS	Personal Navigation Systems

POI	Point of Interest
R&R	Ruin and Recreate
RCL	Restricted Candidate List
SOP	Arc Orienteering Problem
SRT	Stay Region Tree
SVNS	Skewed Variable Neighbourhood Search
TD-FC	Time-Distributed Fully Connected layer
TMS	Traffic Management System
TOP	Team Orienteering Problem
TOPTW	Team Orienteering Problem With Time Windows
TRP	Travelling Repairman Problem
TSP	Transit Signal Priority
TSPP	Travelling Salesman Path Problem
TSPTW	Travelling Salesman Problem with Time Windows
TTDP	Tourist Trip Design Problem
VDC	Vehicle Data Collection
VNS	Variable Neighbourhood Search
VRP	Vehicle Routing Problem
WSN	Wireless Sensor Network

Chapter 1

Introduction

Today, more than a half of the world's population (54%) lives in cities and urban areas [1]. This proportion is expected to rise to 66% by 2050 which can have substantial implications for the world in terms of environmental impact and quality of life [1, 2]. Rapid and unplanned urban growth puts intense pressure on city resources and infrastructures which can threaten sustainable development if policies are not implemented to ensure that the benefits of city life are equitably shared [1]. Since urban mobility is often considered as one of the main facilitators for sustainable development, most of these policies encourage a shift towards cleaner and more efficient urban transport which would support social and economic concentration of resources in cities. Today, the world's population makes more than 64% of all travel kilometers within urban environments which accounts for 40% of all CO_2 emissions of road transport and up to 70% of other pollutants from transport [3, 4]. Furthermore, the travel within urban areas is expected to triple by 2050 [3] and it is obvious that cities will not be able to absorb such amount of traffic without efficient urban mobility management. Therefore, a high priority for cities around the world is to support citizens' mobility within the urban environments while at the same time reducing congestions, accidents, and pollution. However, developing a more efficient and greener (or in one word, *smarter*) urban mobility is one of the most difficult topics to face in large metropolitan areas. Fortunately, science and technology naturally arise as a means to achieve these goals.

In a recent survey [5], Benevolo et al. describe *Smart Mobility* as a pivotal component of *Smart City* vision which greatly assists in achieving its most common long-term objectives such as reducing the environmental footprint of the city and improving the citizens' quality of life (by reducing traffic congestions and noise pollution, improving transfer speed and costs, and increasing people safety). Obviously, as a crucial method for Smart City development, Smart Mobility has been a focus of the many research studies in the last 10 years (e.g., [6–12]). Therefore, different ways to achieve Smart Mobility have been proposed in the literature – from infrastructural projects which change the urban landscape to Intelligent Transport Systems (ITS) which heavily rely on Information and Communications Technologies (ICT). The rapid evolution of ICT and widespread accessibility of Internet make it easy to adopt new solutions which can improve mobility in Smart Cities without the need for large investments or sophisticated sensor technologies. In the wake of this, throughout this work we further promote the extensive use of ICT in achieving Smart Mobility. In particular, we propose to leverage Mobile Crowdsensing (MCS) paradigm in which citizens use their mobile communication and/or sensing devices to collect, locally process and analyse, as well as voluntary distribute geo-referenced information. The mobility data crowdsensed from volunteer citizens can provide valuable information about the current mobility conditions in the city. Eventually, with the adequate data processing algorithms, the collected information (e.g., events, traffic intensity, noise and air pollution, etc.) could be used to route and manage people flows in urban environments. This way we may be able not only to find the shortest or cheapest routes, but also to find "smarter" routes, while being aware of the current mobility conditions. Therefore, in this work, we primarily focus on route optimization and scheduling algorithms with the aim of extending the state-of-the-art methods by integrating mobility data crowdsensed from volunteer citizens. However, before discussing the research gaps and contributions of this thesis, in the next section we give a brief introduction to each of the addressed topics. We first introduce the concept of Smart City as a long-term vision of a green and efficient digital urban area. Thereafter, we introduce MCS paradigm and discuss how it can be leveraged to achieve Smart Mobility. Finally, we give a short introduction to state-of-the-art route optimisation algorithms.

1.1. Background

1.1.1. Smart Mobility in Smart City

Smart City Concept

ICT revolution has offered city governments the opportunity to reduce the scale of environmental and quality of life issues caused by urbanization and rapid population growth. Over the last decade, cities have become more digital and information-based, and ICT has integrated into almost all aspects of modern city life. Such widespread technology and Internet accessibility have not only introduced daily convenience to the urban population, but also possibility to collect and process different data as a basis for sustainable city evolution [13].

In recent years, different information technologies such as cloud computing, big data, data visualisation, Internet of Things (IoT), and mobile computing have significantly speeded up the process of city digitization and evolution of Smart City vision as we know it today. The cloud computing paradigm delivers hardware and software resources over Internet based on on-demand utility model, enabling developers to provide new smart urban services without the need for large investments. At the same time, the distributed nature of cloud computing infrastructures allows massive amounts of city data to be collected and processed. Here, the big data analytics arise as a way to examine large and varied data sets to uncover hidden patterns, unknown correlations, and other useful city information that can help city governments make informed development decisions. Furthermore, advanced data visualisation techniques enables decision makers to see analytics presented visually, so they can grasp more difficult concepts or identify new patterns. Over the past few years, the convergence of wireless technologies, micro-services, and Internet, gave rise to IoT, a paradigm that relies on the identification and use of a large number of heterogeneous physical and virtual objects which are connected to Internet [14]. IoT makes it possible to control physical world by accessing remote sensor data, meaning that cities can effectively sense and manage essential city resources and infrastructures such as the water supply, building operations, road and transport networks, etc. [13, 15]. Therefore, IoT and cloud computing are often seen as two critical technologies for realizing the Smart City vision. The cloud can provide large-scale and long-lived storage and processing resources for the ubiquitous applications delivered through the IoT [16]. Finally, with the powerful multi-sensing capabilities of mobile communication devices and user mobility, a new trend of development in IoT is emerging – Mobile Crowdsensing (MCS). MCS allows everyday sensor-rich mobile devices to form interactive, participatory sensor networks that enable public and professional users to gather, analyse and share local knowledge. The high population density of modern cities and widespread availability of interconnected mobile communication devices encourage ordinary citizens to participate in community sensing and open up new possibilities for effective city management.

The above information technologies, coupled with the urban challenges, have enabled us to form the vision of Smart City and to develop a winning urban strategy to cope with rapidly increasing urbanisation. However, since different technologies have been implemented under the Smart City label, the definition of Smart City has evolved to mean different things in different contexts. In their view on shift from intelligent to Smart Cities, Deakin and Al Wear [18] listed four components that contribute to Smart City development:

1. the application of a wide range of electronic and digital technologies to communities and cities,
2. the use of ICT to transform life and work within a region,
3. the embedding of such ICTs in government systems,
4. the territorialization of such practices in a way that bring ICTs and people together, so as to enhance the innovation, learning, knowledge and problem solving which they offer.

Table 11: Central pillars of Smart Cities

Smart Economy **(Competitiveness)**	**Smart People** **(Social and Human Capital)**
• Innovative spirit • Entrepreneurship • Economic image & trademarks • Productivity • Flexibility of labour market • International embeddedness • Ability to transform	• Level of qualification • Affinity to life long learning • Social and ethnic plurality • Flexibility • Creativity • Cosmopolitanism/Open-mindedness • Participation in public life
Smart Governance **(Participation)**	**Smart Mobility** **(Transport and ICT)**
• Participation in decision-making • Public and social services • Transparent governance • Political strategies & perspectives	• Local accessibility • (Inter-)national accessibility • Availability of ICT-infrastructure • Sustainable, innovative, and safe transport systems
Smart Environment **(Natural Resources)**	**Smart Living** **(Quality of Life)**
• Attractivity of natural conditions • Pollution • Environmental protection • Sustainable resource management	• Cultural facilities • Health conditions • Individual safety • Housing quality • Education facilities • Touristic attractivity • Social cohesion

Source: Giffinger et al. [17]

In short, Deakin and Al Wear see ICT as a crucial instrument for Smart City development, however, the precise goals and challenges need to be further elaborated. Since Smart Cities develop along many dimensions, Giffinger et al. [17] established six major characteristics which can be thought of as a track of development or success indicator of Smart City transformation. Since the proposed characteristics are measurable and well defined in scope and nature, they were soon adopted by the European Union as the pillars central to the existence of Smart Cities. These are: smart economy, smart people, smart governance, smart mobility, smart environment, and smart living. The characteristics are summarised in Table 11, along with the factors which define them. Smart economy gathers ICT-enabled practices for increased productivity such as e-business and e-commerce, but also new ways of enhancing and stimulating economy to secure local and global inter-connectedness and competitiveness [19]. Smart people, i.e., social and human capital, is a critical segment of Smart City development. Citizens in Smart Cities should be able to acquire e-skills, working in ICT-enabled jobs, having access to education and training, human resources and capacity management. Finally, smart people should be able to participate in city development by engaging into transparent decision-making process as a part of smart governance strategy. Besides citizens participation, the smart governance includes technologies to link public, civil and national organisations, and to transparently share the city data between them. Smart governance dimension to some extent overlaps with smart environment which ensures sustainable urban development and resource management. Smart living includes different aspects of quality of life such us culture, health, safety, housing, etc. In addition, ICT-enabled life styles introduce daily convenience to everyday life. Finally, Smart Mobility is only seen as one of the dimensions of Smart City, however, it is a crucial one, impacting on numerous aspects of quality of life, sustainable development, and economy. In other words, smart mobility is seen like a slice of Smart City, crossing almost all of the dimensions listed above. In the rest of this subsection we will analyse the smart mobility initiatives in more details and discuss the role of ICT in supporting Smart Mobility actions.

Smart (Urban) Mobility

Urban mobility is one of the most promising challenges faced by city governments, presenting economic, social and environmental implications. Therefore, developments in the field of Smart Mobility are occurring rapidly, and provide many opportunities in both short and long term, such as the improvement of traffic flow and safety, as well as the possibility to contribute to environmental sustainability. Smart Mobility is not a unique initiative, but a complex set of actions, different in goals, contents and technology intensity. In their survey on Smart Mobility [5], Benevolo et al. gathered the most important Smart Mobility objectives and summarised them into six categories:

1. reducing pollution,
2. reducing traffic congestion,
3. increasing people safety,
4. reducing noise pollution,
5. improving transfer speed,
6. reducing transfer costs.

While these goals can be achieved in many different ways, here we will focus only on those that heavily rely on consistent and systematic use of ICT. Below, we group the most promising technologically advanced services that enable Smart Mobility. Notice that these services offer a wide range of capabilities that benefit all mobility stakeholders, i.e., travellers, transport operators, urban planners, and city governments.

Smart parking systems In traditional smart parking implementations, sensors are deployed to detect the presence or absence of a vehicle, while local and global positioning technology is usually leveraged to navigate drivers to available locations. These traditional systems are referred to as Parking Guidance Information Systems (PGI). Other forms of smart parking implementation include Transit-Based Information Systems, Smart Payment Systems, E-parking, and Automated Parking. Transit-based information systems are designed specifically for Park and Ride facilities, providing parking space information and public transport schedules. This way commuters are encouraged to park their vehicles and use public modes of transport for their transit, which in turn reduces traffic congestion, pollution, and fuel consumption. Smart payment systems employ advanced technologies, such as contactless cards and mobile communication devices, to allow faster and more convenient payment. On the other hand, E-parking systems combine and streamline parking reservation and payment systems. E-parking systems heavily rely on mobile communication technologies, allowing travellers to find, reserve, and pay for parking from their cell phones. Finally, the most technologically advanced form of smart parking include fully-automated mechanical systems for storing vehicles. So-called automated parking allow drivers to avoid difficult parking maneuvers and instead to rely on computer-controlled system which efficiently manages parking spaces.

Smart/Integrated ticketing Smart/Integrated ticketing allows travellers to make a journey that involves transfers within or between different transport modes (e.g., buses, trains, subways, bicycles, ferries, etc.) with a single ticket that is valid for the complete journey. Usually, integrated ticketing system relies on electronic ticketing technologies such as magnetic stripe cards, contactless smart cards, or, lately, mobile communication devices. Such systems encourage travellers to use public transport by allowing seamless payment for transport services across modes.

Real-time journey/route planners Since the 1970s, browser-based journey planners have been widely used in the travel industry by airline companies and booking agents. In recent years, ICT revolution, widespread access to Internet, and proliferation of geospatial data opened new possibilities for mobile-based real-time journey planners. Nowadays, mobile planners are designed to help users in their everyday commute by providing real-time travel information across all transport modes. Since they are usually used only for short-term travel planning, i.e., choosing the most convenient route between two locations within the same city, they are often referred to as route planners. Using the planners, travellers can re-gain control over their own journey time and make adaptive choices to avoid congestions and overcrowding in busy urban areas. Additionally, by being aware of different transport modes and their pricing, travellers are able to optimise their travel costs. Finally, lower carbon travel options are promoted through raised awareness about public transport options.

Driving guidance systems Also known as automotive navigation systems, driving guidance systems typically use satellite navigation technology, i.e., GPS, to direct drivers in a road network. Modern driving guidance systems are also connected to Internet, thus having real-time information about traffic conditions on the road. This way, drivers can be warned about congestions, traffic accidents, or road maintenance, while driving. Advanced systems also allow drivers to choose environment friendly routes with lower fuel consumption and CO_2 emission.

Intelligent Transport Systems Intelligent transport systems (ITS) provide digital, i.e., virtual, solutions in different fields of transport system, such as transport management, control, infrastructure, operations, policies and control methods, etc. [20]. Over the last decade, they have become an integral part of road infrastructure which play a major role in reducing risks, high accidents rate, traffic congestion, carbon emissions, air pollution, and, on the other hand, increasing safety, reliability, travel speeds, traffic flow, etc. [20]. More notable examples of ITS applications include Electronic Toll Collection (ETC), Highway Data Collection (HDC), Traffic Management Systems (TMS), Vehicle Data Collection (VDC), Transit Signal Priority (TSP), and Emergency Vehicle Preemption (EVP). etc. Since ITS comprises a wide range of applications, different technologies have been adopted, e.g., computing hardware, local and global positioning, sensor technologies, data processing, mobile communications, planning techniques, etc. Over the past several years, research efforts have been focused on integrating different ITS applications and exchanging real-time traffic information between them. In other words, traffic data from various sources such as traffic lights and signals, car parks, traffic monitoring station, air quality sensors, etc., are collected and stored in a central database accessible to all deployed applications. Naturally, this opens possibilities for new smart services which would rely on real-time traffic data.

Real-time ride-sharing systems In ride-sharing/carpooling, passengers with matching itineraries and schedules share a ride in a personal vehicle. The passengers usually share the associated costs (e.g. fuel, tolls, parking fees) so that each benefits from the shared ride [21]. Additionally, passengers may save time because they are allowed to use high-occupancy vehicle lanes reserved exclusively for carpooling. Real-time ride-sharing systems utilize mobile communication technologies to add flexibility to ride-share arrangements by allowing drivers and passengers to arrange occasional shared rides ahead of time or on short notice [22]. Real-time ride-sharing systems typically rely on global positioning systems (i.e., GPS) to determine passengers' current locations. Additionally, they make use of social networks to establish trust and accountability between drivers and passengers. Finally, route optimisation and scheduling algorithms are employed to match itineraries and plan the final route.

Taxi booking services Taxi booking systems (e.g., Uber, Cabify, MyTaxi) provide convenient access to taxi services at any place or time. They heavily rely on mobile communication technologies, GPS, and routing and scheduling algorithms to offer users the shortest, fastest, greenest or cheapest routes, depending on their taste or need. Additionally, electronic and fully integrated payment solutions allow users to seamlessly pay for their trips. On the other hand, taxi booking systems collect huge volumes of data that reflect mobility trends in city and as such can be used to inform infrastructure planning and public policy, or to directly influence other public transit services.

Bicycle sharing systems Bicycle sharing system refers to one of the most sustainable public transport service systems which offers short-term urban bicycle rentals by enabling bicycles to be picked up at any self-serve bicycle station and returned to any other bicycle station [23]. Recent ICT advances allowed bicycle sharing system to be smarter by providing real-time information about bicycle usage and availability.

Car sharing systems Similar to bicycle sharing systems, car sharing systems allow people to rent cars for short period of time, typically by hour. They can be classified into "one-way" or "two-way" types, depending whether the users should return the rented vehicle at a different or at the location they picked it up [24]. Obviously, the car-sharing scheme decreases average vehicle kilometers travelled within a city, and thus can improve traffic flow and reduce CO_2 emissions. At the same time, users can benefit from reduced personal transport cost and enhanced mobility [24].

Autonomous vehicles Autonomous vehicles are capable of sensing their surroundings and navigating without human input. They make use of advanced technologies such as computer vision, GPS, odometry, radars, etc., to sense the environment and to identify appropriate navigation paths. It is likely that the technology of autonomous vehicles will dramatically affect ride-sharing platforms. In other words, within a decade or two, autonomous vehicles could completely displace conventional vehicles for ride-sharing, while many citizens will not even own cars. Furthermore, it is believed that this trend will reshape the car manufacturing industry and will disrupt the business models of petrol stations, service centres, car rental companies, car park operators, and many others.

Although very different in nature and scope, the above services and technologies are developed with the same perspective - to build a greener and safer traffic system which would improve transfer speeds and lower transfer costs. As such, the introduced services to some extent rely on the same technologies and similar algorithms. The careful reader might have noticed that most of the above services employ some form of route optimisation. Indeed, reducing transfer speeds and costs can be only in rare cases achieved without any route optimisation or scheduling algorithm. In this thesis, we will focus primarily on such algorithms with the aim of extending the state-of-the-art methods by integrating mobility data collected from citizens. Although most of the algorithms proposed in this thesis will be designed for real-time journey/route planners, they can be easily tweaked for different applications and employed in any of the above systems. Before discussing the state-of-the-art algorithms, we will first explain how and what kind of mobility data can be collected through mobile crowdsensing paradigm and used as an input to more informed route optimisation.

1.1.2. Mobile Crowdsensing Paradigm

Modern mobile communication devices have matured as powerful and versatile computing platforms and they acquired high performance multisensing capabilities. The widespread availability of sensor-enhanced mobile devices, such as smartphones, tablets, or smartwatches, has opened a possibility for a *participatory sensing*. Burke et al. [25] introduced the concept which allows everyday sensor-rich mobile devices to form interactive, participatory sensor networks that enable public and professional users to gather, analyse and share local knowledge. However, traditional participatory sensing infrastructures usually do not consider device carriers as intelligent participants in a sensing process. Over the past few years, the research efforts of combining human and machine intelligence in mobile sensing systems have resulted in a novel exciting sensing paradigm called Mobile Crowdsensing (MCS). As a people-centric evolution of participatory sensing, MCS involves both implicit and explicit participation. In other words, MCS distinguishes two different data generation modes: mobile sensing and user-generated data in mobile social network services [26]. The integration and fusion of the two cross-space data sources is one of the main features (and challenges) of MCS. MCS systems have gone beyond the traditional environmental monitoring where humans are now the device carriers, the information sources, and the consumers of collected data [27].

In their position paper [26], Guo et al. described the evolution of participatory sensing and proposed the first formal definition of MCS: *MCS is a new sensing paradigm that empowers ordinary citizens to contribute data sensed or generated from their mobile devices, aggregates and fuses the data in the cloud for crowd intelligence extraction and people-centric service delivery.* Additionally, in order to better explain the main characteristics of MCS, Guo et al. [26] used an example of route planning problem in urban environment, which fits nicely with the topic of this thesis. As described in the previous subsection, journey/route planning is one of the most common ICT approaches that enable Smart Mobility. However, real-time planning requires up-to-date information about traffic conditions in the city. This is usually achieved by deploying static sensor networks, or, more commonly, leveraging citizens' personal devices to gather traffic data (i.e., using participatory sensing). For example, to answer a route query we can simply use traditional participatory sensing and collect GPS trajectories from vehicles and compute the optimal route by taking in consideration traffic intensity. However, to answer a more complex query, e.g., to generate an itinerary for a visitor given the time and budget restrictions, single trajectory dataset will not be sufficient. Additional information, such as points of interest (POIs) in the city, their operating hours and popular visiting times, user preferences, recommendations, information about scenic routes, etc., are further needed. This information can be obtained from user-generated data from mobile social networks (e.g., FourSquare, Yelp, Twitter) or directly through a route planning mobile application if adequate interface is provided. MCS arises as a result of combining human and machine intelligence, and sensor and user-generated data. By involving humans in the loop, MCS opens up new technological problems and numerous research challenges, such as quality of user-generated data, data collection techniques, data redundancy, users' trust and reputation, privacy, incentive mechanisms, etc. However, before discussing them in more details, we first need to understand the unique characteristics of MCS paradigm which differentiate it from traditional static sensor networks.

MCS characteristics and challenges

Compared to traditional sensor networks, MCS offers several very important advantages. MCS leverages already existing sensing and communication infrastructures (i.e., mobile communication devices and networks), and thus its deployment costs are significantly lower. At the same time, mobile devices have significantly more computing, communication, and memory resources than any of the traditional sensors. Furthermore, the widespread availability of these mobile communication devices and the inherent mobility of their owners provide large-scale spatiotemporal coverage. On the other hand, this also means that the participating mobile devices, the type of sensor data they can produce, and the quality in terms of accuracy, latency, and confidence can change constantly due to device mobility, variations in communication channels, and device owners' preferences [28]. Therefore, allocating sensing task to the right set of devices to produce the data of desired quality is common problem in MCS settings. In traditional sensor networks, where the population of sensors and the data they can produce are mostly known a priori, allocating sensing tasks and controlling the quality of the collected data is usually straightforward. Nevertheless, involving device owners in sensing processes also has its benefits. Human intelligence and mobility allow us to gather higher-quality or semantically complex data that might otherwise require sophisticated hardware and software resources [28]. For example, humans can easily identify traffic conditions in the city (e.g., congestions, accidents, parking availability, road constructions, etc.) and report them by generating some form of participatory media (e.g., pictures, texts, sound records). However, such active participation has also its implications. Mobile communication devices consume their own resources such as battery and computing power. In addition, device owners are exposed to potential privacy threats by voluntary sharing sensor data and personal information. Therefore, a user should receive a satisfying reward to compensate resource consumption and potential privacy breach. An incentive mechanism is needed to assure adequate user participation which is necessary for successful sensing in MCS. Finally, the involvement of human participation in the sensing and computing process leads to heterogeneous data collections and a mixture of machine intelligence (e.g., machine learning, data mining) and human intelligence (e.g., cognition, reasoning). Efficient integration of human and machine intelligence and fusion of cross-space data are some of the most complex challenges to be faced by the research community. MCS systems should be balanced hybrid human–machine systems in which human and machine intelligence complement each other. The differences between MCS and traditional wireless sensor networks (WSN) are summarised in Table 12.

MCS applications

Nowadays, traditional sensor networks are increasingly being replaced by MCS systems which benefit a number of application areas regarding environmental monitoring, public safety, healthcare, location services, mobile social recommendations, etc. However, in this thesis we will only focus on Smart Mobility applications which rely on different types of mobility data that can be collected in urban environments, and then used to measure and map large-scale phenomena related to public traffic infrastructures. We also refer the interested reader to recent surveys on MCS [28–30] for a more comprehensive list of MCS applications.

In Figure 11 we illustrate a typical resource organisation in MCS environments. Raw sensor data and user-generated data are collected on participating mobile devices and processed by local analytic algorithms to produce consumable data for applications. Addi-

Table 12: Comparison between MCS and traditional wireless sensor networks (WSN)

	Operators	**Deployment cost**	**Coverage**	**Data quality**
WSN	Government agencies, public institutions	High: Expensive sensors and infrastructure to deploy the network	Limited coverage with static sensor nodes	High, sound level sensors
MCS	Potentially everyone	Low: Leveraging existing infrastructure, i.e., broad proliferation of cellular network and mobile device usage	The inherent mobility of the phone carriers provides unprecedented spatiotemporal coverage	Low, suffering from issues such as built-in sensor performance and the trustworthiness of user-contributed data

Source: Guo et al. [29]

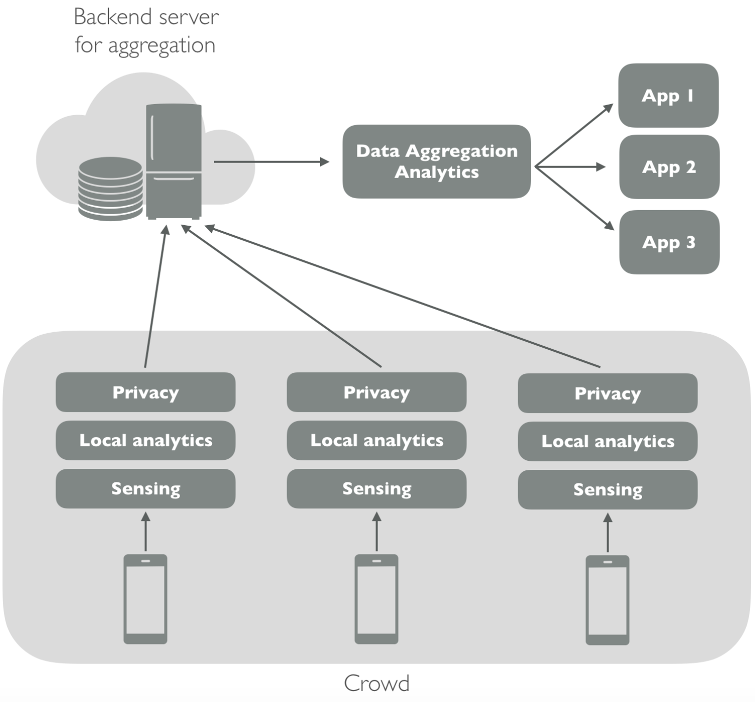


Figure 11: Typical resource organisation in MCS environments.

tionally, before being sent to a backend or cloud, the sensitive data can be modified and protected to conserve privacy. Thereafter, in the cloud, the data is aggregated and fused. Finally, depending on an application, different data mining techniques can be employed to extract relevant information and eventually to deliver adequate people-centric services. The illustrated organisation is basis of any MCS system, while the type of data or delivered service can differ depending on application scope. For example, in typical traffic MCS applications, movement data collected from GPS sensors, accelerometers, and gyroscopes, fused with user-generated data such as check-ins, reflects the mobility conditions in the city and as such can be used for traffic forecasting, public transport, travel planning, traffic monitoring, etc.

MCS paradigm is proved to be a valuable tool for achieving Smart Mobility. Over the past several years, different MCS applications have been proposed to improve the mobility conditions in urban areas. Early MCS deployments measured traffic dynamics and congestion levels in cities, examples of which include MIT's CarTel [31] and Microsoft Research's Nericell [32]. CarTel was a mobile sensor computing system designed to collect, process, deliver, and visualize data from GPS sensors located on mobile units such as automobiles. It has been deployed only on six cars, running on a small scale in Boston and Seattle for over a year, but nevertheless it has been proved to be very successful in analysing commute times, metropolitan Wi-Fi deployments, and automotive diagnostics. On the other hand, Nericell was a system that performed richer sensing by exploiting multisensing capabilities of modern mobile communication devices. In particular, Nericell used the accelerometer, microphone, GSM radio, and/or GPS sensors to detect potholes, bumps, braking, and honking, and to localize the phone in an energy-efficient manner. The effectiveness of the system has been evaluated on the roads of Bangalore, India. Another example of a similar application is the Real Time Rome project [33], a real-time monitoring system that collected and processed data provided by telecommunications networks and transport systems. In partuclar, the proposed system interpolated the aggregate mobility of people according to their mobile phone usage and visualised it synchronously with the flux of public transit, pedestrians, and vehicular traffic. By overlaying mobility information on geographic references of the city of Rome, the system was able reveal the relationships between fixed and fluid urban elements. Similarly, Liu et al. proposed so-called Urban Mobility Landscape System (UMLS) [34] which collected data from real-time sources (5,000 GPS-equipped floating cars and 5 million bus and metro smart cards). The system was deployed in a large urban area in the city of Shenzhen in South China with the aim of examining urban mobility patterns and providing real-time decision support for the city government.

Public transport is another interesting traffic application domain where MCS raises as an efficient tool to improve its design and offer real-time traffic information to the travellers. For example, Zhou et al. [35] managed to successfully predict bus arrival times by using data from generally available and energy efficient sensing resources, such as cell tower signals, movement statuses, audio recordings, etc. This way they brought less burden to the participatory party and encourage their participation. Evaluation of the system performed over 7-week period in Singapore suggested that the proposed approach achieves outstanding prediction accuracy compared with GPS-supported solutions. On the other hand, B-Planner proposed by Chen et al. [36] uses crowdsensed GPS data and pick-up/drop-off records from taxis to plan travel routes for night-buses.

In addition to traffic monitoring and public transport, MCS opens up new possibilities for personal journey/route planners and guidance systems. As previously mentioned, such systems, along with the supporting routing algorithms, will be in the main focus of this thesis. Most of the previous works in this field investigated the optimal route generation based on sensed traffic conditions in the city. For example, in one of the earlier works, Thiagarajan et al. [37] proposed VTrack system which was able to detect hotspots and to precisely estimate travel times for different road segments using positioning data from GPS sensors, WiFi, and cellular triangulation. In order to reduce energy consumption, the system was able to tolerate noisy data from less energy-consuming positioning sensors. Using a hidden Markov model (HMM)-based map matching scheme, Vtrack could interpolate sparse data to identify the most probable road segments driven by the users, and as such was able to rely on streams of timestamped inaccurate position samples recorded at time-varying periodicity. Finally, based on the sensed traffic information, VTrack recommended the fastest travel routes which avoid hotspots and congested areas. Another interesting MCS application was T-Share, a crowd-powered taxi ridesharing service proposed by Ma et al. [38]. T-Share efficiently served real-time requests sent by taxi users and generated ridesharing schedules that could significantly reduce the total travel distance. The system has been evaluated using a GPS trajectory dataset generated by over 33,000 taxis. Costa et al. [39, 40] proposed and developed SmartTrace, a distributed framework for finding similar trajectories in a MCS networks. The purpose of the framework was to look for the most common trajectories and thus provide more informed transit planning. SmartTrace was able to successfully execute distributed similarity search queries on trajectories that are stored in-situ on smartphones, without disclosing the traces of participating users to the querying node. Another worth-mentioning example of MCS-powered route planning application is personal travel itinerary recommendation platform proposed by Yu et al. The platform extracts user preferences, discovers POIs, and determines location correlations from check-in records. Thereafter, it generates personalized travel itineraries by considering spatio-temporal constraints such as travel time and starting location.

Described application examples suggest that the MCS paradigm has been raised as a powerful and easily implementable tool for achieving Smart Mobility objectives. In this thesis we will introduce several new MCS applications within route planning domain, however, we will mainly focus on algorithmic aspects of the proposed designs, rather than investigating data collection and fusion techniques. To put it differently, throughout the thesis we will use crowdsensed data as an already available input to different route optimisation algorithms. We aim to demonstrate that a simple volunteer data collected from mobile communication devices, such as check-ins or user preferences, can be efficiently processed and used to improve mobility in urban areas.

1.1.3. Route Optimisation Algorithms

Network routing is often considered as one of the most intriguing areas of mathematical optimisation and theoretical computer science. Network routing problems can be defined on different kinds of networks such as roads, utilities, electricity, telecommunications, etc. However, from a theoretical/mathematical point of view, a network is simply a graph with a set of nodes and weighted edges representing locations and connections between them. Here, network routing problems consist of finding a set of optimal paths or cycles between two or more nodes in a graph in relation to total time, cost, or distance. Different configurations of the underlying graphical structure, different objectives as well as different constraints have lead to an immense number of problem formulations and developed algorithms [41]. However, in this thesis, we will mainly focus on a set of network routing problems that commonly arise in transportation systems.

Since real routing problems in transportation systems involve finding efficient routes for vehicles (e.g., cars, trucks, buses, etc.), they are often referred to as *Vehicle Routing Problems* (VRP). Over the past several decades, many different variants of VRP have been proposed in the scientific literature. The classical VRP, also known as the Capacitated VRP (CVRP), is usually used to formulate new VRP variants. In CVRP, a fixed fleet of delivery vehicles of uniform capacity must service known customer demands for a single commodity from a common depot at minimum transit cost. Over the years, this classical VRP has been extended in many different ways by introducing additional real-life aspects and constraints, resulting in a large number of variants of the VRP. For example, it is possible to extend CVRP by varying the vehicle capacities, which results in the Heterogeneous Fleet VRP (HFVRP), also known as the Mixed Fleet VRP. In another popular extension, the VRP with Time Windows (VRPTW), each customer is assigned with the specific time interval (i.e., time window) within which she needs to be served by one of the vehicles from the fleet. Further, in the VRP with Pickup and Delivery (VRPPD), goods need to be moved from a certain pickup location and dropped off at another delivery location. A related variant is the VRP with backhauls (VRPB) where only difference is that all deliveries for each route must be completed before any pick-ups are made. The Multi-Depot VRP (MDVRP) assumes that multiple depots are geographically spread among the customers. In the Open VRP (OVRP), vehicles are not required to return to the depot. Recently, some of these variants have been combined into even more complex VRP variants, simultaneously combining multiple real-life aspects. However, instead of going through all possible VRP variants, we refer the interested reader to some of the recent surveys on the VRP classifications (e.g., [42, 43]). Later in this thesis, we will introduce several new routing problems and formulate them based on some of the aforementioned VRP variants. The new formulations represent the pivotal components of some of the novel MCS-driven route planners proposed in this thesis. However, since we will mainly focus on finding efficient solutions to these problems, we continue by discussing the common solution methods and algorithms for network routing problems in transportation systems.

Determining the optimal solutions to VRPs is NP-hard, so the size of the problems that can be solved to optimality within reasonable computing time is limited. Therefore, modern VRP solvers tend to rely on heuristics and metaheuristics due to the size and frequency of real world VRPs they need to solve. We will discuss the most common solution approaches for VRPs, which can be seamlessly applied to many other combinatorial optimisation problems. We group them into following four categories: exact methods, construction heuristics, improvement heuristics, and metaheuristics.

Exact methods

As the name suggests, the exact methods propose to compute every possible solution until the optimal one is reached. One of the most successful exact approaches for solving large scale NP-hard combinatorial optimization problems, such as the VRPs, is the Branch and Bound method (BnB) proposed by Land and Doig [44]. The BnB method is a divide-and-conquer approach based on the principle that the total set of feasible solutions can be partitioned into smaller subsets of solutions. These smaller subsets can then be evaluated systematically until the best solution is found. The systematic enumeration of candidate solutions is achieved by forming a rooted tree with the full set of solutions at the root. The method then explores branches of this tree, which represent subsets of the complete solution set. Before enumerating the candidate solutions of a branch (i.e., solution subset), the branch is checked against estimated bound on the optimal solution, and is discarded if it cannot produce a better solution than the best one found so far. In other words, the bound represents the best possible solution that we can obtain if we follow the current branch. Computing a bound on the optimal solution is usually achieved by relaxing the constraints which are making the problem hard to solve in the first place. In order to improve the bounds computed by such constraint relaxations, the Branch and Cut method (BnC) [45] has been proposed as a special type of BnB method. The BnC uses so-called cutting planes [46] to tighten the introduced relaxation by adding new linear constraints. Many other similar exact methods have been proposed for different variants of the VRP (e.g., Branch and Price (BnP) [47], Branch and Cut and Price (BnCnP) [48], etc.), but they were all based on the same principles introduced by BnB method. Later in the thesis, we will employ the BnB and BnC methods to find the optimal solutions to newly proposed route planning problems, and then use these solutions to evaluate performance of the developed heuristics.

Construction heuristics

Starting with an empty solution, construction heuristics gradually extend the current solution until a complete near-optimal solution is obtained. In case of VRPs and similar network routing problems, a feasible solution is usually created by sequentially adding new nodes (or edges) to the solution routes based on some cost optimisation criterion and subject to a set of restrictions that make sure that the original problem constraints are not violated.

One of the best-known constructive heuristics for VRP is the Savings algorithm proposed by Clarke and Wright [49]. The Savings algorithm starts with a solution in which each customer is served in a separate route. At each iteration, the algorithm computes and sorts the cost savings (e.g., time, distance) obtained by merging each pair of routes in the current solution. The merging with the highest cost saving is carried out given that it produces a feasible solution. These steps are repeated until no additional saving can be made. In [50], Solomon proposed another very simple constructive heuristic based on the nearest-neighbour approach. The proposed method begins with an empty route containing only depot node, and then iteratively searches for the customer closest to the last customer added into the route and adds it at the end of the route. A new route is created whenever the search fails to add a new customer without violating feasibility of the solution. In the same work ([50]), Solomon proposed another successful heuristic algorithm called I1. The algorithm starts with a route with a single "seed" customer, usually chosen to be geographically farthest from the depot. At each iteration, I1 algorithm searches for an un-served customer that can be inserted into the current route without violating any of the predefined constrains. If it fails to find such customer, the algorithm creates a new route and chooses a new "seed" customer. The algorithm stops when there are no un-served customers.

The aforementioned heuristics were later improved in many different ways, e.g., by modifying insertion criteria and selection strategies (e.g., [51–53]), by parallelizing the insertion process (e.g., [54, 55]), by clustering the customers based on their geographical position (e.g., [56]), etc. Even though these modifications improve the quality of the obtained solutions, constructive heuristics are still incomparably inferior to more sophisticated heuristic approaches such as the local search or metaheuristics. However, thanks to their simplicity and low computational complexity, they are often employed in more complex heuristic algorithms to find initial solutions in short computation time.

Improvement heuristics (Local search)

Local search methods are based on the concept of iteratively improving the solution to a problem by exploring neighbouring ones. Starting from the initial feasible solution, local search algorithms move from one solution to another in the space of candidate solutions by applying local changes, until an acceptable solution is found. Here, the local changes consist of a set of moves which create the neighbouring solutions by changing some of the attributes of a given solution. For example, in the context of network routing problems, a local search move could remove or replace some of the nodes or edges in the solution routes. Once a new neighbouring solution is created, it is compared against the current solution. If the neighbouring solution satisfies a certain acceptance rule (e.g., the cost is reduced by some percentage), it replaces the current solution, and the search starts over. Since local search methods sequentially accept solutions that reduce the objective function value, they can easily stuck in the local optimum which can be far from the optimal solution. Therefore, the success of local search methods depends heavily on initial solution and the neighbourhood generation mechanism. In case of network routing problems, some of the previously mentioned constructive heuristic are often employed to build initial feasible solutions. These solutions are then modified by applying different edge-exchange moves and algorithms, some of which we will discuss later in the thesis.

Metaheuristics

Metaheuristics are high-level problem-independent algorithmic frameworks that provide a set of guidelines or strategies to develop heuristic optimization algorithms [57]. They are not algorithms, but rather a consistent set of ideas, concepts, and operators that can be followed to design algorithms. Similar to previously mentioned local search methods, heuristics iteratively evaluate potential solutions and perform a series of operations on them in order to find new, better solutions. However, metaheuristics are also composed of different strategies for systematic neighbour exploration which prevent them to stuck in local optima. Therefore, local search methods are often incorporated merely as a step in much larger algorithmic framework. Some of the notable examples of metaheuristics include genetic/evolutionary algorithms [58], tabu search [59, 60], simulated annealing [61], variable neighbourhood search [62], (adaptive) large neighbourhood search, greedy randomized adaptive search procedure (GRASP) [63], and ant colony optimization [64], although many other exist. Given that these metaheuristics are actually a general problem-independent algorithmic frameworks, they have been successfully applied in many different domains. Over the past decade, they have also showed outstanding results in solving various transportation and scheduling problems. In this thesis, we will further promote the use of such tools in both online and offline route planners that use MCS paradigm as a data-collection mechanism. We will mainly rely on some of the well-known heuristics (namely, variable neighbourhood search and greedy randomized adaptive search procedure, maybe some other) and propose our own implementations to solve network routing problems arising from new MCS applications.

1.2. Research Objectives

In this subsection we summarise the main research objectives of this thesis. Each of the objectives is addressed in a separated chapter, while the results of the conducted research were published in at least one international research conference or workshop. Since we aim to combine two very promising Smart Mobility enablers, MCS and journey/route planning, and thus bring together to some extent distinct research challenges, we separate our research objectives into two parts, i.e., research stages: (1) *architectural challenges in designing MCS systems* and (2) *algorithmic challenges in MCS-driven route planning applications*. This way, we aim to demonstrate a logical research progression over time, starting from fundamentals of human-in-the-loop sensing systems such as MCS, to route optimisation algorithms tailored for specific crowdsensing applications that can eventually improve mobility in urban areas.

1.2.1. Architectural challenges in designing MCS systems

***Objective A-1*: Conceptual architecture of MCS systems**

Over the past several years, there have been several attempts to conceptualise general MCS system architectures. However, these attempts were usually limited to certain application scope, oriented to specific types of sensing tasks, or simply too general to address current research trends in the field ([28–30, 65, 66]). Therefore, the first objective of this thesis is to identify the main requirements for MCS and to design a conceptual architecture which would meet these requirements and serve as extensible and open reference model and guideline for implementing MCS systems.

***Objective A-2*: Task allocation methods for improving the reliability of user-generated data**

Allocating sensing tasks to the right set of devices/users to produce the data of desired quality has been a common problem in MCS settings. Recall from the previous section that the type of data participating mobile devices can produce in MCS environments, and the quality in terms of accuracy, latency, and confidence can change constantly due to device mobility, variations in communication channels, and device owners' privacy preferences. Additionally, the anonymous participants in MCS systems very often "contribute" with incorrect and low-quality data, and thus create the need for post-processing data selection methods which would improve the quality of contributed data. As the second objective of this thesis, we aim to reduce such need by proposing recruitment strategy to identify and encourage expert contributors in MCS systems. More precisely, we aim to model participants' spatio-temporal competences by analysing their mobility traces. By allocating MCS tasks only to participant who are familiar with the target location we can significantly increase the reliability of contributed data and reduce the total communication cost.

***Objective A-3*: Data post-processing methods to improve the quality of crowdsensed data**

Although efficient task allocation algorithms can improve confidence and reliability of the user-generated data, they cannot guarantee high level of quality. Therefore, the logical extensions to the previous research objective is to investigate data post-processing methods to further improve the quality of the collected data. However, given the wide range of different data processing and filtering techniques, we focus our research on a specific use case within Smart Mobility domain, and demonstrate how crowdsensed data can be processed and used to predict traffic conditions in urban areas.

1.2.2. Algorithmic challenges in MCS-driven route planning applications

Once we have investigated MCS system architectures and accompanying data collection and processing techniques, we focus on putting the crowdsensed data into motion. We study how the volunteer data can be used to route different actors in traffic (tourists, citizens, freight vehicles, cyclists) and thus to achieve fluent mobility conditions and more balanced city environments. While we mainly focus on algorithms and heuristics, we use real world application examples to showcase the advantages of the proposed algorithms and infrastructures.

***Objective B-1*: Bi-objective route planning to find a trade-off between travellers' needs and city's global mobility objectives**

We first consider a MCS application in which route planning is regulated by city administrators. More precisely, we aim to allow the administrators to formulate mobility policies, e.g. to promote or constrain certain city areas, POIs, or routes between them, and to allocate them to the participating devices, thus influencing travellers' journey planning. At the same time, we want to make use of MCS paradigm and capture users' interests and travel preferences. The overall objective here is to find an algorithm to help both individuals in planning their trips and city governments in achieving sustainable mobility objectives.

***Objective B-2*: Multi-query route planning to balance the use of public traffic infrastructures**

While the previous objective aims to find individual routes which are conformant with the city's global mobility objectives, it does not consider batch queries, and as such it suffers from lack of coordinated planning. In other words, different travellers can be in parallel guided to the same parts of the city, and thus collapse them by creating congestions, which is usually opposite to the city's global mobility objectives. Therefore, as an extension to the previous objective, we aim to balance the use of public traffic infrastructures by developing multi-query route planning algorithms. As a case study we propose another MCS application to improve the circulation of freight vehicles and to avoid congestions in urban freight transport systems. An efficient algorithm needs to takes into consideration

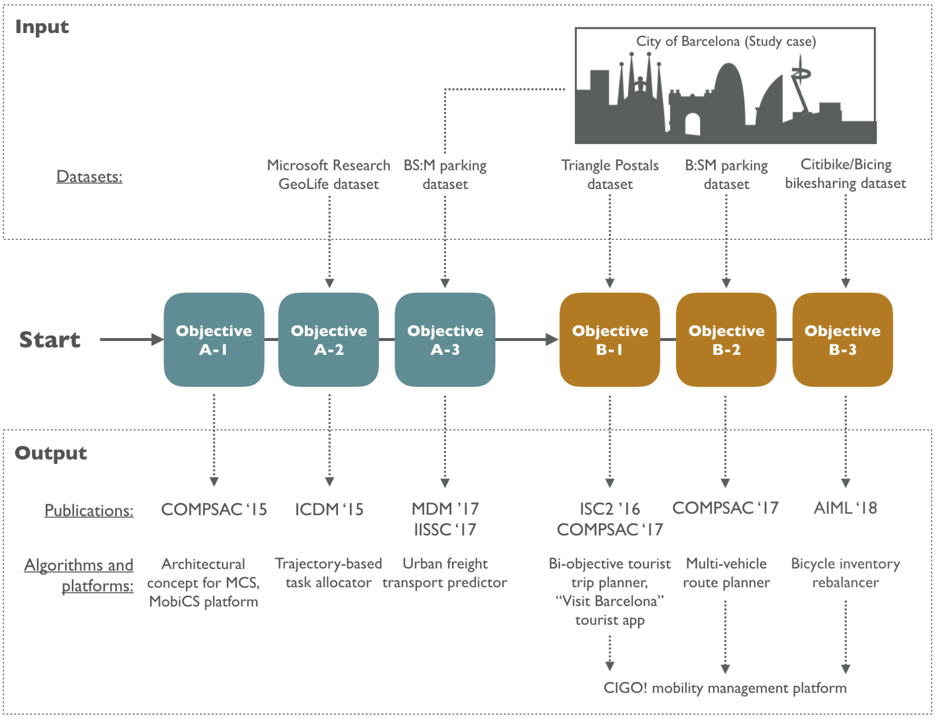

Input
City of Barcelona (Study case)
Datasets:
Microsoft Research GeoLife dataset
BS:M parking dataset
Triangle Postals dataset
B:SM parking dataset
Citibike/Bicing bikesharing dataset
Start
Objective A-1
Objective A-2
Objective A-3
Objective B-1
Objective B-2
Objective B-3
Output
Publications:
COMPSAC '15
ICDM '15
MDM '17 IISSC '17
ISC2 '16 COMPSAC '17
COMPSAC '17
AIML '18
Algorithms and platforms:
Architectural concept for MCS, MobiCS platform
Trajectory-based task allocator
Urban freight transport predictor
Bi-objective tourist trip planner, "Visit Barcelona" tourist app
Multi-vehicle route planner
Bicycle inventory rebalancer
CIGO! mobility management platform

Figure 12: Research path and methodology.

capacity constraints imposed by dense urban environments and to respectively coordinate a fleet of freight vehicles avoiding traffic jams and congestions.

Objective B-3: Online route planning to support demand and supply balancing in shared transport systems

Natural extension to the previous objective, which aims to balance the use of public traffic infrastructures, is to consider a similar route planning approach which would balance the use of moveable assets in shared transport systems. Shared transport systems usually consist of vehicle (e.g., bicycles, electric mopeds) rental stations strategically distributed over the service area. Various factors, such as daily commuting patterns or topographical conditions, can lead to an unbalanced state where the numbers of rented and returned vehicles differ significantly among the stations. This can cause spatial imbalance of the inventory which becomes critical when stations run completely empty or full, and thus prevent users from renting or returning vehicles. Therefore, public vehicles need to be regularly redistributed between the stations using a fleet of freight vehicles. To support this activities, we need to extend the previous objective and develop an online system which will be able to compute optimal routes for a fleet of freight vehicles based on forecast user demand and estimated stations' inventory target levels.

1.3. Research Methodology

The conducted research is empirical, i.e., based on evidence, data, or knowledge/experience from observation and experimentation. The complete research path, along with the employed data sources and outcomes of each step, is shown in Figure 12. Notice that each of the steps in the path corresponds to one of the research objectives covered in the previous section. We started our path by analysing participatory sensing and MCS paradigms with the goal of envisioning the main system requirements based on which we can conceptualise the architecture of MCS systems (*Objective A-1*). As a proof of concept we developed MobiCS system and put it under experiments to prove the potential benefits of the proposed architecture. In the next step we focused on *Objective A-2* and proposed trajectory-based task allocator for reliable MCS systems. In order to evaluate the allocator and to show the advantages of the proposed approach, we used the real-life trajectory dataset from Microsoft Research's GeoLife project. The GeoLife dataset ([67–69]) consists of 17,621 trajectories collected by 182 users in a period of over five years (from April 2007 to August 2012). The trajectories cover a total distance of about 1.2 million kilometers and a total duration of more than 48,000 hours. In the final step of the first research stage dedicated to architectural challenges, we tackled the problem of data post-processing in MCS environments. As a result, we collected and analysed data on urban freight deliveries and parking areas towards an optimized urban freight transport system. We also built a predictor to forecast the availabilities of parking infrastructures, and thus opened up new possibilities for delivery route planning and better managing of public transport infrastructures.

The second stage of our research (*Objectives B-1* to *B-3*) was dedicated to MCS-driven route optimisation algorithms. However, the research activities were considerably application oriented. Since the part of our research work was carried out at Polytechnic University of Catalonia (UPC) in Barcelona, we introduced the city of Barcelona as a study case. Therefore, the defined research problems and proposed solutions arise from real-life contexts. Additionally, we use real-world datasets related to the city of Barcelona to evaluate the proposed algorithms and architectures. In the first step, we focused on *Objective B-1* and proposed crowdsourcing-driven route planning algorithm that can combine users' preferences and city's global mobility policies. To investigate the usability of the proposed algorithm we developed a mobility management platform which allows city administrators to formulate mobility policies, and thus to promote or reduce the traffic of people flows in certain areas. In our pilot project, we upgraded the existing mobile application, *Barcelona Official Guide* provided by *Turisme de Barcelona* and *Triangle Postals S.L.*, and connected it to the developed mobility management platform. In the second step of this research stage, we focused on the drawbacks of the previously proposed bi-objective route planning algorithm, and propose a multi-query route planning approach to balance the use of public traffic infrastructures (*Objective B-2*). Once again, we put the developed algorithm in the real-life context of urban freight transport in Barcelona. The experimental studies conducted on real-world dataset suggest that the proposed planner is able to significantly reduce the amount of traffic in dense urban freight transport networks. Finally, in the last step (Objective B-3) we considered an online route planning approach to support demand and supply balancing in shared transport systems. As a case study we introduced two public bicycle sharing system (Barcelona's Bicing and New York's Citibike). Therefore, we were able to evaluate our demand forecasting and route planning algorithms on real-world dataset from actual shared transport systems.

The results of the second research phase, i.e., set of route planning algorithms and supporting infrastructures, were integrated into CIGO!, MCS-based mobility management platform developed at *Sparsity Technologies* in Barcelona. We discuss CIGO! platform in more details in the last chapter of this thesis.

1.4. Research Contributions

1.4.1. Summary of Contributions

The contributions of this thesis are as follows:

- **Conceptual architecture for MCS systems.** We identify the main requirements for MCS, such as data representation, quality control, trust and reputation management, task allocation, etc., and propose a conceptual architecture which meets these requirements and serve as extensible and open reference model and guideline for implementing MCS systems.

- **Trajectory-based task allocation approach for reliable MCS systems.** We propose a novel trajectory-based approach for task allocation in MCS environments and model participants' spatio-temporal competences by analysing their mobility traces. By allocating MCS tasks only to participant who are familiar with the target location we significantly increase the reliability of contributed data and reduce the total communication cost. We introduce novel metric to estimate participants' competence to conduct MCS tasks and propose fair ranking approach allowing newcomers to compete with experienced senior contributors. Additionally, we group similar expert contributors and thus open up new possibilities for physical collaboration between them.

- **Analysis of crowdsensed mobility data.** We collect and analyse crowdsensed data on urban freight deliveries and parking areas towards an optimized urban freight transport system. We explore the relationship between areas' availability patterns and underlying traffic behaviour in order to understand the trends in urban freight transport. By applying the detected patterns we predict the availabilities of loading/unloading areas, and thus open up new possibilities for delivery route planning and better managing of freight transport infrastructures.

- **Bi-objective route planning algorithm for sustainable city mobility.** We propose a MCS application in which route planning is based on user-generated data, but at the same time regulated by city administrators. The planning problem is modelled as an extension of the mixed orienteering problem and can be controlled by deployment of mobility policies which put restrictions on points of interest and routes between them. We propose an algorithmic approach and a software tool to solve this hard combinatorial optimisation problem using Variable Neighbourhood Search (VNS). The performance of the proposed algorithm and the tool is assessed over a real-life dataset related to the city of Barcelona. Computational results confirm the efficiency of the algorithm and ability to help both individuals in planning their trips and city governments in achieving sustainable mobility objectives.

- **Multi-vehicle route planning algorithm for balancing public freight transport infrastructures.** We study the problem of coordinated route planning with the aim of balancing the use of public traffic infrastructures. As a study case, we consider urban freight transport, i.e., one of the least sustainable components of urban mobility, and propose a novel multi-vehicle route planner that avoids congestions at the loading/unloading areas and minimizes the total duration of the delivery routes. We formally define the problem as a variant of the multiple minimum Hamiltonian Path

Problem (HPP) where each deliverer needs to visit a set of predefined locations while the route choice affects the delay of other deliverers in the system. Since the problem is NP-complete, we propose an affective solution based on combining metaheuristic frameworks - Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighbourhood Search (VNS). Experimental study conducted on a novel real-life dataset collected from citizens and deliverers in the city of Barcelona confirms the effectiveness of the proposed solution.

- **An online framework for adaptive rebalancing of public bicycle sharing systems.** We propose an online framework which takes in consideration the real-time information from shared transport systems to estimate inventory target levels and accordingly to plan and assign routes to rebalancing vehicles. This way, the framework can efficiently adapt to unplanned events (e.g., weather changes, traffic jams, system failures, etc.), and track repositioning progress of each vehicle in order to balance their workload. Given that the optimal inventory levels aim at maximising the time within which the stations remain balanced (i.e., with enough bicycles and empty stands to meet the future user demand), we propose to forecast user demand multiple time steps into the future. Such approach allows for more accurate long-term inventory planning which can lower the number of redistribution runs and thus significantly reduce the operating cost. As a study case, we consider two very different public bicycle sharing systems and collect real-world data to evaluate the proposed solutions.

1.4.2. Software and Platforms

The following software have been developed in the course of this thesis:

- **MobiCS platform[1,2].** To explore the opportunities and to prove the potential advantages of the proposed conceptual architecture of MCS systems, we developed MobiCS, a prototype platform which allows users to formulate and distribute sensing tasks to Android-powered mobile communication devices.

- **Itinerary recommender within *Barcelona Official Guide* mobile application[3,4,5,6].** We upgraded the existing mobile application, *Barcelona Official Guide* provided by *Turisme de Barcelona* and *Triangle Postals S.L.*, by introducing a new feature which allows users to express their preferences and builds custom-tailored travel itineraries.

- **CIGO! mobility management platform[7].** CIGO! platform integrates mobility data from various sources such as Open Data, mobile applications, sensors and government data, allowing for its visualisation and analysis while making it actionable through associated third party mobile applications. We allow injection of the city mobility policies to the third party mobile applications which provide services related to the city resources. In this way we form a value chain which connects different actors (city governments, mobile application providers, POI owners, companies that require

[1] github.com/pmrazovic/mobics

[2] github.com/pmrazovic/mobics-backend

[3] play.google.com/store/apps/details?id=cat.triangle.bcn

[4] itunes.apple.com/us/app/id594589405

[5] github.com/pmrazovic/ls-top-java

[6] github.com/pmrazovic/ls-top-objc

[7] www.smart-cigo.com

logistics in cities, and final users) who both take a part in improving the mobility in urban areas, and benefit from the way mobility policies being executed. The platform implementation was funded by the *frontierCities* accelerator programme under the FP7.

1.4.3. Publications

Results presented in this thesis have been published in international conference and workshop proceedings:

[1] P. Mrazovic and M. Matskin, “Mobics: Mobile platform for combining crowdsourcing and participatory sensing”, in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, IEEE, vol. 2, 2015, pp. 553–562.

[2] V. Granfors, J. Waller, P. Mrazovic, and M. Matskin, “CrowdS: Crowdsourcing with Smart Devices”, in *Internet Computing & Internet of Things, 2018 International Conference on*, CSCE.

[3] P. Mrazovic, M. Matskin, and N. Dokoohaki, “Trajectory-Based Task Allocation for Reliable Mobile Crowd Sensing Systems”, in *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, IEEE, 2015, pp. 398–406.

[4] P. Mrazovic, I. De La Rubia, J. Urmeneta, C. Balufo, R. Tapias, M. Matskin, and J. L. Larriba-Pey, “CIGO! Mobility management platform for growing efficient and balanced smart city ecosystem”, in *Smart Cities Conference (ISC2), 2016 IEEE International*, IEEE, 2016, pp. 1–4.

[5] P. Mrazovic, J. L. Larriba-Pey, and M. Matskin, “Improving mobility in smart cities with intelligent tourist trip planning”, in *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*, IEEE, vol. 1, 2017, pp. 897–907.

[6] P. Mrazovic, B. Eravci, J. L. Larriba-Pey, H. Ferhatosmanoglu, and M. Matskin, “Understanding and Predicting Trends in Urban Freight Transport”, in *Mobile Data Management (MDM), 2017 18th IEEE International Conference on*, IEEE, 2017, pp. 124–133.

[7] B. Kolbay, P. Mrazovic, and J. L. Larriba-Pey, “Analyzing Last Mile Delivery Operations in Barcelona’s Urban Freight Transport Network”, in *Cloud Infrastructures, Services, and IoT Systems for Smart Cities*, Springer, 2017, pp. 13–22.

[8] P. Mrazovic, E. Eser, H. Ferhatosmanoglu, and J. L. Larriba-Pey, “Multi-vehicle Route Planning for Efficient Urban Freight Transport”, in *Intelligent Systems (IS), 2018 9th IEEE International Conference on*, IEEE, 2018.

[9] P. Mrazovic, J. L. Larriba-Pey, and M. Matskin, “A Deep Learning Approach for Estimating Inventory Rebalancing Demand in Bicycle Sharing Systems”, in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, IEEE, 2018, pp. 110–115.

1.5. Thesis Organization

The rest of this thesis is organised as follows. In Chapter 2, we investigate the main requirements of MCS paradigm and propose conceptual architecture to serve as a guideline for implementing MCS systems. We address issues related to data representation, quality control, trust and reputation management, and task allocation. Thereafter, in Chapter

3, we propose the trajectory-based task allocation approach to improve the reliability of user-generated data in MCS environments. Chapter 4 brings the first example of MCS application within Smart Mobility initiative and deals with analysis of crowdsensed traffic data. In Chapters 5, 6, and 7, we introduce three new MCS applications where each of them serves as a case study for several research challenges related to MCS-driven route planning. First, in Chapter 5, we consider the bi-objective route planning algorithm to find a trade-off between satisfying travellers' preferences and meeting city's global objectives towards achieving sustainable urban mobility. Next, in Chapter 6, we address the problem of coordinated route planning with the aim of balancing the use of public traffic infrastructures. In Chapter 7, we introduce time-dependent route planning algorithm for inventory rebalancing problems in bikesharing systems. In Chapter 8, we envision the mobility management platform to integrate the technologies proposed in this thesis. Additionally, we discuss the value chain which connects different actors (city governments, mobile application providers, POI owners, companies that require logistics in cities, and final users) who both take a part in improving the mobility in urban areas, and benefit from more balanced environment.

Most of the content in this thesis are based on material previously published in peer reviewed conferences (see section 1.4.3 for the complete list of produced publications).

Chapter 2

Conceptual architecture of application-independent MCS systems

This chapter is based on materials from the following peer-reviewed papers:

> P. Mrazovic and M. Matskin, "Mobics: Mobile platform for combining crowdsourcing and participatory sensing", in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, IEEE, vol. 2, 2015, pp. 553–562.
>
> V. Granfors, J. Waller, P. Mrazovic, and M. Matskin, "CrowdS: Crowdsourcing with Smart Devices", in *Internet Computing & Internet of Things, 2018 International Conference on*, CSCE.

2.1. Introduction

The mobile sensing infrastructures have been discussed for a number of years in both industrial and research communities, however, the major boost was enabled by recent technological advances in mobile communication. Widely available smartphones, smartwatches, and tablets are replacing dumb sensors which are usually carried or attached to mobile participants in traditional mobile sensing networks. At the same time, smart devices significantly contribute to computation, memory and data resources in mobile sensing infrastructures. Such autonomous devices are programmable and thus offer access to cheap embedded sensors in form of local sensing intelligence. Traditional mobile sensing infrastructures usually do not consider carriers of devices as intelligent participants in a sensing process. However, smart devices can allow users to express their opinions and judgements which can complement captured sensor data. We argue that such information can represent sensing context and serve as a valuable resource for quality control, reputation management and task allocation. Therefore, in this chapter we aim to put humans in the center of a novel sensing infrastructure.

We adopt a concept of human intelligence task (HIT) typically designed by older web-based crowdsourcing platforms to tackle problems that are difficult to solve for a machine but should be easy for humans. Examples of such tasks include image tagging, media transcription, poll answering, etc. We extend HITs with sensing context and physical interaction with mobile devices. This allows us to design a general mobile platform for versatile context-aware sensing and we believe that powerful MCS platform needs to be able both to efficiently deliver sensing tasks to mobile communication devices, and, when needed, to

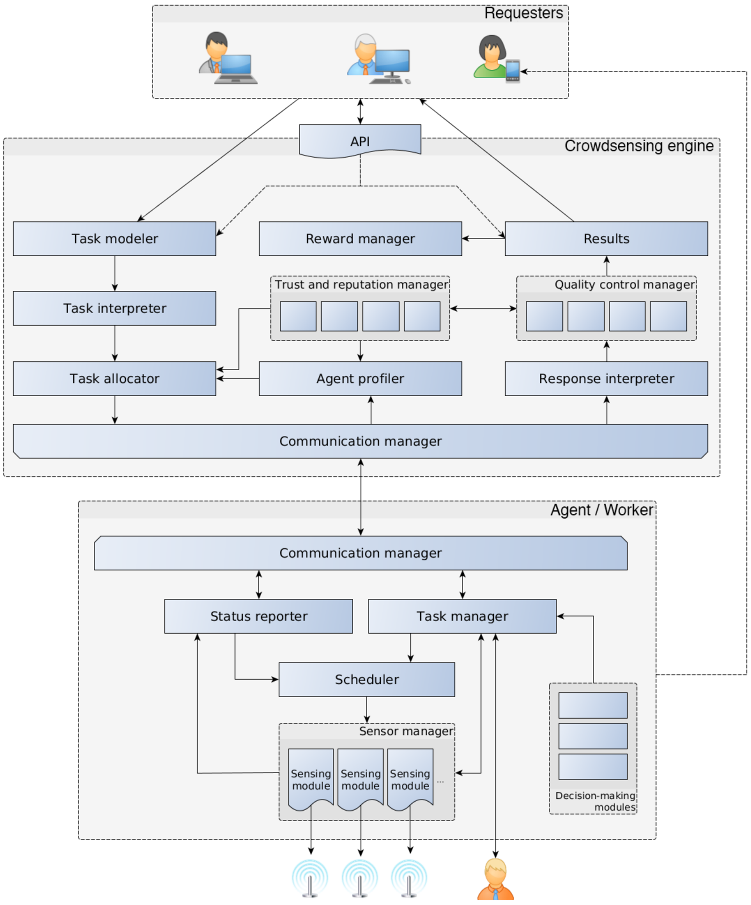


Figure 21: Conceptual architecture of the platform

treat their carriers as intelligent participants in sensing process. The diversity of crowdsensing tasks in our platform presents a great challenge in conceptual architecture modelling. Different types of tasks require distinct approaches to data representation, quality control, trust and reputation management, and task allocation. Proposed conceptual architecture is designed to eventually meet these requirements and serve as extensible and open reference model and guideline for implementing MCS systems, regardless of their specific application. To prove our concept, we implemented platform prototype which supports versatile mobile sensing and conducted several experiments to demonstrate its potential benefits. However, in this thesis we only briefly discuss the implementation, while we refer the interested reader to [70] for more technical details and use case examples.

2.2. Conceptual architecture

There have already been some attempts to conceptualise general crowdsensing system architectures, however these attempts were usually limited to certain application scope, or oriented to specific types of crowdsensing tasks ([65, 71–73]). Korthaus and Dai [74] explored opportunities and challenges of crowdsourcing in the heterogeneous network environments

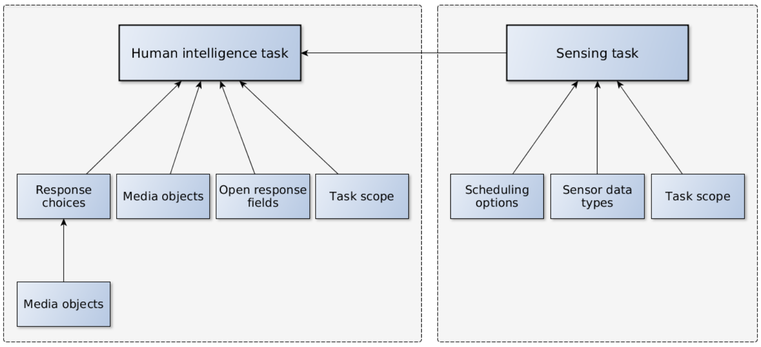


Figure 22: Proposed task model

and proposed a general conceptual architecture for mobile crowdsensing systems. They successfully detected the key roles in such environments, however, they retained at a high-level of abstraction. In this section we propose a detailed general architecture which brings together traditional (web-based) crowdsourcing and participatory sensing.

Figure 21 shows conceptual architecture of our platform. At the top level it represents a crowdsensing workflow. Requesters, as process initiators, formulate tasks (usually from previously defined templates) which are handed to central coordinating component – the crowdsensing engine. The engine translates tasks into persistent objects and distributes them to potential contributors. The engine needs to carry out these responsibilities while assuring high quality information harvesting. In order to support the quality, the engine profiles and rates contributors. Obviously, contributors also play important role in the process. They are programmable intelligent agents with sufficient computational and memory capabilities. Additionally, modern mobile communication devices provide opportunity for user interactions and thus include humans as resourceful participants in the sensing process. Device carriers not only act as intelligent information providers, but also have ability to initiate sensing processes by formulating new crowdsensing tasks. The crowdsensing engine implements APIs which allow agents to act as both data contributors and process initiators (i.e., requesters), and thus create a large pervasive community. Below we use top-down approach to explore components of the proposed system architecture.

2.2.1. Task modelling

One of the major responsibility of the engine is to model a variety of crowdsensing tasks which are stored, distributed, and interpreted by agents. The task model needs to be both extensible and open to new types of sensing paradigms. *Task modeller* and *task interpreter* are two highly dependent components whose responsibility is to meet these requirements. The task modeller uses predefined models to provide guidelines for correct task formulation. Properly formulated tasks are then stored by the task interpreter. The common approach to task modelling is to classify tasks into groups by their content or expected response format. The aim of such classification is to eventually assign a template to each group which will serve as a guide in task formulation. Amazon, for example, defines such templates for several answer formats in their Mechanical Turk crowdsourcing platform. It is important to note that Amazon requires tasks to be presented as HTML forms, which is apparently not applicable in MCS environments where we have plurality of sensor-enhanced participants. We approach this problem by classifying crowdsensing tasks into two main categories – *sensing tasks* and *human intelligence tasks*. Sensing tasks only rely on agents' sensor equipment and do not require active and conscious user involvement. Their purpose is to simply capture sensor data. On the other hand, human intelligence tasks heavily rely on humans as problem solvers, which makes them significantly more complex. However, these two categories are not mutually exclusive since we sometimes want to use sensor data to complement human intelligence tasks with information context and verify the results with sensing data.

Figure 22 shows the proposed task modelling approach. Sensing tasks are defined as a list of requested *sensor data*. For example, initiator can request temperature, location and lightning data at the same time. Additionally, sensing task can be scheduled to collect sensor data over some time period. *Scheduling options* can be used by crowdsourcing engine to coordinate and synchronize agents by sending periodical requests. They can also be distributed to agents and used as pattern for autonomously scheduled sensing. Sensing task can as well be linked to human intelligence task and serve as a sensor context description. Both sensing and human intelligence tasks are often assigned with *scope* to define set of relevant contributors. Location is commonly used to narrow set of potential contributors, but sensor-rich devices allow us be more accurate and define scope using sensor-specific information. For example, we might want to distribute certain task only to agents in quiet surroundings.

While modelling sensing tasks is relatively straightforward, modelling human intelligence tasks requires expressing human judgements and opinions in crowdsensing environments. Experience in traditional crowdsourcing with humans has shown that establishing social belief is more effective when individual human opinions are collected as answers to multiple choice questions. However, it is obvious that we can not always assume prior knowledge of all possible outcomes of the tasks, and therefore it is impossible to formulate everything as a simple multiple choice question. In such cases, it is necessary to use open questions. The open questions assume free-form answers which are usually presented as unconstrained textual content. Unfortunately this kind of questions is less suitable when deriving highly reliable social beliefs. Finally, requester can ask agents to perform certain action which can not be expressed as a direct response. An example of such tasks is "Create user account by

Table 21: Example of task description

Task	*What is the weather like at your current location?*

Task context	Latitude = 45.840196
	Longitude = 15.9643316
	Radius = 20km

Response choices	(1) Sunny, (2) Rainy, (3) Cloudy, (4) Snowing
Open response fields	Express it in your own words
Requested data	Location, Temperature, Lightning
Scheduling options	

Task	Sensing weather data

Task context	Latitude = 45.840196
	Longitude = 15.9643316
	Radius = 20km

Response choices	
Open response fields	
Requested data	Location, Temperature, Lightning
Scheduling options	Sampling frequency = 5 min Interval = [1:00pm, 3:00pm]

following link: www.mobics.com!". In this case it is impossible to directly prove that the agent has taken the requested action. Looking back to Figure 22, human intelligence task model enables different ways of expressing human judgements. For example, it can be formulated with number of *response choices*. Our model also considers extra multimedia data describing both task and response choices, which allows us to formulate such multimedia tasks as transcription from video or sound, image tagging, categorization, moderation, etc. In case of open questions, human intelligence tasks rely on number of *open response fields* which describe free-form responses. For example, if we want agents to tag arbitrary number of objects in an image, we will assign one open response field to each tag. Table 21 contains some examples of tasks formulated using the proposed model.

2.2.2. Response modelling

Similar to the above modelled tasks, contributed data, i.e. task responses, need their own structural representation. We use the same approach and distinguish two types of task contributions – sensor data and human judgements. Each sensing response consists of a number of sensor-specific data entities which together describe agent's surrounding. Additionally, sensor data can complement human judgements and serve as information context which can be used for managing trust relations and assuring quality of contributions. The *response interpreter* parses contributed raw data with respect to described model and constructs logical entities which can be stored in a database.

2.2.3. Task allocation

Task allocator is a coordinating component whose responsibility is to identify a set of task-relevant information providers. The task allocator relies on task-specific scope and takes inputs from trust and reputation manager which greatly affects the quality of the collected data. Task allocation also plays important role in agent load balancing and energy conserving, and thus needs to take into consideration the real-time agent context. Therefore, battery status, processing power, network connectivity and other valuable information provided by *user profiler* are of great use. It is important to remember that primary usage of mobile communication devices is reserved for the owners' regular activities such as making calls, Internet access, etc. Users will participate in crowdsensing process if it does not consume significant resources by preventing access to usual services [75].

In the next chapter, we will learn more about task allocation mechanisms in MCS systems and introduce a novel algorithm which is able to find expert contributors by profiling them based on their movements.

2.2.4. Trust and reputation management

MCS systems should effectively attract new contributors to ensure high level of participation. However, openness exposes systems to erroneous and malicious contributions. Ensuring the quality of contributed data is a great challenge in heterogeneous systems. We assume that managing of trust relations and a reputation mechanism can be seen as an opportunity to provide resistance to errors and malicious behaviours. There has been many works on trust and reputation management in both participatory sensing and traditional crowdsourcing systems ([76–79]), but these works do not support the diversity of crowdsensing tasks. Therefore, it is necessary to design model which will separately evaluate devices as autonomous agents and their carriers as biased human contributors. HaoFan

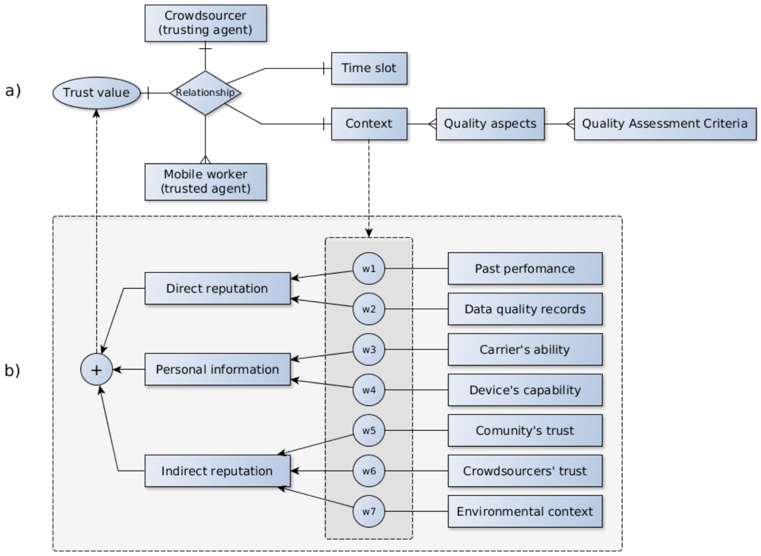


Figure 23: Proposed trust (a) and reputation model (b)

Yang et al. in [79] proposed novel approach to modelling trust and reputation in participatory sensing systems. They emphasize the importance of willingness and precision as two essential factors that affect the data quality. Their model involves both direct and indirect measures of reputation, coupled with personal information to help in classifying individuals. For more details of the proposed model we refer to [79], while here we present an extension and adaptation of their model to our conceptual architecture.

Our architecture requires a flexible trust model which will support various trust assessment approaches related to different types of crowdsensing tasks. Therefore, we extend managed trust relations with the *context* defining the nature of agent and crowdsensing task. For example, we might believe that a certain device will contribute with accurate sensing data, i.e. we trust the agent *in the context of conducting sensing tasks*. However, at the same time we might not believe that device's owner provides trustful information, i.e. we do not trust the agent *in the context of conducting human intelligence tasks*. The context can be usually decomposed into several *quality aspects* for the purpose of quality assessment or measurement. The context of conducting sensing tasks can, for example, be decomposed into "sensor power", "sensor resolution", "maximum sensing range", etc.. With each quality aspect, there is always a set of *quality assessment criteria* that can be used to measure the quality aspect under a particular context [80]. For example we might additionally require that the resolution of the ambient temperature sensor needs to be less than $0.5°C$ before we treat it as an accurate, i.e. trustworthy. Moreover, trust relations need to be time-dependent as our trust to certain agent can change over time. We therefore extend trust relations with the time dimension. Figure 23a illustrates the described trust model.

Assigning values to the trust relations will give us reliable measure of how much we believe that the agent will conduct certain task truthfully. For this purpose a reputation model is needed. The reputation information is usually computed from the past behaviour of the targeted agent, its ratings and recommendations, or from their combination. Our reputation model relies on the one proposed in [79], which derives trust from a combination of direct and indirect reputation measures, coupled with personal information (Figure

23b). Direct reputation is derived from the previous data quality records and agent's past performance. Indirect reputation includes more subjective measures such as requesters' and community's opinions. We introduce *environmental context* as an additional source of indirect reputation which allows us to assess the influence of the current agent's surrounding to the quality of contributions. In this manner sensing context of contributions is used to learn agents' behaviour patterns. For example, it is possible that some agent usually contributes low quality data in some detectable cases, e.g. at low temperature. This extension makes our model more dynamic and context-aware which is desirable in unstable and heterogeneous networks such as MCS environments. Finally, the direct and indirect reputation is coupled with agent's personal information, e.g. device's capability and carrier's ability and competence, which is usually collected during registration process.

Similarly to [79], we assign a weight to each component in our reputation model. However, the weights in the model present the quality aspects which define relevance of reputation sources in each trust context. For example, it is obvious that we put more weight to device's capability than to carrier's ability when assessing trust in the context of conducting sensing tasks. Similarly, we will value carrier's competence more in the context of conducting human intelligence tasks. Figure 23 gives an overview of the proposed trust and reputation model.

2.2.5. Agent profiling

Agent profiler is a component playing important role in various crowdsensing mechanisms such as task allocation, trust assessment, reputation management, quality control etc. Its primary responsibility is to construct agent profiles by analysing agents' contributions and sensor context data. Inferring surrounding context and activities by discovering patterns or correlations in large quantities of data is mainly a machine learning problem. We emphasize the importance of this component by accommodating it into our conceptual architecture. As shown in Figure 21, the agent profiler has a central position in the crowdsensing engine. Apart from providing categorized data to task allocator and trust and reputation manager, the agent profiler directly handles the stream of context data from agents.

2.2.6. Quality control

The main purpose of agent profiling, as well as trust and reputation management, is to eventually assure better quality of contributed data. The diversity of crowdsensing tasks in our approach involves different ways for controlling quality of contributions. The *quality control manager* is presented as a modular component (see Figure 21). The quality of sensor data in participatory sensing networks is usually assured by applying different outlier detection algorithms, while the traditional crowdsourcing systems use weighted voting as a more effective tool for improving the result quality of human intelligence tasks (Figure 24). Additionally, we introduce the *sensing context information* as important source which can directly inform about the contribution's relevance to the certain task's scope. For example, we can ask contributors to describe traffic situation at a specific location, and then we can value more the data coming from locations closer to the requested one. This simple extension makes our quality control manager highly applicable in context-aware crowdsensing systems. We also emphasize the importance of modularity in quality control management. It allows us to easily extend the quality control with more advanced mechanism such as gold standards, peer consistency [81], confusion matrices [82, 83], etc.

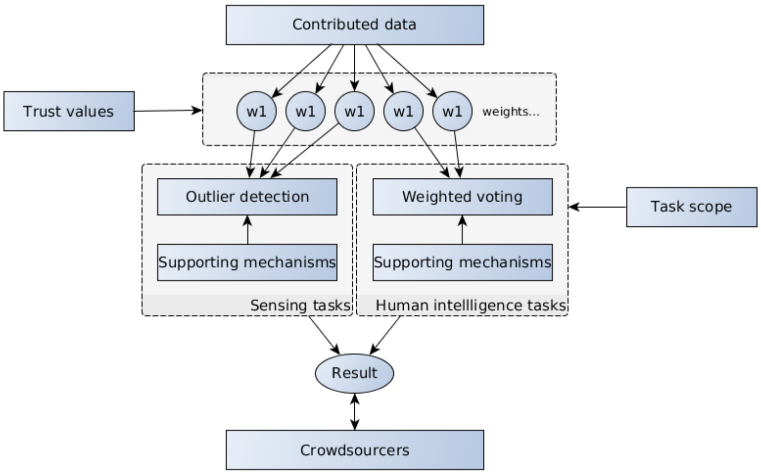


Figure 24: Quality control manager

2.2.7. Reward management

While participating in crowdsensing tasks, mobile communication devices consume their own resources such as battery and computing power. In addition, device carriers are exposed to potential privacy threats by voluntary sharing sensor data and personal information. Therefore, a user should always receive a satisfying reward to compensate resource consumption and potential privacy breach. An incentive mechanism is needed to assure adequate user participation which is necessary for successful sensing. While the most of web-based crowdsourcing platforms offer fixed micropayments to attract contributors, more advanced platforms employ game-theoretical approach to maximize the utility of the platform and still attract enough participants.

2.2.8. Agents

Our MCS concept treats programmable mobile communication devices and their carriers as highly intelligent and resourceful agents. This is why task handling, capturing sensor data and user interaction require an intelligence on the client side . In Figure 21 we identify main components implementing agents as autonomous information sources. Once again, our model is designed with respect to different types of crowdsensing tasks. *Task manager* handles allocated tasks and coordinates device carrier and *sensor manager*. The sensor manager is modular component whose responsibility is to capture sensor data. Modules are autonomous and sensor-specific which allows developers easily upgrade agents with new sensing capabilities. Additionally, sensing tasks can be delivered with scheduling options which are then interpreted and processed by *task scheduler*. In addition to task processing, agent periodically reports its status to crowdsensing engine in the form of sensor data. This data is usually processed by agent profiler which infers agent activities and surrounding context.

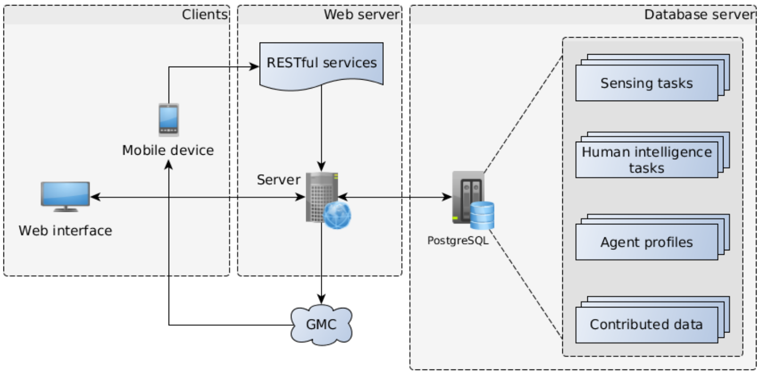


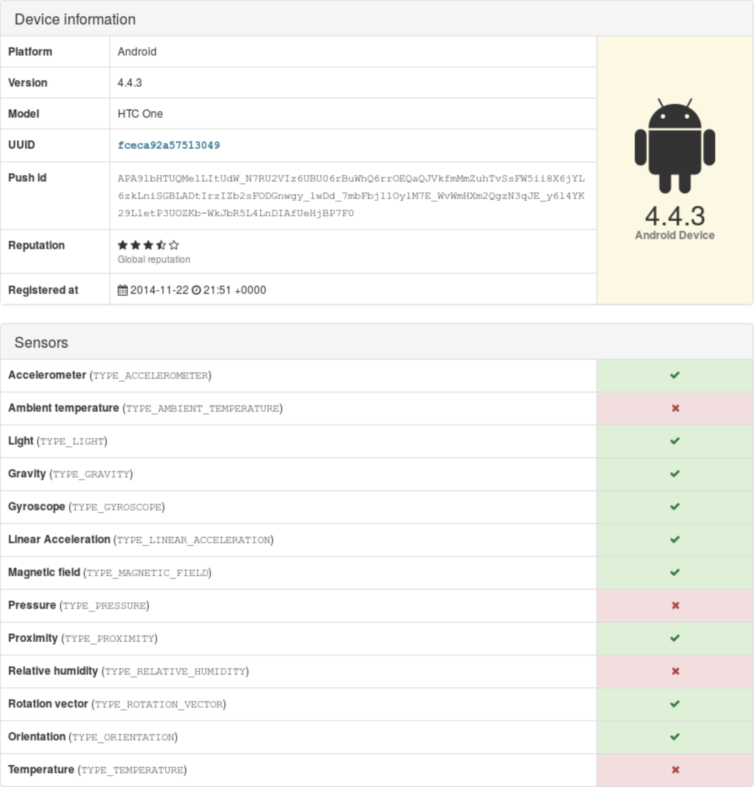
Figure 25: MobiCS architecture

2.3. MCS platform prototype

To investigate the advantages of the proposed conceptual architecture, we developed *MobiCS*, a prototype platform for versatile MCS. MobiCS allows requesters to formulate and distribute both sensing and human intelligence tasks to Android-powered mobile communication devices. It consists of server-side crowdsensing engine, client web interface and client mobile application (Figure 25). Requesters can use both mobile and web application to formulate and publish arbitrary tasks associated with specific sensor context information. Server-based engine stores tasks in database and distributes them to potentially relevant contributors. Mobile application handles incoming tasks by capturing sensor readings and carriers' judgements and actions. In this section we only briefly discuss implementation in order order to illustrate how the the proposed conceptual architecture can be realised in practice. For more technical details and use case examples, we refer the interested reader to [70].

The central and coordinating component of the system is a server-based crowdsensing engine. The server-side is a Ruby-based platform consisting of a PostgreSQL database and several RESTful web services used by mobile and web applications. The database includes collections of all registered agents, i.e. mobile communication devices, crowdsensing tasks and agents' contributions. Each participant in MCS system needs to be properly registered with enough information to be later used for successful allocation of crowdsensing tasks. Therefore, our implementation collects information about device model, operating system and built-in sensors. Each sensor is defined by Android sensor type constant, manufacturer, version, power, maximum range and resolution. Additionally, device's universally unique identifier (UUID) is adopted for identification purposes. This information about device's capabilities is collected during registration process but it can also be updated later during the system operation. Example screenshots of web interface in Figure 26 show information collected during registration process for device HTC One M7.

Before storing registration information in database, the engine register device with appropriate push notification services to enable sending tasks directly to applications on mobile devices. The engine will register devices with Google Cloud Messaging service (GCM). Push notification services return device token to be used for identification purposes in push communication protocols. In Figure 26 the device token is shown in `Push id` field. After successful registration, devices are assigned with the system's unique ID which makes them ready to communicate and participate in MCS process. Tasks are directly distributed



Device information

Platform	Android	4.4.3 Android Device
Version	4.4.3	
Model	HTC One	
UUID	fceca92a57513049	
Push id	APA91bHTUQMe1LItUdW_N7RU2VIz6UBU06rBuWhQ6rrOEQaQJVkfmMmZuhTvSsFW5ii8X6jYL6zkLniSGBLADtIrzIZb2sFODGnwgy_1wDd_7mbFbj11Oy1M7E_WvWmHXm2QgzN3qJE_y614YK29LletP3UOZKb-WkJbR5L4LnDIAfUeHjBP7F0	
Reputation	★★★⯪☆ Global reputation	
Registered at	2014-11-22 21:51 +0000	

Sensors

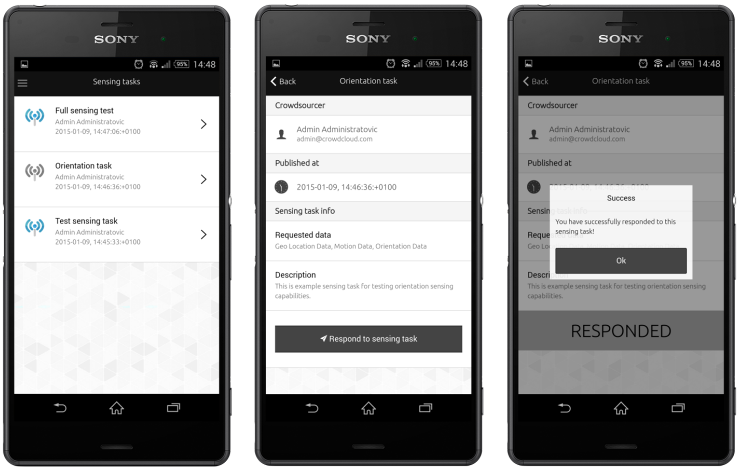
Accelerometer (TYPE_ACCELEROMETER)	✓
Ambient temperature (TYPE_AMBIENT_TEMPERATURE)	✗
Light (TYPE_LIGHT)	✓
Gravity (TYPE_GRAVITY)	✓
Gyroscope (TYPE_GYROSCOPE)	✓
Linear Acceleration (TYPE_LINEAR_ACCELERATION)	✓
Magnetic field (TYPE_MAGNETIC_FIELD)	✓
Pressure (TYPE_PRESSURE)	✗
Proximity (TYPE_PROXIMITY)	✓
Relative humidity (TYPE_RELATIVE_HUMIDITY)	✗
Rotation vector (TYPE_ROTATION_VECTOR)	✓
Orientation (TYPE_ORIENTATION)	✓
Temperature (TYPE_TEMPERATURE)	✗

Figure 26: Registered device information

to selected mobile agents via push messages, but can be also pulled on demand via web services.

All the tasks in our MCS system are formulated through the web and mobile applications. However, server-based engine plays important role in task modelling as well. Domain model of our solution is designed by following the concept proposed in 2.2.1. Each task is specified by a set of properties including task ID, title, description, requester, list of requested sensor readings, task scope and other information which closely describes specific types of crowdsensing tasks. Both sensing and human intelligence tasks are modelled to be stored in database and they are distributed as push messages among the crowd. The following JSON content presents an example of human intelligence task distributed to agents via push message.

```
"id": 3,
"created_at": "2015-01-08T14:14:01.186Z",
"published_at": "2015-01-08T14:18:16.963Z",
"question": "What is the weather like
             at your current location?",
"description": "Task is published for integration
                testing purposes...",
"choices": [
  { "description": "Sunny", "id": 9 },
  { "description": "Cloudy", "id": 10 },
  { "description": "Rainy", "id": 11 },
  { "description": "Snowy","id": 12 } ],
```



Full sensing test
Admin Administratovic
2015-01-09, 14:47:06:+0100

Orientation task
Admin Administratovic
2015-01-09, 14:46:36:+0100

Test sensing task
Admin Administratovic
2015-01-09, 14:45:33:+0100

Crowdsourcer
Admin Administratovic
admin@crowdcloud.com

Published at
2015-01-09, 14:46:36:+0100

Sensing task info

Requested data
Geo Location Data, Motion Data, Orientation Data

Description
This is example sensing task for testing orientation sensing capabilities.

Success
You have successfully responded to this sensing task!

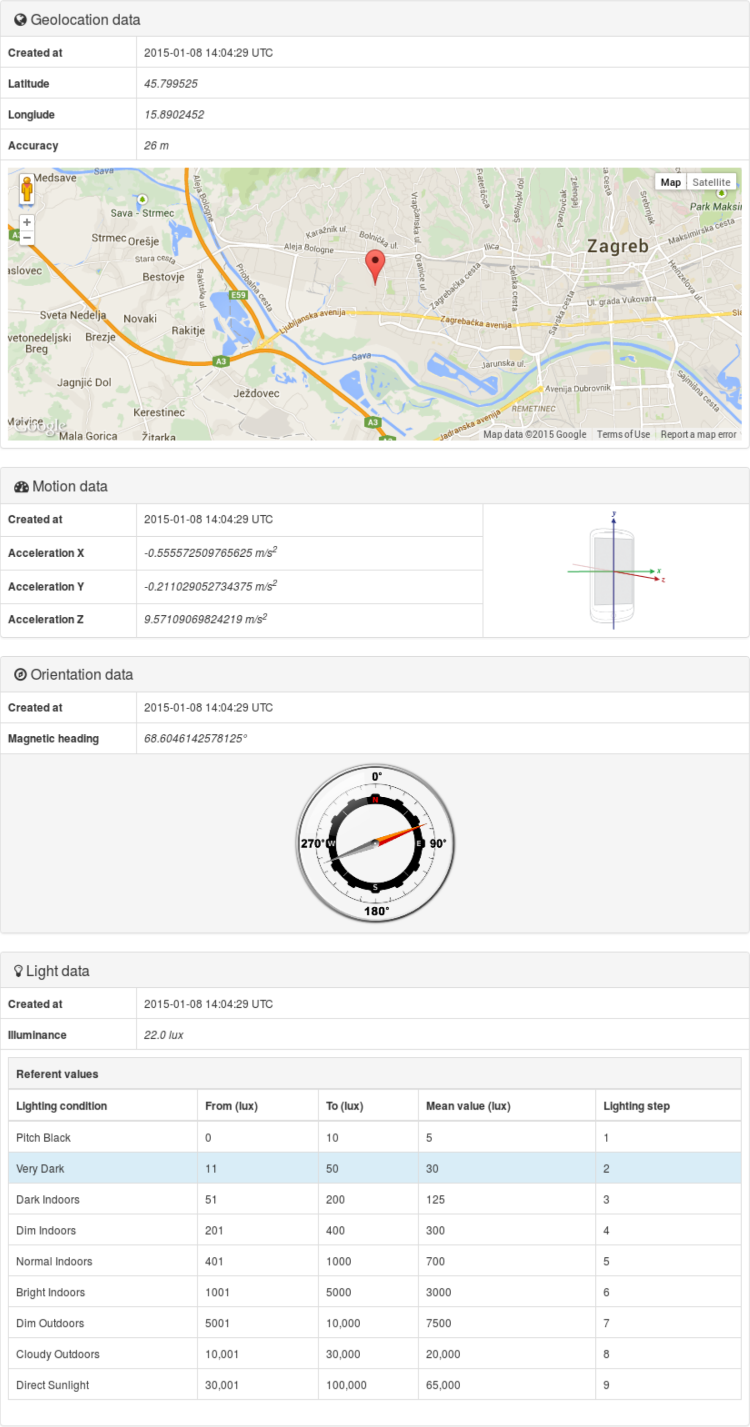
Figure 27: Cross-platform mobile application

```
"context_data_types": [
  "GeoLocationData", "MotionData",
  "OrientationData", "LightData" ],
"crowdsourcer_email": "petar@mobics.com",
"crowdsourcer_name": "Petar Mrazovic",
"responded": false
```

The push message contains all information needed by the devices to act as autonomous agents and to contribute by conducting the crowdsensing task. Above example describes simple human intelligence task with specified context sensor readings (GeoLocationData, MotionData, OrientationData, LightData). Upon obtaining requested data, agents respond to allocated tasks by calling designated web service. The example below shows response data collected by one of the testing devices (Sony Xperia Z3).

```
"device": { "uuid": "1b42bd6aea20ccde"},
"task_id": 3,
"choice_id": 10,
"sensing_response_items":[
  { "LightData": {"lumen": 18} },
  { "OrientationData":
    { "magnetic_heading": 69.1845703125 } },
  { "MotionData":
    { "acceleration_x": -1.9742431640625,
      "acceleration_y": 7.86016845703125,
      "acceleration_z": 5.12249755859375 } },
  { "GeoLocationData":
    { "latitude": 45.79959,
      "longitude": 15.890267,
      "accuracy": 24.041000366210938 } }]
```

Client mobile application captures sensing context and enables carriers to express their judgements and opinions. We developed a mobile application using cross-platform application development framework Apache Cordova (Figure 27). Even though the implementation heavily relies on web technologies (HTML5, CSS and JavaScript), each sensing module is developed as native plug-in which makes the implementation open to new types of sensors and mobile platforms. We developed native plug-ins for capturing device motion and orientation, ambient light and temperature, surrounding noise level and global geographic



Geolocation data

Created at	2015-01-08 14:04:29 UTC
Latitude	*45.799525*
Longitude	*15.8902452*
Accuracy	*26 m*

Motion data

Created at	2015-01-08 14:04:29 UTC
Acceleration X	*-0.555572509765625 m/s^2*
Acceleration Y	*-0.211029052734375 m/s^2*
Acceleration Z	*9.57109069824219 m/s^2*

Orientation data

Created at	2015-01-08 14:04:29 UTC
Magnetic heading	*68.6046142578125°*

Light data

Created at	2015-01-08 14:04:29 UTC
Illuminance	*22.0 lux*

Referent values

Lighting condition	**From (lux)**	**To (lux)**	**Mean value (lux)**	**Lighting step**
Pitch Black	0	10	5	1
Very Dark	11	50	30	2
Dark Indoors	51	200	125	3
Dim Indoors	201	400	300	4
Normal Indoors	401	1000	700	5
Bright Indoors	1001	5000	3000	6
Dim Outdoors	5001	10,000	7500	7
Cloudy Outdoors	10,001	30,000	20,000	8
Direct Sunlight	30,001	100,000	65,000	9

Figure 28: MobiCS's web interface

position. The application has been initially tested on several Android-powered mobile devices (HTC M8, Sony Xperia Z3, Samsung Galaxy S5, LG G3, etc.), but still might require more extensive testing and evaluation. The application allows users to act as sensing initiators and contributors at the same time. The exposed web services enable mobile agents to register in MCS system, pull published tasks, create new tasks and respond to accepted tasks. The same web services are used by web applications which provide additional functionalities such as filtering, statistical overviews, message sending etc. Figure 28 shows a sensing context presented in web interface.

2.4. Conclusion

The proposed conceptual architecture is independent of any specific application domain, and as such can be used as an extensible reference model and guideline for implementing any MCS system. Although we focus on smart mobility applications, understanding the general MCS paradigm is of great importance. Throughout the thesis will touch some of the introduced components and discuss them in more details (e.g., task allocation and quality control), but we will mainly concentrate on contributed data and study how it can be used to solve practical mobility problems in urban areas. However, before doing so, in the next chapter we study how to use MCS to collect this data by allocating crowdsensing tasks only to reliable contributors.

Chapter 3

Trajectory-based task allocation approach for MCS systems

This chapter is based on materials from the following peer-reviewed paper:

> P. Mrazovic, M. Matskin, and N. Dokoohaki, “Trajectory-Based Task Allocation for Reliable Mobile Crowd Sensing Systems”, in *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, IEEE, 2015, pp. 398–406.

3.1. Introduction

In the previous chapter we introduced an application-independent conceptual architecture for MCS systems which puts humans in the center of sensing process and allows for their implicit and explicit participation. How to optimize the combination of human and machine intelligence is another important and challenging research problem for MCS. In this chapter we contribute to this problem by tackling one of the main issues of people-centric infrastructures – the reliability of contributed data. The anonymous participants in MCS systems very often "contribute" with incorrect and low-quality data, and thus create the need for post-processing data selection methods which would improve the quality of contributed data. Here, we try to reduce such need by proposing recruitment strategy to identify and encourage expert contributors in MCS systems.

Expert user selection is a common technique for assuring higher quality of collected data in crowdsourcing settings, but however it still has not been sufficiently explored in MCS. The sensing tasks and campaigns in MCS systems are spatio-temporal correlative, i.e. they typically refer to a certain location and time frame. By allocating such tasks only to participant who are familiar with the target location and within specified time frame, we can significantly increase the reliability of contributed data and reduce total communication cost. Therefore a recruitment framework in MCS should rely on participants’ past experience with visited locations and discover their regularities. In this chapter we will model participants’ spatio-temporal competences by analysing their mobile traces (GPS trajectories) and learning their individual behaviour patterns. Unlike traditional recruitment approaches in participatory sensing where participants’ current location is only criteria, we also recruit participants who often visit specified location or spend there significant time. Additionally, we propose fair ranking approach allowing new community members to compete with experienced senior contributors. Finally, our framework groups similar expert contributors and opens up new possibilities for physical collaboration between them.

Despite all the studies on MCS and participatory sensing, only few works have focused on task allocation and recruitment of workers. Reddy et al. [84] developed a recruitment framework to enable requesters in participatory sensing environments to identify best suited contributors based on their geographic and temporal availability. They transform participants' raw mobility information (GPS trajectories) into context-annotated mobility profiles that model transportation mode, location, and time. The proposed recruitment engine compares participants' profiles to determine which individuals maximize coverage over a specific area and time period. However, the coverage is evaluated on predefined area split into a number of spatial blocks. Such pre-partitioned grids are not scalable and limit sensing systems only to predefined geospatial areas. Therefore, in our work we replace grid partitioning with hierarchical organized regions of interest.

Similar to our work, Hao et al. [85] propose trajectory-based recruitment strategy of social sensors for participatory sensing. They collect participants' GPS trajectories within certain time frame and represented them with time-series of third-order tensors. Furthermore, the dynamic tensor analysis algorithm (DTA) algorithm is adopted to learn this time series tensor and predict the future trajectory information. Again, the proposed recruitment strategy works within a given monitoring area pre-partitioned into a number of cells. Additionally, prediction of users' locations and inferring their availability is not enough for efficient task allocation in MCS systems. Location-based MCS tasks do not always require real-time information from the field (opposed to participatory sensing), and thus we also need to model users' past experience and general familiarity with visited locations.

Cardone et al. ([66, 86]) started the ParticipAct Living Lab testbed, an experiment at the University of Bologna involving 300 students for one year in crowd sensing campaigns where it was possible passively access smartphone sensors and require active user collaboration. To the best of our knowledge this is the first participatory sensing infrastructure to consider humans as intelligent problem solvers and involves them into information retrieval processes. Cardone et al. in [66] propose an algorithm to autonomously assign tasks to well-suited participants. They propose three different task assignment policies: random policy, attendance policy, and recency policy. The random policy selects random group of available participants. The attendance policy relies on knowledge about the time previously spent by participants in the task area. Finally, the recency policy favours participants who have more recently traversed the sensing task area.

Our task allocation approach relies on discovering participants' geospatial regions of interest. In their inspiring work Li et al. [87] mine similarity between users based on their geographic location histories, considering the sequential, hierarchical and popularity properties of location. They show great advantages of hierarchical organized regions of interest for inferring similarities between users. We adopt their concept of hierarchical (multi-level) framework and define our own tree structure to infer participants' spatio-temporal competences. We treat regions of interest as tree nodes and employ a tree climbing approach to find the most suited contributors from a large set of voluntary participants.

3.2. Framework overview

As shown in Figure 31, our task allocation framework contains the following six steps:

Step 1: Trajectory data collection. The trajectory data from volunteer participants is the only input to our system. Luckily, the advances in positioning technologies make it possible to seamlessly collect a large number of spatio-temporal points from modern communication devices.

Step 2: Stay point discovery. The discovery of participants' regions of interest involves obtaining the physical locations which matter to their daily life and routines. Here, we refer to them as *stay points*. In the second step we search for the stay points in previously collected trajectory data using spatio-temporal density-based clustering algorithm.

Step 3: Stay point clustering. Using OPTICS clustering algorithm [88], we hierarchically cluster stay point dataset from previous stage into a number of geospatial regions in a divisive manner. The similar stay points from various participants are assigned to the same *stay regions* on different layers in obtained cluster hierarchy.

Step 4: Data tensorization. The 3-dimensional correlations between participants, stay regions and time of day are represented with third-order tensors. Stay region hierarchy obtained in the previous step is used to build a hierarchy of third-order tensor. Correlations on each level in stay region hierarchy are represented with one tensor.

Step 5: Assessment of spatio-temporal competences. Using tensor hierarchy we estimate participants' familiarity with each level's stay regions in different times of day by computing *competence scores*. We take in consideration personal significance of the stay region, but also participant's amount of experience with stay region in certain time period.

Step 6: Ranking and grouping. We use computed competence scores to rank participants and identify experts for each stay region in different times of day. Additionally, we compute cosine similarity between experts and group them into collaborative social network. Finally, we allocate task to most suited group of experts with respect to assigned stay region and time of day.

In the next section we describe in more details each stage of proposed task allocation framework.

3.3. Framework details

The steps involved in proposed task allocation approach are detailed in this section. We place extra emphasis on describing the underlying algorithms. We also explain the inputs and outputs of each stage along with tuning parameters which affect the final outcome of the task allocation process.

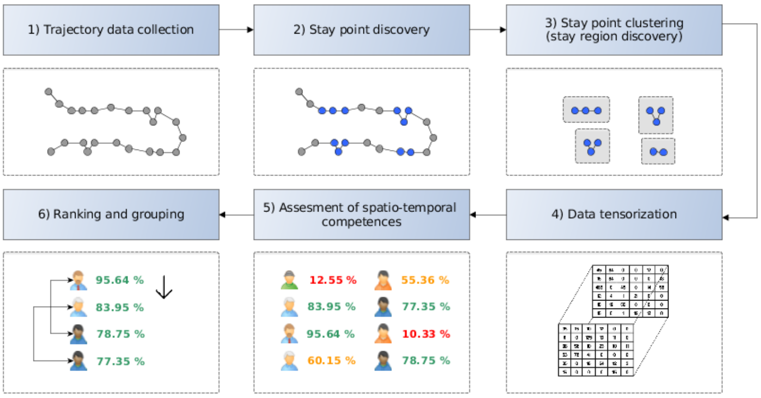


Figure 31: Overview of proposed trajectory-based task allocation framework

3.3.1. Trajectory data collection

The research studies have shown high regularity in the human mobility behaviours, suggesting that most individuals follow a simple and reproducible pattern [89]. In this paper we rely on high resolution positioning data to build participants' mobility profiles and to find coverage areas in MCS systems. Global Positioning System (GPS) allows us to track the movements and obtain trajectories that represent spatial paths followed by individuals through certain time periods.

Definition 3.3.1 (GPS trajectory) *GPS trajectory is a chronologically ordered sequence of spatio-temporal points* $(p_0, ..., p_N)$ *where* $p_i = \langle lat_i, lng_i, t_i \rangle$ *with* t_i *as a time point and* (lat_i, lng_i) *as a GPS point.*

3.3.2. Stay point discovery

Discovering regions of interest, i.e., coverage areas, starts with mining important locations from GPS trajectories. We employ spatio-temporal density-based algorithm to find personal places of interest. The algorithm detects sequences of spatially close points in GPS trajectories where individuals stay for a reasonable time. We refer to them as *stay sequences* and treat them as locations of personal significance [67]. Participants are usually familiar with these locations and can contribute with valuable information when conducting location-based MCS tasks.

The proposed algorithm is based on density-based clustering. For more details we refer reader to [90] where representative algorithm DBSCAN was introduced. We have chosen density-based clustering approach for a number of its advantages. The extraction of stay sequences in our approach depends on only two input parameters, a time threshold (ε_t) and a distance threshold (ε_d). Second, the algorithm does not require prior knowledge of the number of expected stay sequences. Finally, it can discover clusters of arbitrary shapes and easily detect outliers.

We adopt the concept of neighbourhood from traditional density-based clustering approach and extend it by introducing temporal information. We define the spatio-temporal neighbourhood (*ST-neighbourhood*) of a point p_k as a set of points in the trajectory whose spatial and temporal distances from p_k are less than predefined thresholds.

Definition 3.3.2 (ST-neighborhood) *Let* $(p_0, ..., p_k, ..., p_N)$ *be a GPS trajectory, where* $p = \langle lat, lng, t \rangle$. *The ST-neighbourhood of point* p_k *is the set of points* p_i *such that* $distance((lat_i, lng_i), (lat_k, lng_i)) < \varepsilon_d$ *and* $|t_k - t_i| < \varepsilon_t$, *where* ε_d *and* ε_t *stand for distance and time threshold.*

By introducing the time threshold we can detect and ignore outliers in a subsequence of spatially close GPS points. Figure 32 illustrates ST-neighbourhood of point p_4. In the first example ST-neighbourhood is consisted of points p_3, p_5 and p_7. Since p_7 was captured not long after p_4, i.e. $t_7 - t_4 < \varepsilon_t$, distant point p_6 is treated as outlier. However, in the second example temporal distance between p_4 and near p_8 is greater than ε_t, and thus distant p_6 and p_7 are not treated as outliers and p_8 belongs to another visit.

Below we use the ST-neighbourhood to formally define the stay sequence as the union of ST-neighbourhoods and describe how we use them to compute the stay points.

Definition 3.3.3 (Stay sequence) *A stay sequence* Q *of a trajectory* T *with respect to* ε_d *and* ε_t *is a non-empty set of chronologically ordered trajectory points, such that* $\forall p_i, p_j \in T$ *if* $p_i \in Q$ *and* $p_j \in$ *ST-neighbourhood(*p_i*) then* $p_j \in Q$.

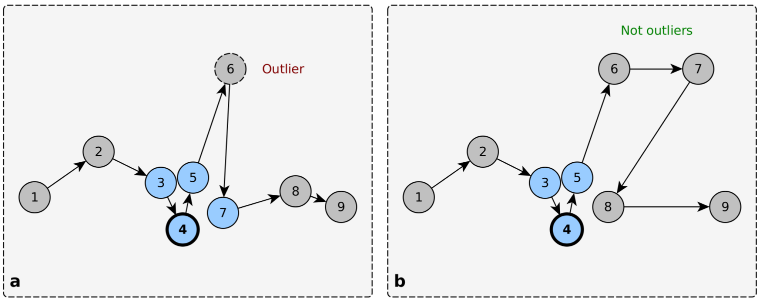


Figure 32: Examples of ST-neighbourhood

Definition 3.3.4 (Stay point) *Stay sequence* $Q = \{p_i | 0 \leq m \leq i \leq n \leq N\}$ *determines the stay point* $s = (lat, lng, t_a, t_l)$ *where* $lat = \frac{1}{n-m} \sum_{i=m}^{n} lat_i$*,* $lng = \frac{1}{n-m} \sum_{i=m}^{n} lng_i$ *stand for the average latitude and longitude of the points in* Q*,* $t_a = t_m$ *is the arriving time at* s *and* $t_l = t_n$ *is the leaving time.*

The Algorithm 1 summarises our approach to extract stay sequences and compute stay points. Obtained stay points usually occur when participants enter a building and lose a satellite signal over a time interval $t \geq \varepsilon_t$ until coming back outdoors, or when they exceed a time limit ε_t at a certain geospatial area. Therefore, the stay points represent both indoor and outdoors locations of interest.

3.3.3. Stay point clustering

In the third step we put together detected stay points and hierarchically cluster them into geospatial areas. By assigning similar stay points from various participants to the same clusters, we can easily discover regions of interest which we refer to as *stay regions*. They represent geospatial areas where participants spend significant amount of time. Obviously these regions include a dozens of points of interest such as restaurants, supermarkets, hotels, fuel stations, etc.

Definition 3.3.5 (Stay region) *Stay region* R_C *is a geospatial area that contains all of the stay points* $s \in C$ *where* C *is a geographically bounded cluster of stay points.*

The relationship between stay regions can be read out from the tree structure obtained as a result of hierarchical clustering. Here we refer to it as *stay region tree (SRT)* and use it as a framework to effectively model participants' competences to conduct location-based MCS tasks. Task allocation problem boils down to assigning the task to the right stay region and finding local participants who are familiar with the location and/or the surrounding area. Later in this section we explain how we use SRT to build a hierarchy of third-order tensor and represent correlations between participants, stay regions and time of day.

Definition 3.3.6 (Stay region tree (SRT)) *Stay region tree (SRT) is a tree representation of a hierarchical clustering whose nodes represent clusters of stay points, i.e., stay regions.*

Selection of the right hierarchical clustering algorithm significantly affects the structure of SRT. We have chosen OPTICS algorithm which generalize density-based clustering and thus is capable of detecting clusters with irregular shapes [88]. It creates an augmented

Algorithm 1: Stay point discovering

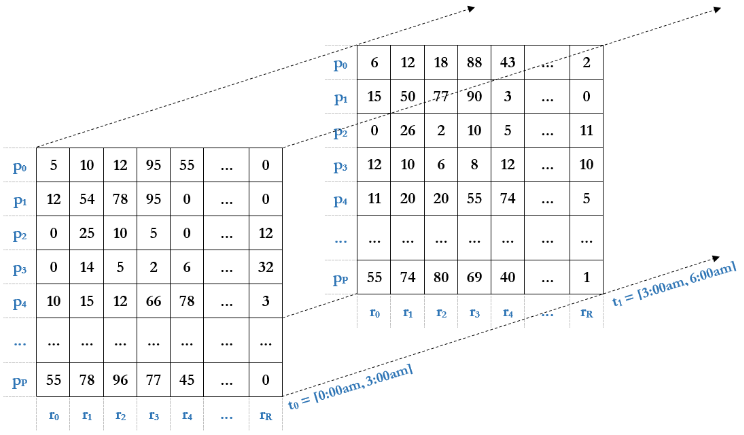
```
Input: set of GPS trajectories $\mathcal{T}$, distance threshold $\varepsilon_d$, time threshold $\varepsilon_t$
Output: set of stay points $\mathcal{S}$
$\mathcal{S} = \emptyset$
for $T_i \in \mathcal{T}$ do
    for GPS point $p_i \in T_i$ do
        if $p_i$ not processed then
            mark $p_i$ as processed
            neighbors = ST_neighborhood($p_i, \varepsilon_d$)
            for $p_j \in$ neighbors do
                for $p_k \in$ ST_neighborhood($p_j, \varepsilon_d$) do
                    if $p_k$ unprocessed then
                        neighbors $\leftarrow p_k$
                    end
                end
            end
            mark $p_j \in$ neighbors as processed
            set neighbors as stay sequence $Q$
            if duration($Q$) $\geq \varepsilon_t$ then
                compute stay point $s$ from $Q$
                $\mathcal{S} \leftarrow s$
            end
        end
    end
end
```

ordering of the points in dataset representing its density-based clustering structure. The output of the algorithm is *reachability plot*, i.e. a bar plot showing clusters as valleys in the plot. Hierarchical structure of the clusters can be obtained easily from reachability plot. We implement automatic cluster extraction algorithm proposed by Sander et al. in [91]. The proposed method tries to find local maxima in reachability plot to separate valleys which represents density-based clusters. Details of implementation has been introduced in [91].

3.3.4. Data tensorization

By introducing time domain into our framework, we can more accurately asses participant's ability to conduct MCS task at different times of day. To put it differently, participants can be within different time frames in varying degrees familiar with the target stay region. For example, participants who use public transportation are usually at the same stations every morning. This means they are familiar with these locations, but only in certain time frames. It is possible that they have never even been there at the different time, and thus can not provide valuable information for the rest of the day. This is why we introduce time domain and model participants' competences for each stay region within different time frames. More precisely, we split day into 8 *time slots* (0:00 am - 3:00 am, 3:00 am - 6:00 am, 6:00 am - 9:00 am, etc.) and use them to asses participants' time-dependent abilities to conduct MCS tasks.

We use SRT obtained in the previous step as a framework to build a hierarchy of third-order tensors which represent 3-dimensional correlations between participants, stay regions



p_0	5	10	12	95	55	...	0
p_1	12	54	78	95	0	...	0
p_2	0	25	10	5	0	...	12
p_3	0	14	5	2	6	...	32
p_4	10	15	12	66	78	...	3
...	...	...	...	...	...	...	...
p_P	55	78	96	77	45	...	0
	r_0	r_1	r_2	r_3	r_4	...	r_R

p_0	6	12	18	88	43	...	2
p_1	15	50	77	90	3	...	0
p_2	0	26	2	10	5	...	11
p_3	12	10	6	8	12	...	10
p_4	11	20	20	55	74	...	5
...	...	...	...	...	...	...	...
p_P	55	74	80	69	40	...	1
	r_0	r_1	r_2	r_3	r_4	...	r_R

Figure 33: Example of third-order tensor $\mathcal{X}_l$

and time slots. Each tensor $\mathcal{X}_l \in \mathbb{R}^{I_p \times I_r \times I_t}$ in tensor hierarchy $\mathcal{X}^{SRT} = \{\mathcal{X}_0, ..., \mathcal{X}_L\}$ represents relations between participants, time slots and stay regions on SRT level $l \in \{0, ..., L\}$. The value of each element $x(p_i, r_j, t_k)$ of third-order tensor $\mathcal{X}_l$ simply represents the amount of time participant p_i spent inside level's l stay region r_j within time slot t_k (Figure 33). Using the tensor hierarchy $\mathcal{X}^{SRT}$, in the next step we build another tensor hierarchy $\mathcal{Y}^{SRT}$ whose elements represent *competence scores*.

3.3.5. Assessment of spatio-temporal competences

In this step we estimate participants' familiarity with each level's stay regions in different times of day and compute so-called *competence scores*. MCS tasks are usually geolocalized and refer to a specific time of day. Obviously, we would like to allocate such tasks to participants who have the most extensive experience with the target location and/or surrounding area at the designated time. Effectively this means we are looking for the participants who spent the largest portion of their time inside the corresponding stay region and within matching time slot. Therefore, for each participant we can simply compute her share in the total amount of time logged inside the target region within target time slot (total being the sum over all the participants). Hence, we use previously obtained hierarchy of tensors $\mathcal{X}^{RST}$ and define

$$a(p_i, r_j, t_k) = \frac{x(p_i, r_j, t_k)}{\sum\limits_{m=0}^{P} x(p_m, r_j, t_k)} \tag{3.1}$$

where P is the total number of participants. Obviously this kind of measure does not assure fair competition between newcomers and experienced participants. Additionally, we also need to take into consideration the regularity of the target location in participant's location history. Therefore we use *regularity measure* b which is the measure of participant's personal significance of the target location. We define

$$b(p_i, r_j, t_k) = \frac{x(p_i, r_j, t_k)}{\sum\limits_{n=0}^{R} x(p_i, r_n, t_k)} \tag{3.2}$$

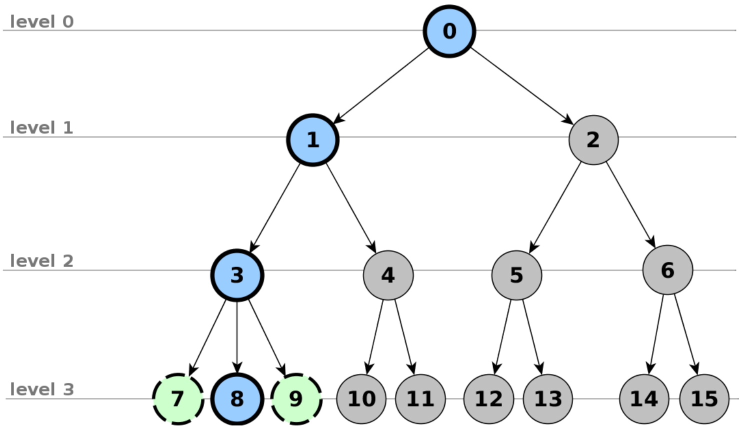


Figure 34: SRT climbing

where R is the total number of stay regions. Finally, we combine (3.1) and (3.2) and formally define competence score as

$$c(p_i, r_j, t_k) = \gamma a(p_i, r_j, t_k) + (1 - \gamma) b(p_i, r_j, t_k) \tag{3.3}$$

where $0 \leq \gamma \leq 1$ is a tuning parameter which determines proportion of the two measures in the final score. Obviously, $0 \leq c(p_i, r_j, t_k) \leq 1$ and for this reason we will sometimes express competence scores in percentages.

Finally, we compute competence scores for each level in the tensor hierarchy $\mathcal{X}^{SRT}$ and build new tensor hierarchy $\mathcal{Y}^{SRT}$ whose elements represent competence scores, i.e. $y(p_i, r_j, t_k) = c(p_i, r_j, t_k)$.

3.3.6. Ranking and grouping

In the final step of the proposed task allocation approach we use the SRT and tensor hierarchy $\mathcal{Y}^{SRT}$ obtained in the previous steps to find the best suited contributors for specific MCS tasks. This boils down to assigning MCS task to the right stay region in SRT and ranking participants by corresponding competence scores.

Definition 3.3.7 (MCS task) *MCS task $g = \langle lat_g, lng_g, t_g \rangle$ is geolocalized crowdsensing task assigned to location $\langle lat_g, lng_g \rangle$ and time slot t_g. In other words, MCS task g requires real-world information from location $\langle lat_g, lng_g \rangle$ within time slot t_g to be successfully solved.*

Naturally we would like to allocate above defined MCS task to the contributors who are familiar with the target location and/or the surrounding area within designated time slot. Therefore, we first assign the target location $\langle lat_g, lng_g \rangle$ to the right stay region, i.e. node in SRT. We search SRT level by level for a stay region which contains location $\langle lat_g, lng_g \rangle$. If the target location can not be assigned to any stay region at the current level, we climb the tree one level up and search among larger (and sparser) stay regions.

Once we have found target stay region r_g on level l, we look up competences scores $y_l(p_i, r_g, t_g)$ in $\mathcal{Y}_l \in \mathcal{Y}^{SRT}$ for each participant p_i. Obviously, participants with the highest competence scores can be treated as preferred contributors for task g. However, they are not the only good candidates in the system. Participants who are familiar with the close locations outside the stay region r_g can also be treated as well-suited contributors. Therefore, we also consider nearby stay regions by climbing the SRT from the node r_g and summing up competence scores for each of the visited target nodes r_l. For example, in Figure 34 region r_8 is found as the target stay region. However, participants who are familiar with close adjacent areas r_7 and r_9 should also be treated as reliable contributors. Therefore, we climb the SRT from node r_8 to r_3 and refer to it as the new target stay region on level 2. We then look up corresponding competence scores in $\mathcal{Y}_2$ and add them to level's 3 competence scores from $\mathcal{Y}_3$. We continue climbing the SRT until we reach the root node containing all the stay regions in the system. Additionally, we involve the level-dependent factor $\alpha(l)$ to weigh the significance of participants' competences on different levels. In other words, we value more competence scores on lower levels where the stay regions are smaller and denser. Hence, we can, for example, model $\alpha(l)$ using exponential function $f(x) = a^x$ where $a \geq 0$ (similar was proposed by Li et al. in [87]). Finally, we formally define the accumulated multi-level competence score as

$$c_m(p_i, g) = \sum_{l=0}^{L} y_l(p_i, r_{g,l}, t_g)\alpha(l) \tag{3.4}$$

where $r_{g,l}$ is the target node on level l. The algorithm 2 summarises the process of calculating multi-level competence scores.

Algorithm 2: Computing multi-level competence scores

```
Input: MCS task $g = \langle lat_g, lng_g, t_g \rangle$, stay region tree (SRT), tensor hierarchy $\mathcal{Y}^{SRT}$
Output: multi-level competence scores $c_m(p_i, g)$
$l = L$ // current level
while $l$ != 0 do
    for stay region $r_j \in SRT(l)$ do
        if $\langle lat_g, lng_g \rangle \in r_j$ then
            $r_g = r_j$
            break
        end
    end
    $l$ -= 1
end
$r_l = r_g$ // current stay region
while $r$ != null or $l$ != 0 do
    foreach participant $p_i$ do
        $c_m(p_i, g)$ += $y_l(p_i, r_l, t_g)\alpha(l)$
    end
    $r_l = r_l.parent$
    $l$ -= 1
end
```

Before we allocate task to the highest ranked participants, we try to group those who usually visit the same areas at the similar times of day, and thus enable them to collaboratively conduct the MCS task. In other words, we try to identify high ranked participants with the similar mobility patterns. This opens up new possibilities for establishing social and trust networks between contributors and data collectors. However, instead of engaging with the issues of these large research fields, we stay within the scope of this work and only present employed similarity measure. We take previously described hierarchical approach and accumulate similarity scores from different level levels of SRT. Hence, we define similarity as

$$sim(u,v) = \sum_{l=0}^{L} sim_l(u,v)\alpha(l) \tag{3.5}$$

where $sim_l(u,v)$ is a similarity score between participants u and v on level l. To estimate similarities between participants' spatio-temporal behaviours on each level of SRT we adopt measure proposed by Yuan et al. in [92]. They extend the cosine similarity measure by incorporating temporal influence and develop time-aware collaborative model for point-of-interest recommendation. The cosine similarity between two vectors is a measure that calculates the cosine of the angle between them. Since it takes in consideration only orientation of the vectors and not magnitude, we can efficiently estimate similarity between senior and new contributors. Therefore, we use obtained tensor hierarchy $\mathcal{X}^{SRT}$ and compute similarity between participants' spatio-temporal behaviours on level l as follows

$$sim_l(u,v) = \frac{\sum\limits_{j=0}^{R}\sum\limits_{k=0}^{T} x_l(u,r_j,t_k)x_l(v,r_j,t_k)}{\sqrt{\sum\limits_{j=0}^{R}\sum\limits_{k=0}^{T} x_l(u,r_j,t_k)^2}\sqrt{\sum\limits_{j=0}^{R}\sum\limits_{k=0}^{T} x_l(v,r_j,t_k)^2}} \tag{3.6}$$

Finally, using the competences scores for MCS task g and similarities between participants we allocate task g to the highest ranked group of similar participants.

3.4. Experiments

MCS system are very difficult to experiment with given that they require a large number of volunteer participants which are willing to invest their time, install required technology, provide resources of their mobile devices when needed, and expose themselves to potential privacy and security threats. Therefore, most of the experimental studies in the literature are conducted on simulated MCS environments or already available datasets. Instead of collecting a new trajectory dataset using the prototype platform introduced in the previous chapter, in this section we will make use of the GPS trajectory dataset collected in *GeoLife* project of Microsoft Research Asia [93].

3.4.1. Dataset

The GeoLife dataset ([67–69]) consists of 17,621 trajectories collected by 182 users in a period of over five years (from April 2007 to August 2012). The trajectories cover a total distance of about 1.2 million kilometres and a total duration of more than 48,000 hours. They were recorded by different GPS loggers and GPS-phones, and have a variety of sampling rates. A GPS trajectory of this dataset is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude. 91% of the trajectories are logged in a dense representation, e.g. every 1-5 seconds or every 5-10 meters per point. The dataset reflects a broad range of users' outdoor movements, not restricted to only simple life routines. The majority of trajectories are located in Beijing (China).

Table 31: Structure of SRT obtained from GeoLife dataset

Level	Number of stay points	Number of stay regions	Number of *leaf* stay regions
1	7759	1	0
2	7694	24	21
3	7393	55	54
4	6857	23	21
5	6429	49	47
6	5459	138	134
7	3439	114	100
8	1824	61	53
9	613	14	13
10	197	12	12
Total	7759	491	455

3.4.2. Constructing stay region tree

For discovering stay points from GPS trajectories in GeoLife dataset we set ε_d to 50 meters and ε_t to 20 minutes (see Section 3.3.2). This results in detecting 7759 stay points in GeoLife trajectory dataset. Obviously, the parameters ε_d and ε_t can significantly affect the structure and size of the obtained SRT, i.e. total performance of the proposed task allocation approach. However, here we naturally assume the time and size limits for stay points and thus leave the scalability evaluation for the future work. Further, we hierarchically cluster stay points using OPTICS density based clustering algorithm. The automatic cluster extraction algorithm proposed by Sander et al. in [91] was used to find local maximas in OPTICS reachability plot and to identify cluster candidates. Additionally, we adopted parameter settings and heuristics from [91] to obtain a *non-binary* hierarchy tree from iterative *binary* partitioning of cluster candidates. To ignore clusters that are too small we also assume a minimum cluster size of 3 stay points.

Finally, we established a 10-level cluster hierarchy which we refer to as stay region tree (SRT). The Table 31 details the structure of the obtained SRT in terms of the total number of stay points and stay regions per each level. Additionally, it contains a number of *leaf* stay regions (leaf nodes in SRT) which are initially assigned to geolocalized MCS tasks in the recruitment process. Only 7.3% of stay regions are represented as internal nodes in SRT. This can be explained by wide distributions of GPS points in GeoLife dataset. In other words, the majority of trajectories are located in Beijing, but the GPS points are distributed over a much larger geospatial area (more than 30 cities around the world). Therefore, only denser areas in Beijing region are represented as internal nodes and further clustered on lower levels of SRT.

3.4.3. Constructing tensor hierarchies

We introduced the time domain into our framework by defining 8 three-hour time slots (0:00 am - 3:00 am, 3:00 am - 6:00 am, 6:00 am - 9:00 am, etc.) and use them to asses participants' time-dependent abilities to conduct MCS tasks. Size of the time slots mainly depends on the nature of MCS task and mobility of participants. In other words, for the MCS tasks that require time-sensitive information, we would normally shorten the time slots. However, in our experiments we assume lower time-sensitivity of MCS tasks and

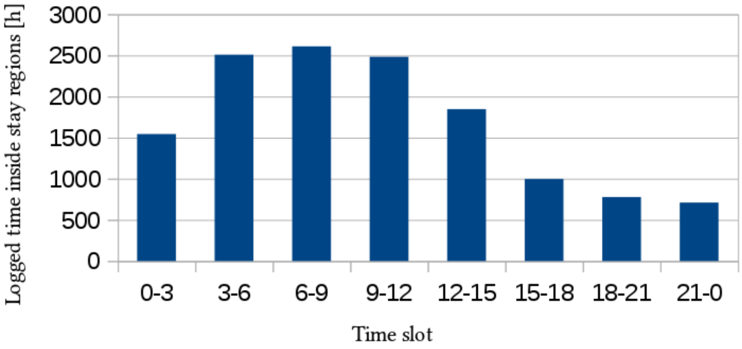


Figure 35: Distribution of time logged inside the stay regions over time slots

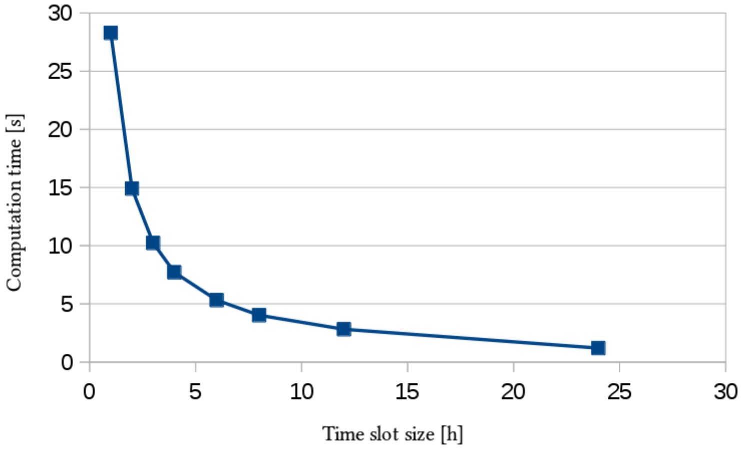


Figure 36: Correlation between computation time of tensor hierarchy construction and size of the time slots, measured on a machine with a dual-core Intel Core i5-3320M 2.6 GHz CPU and 4 GB RAM running Linux Mint 17.1.

define three-hour time slots. Figure 35 shows total amount of time logged inside inferred stay regions by 182 users from GeoLife dataset, per each of three-hour time slots. As expected, the most of the stays were recorded in forenoon when people are usually stationed around their workplaces.

Obviously, the number of time slots significantly affect the size of tensors in hierarchies $\mathcal{X}^{SRT}$ and $\mathcal{Y}^{SRT}$, and thus the computation load performed in recruitment process. The hierarchies constructed in our experiments contain 10 third-order tensor with dimensions $182 \times R_l \times 8$ where R_l is a number of stay regions on level l. Figure 36 shows computation time required to construct tensor hierarchies $\mathcal{X}^{SRT}$ and $\mathcal{Y}^{SRT}$ in a relation to different time slot sizes.

3.4.4. Recruitment

We naturally assume that participants' reliability to conduct MCS task is proportional to their experience with the target location. To put it differently, participants who spent

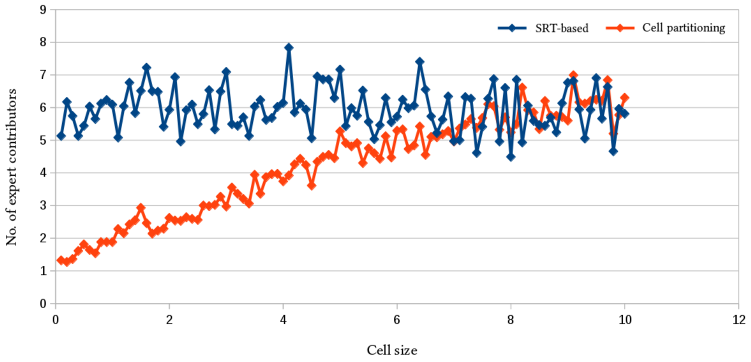


Figure 37: Number of experts found with SRT-based approach and cell partitioning

more time at the target location are likely to be more competent and contribute with high quality information.

We compare our *hierarchical* SRT-based approach with traditional *flat* cell partitioning used for inferring sensing coverage in mobile sensor networks. In participatory sensing environments predefined monitoring areas are usually pre-partitioned into a number of fixed rectangular cells. In our experiments we use 100 different partitioning configurations, i.e. 100 different cell sizes (0.10, 0.20, 0.30, ... , 10.00 km^2). In other words, for each of 100 cell configurations we create 1,000 randomly geolocalized MCS tasks (100,000 in total) and try to assign them to expert contributors using both SRT-based approach and flat cell partitioning. In the experiments we identify experts as a contributors with competence scores above 30%. The tuning parameter γ is set to 0.5, i.e. we value equally participant's personal significance of target location and total experience compared to all other participants.

Figure 37 shows the average number or expert contributors found in 100 experimental configurations. The proposed SRT-based approach finds 6-7 expert contributors for each of randomly geolocalized MCS tasks. On the other hand, traditional cell partitioning approach depends heavily on cell configuration and therefore performs worse with smaller cell sizes. However, powerful MCS system needs to efficiently allocate location-sensitive tasks which are very often associated with exact GPS coordinates or very small geographic areas. Therefore only a very fine cell partitioning approach can assure high quality information harvesting.

As shown in Figure 37, if we increase enough the cell size (> 8 km^2), cell partitioning approach will perform as good as SRT-based approach in terms of the number of recruited expert contributors. However, it is important to understand that the set of participants identified as experts by the two compared approaches differ one from another. Obviously, flat cell partitioning approach takes in consideration only participants who have been inside the target cell. On the other hand, hierarchical SRT-based approach also considers participants with extensive experience with nearby locations. Therefore, proposed approach finds expert contributors from a much larger set of candidates and thus outperforms traditional cell partitioning with smaller and less populated cells. This effect can be seen as different scatters of stay points in Figure 38 which shows all of the stay points from expert contributors found by the two compared methods in one of the conducted experiments. The stay points represent candidates' points of interest and thus are concentrated around the target location. In Figure 38a they are spread over a larger area and denser around the target location. This can be explained as a result of hierarchical competences assessment which

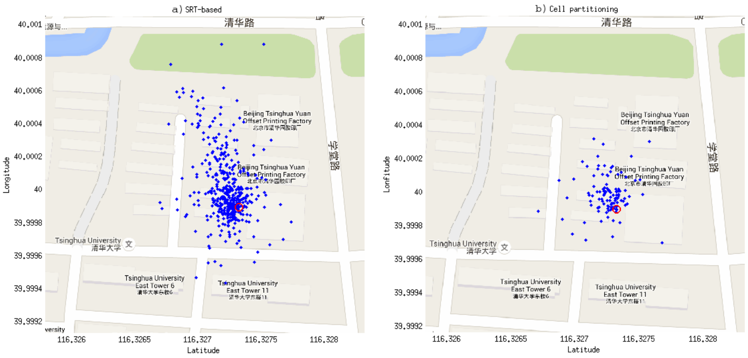


Figure 38: Stay points from expert contributors found with SRT-based approach and cell partitioning

values more candidates who are familiar with the areas closer to the target location.

3.5. Conclusion

In this chapter we investigated the correlation between users' mobility and their role as contributors in MCS systems. We find that human mobility significantly affects sensing coverage and opens up new possibilities for task allocation and recruitment of contributors. Sensing coverage can be to a large extent inferred from users' history of visited locations. Therefore, we modelled 3-dimensional correlations between participants, regions of interest and time of day as a hierarchy of third-order tensors. Obtained structure allows us to fast and efficiently asses participants' competence to conduct location-based MCS tasks. Finally, proposed competence scores assure fair ranking and allow newcomers to compete with experienced participants.

Although efficient task allocation can improve confidence and reliability of user-generated data, they cannot guarantee high level of quality. In the next chapter we will introduce an example of a very simple MCS system which collects check-ins from drivers in Barcelona's urban freight transport and show that post-processing mechanisms are usually required to improve the quality of crowdsensed data.

Chapter 4

Analysis of crowdsensed mobility data

This chapter is based on materials from the following two peer-reviewed papers:

> P. Mrazovic, B. Eravci, J. L. Larriba-Pey, H. Ferhatosmanoglu, and M. Matskin, "Understanding and Predicting Trends in Urban Freight Transport", in *Mobile Data Management (MDM), 2017 18th IEEE International Conference on*, IEEE, 2017, pp. 124–133.
>
> B. Kolbay, P. Mrazovic, and J. L. Larriba-Pey, "Analyzing Last Mile Delivery Operations in Barcelona's Urban Freight Transport Network", in *Cloud Infrastructures, Services, and IoT Systems for Smart Cities*, Springer, 2017, pp. 13–22.

While so far we studied only general-purpose MCS systems, in this chapter we will narrow our focus to their applications within smart mobility domain. In particular, we will introduce the first of the three MCS systems proposed in this thesis. Recall that our research is to large extent application-driven, meaning we will study real-world examples of MCS applications which aim to solve specific mobility problems, and use them to demonstrate and evaluate developed algorithms. In this chapter we will study a very simple MCS system which collects check-ins from drivers in Barcelona's urban freight transport network. This system will help us demonstrate how simple user-generated data from MCS environments, such as check-ins in this particular case, can be processed and analysed to produce valuable insights into urban mobility trends. Notice here that even though we are trying to solve very specific problem, the proposed data management techniques can be used in other MCS systems to post-process crowdsensed data in order to improve its quality and obtain meaningful results.

4.1. Introduction

High population densities in urban environments are responsible for increasing consumption of goods and services that often need to be delivered from the less crowded city periphery. However, the traffic infrastructures and the possibilities for their extension in dense urban environments are limited, and thus deliveries tend to be made in smaller loads and frequent trips which additionally increases the traffic in already crowded urban grids of streets [94]. This also results in generating noise and pollutant emissions, increasing congestion and posing a threat to the safety of road users [95]. The social and economic concentration of resources in cities poses a strong need for efficient and sustainable freight distribution processes.

In this context, mobile data management and analytics arise as a means to achieve more efficient freight distribution systems, without the need for large investments or sophisticated technologies. In this chapter, we are taking the first step towards such a solution by analysing and predicting usage of urban freight transport infrastructure over data collected by a new MCS system developed for this purpose.

The provision of on-street and off-street loading/unloading areas designated for the freight vehicles has been recognized as the most effective infrastructure measure for optimising last-mile delivery operations and improving mobility conditions in urban areas [95]. Nevertheless, with the increasing urban population, urban transport demands are becoming difficult to absorb due to limitations of the urban environments. The loading[1] areas are becoming over-occupied and can not meet high transport demands, and allocating new spaces for logistics activities is not possible in already dense urban environments. Therefore, the pervasive information and communication technologies naturally arise as a means to improve circulation of the freight vehicles at the loading areas.

We will analyse data from a new MCS system that allows deliverers to check-in at the designated loading area, and thus provide real-time occupancy information. The system has been released as a part of Barcelona's sustainable urban mobility plan. We will focus on exploratory analysis of the check-in dataset collected from the system in order to detect temporal and spatio-temporal patterns in urban freight transport. Discovered patterns can allow us to detect hot spots in the city, as well as morning and evening rush hours for individual loading areas. By applying the detected patterns we will develop several availability prediction models for the loading areas in the city. In particular, we propose linear regression (LR) based models to incorporate both the historical delivery trends for the specific area and the current availability patterns for that specific day to better estimate the current availability. This work naturally opens up new possibilities for delivery route planning and better managing of parking spaces and freight transport infrastructure.

4.2. Barcelona's urban freight transport scheme

Freight requires dedicated urban spaces, such as loading areas. Insufficient delivery spaces transfer delivery activities on traffic lanes and pavements, which leads to congestions and traffic accidents. Therefore, the city of Barcelona, as well as many other European cities, has adopted land use planning and infrastructure measures for achieving a more efficient integration of urban freight in the urban transport system. In 2015 Barcelona city authorities launched a new scheme for urban freight transport which prescribes the obligatory usage of newly regulated areas for delivery operations, locally known as *areaDUM* (Distribució Urbana de Mercaderies in Catalan). Roughly 9,000 parking spaces across 2,000 areas have been designated for the freight transport purposes, while an estimated 45,000 parking manoeuvres take place in these zones each day. In other to improve the road network and parking capacity, different parking regulations have been introduced for each of the loading areas. For example, some of the lateral road lanes are devoted to traffic during peak hours, deliveries during off peak hours, and residential parking during the night [96]. Additionally, different parking time limits have been prescribed for the different areas. The parking enforcement unit was formed to ensure compliance of these parking regulations to facilitate availability of the loading areas for freight vehicles.

In order to facilitate and regulate the usage of loading areas, and to improve the circulation of vehicles, the city of Barcelona introduced a novel parking management solution based on MCS paradigm. In order to use the designated areas, deliverers need to check-in

[1] We use "loading" in short while referring to both loading and unloading activities.

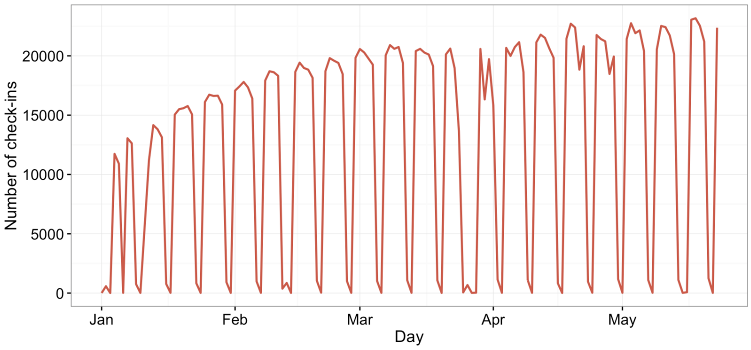


Figure 41: Total number of check-ins per each day of the observed period between January 1st, 2016 and May 23rd, 2016

with their vehicle's registration plate number using either mobile application or SMS service. In addition, the mobile application, available for both Android and iPhone devices, helps deliverers in managing their parking operations. The application informs deliverers about the time remaining for parking and also includes a system of warnings to help them avoid fines. The new service aims to offer more availability of spaces and, in the future, recommend the areas in which deliverers are most likely to find parking in specific time ranges.

4.3. AreaDUM dataset

In this section, we introduce the *areaDUM* check-in dataset, and give some basic characteristics of it. To the best of our knowledge, this is the first dataset crowdsensed from citizens and private and professional deliverers that reflects the usage of the urban freight infrastructure. The dataset consists of 49,172 users, i.e. deliverers, 1,990 loading areas and 3,014,610 check-ins over the period between January 1st, 2016 and May 23rd, 2016. Each check-in consists of the ID of the deliverer who made the check-in, her vehicle's plate number (anonymised), the loading area to which the check-in was made, geographic coordinates, and a timestamp. Each of the loading areas in the dataset is geocoded, and deliverers could check in only if they were in the vicinity of the corresponding area. Therefore, the check-ins' geographic coordinates are introduced only to confirm their validity. Further, the number of authorized parking spaces for each of the loading areas is known from the dataset. There are total 8,491 parking spaces across 1,990 loading areas, while the average number of parking spaces per loading area is 4.27.

In spite of the large amount of check-ins, the collected data is still sparse, as it is the case for most ICT based data collection and smart city applications. The new system based on mobile technologies was recently introduced to replace the old parking discs. However, the two systems coexisted for a short period of time allowing users to adapt to the new system. Nevertheless, during the observed time period, the estimated percentage of deliverers using the new system has grown to around 60%. This trend could be seen in Figure 41 where we plotted the total number of check-ins per each day of the observed period. Also notice the occasional valleys in the plot which represent holidays and weekends when the delivery operations were kept to minimum. However, in order to further address the issue of data sparsity, we have to look at the distribution of these check-ins among the loading areas. As

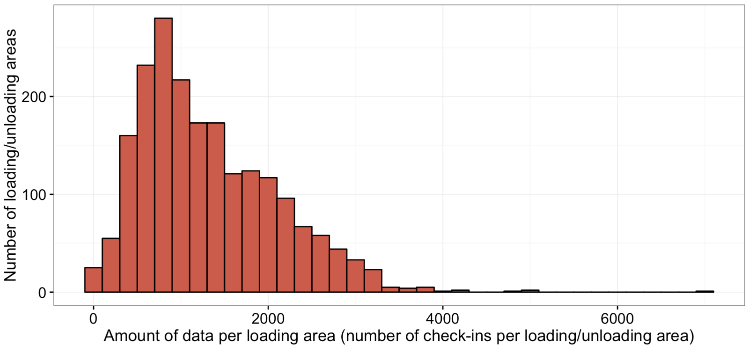


Figure 42: Distribution of check-ins among loading areas

we will see in the next section, not all of the areas have the same turnover, and thus some of them stand out with a lower number of check-ins in the dataset. Figure 42 depicts such unequal distribution of data among different loading areas. Each bar in the plot indicates the number of areas with the certain amount of check-ins (plotted on the horizontal axis). From the plot we can see there are almost 200 areas with less than 500 check-ins (~3.49 check-ins per day). Usage of these areas is harder to predict due to sparseness of collected information, and thus one needs to put an extra effort to deal with the data sparsity.

4.4. Parking availability estimation

The new service aims to eliminate congestions at loading areas and balance the total amount of traffic in urban freight network. The first step towards achieving these goals is understanding how the parking availability changes over time and designing a prediction algorithm which can help deliverers in planning their delivery routes. Therefore, we study how to compute and predict real-life availabilities of loading areas.

We start by formally defining the availability of loading area as the total number of unoccupied parking spaces.

Definition 4.4.1 (Availability of loading area) *Availability of loading area $\hat{a}(d_i, t_j)$ is the total number of unoccupied parking spaces at area d_i at time t_j. Formally,*

$$\hat{a}(d_i, t_j) = c_{d_i} - o(d_i, t_j) \tag{4.1}$$

where c_{d_i} is the capacity of area d_i (i.e. total number of parking spaces), and $o(d_i, t_j)$ is the total number of occupied parking spaces at area d_i at time t_j.

Therefore, if we want to compute the real-life availability of a loading area d_i, we need to know the total number of occupied spaces $o(d_i, t_j)$. However, since not all deliverers have adopted the new system based on mobile technologies, this information cannot be inferred directly from the check-in dataset. Additionally, authorized emergency vehicles (e.g. police cars, fire trucks, ambulances, etc.) and vehicles with disabled parking permits can be legally parked at loading areas without checking in. Finally, the parking spaces can be also occupied by illegal parkers.

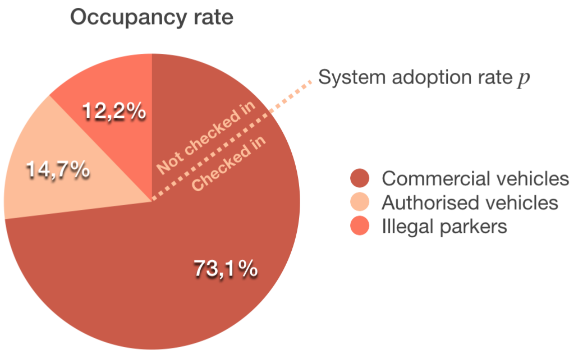


Figure 43: Occupancy data collected in field work

Definition 4.4.2 (Occupancy of loading area) *Occupancy of loading area $o(d_i, t_j)$ is the total number of occupied parking spaces at area d_i at time t_j. Formally,*

$$o(d_i, t_j) = o_c^+(d_i, t_j) + o_c^-(d_i, t_j) + o_a(d_i, t_j) + o_i(d_i, t_j) \tag{4.2}$$

where $o_c^+(d_i, t_j)$, $o_c^-(d_i, t_j)$, $o_a(d_i, t_j)$ and $o_i(d_i, t_j)$ stand for the total number of parking spaces occupied by checked-in commercial vehicles, not checked in commercial vehicles, authorized vehicles and illegal parkers, respectively.

In order to estimate the real-life availability and occupancy of loading areas, Barcelona City Council conducted a field research where parking behaviours in a representative sample of loading areas have been observed. The collected data is reported in Figure 43. 73.1% of parking spaces were occupied by commercial vehicles, 14.7% by authorized vehicles and 12.2% by illegal parkers. Additionally, notice that the ratio between checked in and not checked in commercial vehicles depends on the system's adoption rate (p) which slowly increases with time (Figure 41). Therefore, when estimating and predicting real-life availability we will simulate different lifespan stages of the new system by adjusting adoption rates.

Definition 4.4.3 (System adoption rate) *We define the system adoption rate $p(d_i, t_j)$ at loading area d_i at time t_j as the ratio*

$$p(d_i, t_j) = \frac{o_c^+(d_i, t_j)}{o_c^+(d_i, t_j) + o_c^-(d_i, t_j)} \tag{4.3}$$

However, we will assume that this ratio is the same for all the d_i and t_j. Hence, we will denote this universal quantity simply by p.

Using the above definitions and results from conducted field research we can estimate availability of loading areas. Since the total number of checked in commercial vehicles $o_c^+(d_i, t_j)$ is directly observable from the check-in dataset, we use it to express other occupancy variables. Using (4.3) we can express the total number of not checked in commercial vehicles as follows

$$o_c^-(d_i, t_j) = \frac{o_c^+(d_i, t_j)(1-p)}{p} \tag{4.4}$$

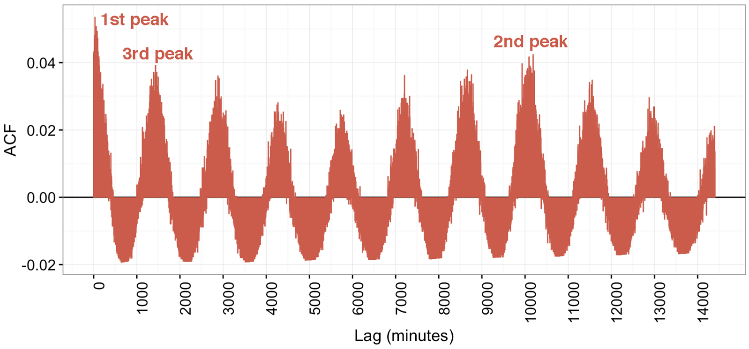


Figure 44: ACF of the time series recovered for the example loading area located in La Rambla street

Similarly, using the statistical result reported in Figure 43 we can express the total number of parking spaces occupied by authorized and illegal vehicles as follows

$$o_a(d_i,t_j) + o_i(d_i,t_j) = \frac{0.269}{0.731}(o_c^+(d_i,t_j) + o_c^-(d_i,t_j)) \tag{4.5}$$

After we substitute $o_c^-(d_i,t_j)$ in (4.5) with the right side of (4.4) we get

$$o_a(d_i,t_j) + o_i(d_i,t_j) = \frac{0.368}{p}o_c^+(d_i,t_j) \tag{4.6}$$

Finally, by using (4.4) and (4.6) in (4.1) we obtain the availability estimation formula

$$\hat{a}(d_i,t_j) = c_{d_i} - \frac{1.368}{p}o_c^+(d_i,t_j) \tag{4.7}$$

In the next section we employ the derived formula in order to obtain the areas' historical availability profiles that reflect the trends in urban freight transport network.

4.5. Historical availability profiles

In this section, we study temporal delivery trends by computing individual historical availability profiles for each of the areas. The historical availability profile (HAP) of a loading area consists of the mean of the area's parking availability as a function of time. However, before computing the profiles, first we need to take a closer look at the dataset and analyse the periodicity of the underlying process in order to choose the appropriate duration of the profiles and the size of the discretisation bin.

4.5.1. Seasonality analysis

We treat the data as time series and analyse the autocorrelation functions (ACF) for each of the loading areas. Initially, we sample the time-stamped data by accumulating the number of check-ins over 1-minute sampling period. In Figure 44 we plot ACF of an example time series recovered by sampling the data from the loading area located in La Rambla, one of the most popular streets in Barcelona. The plot reveals the periodicities (seasonality) of vehicle arrivals at the observed area. The AFC exhibits three significant peaks at lags of

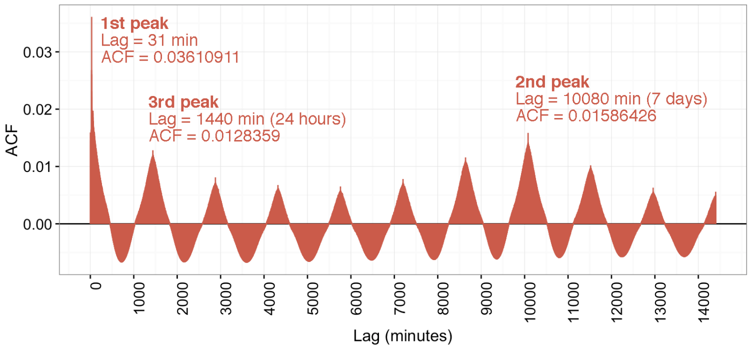


Figure 45: Average ACF

31, 10,188 and 1,394 minutes. The largest peak occurs at lag of 31 minutes which indicates the strongest correlation caused by the present parking regulations. More specifically, for the most of the loading areas, including the observed one, maximum allowed stay time is 30 minutes. Further, the areas are very often fully occupied and thus deliverers need to wait for a parking space. Therefore, the strong 31-minute periodicity suggest that deliverers tend to stay at the area for the maximum allowed time, whereupon the newly freed parking space is immediately taken by another vehicle. The second largest peak occurs at lag of 10,188 minutes, i.e. 7.075 days, and indicates weakly seasonality of the process. The third peak at lag of 1,394 minutes, i.e. 23.23 hours, indicates daily seasonality, while the rest of the peaks in Figure 44 represent its harmonics. Due to the seasonal nature of the urban freight demand and the established urban delivery routines, daily and weakly periodicities in the dataset have been expected a priori. For example, shops, bars and restaurants are replenished daily and weekly with stocks of fresh and dry food, and other retail goods. Finally, the discovered periodicities can be observed more clearly in Figure 45 where we average the ACFs computed for all 1,990 loading areas.

4.5.2. Constructing HAPs

For most of the loading areas, parking is permitted only during daytime, i.e. from 08:00 to 20:00. We refer to this period as operating cycle. Therefore, when constructing HAPs we compute the parking availability only for the operating cycles, assuming that for rest of the time all parking spaces are available. In other words, we split the time domain into a number of operating (daily) cycles.

Definition 4.5.1 (Operating cycle) *Operating cycle is time period in which parking at loading area is permitted.*

Definition 4.5.2 (Historical availability profile (HAP)) *Historical availability profile of a loading area d_i consists of the mean of the area's parking availability as a function of time,*

$$\hat{h}(d_i, t_j) = \frac{\sum_{k=1}^{m} \hat{a}_k(d_i, t_j)}{m} \tag{4.8}$$

where m is the number of operating cycles for which parking availability is collected.

Since time is discrete we need to compute the parking availability for each time point of the operating cycle. However, in order to capture the previously discovered periodicities

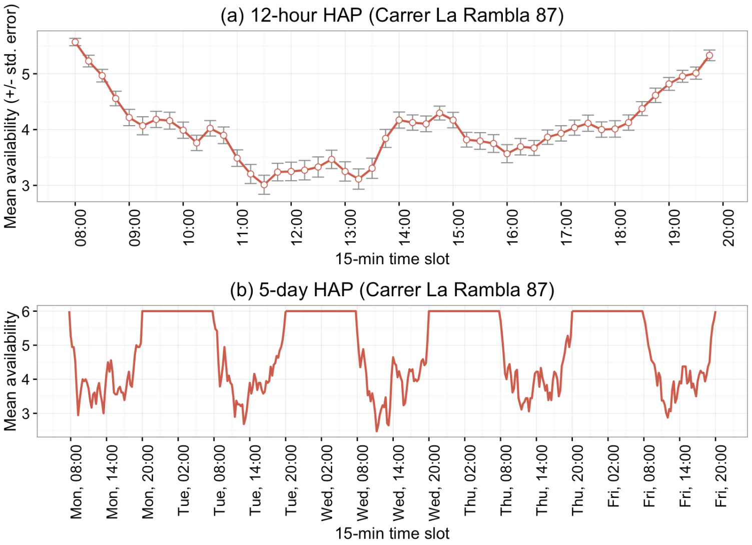


Figure 46: 12-hour (a) and 5-day (b) HAPs for example loading area located in La Rambla street

we use 15-minute discretisation bin. To put it differently, we first sample the time-stamped check-in data by accumulating the number of checked in vehicles over 15-minute sampling period. Thereafter, we compute the parking availability using (4.7) for each 15-minute time slot. Finally, HAPs are obtained by averaging computed availabilities using (4.8). For example, in Figure 46a a 12-hour HAP for previously observed loading area in La Rambla street is obtained by computing the average availability for each of the 15-minute time slot within 12-hour operating cycle (from 08:00 to 20:00) over the 20-week monitoring period. Similarly, in Figure 46b we obtained a 5-day (weekly) HAP for the same area and operating cycle by computing averages for each of the 384 time slots in 5-day window. The computed availability patterns depend significantly on the area's location and the surrounding urban environment. For example, the observed area in La Rambla street is located next to the famous market place La Boqueria, one of the Barcelona's foremost tourist landmarks. The market has a very diverse selection of goods, mostly fresh food that needs to be replenished very frequently to avoid out-of-stock position. The 12-hour HAP plotted in Figure 46a reflects the high turnover of deliveries throughout a whole day. However, notice that the mornings and evenings are less busy due to the lower trading volumes and, consequently, lower replenishment needs. Further, the popular visit times (12:00 - 14:00 and 16:00 - 19:00) are reflected in the plot as the busiest delivery periods. The subtle availability increase in the short period between 14:00 and 16:00 most likely reflects the lunch break and end of the first delivery shift. As we will see later in this section, this increase is common for all loading areas and can be observed more clearly in the global HAP aggregated for all loading areas. The weekly HAP plotted in Figure 46b reveals very similar usage trends across all weekdays. As we expected, morning hours on Mondays are slightly busier than on the rest of the days due to the regular weekly stock replenishments.

As we previously noted, the usage trends depend on the loading areas, and thus we can observe different patterns in the different areas. However, instead of analysing each of the loading areas individually, we will take a look at the aggregated HAP computed by averaging the individual HAPs across all 1,990 loading areas. Such global view on availability

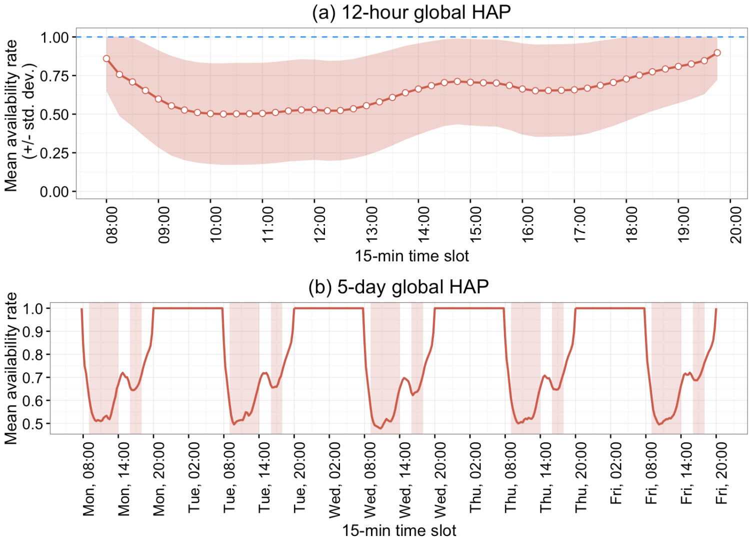


Figure 47: 12-hour (a) and 5-day (b) global HAPs

fluctuations will give us a better understanding of the global mobility and delivery trends in the city of Barcelona. However, we noticed from the dataset that the parking availability at the particular loading area is closely correlated to the total number of authorized parking spaces, i.e. capacity of the area (plot is omitted to conserve space). Therefore, instead of aggregating previously computed availabilities, we also take in consideration the capacity of each of the areas and compute relative availabilities, i.e. $\hat{a}(d_i, t_j)/c_{d_i}$. In Figure 47 we plot the aggregated, i.e. global, 12-hour and 5-day HAPs. Recall from Figure 42 that there a large number of less popular loading areas with only couple of check-ins per day which explains high availability percentages in the plot. Nevertheless, the plot reveals two repeating valleys between 09:00 and 14:00, and 16:00 and 19:00, which corresponds to the first and second (i.e. morning and afternoon) delivery shift. As expected, the second delivery shift is significantly shorter and less busy. Also notice that the second shift on Friday is less active than on the other weekdays, most likely due to reduced working hours in many workplaces in Barcelona. The shift change period and late lunch break can be now observed more clearly as a sudden 30% availability increase between 14:00 and 16:00. Also, notice that all of the weekdays have very similar usage patterns. However, in Figure 47a relatively large standard deviation (red area) indicates instability of the observed HAPs. To put it differently, the individual HAPs significantly differ one from another, especially in busy time intervals.

4.5.3. HAP clustering

The geographic layout of the city of Barcelona is distinguished by its balanced mixture of residential, commercial and touristic areas, and as such has a very strong influence on the delivery operations and the underlying mobility patterns. In order to understand them better, we group areas with similar HAPs and then analyse the geographic layout of the obtained clusters. We again normalise the individual HAPs with respect to areas' capacities (recall that we did the same in Section 4.5.2), and thereafter apply average-linkage hierarchical clustering algorithm to partition loading areas based on their daily usage

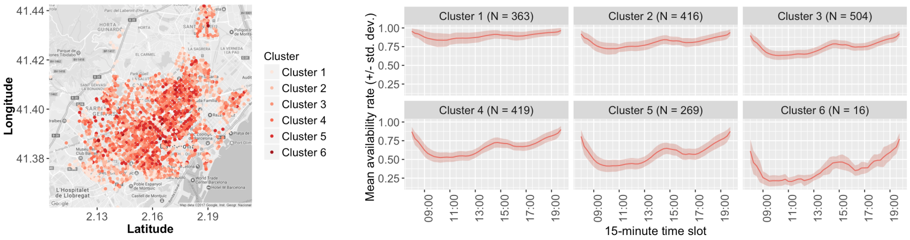


Figure 48: Loading areas clustered by normalised HAPs

trends. The algorithm starts with 1,990 clusters, one for each area. The clusters are then sequentially combined into larger clusters until all loading areas end up being in the same cluster. At each step, the two clusters separated by the shortest distance are combined. The cluster-to-cluster distance is computed as the average pairwise Euclidean distance between 12-hour HAPs belonging to the areas in two different clusters. Finally, the result of the clustering is illustrated in Figure 48. We cut the obtained dendrogram into 6 clusters. The very small standard deviations of per-cluster aggregated HAPs indicate high intra-cluster similarities. On the other hand, the aggregated HAPs exhibit very similar shapes with two valley in morning and afternoon, however with the different magnitudes. Therefore, the clusters mainly differ from each other by the usage intensity, i.e. the magnitude of the availability rate. Consequently, we order the clusters based on the average availability rate, i.e. from the least to the most occupied. The first cluster with 363 loading areas is the least popular one. This corresponds to the plot in Figure 41 where we noticed the large number of less popular areas with only couple of check-ins per day. Similarly, the sixth cluster is the smallest, but the most popular one. It contains 16 most occupied loading areas. In order to detect the busiest delivery areas in the city (hotspots), in Figure 48 (right) we plotted location of each area on the map of Barcelona. Additionally, we coloured the markers based on their belonging clusters. The most popular areas (coloured in darker reds) are located in the central and the most developed part of the city. It is also interesting to notice that most of these popular areas are spread across only two districts - Ciutat Vella and Eixample, the most touristic and developed one. On the other hand, the less busy loading areas are located in the peripheral part of the city.

4.6. Availability prediction

We focus next on the short-term prediction of loading area availability. In particular, we are interested in predicting the total number of available parking spaces in the next time slot, i.e. 15-30 minutes ahead of time. This setup corresponds to most of the delivery scenarios in Barcelona where parking is planned on the fly, while all of the areas are reachable within 30 minutes from any part of the city. Additionally, for most of the loading areas maximum allowed parking time is 30 minutes. The benefits of the availability prediction are threefold: (1) it allows more accurate load balancing of loading areas, (2) it assists city governments in mobility management by providing information about expected delivery activity, and (3) it opens up new possibilities for mobile services such as delivery route planning. We compare several simple prediction models, and establish them as baselines with which other (more complex) models can be compared. We then present a more advanced prediction techniques that combine both historical and online availability data.

We denote by $x(d_i, t_j)$ a random variable representing the true parking availability at loading area d_i at 15-minute time slot t_j of an operating cycle. Similarly, we denote by $\hat{x}(d_i, t_j)$ the predicted parking availability for time slot t_j of the same operating cycle.

4.6.1. Historical model

Historical prediction model (HM) $\hat{x}_{HM}(d_i, t_j)$ dynamically constructs a HAP based on all observations until the current time slot $t_j - 1$, and returns the value at the corresponding HAP's time slot t_j. More formally,

$$\hat{x}_{HM}(d_i, t_j) = \hat{h}(d_i, t_j) \tag{4.9}$$

4.6.2. Last value model (LV)

Last value prediction model (LV) $\hat{x}_{LV}(d_i, t_j)$ predicts that the availability at the current time slot $t_j - 1$ remains constant as its last measured value $\hat{a}(d_i, t_j - 1)$ for the next time slot t_j, i.e.

$$\hat{x}_{LV}(d_i, t_j) = \hat{a}(d_i, t_j - 1) \tag{4.10}$$

4.6.3. Linear regression model (LR)

This model uses a mixture of the online data we acquire from the application and past averages of the availability data to train a linear regression model. The model formed for estimation of availability at area d_i and timeslot t_j, $x_{LR}(d_i, t_j)$, is as follows

$$\hat{x}_{LR}(d_i, t_j) = \beta_1 \; \hat{a}(d_i, t_{j-1}) + \beta_2 \; \hat{h}(d_i, t_j) + \gamma \tag{4.11}$$

This model incorporates both the historical model for the specific area and the current availability patterns for that specific day to better estimate the current availability.

The parameters are estimated for each parking area and for each time slot using the previous data for the particular day. The following matrix denotes the training matrix for LR model for parking area :

$$X_{d_i, t_j - 1} = \begin{bmatrix} \hat{a}(d_i, t_1) & \hat{h}(d_i, t_2) \\ \hat{a}(d_i, t_2) & \hat{h}(d_i, t_3) \\ \vdots & \vdots \\ \hat{a}(d_i, t_{j-2}) & \hat{h}(d_i, t_{j-1}) \end{bmatrix}$$

Corresponding ground truth vector for training is as follows:

$$Y_{d_i, t_{j-1}} = \begin{bmatrix} a(d_i, t_2) \\ a(d_i, t_3) \\ \vdots \\ a(d_i, t_{j-1}) \end{bmatrix}$$

where t_1 time slot value corresponds to first time slot of the operating cycle. We can find the parameters of the model using regular solutions to linear regression problems such as least-squares estimation.

Intercept value (γ) is also added to the model but from the results of our data we have seen that a model without intercept seems to decrease the estimation error.

One also notes that we have scarce data to estimate from when t_j is small, i.e. at the beginning of operating cycle. This scarcity can cause unstable and unreliable results in the estimation phase. To tackle this issue we have used the p-value of the parameters to measure the significance of the relations of the variables. We have chosen $p = 0.05$ as a threshold for significance which is the value used generally in the literature. According to the result of estimation if calculated significance is less than the threshold we use the the historical model estimate, $\hat{x}_{HM}(d_i, t_j)$, instead. In summary:

$$\hat{x}_{LR}(d_i, t_j) = \begin{cases} \hat{x}_{LR}(d_i, t_j), & \text{if } p <= 0.05 \\ \hat{x}(d_i, t_j), & \text{otherwise} \end{cases} \tag{4.12}$$

4.6.4. Linear regression model with aggregated HAPs (LR-A)

Recall from Section 4.3 that for some of the loading areas we have very few observed check-ins. The availability of these areas is harder to predict with the proposed models due to sparseness of the observations. On the other hand, in Section 4.5.3 we saw that loading areas can be efficiently clustered using normalised HAPs, and thus less sparse aggregated HAP can be obtained for each cluster. Therefore, in order to overcome data sparsity issues, we propose to extend previously defined LR model by aggregating historical availability information. To put it differently, instead of using a single HAP of the specific area, we cluster areas with similar HAPs and compute aggregated representative HAP. Forecasts are then performed by applying LR model with the corresponding representative HAP to each loading area in the cluster, individually. As we will see in the next section, in certain cases these grouped models can avoid overfits, reduce the impact of outliers, and successfully capture common trends.

We compute cluster representative HAP by averaging historical availabilities across all loading areas contained in the corresponding cluster, i.e.

$$\hat{h}(C_k, t_j) = \frac{1}{|C_k|} \sum_{d_i \in C_k} \hat{h}(d_i, t_j) \tag{4.13}$$

where C_k stands for arbitrary cluster. The ground truth vector $Y_{d_i, t_{j-1}}$ remains the same, while the training matrix $X_{d_i, t_j - 1}$ is updated as follows

$$X_{d_i, t_j - 1} = \begin{bmatrix} \hat{a}(d_i, t_1) & \hat{h}(C_k, t_2) \\ \hat{a}(d_i, t_2) & \hat{h}(C_k, t_3) \\ \vdots & \vdots \\ \hat{a}(d_i, t_{j-2}) & \hat{h}(C_k, t_{j-1}) \end{bmatrix}, \forall d_i \in C_k$$

We experimented with different clustering algorithms and similarity measures, and we found simple k-nearest neighbour method (kNN) and cosine similarity to give accurate results. Therefore, we compute the similarity between areas as follows

$$sim(d_i, d_j) = \frac{\sum_{k=1}^{48} \hat{h}(d_i, t_k) \hat{h}(d_j, t_k)}{\sqrt{\sum_{k=1}^{48} \hat{h}(d_i, t_k)^2} \sqrt{\sum_{k=1}^{48} \hat{h}(d_j, t_k)^2}} \tag{4.14}$$

4.7. Experimental evaluation

4.7.1. Experimental setup

Our simulations and experiments are based on real-world check-in data introduced in Section 4.3. In order to simulate ground truth data we assume 100% adoption rate and

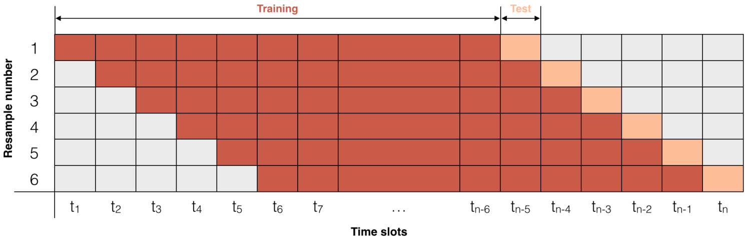


Figure 49: Example of rolling forecasting origin strategy

estimate the true availabilities using (4.7), i.e.

$$x(d_i, t_j) = c_{d_i} - 1.368 o_c^+(d_i, t_j) \tag{4.15}$$

However, we also generate the input data stream by sampling the original data with respect to different adoption rates p. In other words, each check-in from the original dataset is randomly selected in the input stream with the probability p. We use the generated input stream to predict the true availability.

In order to estimate how accurately the prediction models will perform in practice, we use time series cross-validation strategy known as *rolling forecasting origin* [97]. With a rolling forecasting origin, we create an initial window that contains the first k data points as the first training set to estimate the prediction model. We compute the error on the forecast for the next data point $k+1$, and then shift the window by one data point. We repeat the procedure until no data point left. In our case, we use $k = 3360$ data points, i.e. time slots, as training set window. This corresponds to 70 operating cycles and 80% of the whole dataset. In Figure 49 we further illustrate the employed evaluation strategy.

Finally, we consider the mean squared error (MSE) between the estimated and true availability as a performance measure for the proposed models, i.e.

$$MSE(d_i) = \frac{1}{n-k-1} \sum_{j=k+1}^{n} (\hat{x}(d_i, t_j) - x(d_i, t_j))^2 \tag{4.16}$$

We have also considered looking at vacant parking space existence; a boolean variable checking if the model has correctly identified the presence of at least one available parking space. However, since users can make more confident decisions to use a parking place based on the number of vacant places (a higher number of vacant space increases the probability of at least one vacant space by the time user has arrived at the specific area) we have opted to use the mean square error instead. Our methods have shown very high accuracy levels in terms of estimation of vacant space existence and estimation of exact vacant space size is more challenging when these two performance criteria are compared.

4.7.2. Results

We evaluate the performance of the proposed predictive models on five input streams sampled from the original check-in dataset. The sampling was conducted using previously described simple random method with $p \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. This way, we simulate different adoption stages of the new system in order to assess how the adoption rate reflects on the performance of the models under evaluation.

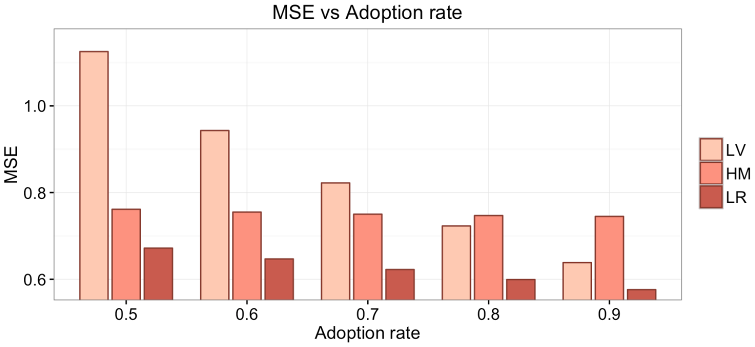


Figure 410: MSE performance for different adoption rates

Figure 410 shows the MSE performance of the models for different adoption rates. Here, the total MSE is aggregated over all time slots in the test dataset. As expected, for all models, the MSE decreases as the adoption rate increases. In other words, the models perform better on less sparse input streams. However, the performance increase is not the same for all models. Notice that the MSE for the LV drops from 1.125 for $p = 0.5$ to 0.638 for $p = 0.9$ (43.29% accuracy increase). On the other hand, the performance increase for the HM is less significant, i.e. the MSE drops from 0.761 for $p = 0.5$ to 0.745 for $p = 0.9$ (2.10% accuracy increase). Also notice that the LV outperforms the HM for $p = 0.8$ and $p = 0.9$. The LV relies on the estimation of the current availability whose accuracy strongly depends on the system's adoption rate, i.e. sparsity of input stream. This is not the case with the HM where the estimated availabilities are aggregated for each time slot of the operating cycle over the entire input stream. Finally, from the plot it can be observed that the LR significantly outperforms the other models for all input streams. In other words, the LR is able to successfully combine online and historical prediction approach, and outperform them both when applied separately. Similar to the LV and HM, the accuracy of the LR increases with the adoption rate, i.e. the MSE drops from 0.672 for $p = 0.5$ to 0.576 for $p = 0.9$ (14.29% accuracy increase).

In addition to the total MSE, we also compute the MSE for each of the 48 time slots of the operating cycle and study how the prediction models perform with respect to different times of day. In Figure 411 we plot the MSE per time slot for different adoption rates p. All of the plots exhibit two distinct peaks, of which the higher (left-hand) peak occurs between time slot 5 and 25 (09:00 - 14:00) which corresponds to morning delivery shift, and the lower (right-hand) peak occurs between time slot 33 and 45 (16:00 - 19:00) which corresponds to afternoon delivery shift. Notice that these patterns are negatively correlated to the ones in Figure 47 where we plotted relative availabilities against different time slots of the operating cycle. This means that all of the models perform slightly worse during busy periods when check-in patterns are less stable. Since it is more difficult to find a parking space during busy periods, deliverers tend to go from one loading area to another in search for a parking space. Obviously, this makes deliveries to some extent disorganized and thus harder to predict.

Figure 411 also shows that at most times of day the LR outperforms both LV and HM. However, the exception to this can be observed at the beginning of the operating cycle (time slots 1-10), i.e. in morning hours (08:00 - 10:00), for all adoption rates p. The reason for this lies in the fact that the LR relies on current trends which can be difficult to infer at the start of a day when we have only few data points at disposal. Recall

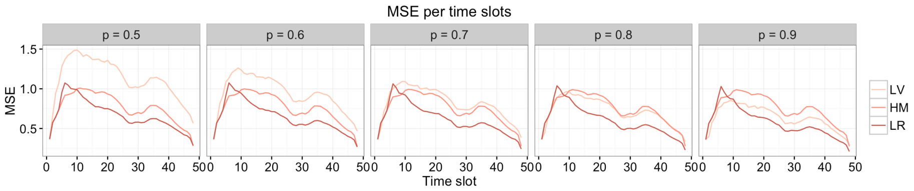


Figure 411: MSE performance per time slots

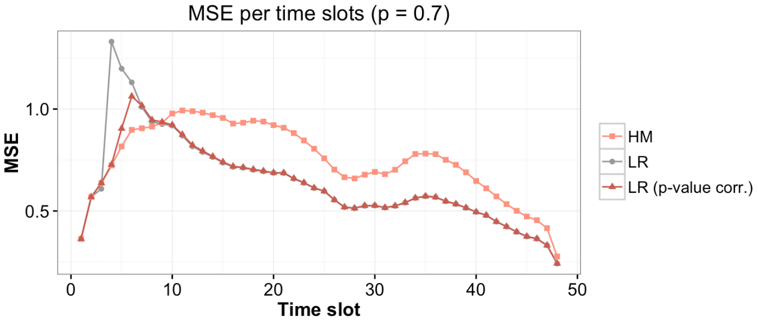


Figure 412: MSE performance of the initial and adjusted LR model

from Section 4.6.3 that we tackle this scarcity problem by using p-values for the model parameters to test the significance of the relation between the variables (i.e. online and historical availability). If the p-value falls above 0.05 threshold, instead of using the trained LR model for prediction, we use the HM model. As shown in Figure 412, with this simple trick we managed to significantly improve the initial performance of the LR model at the beginning of the operating cycle.

As we discussed in the previous section, the performance of the LR model can be further improved in certain cases by grouping similar loading areas and computing aggregated HAP (LR-A model). Recall from Section 4.3 that check-ins are unequally distributed among loading areas, and thus for some of the less popular areas in city's periphery, the collected data can be sparse and bursty. In these special cases, clustering and aggregating arise as a means to reduce the noise and avoid overfitting. We will demonstrate the performance of the LR-A model on an isolated case depicted with Figure 413.

In Figure 413a we plot individual HAPs of 4 similar loading areas grouped together using k-nearest neighbour (kNN) method. Interestingly, the typical 2-valley shape is harder to observe in these unsmooth plots. This suggests that data recorded at these specific areas is quite noisy and bursty. Additionally, we notice that the average number of check-ins per day for the grouped areas is only 10.13. Therefore, instead of building 4 individual models that might be vulnerable to overfitting, we compute aggregated HAP and use it to build a single, but more robust, prediction model. In Figure 413b we compare the performance of individual LR models and aggregated LR-A model. The plot shows that LR-A outperforms LR for all of the grouped areas. More precisely, we reduce the MSE for 7.52% for area 8656, 6.65% for area 9463, 12.32% for area 9987, and 11.46% for area 10527.

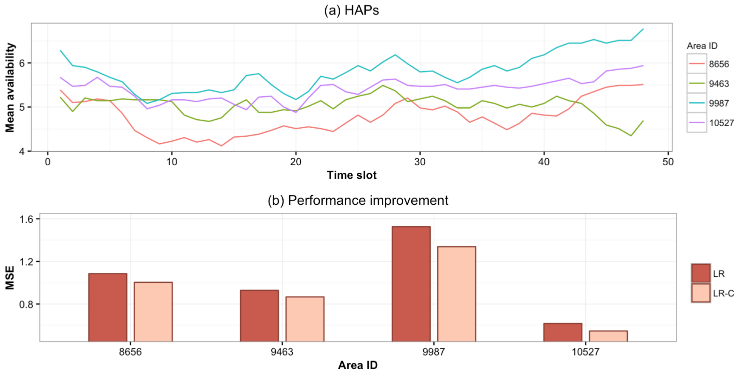


Figure 413: MSE performance of LR and LR-A model

4.8. Related work

In recent years, the mobility patterns in urban freight networks have attracted a certain amount of attention in the research community (e.g. [98–101]). However, obtaining data which would allow a large scale study is often very difficult (or impossible), mostly due to privacy issues. Therefore, many studies relies on information gathered from surveys carried out among freight distributors (e.g. [102, 103]). In our work, we use real-life check-in dataset collected from the novel parking regulation system based on mobile technologies.

We use historical availability profiles (HAPs) of the loading areas in order to understand daily and weekly delivery trends. The similar concepts can be often found in research works that focus on analysis of human mobility patterns. For example, Froehlich et al. [104] employ the concept of *DayView* to study bicycle station usage from Barcelona's shared bicycling system. A DayView is calculated by averaging station data that matches certain criteria into a 24 hour window, discretised into five-minute bins (288 bins/day). Additionally, in order to investigate how usage patterns are shared across stations and geographically distributed in the city, Froehlich et al. cluster stations based on their DayViews. They show that the spatial layout of a city has a strong influence on the movement patterns and social behaviours. A similar research has been conducted in [105], where the DayViews are replaced with local *ActivityCycles* computed for each station and day of a week. Again, in order to get a spatial picture of the mobility pattern in the city, these local ActivityCycles are paired with the station's geo-coordinates and visualised on a heat map.

In [106] Xu et al. introduce an algorithm to construct the historical availability profiles (HAP) by computing estimated values for the mean and the variance of true parking availability for an arbitrary street block. The algorithm relies on PhonePark system which is able to automatically detect parking events using drivers' mobile phones equipped with GPS and/or accelerometer. Here, detected parking events correspond to check-ins in our case. Additionally, since not all drivers are equipped with PhonePark system, its *market penetration ratio* is also considered in [106] (i.e. share of drivers with PhonePark enabled phones). Similarly, we use adoption rate in our work to simulate different adoption stages of the new system. Finally, while Xu et al. estimate the current availability of street blocks, we focus on short-term availability prediction of loading areas using obtained historical and current trends.

Several recent studies have investigated the problem of predicting parking availability. In [107] and [108], a continuous-time Markov model is built to predict the parking availability in ad hoc networks. In [109], a recurrent neural network is employed to forecast the parking occupancy using data collected from a wide network of parking sensors installed on-street in the city of Santander, Spain. Similarly, Zheng et al. [110] analyse the performance of different machine learning methods (neural network, regression tree, support vector regression) in predicting parking availability from the real-time car parking information that has been collected and disseminated by the City of San Francisco, USA and the City of Melbourne, Australia. Additionally, they consider several feature sets consist of past observations, time of day and day of week. In [111], Chen et al. propose Generalized Additive Models for availability prediction of shared bike and car parking lots in the city of Dublin, Ireland. They consider even more features such as time of day, time of year, day type, weather, temperature, humidity and past occupancy observations. Some other studies (e.g. [105, 112]) view the parking data as a one-dimensional time series and explore the use of time series models such as Autoregressive Integrated Moving Average (ARIMA).

4.9. Conclusion

In this chapter we introduced an example of MCS system which has been released as a part of Barcelona's sustainable urban mobility plan. The system allows urban deliverers to check-in at the designated loading areas using a mobile application, and thus provide real-time occupancy information. As such it eliminates the need for hardware based investments or technologies such as sensor infrastructures for parking lots, and provides more information about occupancy such as the licence of the trucks in the parking area. However, we showed that in MCS environments user-generated data, or in this case, check-ins, can be unreliable, uncertain, or insufficient to make conclusion based only on it. Therefore, different post-processing techniques are usually required to improve the quality of the data and increase its applicability.

For this particular MCS example, we presented an exploratory analysis of the crowd-sensed data in order to detect temporal patterns in urban freight transport. Due to phased adoption of the MCS system and illegal parking practices, we had to collect additional occupancy information from the field to estimate the real availability of loading areas. We built individual historical availability profiles (HAPs) for each parking area. We showed that HAPs can provide valuable information about delivery trends and can be efficiently used for detecting hotspots and rush hours, as well as forecasting the future availability for any specific area.

Finally, we proposed a prediction model which incorporates both the historical model (i.e. HAP) for the specific loading area and the current availability patterns for that specific day to better estimate the current availability. The estimated availability values are considered accurate enough to efficiently utilize a parking area recommendation system in Barcelona according to the city government. Later in Chapter 5 we will come back to the problem introduced in this chapter and show how it can open up new possibilities for delivery route planning and itinerary generation for more effective urban freight transportation.

Chapter 5

Bi-objective route planning approach for sustainable urban mobility

This chapter is based on materials from the following peer-reviewed paper:

> P. Mrazovic, J. L. Larriba-Pey, and M. Matskin, "Improving mobility in smart cities with intelligent tourist trip planning", in *Computer Software and Applications Conference (COMPSAC), 2017 IEEE 41st Annual*, IEEE, vol. 1, 2017, pp. 897–907.

While so far we studied fundamentals of human-in-the-loop sensing systems such as MCS, we now move to the second topic addressed by this thesis and focus on algorithmic challenges in MCS-driven route planning applications. We will study different route optimisation algorithms tailored for specific crowdsensing applications that can eventually improve mobility in urban areas. We first consider a MCS application in which route planning is regulated by city administrators. More precisely, we aim to allow the administrators to formulate mobility policies, e.g. to promote or constrain certain city areas, POIs, or routes between them, and to allocate them to the participating devices, thus influencing travellers' journey planning. In this particular example we will show how to put the crowdsensed data into motion and make it actionable.

5.1. Introduction

Urban tourism becomes very popular nowadays and for many cities it is one of their main pillars and economic drivers. The number of urban tourists increases every year and this becomes one of the key factors in city development [113]. Obviously, the increasing number of arriving tourists in already crowded urban areas requires efficient and sustainable management. The massive amounts of tourists can collapse busy areas causing transport inefficiency, unbalanced economic growth and nuisance among tourists and citizens. Therefore, the city governments need to deliver expected touristic experience, but at the same time assure balanced and efficient mobility in the city. In this chapter we study this challenging bi-objective problem and propose a MCS-driven solution to achieve a balance between the *individual* goals of the visitors and the *global* needs for an efficient citizen-oriented city.

Tourists are usually constrained in time and budget. For them, identifying Points of Interest (POIs) worth visiting and planning their daily activities can be a daunting and time consuming problem. The problem of generating personalized tourist itineraries has been defined in the literature as the *Tourist Trip Design Problem* (TTDP) [114] and has been extensively studied in the last decade (a recent survey can be found in [115]). The goal of the problem is to recommend a list of POIs and generate personalized tourist itineraries by taking into consideration common constraints such as the time available for sightseeing, the travelling distances between POIs, the visiting time for each POI, and the estimated degree of satisfaction associated with the visit to each POI. However, to the best of our knowledge, the attempt to use TTDP to influence and control tourist movements in urban areas has not been addressed in research communities. In this chapter, by relying on MCS technologies we propose a novel model of TTDP and open up new possibilities for city governments to affect tourists' trip planning by promoting or putting restrictions on certain POIs or routes between them.

Finding personalized tourist itineraries that achieve also mobility goals of city governments is the challenging combinatorial problem which can be formulated as a variant of the Mixed Orienteering Problem (MOP). MOP is an NP-hard combinatorial optimization problem and to the best of our knowledge there have not been many research studies focusing on achieving a short response time for it. Furthermore, we consider the problem on complex instances formulated on complete graphs and multigraphs. However, the state-of-the-art approaches for solving MOP and its variants can only be efficiently applied on low degree graphs usually employed to represent traffic road networks.

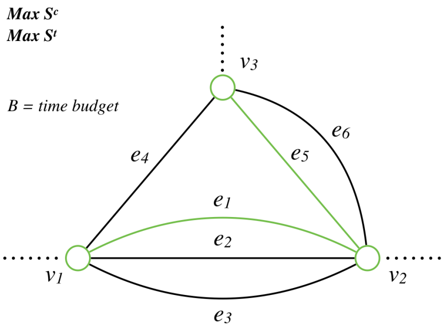
The problem of designing a single tourist itinerary can be considered as an Orienteering Problem (OP) [116]. In an OP, a set of locations is given and it is associated with certain benefit scores representing the estimated tourist's satisfaction associated with the visit of the locations. The goal is to determine a single path (itinerary) of limited length that visits some of the locations and maximizes the total sum of collected scores. The OP has been proven to be an NP-hard problem [117] and thanks to its algorithmic complexity, it has been in the focus of many research studies. Therefore, different versions and extensions of the OP have been successfully applied to model various trip design problems.

The arc routing version of the OP has been introduced by Souffriau et al. in [118] as the Arc Orienteering Problem (AOP). AOP has been used to solve the problem of finding personalized travel routes on large-scale road networks ([118–120]). In AOP each arc is assigned with the profit score usually representing the personalized scenic value associated with a segment of road. The goal of the problem is to construct an optimal travel route by selecting a subset of edges. In this section, we combine the OP and the AOP and solve the problem introduced under the name Mixed Orienteering Problem (MOP) in [121]. In MOP, profit scores are associated both with the nodes and with the edges. MOP is usually applied to solve TTDP when not all the attractions are represented POIs, but when a walk along a scenic route between them can also be considered as an attraction.

We use MOP to formulate the variant of TTDP where each POI and route, i.e. node and edge, are associated with two profit scores representing the importance of a visit or walk for the tourist and the city government respectively. In this way, we enable city administrators to control the tourists' trip planning by assigning their own scores, i.e. representing mobility policies, on POIs and routes between them. We also allow parallel edges between nodes representing alternative routes between POIs. Similar to other orienteering formulations, the goal of the problem is to find a single path limited in length, with favourable allocation of collected city and tourist scores. Here we are primarily focused on solving the introduced optimization problem, and therefore we omit any discussion on how the city and tourist scores can be defined.

5.1.1. Formal problem description

Our problem can be formally defined as finding a path on an undirected multigraph $G = (V, E, sig : E \to \mathcal{P}_2(V))$, where $V = \{v_1, ..., v_N\}$ is the set of nodes representing POIs,



	S^c	S^t	t/d
v_1	0.3	0.8	30
v_2	0.2	0.9	90
v_3	0.7	0.2	60
e_1	0.8	0.4	12
e_2	0.3	0.2	14
e_3	0.5	0.4	15
e_4	0.1	0.2	30
e_5	0.7	0.5	11
e_6	0.4	0.3	10

Figure 51: Problem formulation on an example subgraph. The table on the right contains the city and tourist profit scores (S^c, S^t), visit durations (d) and travel times (t) associated to nodes and edges. A possible segment of a solution path is highlighted with lighter colour.

and $E = \{e_1, ..., e_M\}$ is the set of edges representing alternative routes between them. Since the given multigraph G can have multiple edges along the same nodes, we define the mapping $sig : E \rightarrow \mathcal{P}_2(V)$ which tells us for a given edge $e \in E$, the two nodes it connects. The travel time $t(e_i)$ and the visit duration $d(v_j)$ are associated with each edge $e_i \in E$ and node $v_j \in V$, respectively. Additionally, each edge and node are also associated with the city and the tourist profit scores $S^c, S^t : V \cup E \rightarrow \mathbb{R}^+$. The city profit score S^c represents the importance of a visit to a certain POI (node), or a travel along a certain route (edge) for achieving the city's global mobility objectives. The tourist profit score S^t represents the estimated degree of the tourist's satisfaction with the visit or travel.

The goal of the proposed problem is to determine a path starting at v_1 and ending at v_N, having a length not exceeding time budget (B), and maximizing the total city and tourist profit scores collected over a path. Here we use the graph theory concept of path to represent a single itinerary. Figure 51 illustrates the problem on an example subgraph where a subset of edges, i.e. a segment of a solution path, are highlighted.

5.1.2. Mixed-integer linear programming model

We further formalize our problem as a Mixed-integer Linear Programming Problem (MILP). We employed the commercial MILP solver (IBM ILOG CPLEX Optimizer v12.6.3) to obtain optimal solutions for generated test instances. Due to its high computational time, the following MILP model is not intended to be used as an efficient solution to the proposed problem, but rather to better understand its complex structure. We first define the following decision variables:

$$x_i = \begin{cases} 1 & \text{if node } v_i \text{ is visited in the solution path} \\ 0 & \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if edge } e_i \text{ is visited in the solution path} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{ij} = \begin{cases} 1 & \text{if, in the solution path, a visit to node } v_i \text{ is} \\ & \text{followed by a visit to node } v_j \\ 0 & \text{otherwise} \end{cases}$$

$$u_i \Longrightarrow \text{position of node } v_i \text{ in the solution path}$$

Using the introduced notation we formulate our problem as follows:
Maximize

$$\begin{gathered} \alpha \left(\sum_{i=1}^{N} S^c(v_i)x_i + \sum_{j=1}^{M} S^c(e_j)y_j \right) + \\ (1-\alpha) \left(\sum_{i=1}^{N} S^t(v_i)x_i + \sum_{j=1}^{M} S^t(e_j)y_j \right) \end{gathered} \tag{5.1}$$

subject to:

$$\sum_{i=2}^{N} z_{1i} = \sum_{j=1}^{N-1} z_{jN} = 1 \tag{5.2}$$

$$\sum_{i=1}^{N} z_{ik} = \sum_{j=1}^{N} z_{kj} \leq 1; \quad \forall k = 2, ..., N-1 \tag{5.3}$$

$$\sum_{i=1}^{N} d(v_i)x_i + \sum_{j=1}^{M} t(v_j)y_j \leq B \tag{5.4}$$

$$2 \leq u_i \leq N; \tag{5.5}$$

$$u_i - u_j + 1 \leq (N-1)(1 - z_{ij}); \quad \forall i, j = 2, ..., N; i \neq j \tag{5.6}$$

Objective equation (5.1) represents the city and tourist profit scores collected over the entire solution path. Factor $0 \leq \alpha \leq 1$ is a tuning parameter which determines proportion of the two measures (city and tourist scores) in the final score. Equation (5.2) ensures that solution path starts at node v_1 and ends at node v_N. Equation (5.3) ensures the connectivity of the path and guarantees that each node and each edge is visited at most once. Equation (5.4) ensures that the length of the path is under preset time budget B. Finally, equations (5.5) and (5.6) represent Miller-Tucker-Zemlin (MTZ) subtour elimination constrains [122].

The proposed model is an extension of MOP, i.e. a variant of OP which has been proven to be NP-hard by Golden et al. in [117]. Therefore, finding the exact solution for even smaller problems formulated with the proposed MILP model would be computationally expensive and time consuming. However, using a heuristic algorithm may allow finding near-optimal solution for this complex combinatorial optimization problem. In the next section we discuss the proposed heuristics.

5.2. Heuristic algorithm

The proposed algorithm consists of two phases: (1) Transformation and (2) Search. In the transformation phase, the initial problem is transformed to an equivalent AOP, where the scores and time costs are associated solely to the edges (routes). In the search phase, we employ the Variable Neighbourhood Search (VNS) metaheuristic [62] to find a near-optimal solution to the transformed problem. Both phases are detailed in this section.

5.2.1. Transformation phase

A special case of the proposed MOP variant where all the nodes have identical scores and visit durations reduces to AOP as follows,

$$\begin{aligned} S^c(v_i) - S^c(v_j) &= 0; \quad \forall v_i, v_j \in V \\ S^t(v_i) - S^t(v_j) &= 0; \quad \forall v_i, v_j \in V \\ d(v_i) - d(v_j) &= 0; \quad \forall v_i, v_j \in V. \end{aligned}$$

We define a reduction in which $S^c(v_i) = S^t(v_i) = d(v_i) = 0$ for all $v_i \in V$, and, hence, all three the above AOP conditions are trivially satisfied. We do this in a way which ensures that this modified problem is equivalent to the original one. Since each node in MOP can be visited only once, its city and tourist scores can be collected by traversing a pair of its incident edges contained in the solution path. Therefore, we take half of the node's score and add it to all of its incident edges. This way, half of the score will be collected by traversing an edge when entering and half when leaving the node. However, two special cases arise from the proposed method. Since the solution path neither enters the start node v_1 nor exits the end node v_N, for these nodes we add the whole value of the associated scores to all of their incident edges. In Figure 52 we illustrate the proposed approach on an example subgraph. Finally, we formalize the reduction as follows:

$$S^{c'}(v_i) = \begin{cases} S^c(v_i) & \text{if } v_i = v_1 \text{ or } v_i = v_N \\ S^c(v_i)/2 & \text{otherwise} \end{cases}$$

$$S^{t'}(v_i) = \begin{cases} S^t(v_i) & \text{if } v_i = v_1 \text{ or } v_i = v_N \\ S^t(v_i)/2 & \text{otherwise} \end{cases}$$

$$d'(v_i) = \begin{cases} d(v_i) & \text{if } v_i = v_1 \text{ or } v_i = v_N \\ d(v_i)/2 & \text{otherwise} \end{cases}$$

$$S^c_{new}(v_i) = S^t_{new}(v_i) = d_{new}(v_i) = 0; \quad \forall v_i \in V$$

$$\begin{aligned} S^c_{new}(e_i) =& S^c(e_i) + \sum_{v_j \in sig(e_i)} S^{c'}(v_j) \\ S^t_{new}(e_i) =& S^t(e_i) + \sum_{v_j \in sig(e_i)} S^{t'}(v_j) \\ t_{new}(e_i) =& t(e_i) + \sum_{v_j \in sig(e_i)} d'(v_j) \end{aligned}$$

We apply the same reduction to the nodes' time costs, i.e. we take half of the node's visit duration time and add it to the travelling costs associated to its incident edges. Notice, that in the case of a direct graph we would add the whole value of the node's score/cost to either all of its incoming or all of its outgoing edges.

5.2.2. Search phase

In the search phase we employ the Variable Neighbourhood Search (VNS) metaheuristic proposed by Mladenovic and Hansen in [62]. The basic idea behind the VNS framework is to

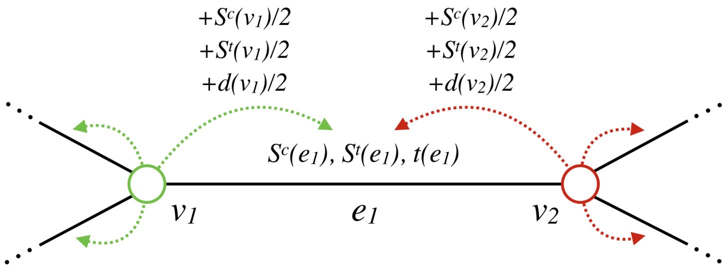


Figure 52: Reducing the MOP to an equivalent AOP. Half of the node's score/cost is added to all of its incident edges. This way half of the score/cost will be collected by traversing an edge when entering and half when leaving the node.

search the solution space with a systematic change of neighbourhood, by both descending to local optima and escaping from valleys which contain those local optima. VNS is often used to find high-quality solutions to numerous combinatorial optimization problems thanks to its basic structure which is still very simple and requires few, and sometimes no parameters. First, we briefly explain the basic scheme of VNS. Then, we discuss how the solution to our problem can be conceptually represented. Finally, we propose search heuristics and implement VNS procedures for search intensification and diversification.

Algorithm 3: The basic VNS

```
Define neighbourhood structures $N_k$ $(k = 1, ..., k_{max})$
Generate initial solution $s \in S$
Choose a stopping condition
while stopping condition is not met do
    $k = 1$
    while $k \leq k_{max}$ do
        $s' = \text{Shake}(s);\ s' \in N_k(s)$
        $s'' = \text{LocalSearch}(s');\ s'' \in S$
        if $Score(s'') > Score(s')$ then
            $s = s''$
            $k = 1$
        else
            $k = k+1$
        end
    end
end
```

The basic scheme of VNS is given in Algorithm 12 where N_k, $k = 1, ..., k_{max}$, represents the k^{th} neighbourhood structure, S represents the set of all feasible solutions, and $N_k(s)$ represents the set of solutions in the k^{th} neighbourhood of solution s. Starting from a chosen initial solution s, the algorithm systematically explores solution space S and tries to find the optimal one. First, the `Shake` procedure randomly selects a new solution s' within a k^{th} neighbourhood $N_k(s)$. Then, a descent from s' is done by the `LocalSearch` routine which leads to a new local optima s''. At this point, the new solution s'' is compared with the

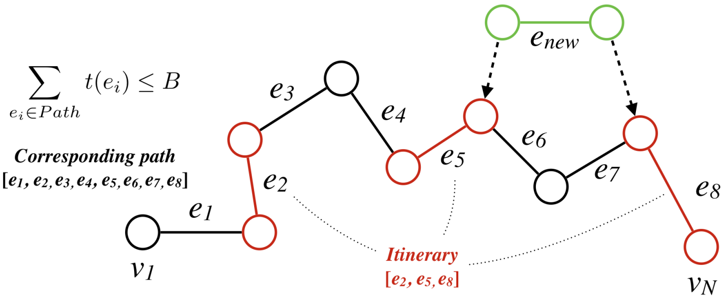

$\sum_{e_i \in Path} t(e_i) \le B$
Corresponding path
[e1, e2, e3, e4, e5, e6, e7, e8]
e1
e2
e3
e4
e5
e6
e7
e8
enew
v1
vN
Itinerary
[e2, e5, e8]

Figure 53: Example of a valid solution. Itinerary contains only three edges ($\langle e_2, e_5, e_8 \rangle$), while its corresponding path connects the gaps between the edges using the cheapest paths ($\langle e_1, \mathbf{e_2}, e_3, e_4, \mathbf{e_5}, e_6, e_7, \mathbf{e_8} \rangle$).

incumbent solution s. If s'' is better than s, the algorithm recenters the search by replacing s with s'', and starts all over from the first neighbourhood structure $N_1(s)$. Otherwise, the algorithm moves to the next neighbourhood structure and draws a new random solution from $N_{k+1}(s)$. Obviously, the choice of neighbourhood structures, as well as the implementation of intensification and diversification routines (`LocalSearch` and `Shake`), are critical for the performance of the algorithm. However, before we discuss them in detail, we need to define how we conceptually represent solutions in our tourist itinerary.

We adopt the idea proposed by Souffriau et al. in [118] and represent the solution to our tourist itinerary problem, as a list of visited edges or list of routes between POIs. These edges are not required to be connected, and thus the score and the cost of an itinerary are determined by the corresponding path which connects non-consecutive edges in the itinerary with the cheapest paths. An itinerary is then a valid solution if:

1. Its corresponding path starts at node v_1.
2. Its corresponding path ends at node v_N.
3. Its corresponding path does not visit any node or edge more than once.
4. The cost of its corresponding path does not exceed the time budget B.

In Figure 53 we give an example of a valid solution. The itinerary contains three edges ($\langle e_2, e_5, e_8 \rangle$). Its corresponding path is created by augmenting the itinerary with the cheapest paths between the non-consecutive edges ($\langle e_1, \mathbf{e_2}, e_3, e_4, \mathbf{e_5}, e_6, e_7, \mathbf{e_8} \rangle$). Therefore, the total cost of the solution equals the sum of the travelling costs associated to the edges contained in the corresponding path, i.e. $cost(s) = \sum_{e_i \in Path} t(e_i)$. Similarly, the total score of the solution equals the sum of the city and tourist scores associated to edges contained in the corresponding path, i.e. $score(s) = \alpha \sum_{e_i \in Path} S^c(e_i) + (1-\alpha) \sum_{e_i \in Path} S^t(e_i)$ where α is the tuning parameter introduced in Section 5.1.2. Finally, it is worth mentioning that in order to reduce the computational time needed to generate the corresponding paths, the network can be preprocessed by computing all pairwise shortest paths within a certain range.

The introduced model also allows us to easily insert a new edge between any pair of non-consecutive edges in the itinerary. For example, in Figure 53 a new edge e_{new} is inserted in the itinerary between edges e_5 and e_8. The corresponding path is then updated by filling the gap between edges e_5 and e_{new}, e_{new} and e_8 with the cheapest paths, eliminating edges e_6 and e_7. If the new itinerary is again a valid solution according to the above conditions, we say that the applied insert move is valid.

Algorithm 4: Shake procedure

```
Input: Solution $s$, neighbourhood index $k$
Output: New random solution $s' \in N_k(s)$
Procedure Shake(s,k)
    $s' = s$
    $i = 0$
    while $i \leq k$ do
        remove edge from $s'$ at random
        $i = i + 1$
    end
    CloseGaps($s'$,RAND)
    return $s'$
```

Algorithm 5: CloseGaps procedure

```
Input: Solution $s$, method of move selection $mode \in \{RAND, BEST\}$
Output: Updated solution $s$
Procedure CloseGaps(s,mode)
    $insertList$ = FindValidInsertMoves($s$)
    while $insertList$ not empty do
        if $mode == RAND$ then
            $chosenMove$ = random move from $insertList$
        else if $mode == BEST$ then
            $chosenMove$ = best move from $insertList$
        end
        Apply $chosenMove$ to $s$
        $insertList$ = FindValidInsertMoves($s$)
    end
    $s$.itinerary = $s$.correspondingPath
    return $s$
```

Using the described solution model and insert move, we can select our neighbour structures for VNS. We define the neighbourhood $N_k(s)$ of a solution s as a set of all feasible solutions created by removing k edges from solution s and then inserting new ones by applying a number of valid insert moves. Moving from one neighbourhood to another is achieved in the `Shake` routine by generating a random solution within the $N_k(s)$ neighbourhood. In Algorithm 4 the `Shake` procedure first randomly removes k edges and then calls the `CloseGaps` procedure (Algorithm 5). The `CloseGap` procedure keeps inserting new edges until no valid insert move can be found. Notice that the selection of insert moves in the `CloseGap` procedure depends on the input parameter. In the diversification phase (shaking) the procedure randomly selects a move from the list of valid insert moves. However, in the intensification phase (local search) `CloseGap` will, in each iteration, choose the best possible insert move which maximizes the new score. In the `FindValidInsertMoves` procedure (Algorithm 6), the list of valid insert move is generated by iterating over all possible insert positions and non-included edges. For each position and edge the validity of the insert move is checked by computing the new corresponding path. Notice that in Algorithm 6 the insert moves are represented as triplets $\langle edgeI, edgeJ, newEdge \rangle$, meaning that *newEdge* has to be inserted between *edgeI* and *edgeJ*.

Algorithm 6: FindValidInsertMoves procedure

```
Input: Solution s
Output: List of valid insert moves insertList
Procedure FindValidInsertMoves(s)
    insertList = $\emptyset$
    foreach edgeI in s.itinerary do
        edgeJ = next edge
        foreach newEdge not in s.itinerary do
            if insert move $\langle$edgeI, edgeJ, newEdge$\rangle$ is valid then
                insertList $\leftarrow$ $\langle$edgeI, edgeJ, newEdge$\rangle$
            end
        end
    end
    return insertList
```

In order to iteratively improve the score of a solution constructed with the `Shake` procedure, and thus lead the search to local optima, we intensify the search by introducing two local search heuristics: `Replace` and `Swap`. Heuristic `Replace` seeks to improve a given solution by replacing a pair of consecutive edges in an itinerary with new non-included edges. The choice of edges to be replaced is selected carefully, i.e each pair of consecutive edges is considered for replacement. As shown in Algorithm 7, this is accomplished by creating a temporary solution s' where each pair of consecutive edges is removed and then replaced by the `CloseGap` procedure. The solution with the highest score is chosen at each iteration. The iteration stops when no improvement can be made by replacing any of the non-consecutive edges in the incumbent solution s_{best}. In the intensification phase, the `CloseGap` greedily selects the best insert move at each iteration until no valid move can be found. This guarantees that the solutions found by replacing edges are valid. Also removing two consecutive edges in the described process is equivalent to removing a single node, i.e. POI, from an itinerary.

The second heuristic, `Swap`, has the objective to improve the solution by exchanging positions of two nodes in the itinerary (Algorithm 8). However, a swap between a pair of nodes can very often generate a solution where no valid corresponding path can be created. For example in Figure 54 nodes v_3 and v_7 are swapped, and thus the corresponding path augments the new itinerary by closing the gaps between nodes v_2 and v_7, v_7 and v_4, v_6 and v_3, and v_3 and v_8. However, it is possible that some of the nodes contained in the shortest paths are already included in the itinerary. Therefore, in Algorithm 8, for each pair of nodes the feasibility of swapping is checked by generating a new corresponding path. If the new path is valid and its score is higher than the score of the incumbent solution, the swap is carried out and the algorithm starts all over from a new incumbent solution. The heuristic stops when no improvement can be made. Notice that in `Swap` we use the

```
Algorithm 7: Replace heuristic
  Input: Solution $s$
  Output: Improved solution $s_{best}$
  Procedure Replace(s)
      $s_{best} = s$
      $improved$ = true
      while $improved$ do
          $improved$ = false
          $s' = s_{best}$
          foreach $edgeI$ in $s'.itinerary$ do
              $edgeJ$ = next edge
              $s'' = s'$
              Remove $edgeI$ and $edgeJ$ from $s''.itinerary$
              CloseGaps($s''$,BEST)
              if $Score(s'') > Score(s_{best})$ then
                  $s_{best} = s''$
                  $improved$ = true
              end
          end
      end
      return $s_{best}$
```

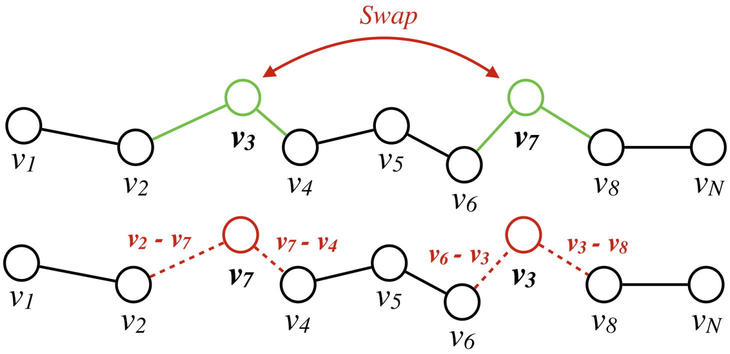


Figure 54: Example of a swap move. Since nodes v_3 and v_7 are swapped, new corresponding path augments the new itinerary by closing the gaps between nodes v_2 and v_7, v_7 and v_4, v_6 and v_3, and v_3 and v_8.

first improvement strategy as opposed to `Replace` where we use the best improvement rule. Finally, in Algorithm 13 we combine the two heuristics in the local search routine.

Algorithm 8: Swap heuristic

```
Input: Solution $s$
Output: Improved solution $s_{best}$
Procedure Swap(s)
    $s_{best} = s$
    $improved$ = true
    while $improved$ do
        $improved$ = false
        foreach $nodeI$ in $s_{best}.itinerary$ do
            $nodeJ$ = next node
            $s' = s_{best}$
            Swap $nodeI$ and $nodeJ$ in $s'$
            Update corresponding path for $s'$
            if $s'$ valid && $Score(s') > Score(s_{best})$ then
                $s_{best} = s'$
                $improved$ = true
                break
            end
        end
    end
    return $s_{best}$
```

5.3. Experimental studies

We implemented the basic variant of VNS where the `Shake` routine diversifies the search with systematic change of neighbourhood, and the `LocalSearch` routine which intensifies the search by moving to local optima. In the next section we experimentally evaluate the proposed algorithmic approach.

5.3.1. Dataset

To evaluate the efficiency and accuracy of the proposed algorithm, we created a real-life dataset[1] related to the city of Barcelona. The dataset contains 800 test instances with different graph topologies. To the best of our knowledge, this is the largest mixed orienteering dataset in terms of the number of non-trivial problem instances with known optimal solutions. The complexity of instances varies in terms of the number of selected POIs and routes between them, the associated scores, and the time budget constraints. 10 test instances were generated for 10 different sized graph topologies (with 10, 20, 30, ... , and 100 POIs) and 8 different time budget constraints (180, 240, 300, 360, 420, 480, 540 and 600 minutes). Given that in TTDP only the most relevant POIs with the highest scores are selected in graphs, the complete graphs with < 100 nodes can be efficiently used to produce

[1] We encourage researchers to freely use the dataset provided at `https://github.com/pmrazovic/BarcelonaMOP`.

Algorithm 9: LocalSearch routine

```
Input: Solution $s$
Output: Improved solution $s'$
Procedure LocalSearch(s)
    $s' = \text{Replace}(s)$
    $s' = \text{Swap}(s')$
    $s' = \text{Replace}(s')$
    return $s'$
```

high-quality itineraries. The tourist scores assigned to POIs were computed for different tourist profiles represented as random distributions of tourists' interests in five categories: Art, Architecture, Museums, Leisure and Shopping. The city scores, on the other hand, were randomly generated for all POIs and routes between them. The cost of each visit to a POI is set to 45 minutes, while the travelling cost between POIs is computed as a walking time needed to traverse Euclidean distance between them. Finally, we randomly create up to 3 alternative routes between each pair of POIs yielding high degree multigraphs where all the edges are profitable (i.e. assigned with score).

5.3.2. Ground truth

In order to use the dataset as a benchmark to compute the accuracy of the proposed algorithm, we obtained the optimal solutions to the test instances by implementing MILP model (5.1)-(5.6) using the IBM's commercial MILP solver ILOG CPLEX Optimizer (v12.6.3). The computation is performed on a machine with 2.40 GHz Intel® Xeon® E5-2630 processor and 265 GB of RAM. Computing the exact solutions to even moderate sized NP-hard problems such as MOP takes very long time. For some of the more complex test instances in our dataset computational time exceeded 15 hours. Therefore, we limited computational time per instance to 3 hours. Consequently, computing the optimal solutions for all 800 instances lasted 20 days in total. Nevertheless, 88% of the instances were optimally solved. For the rest of the instances, solutions are proven to be within 3.54% from optimal (in average), meaning there is high probability that these solutions are optimal since only a small part of the solution space has been left unexamined. In order to demonstrate how the instance complexity is affected by the graph topology and time budget restrictions, in Figure 55 we show the correlation between (a) computational time and size of the graph and (b) computational time and time budget constraint. Notice that computational time. i.e. solving complexity, generally grows with graph size (number of nodes) and budget constraint. However, occasional peaks indicate that computational time is also heavily influenced by the graph structure and distribution of scores among nodes and edges, which both differ among generated instances.

5.3.3. Baseline algorithms

Since the MOP is reduced to AOP, we compared our heuristic approach with two recent heuristic algorithms for AOP: Greedy Randomized Adaptive Search Procedure-based algorithm (GRASP) [118] and Iterated Local Search (ILS) [119]. Since the baseline algorithms were commercialized, we reimplemented them based on published descriptions. However, in order to assure a fair comparison, we increased the allowed execution time to 30 seconds

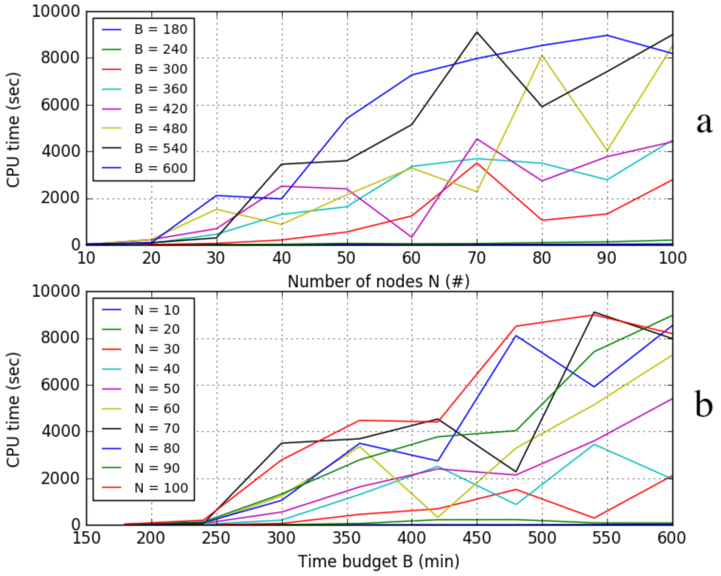


Figure 55: Correlation between (a) computational time and size of the graph (number of nodes) and (b) computational time and time budget constraint.

for all algorithms. Additionally, we precomputed all pairwise shortest paths between all the nodes in graphs using the Dijkstra's algorithm. Finally, we measured the performance of the algorithms with the accuracy percentage as $accuracy = \frac{score}{optimal\_score} \cdot 100\%$, where *score* and *optimal_score* stand for the score of the optimal solution found by the algorithms under evaluation and CPLEX Optimizer, respectively. The algorithms were developed in JAVA, and all experiments were conducted on the same machine with 2.50 GHz Intel® Core i7® 4870HQ processor and 16 GB of RAM.

5.3.4. Computational results

We first present the experimental results on our novel dataset with respect to the complexity of the generated instances. In Figure 65a we plot the average accuracy when varying the size of the corresponding complete graph in terms of the number of nodes. Similarly, in Figure 65b we plot the average accuracy when varying the time budget restriction. In Figure 57 we also show non-averaged accuracy results for the same experiments. The figures show that the proposed heuristic approach (VNS) significantly outperforms the baseline AOP algorithms GRASP and ILS. In this experiment VNS achieves 99.53% average accuracy within 30 seconds of computational time. On the other hand, GRASP and ILS achieve 92.22% and 91.78% average accuracy, respectively. Additionally, the plots show better scalability of VNS compared to baseline algorithms. In Figure 65a we see that accuracy of GRASP and ILS drops 10.41% and 10.69%, respectively, when increasing the size of the complete graph from 10 to 100 nodes. The performance of VNS drops only 0.65%. Similarly, Figure 65b shows an accuracy drop of 12.05% and 0.75% in case of GRASP and ILS, respectively, when increasing the time budget restriction from 180 minutes to 600 minutes, and 0.54% drop in case of VNS. However, in Figure 65b the average accuracy of ILS for instances with 180-minute time budget restrictions (beginning of the ILS curve) stands out with a

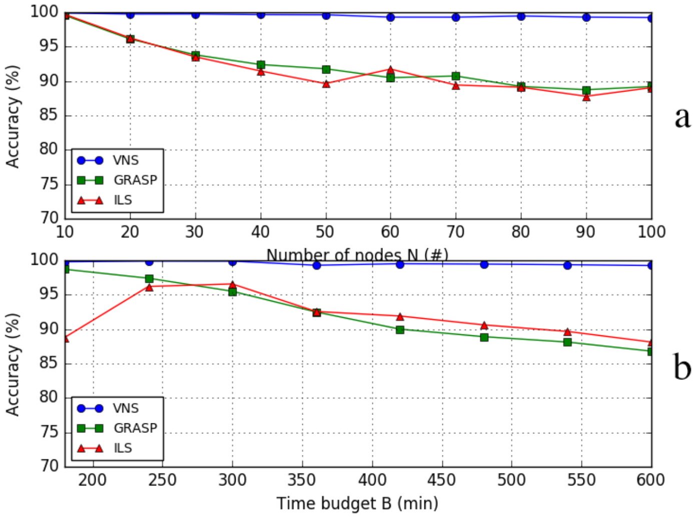


Figure 56: Average accuracy of the heuristic algorithms when varying the graph size (a) and the time budget restriction (b).

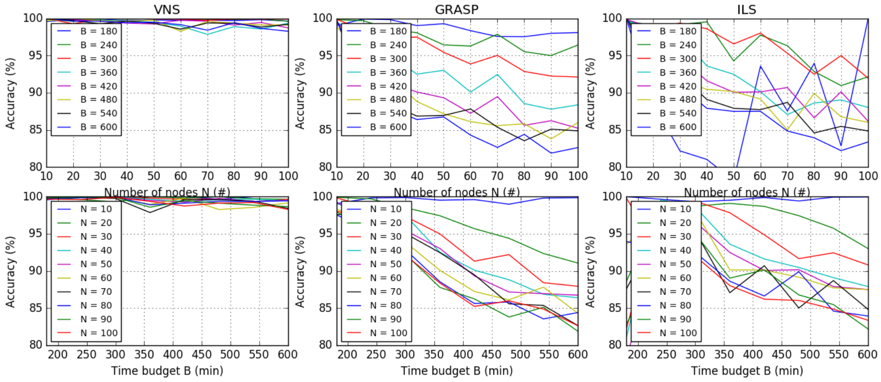


Figure 57: Accuracy of the heuristic algorithms when varying the graph size and the time budget restriction.

significantly lower percentage (88.77%). We explain this irregularity further in this chapter. Besides that, GRASP and ILS perform similarly on our dataset. Their application scope is, to some extent, limited to sparse (low degree) graphs typically used to represent travel road networks. However, here we show their shortcomings when applied on completed graphs with large number of edges, and thus we justify the need for a new algorithm.

In order to further explain the obtained results we briefly discuss implementation details of GRASP and ILS algorithms, and experimentally show their effects in Figure 58. For more information about the baseline algorithms we refer the reader to [118] and [119]. Figure 58 plots the accuracy versus (a) the elapsed time and (b) the number of iterations for each of the algorithms under evaluation. We conducted 100 experiments on a medium complex

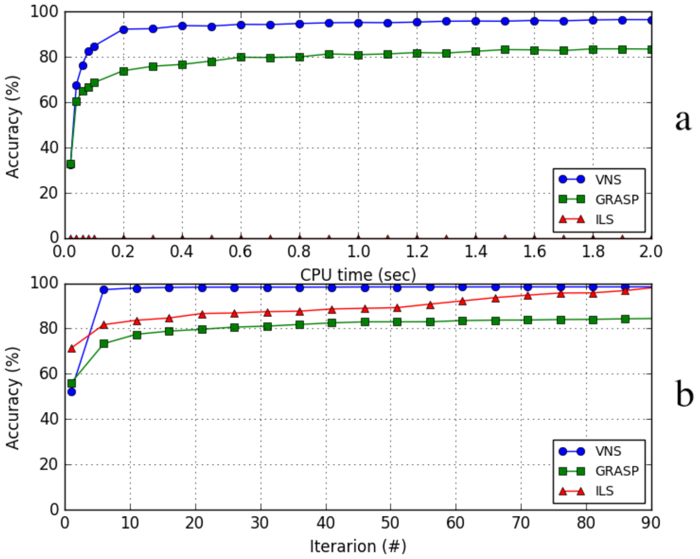


Figure 58: Accuracy versus the elapsed time (a) and the number of iterations (b)

test instance with 70 nodes and 6-hour time budget. In Figure 58b we see that GRASP performs worst in terms of accuracy per iteration, due to its independence of subsequent iterations. Starting from an initial solution, GRASP generates a set of candidate solutions in each iteration, and calculates their total score. The candidate solutions are generated by inserting new non-included edges between pairs of adjacent edges in the current solution. Then, among the candidate solutions whose score values are larger than a random threshold, GRASP selects a new solution at random. Obviously, GRASP only remembers the best solution, while every new iteration starts again from an empty solution. This means that iterations in the GRASP approach can be too random for larger datasets, and thus they slowly converge to optimal solutions. On the other hand, even though the VNS approach also relies on random move selection, it does not ignore the previously detected promising regions in the search space. On the contrary, using a local search procedure, it leads the search to local optima. This effect can be seen as a rapid slope increase at the beginning of the VNS curve in Figure 58b. Notice that VNS can find high-quality solutions ($\sim 97\%$ accuracy) in only a couple of iterations (< 10). Additionally, in Figure 58a we see that VNS needs only a couple of milliseconds to find such solutions.

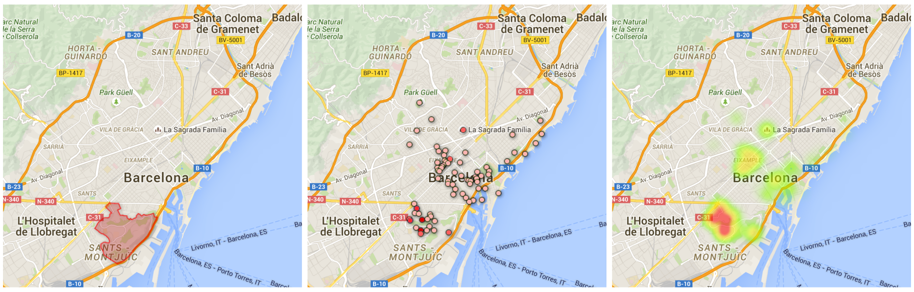
The ILS algorithm was proposed in [119] to speed up iterations. However, as we see in Figure 58a this was not the case in our experiments. Even though it performs better than GRASP in terms of accuracy per iteration (Figure 58b), the high computational time of ILS makes it unsuitable for online applications. In Figure 58a we see that 2 seconds was not enough to complete any iteration, and therefore the accuracy is 0%. ILS implements the iterated local search framework by searching for a new solution at each iteration using a depth-first search (DFS) strategy. Namely, at each iteration ILS removes part of the current solution and tries to close the created gap using DFS. To reduce the DFS search space, ILS checks the feasibility for the edges to be inserted in the current solution. Additionally, it also restricts the depth of the search. Nevertheless, the algorithm performs poorly on complete and high degree graphs. The reason for this is the size of the search tree which is

$O(degree^{max\_depth})$ where *degree* is the average degree of node in graph and *max_depth* is the parameter used to restrict the depth of the DFS tree. In [119] Verbeeck et al. explain that ILS performs well on real-life road network, where usually only four roads come together at a vertex, and thus the size of the search tree is $O(4^{max\_depth})$. Even though Figure 58a does not tell us much about ILS except that it is not capable to produce a solution within 2-seconds computational time, when measuring performance of each iteration in ILS (Figure 58b) we noticed that, in average, computational time exceeds 10 seconds per iteration ($max\_depth = 20$ [119]). Also notice that in each iteration DFS terminates after the first improvement has been found. This means that the iterations in which the current solution is close to optimal consume a lot of computational time, since a large part of the search tree needs to be traversed in order to improve the current solution. This explains the beginning of the ILS curve in Figure 65b. There are few feasible solutions under a 180-minute time budget restriction, which is why iterations last longer and ILS does not succeed to improve the solution significantly in 30 seconds.

5.3.5. Application case

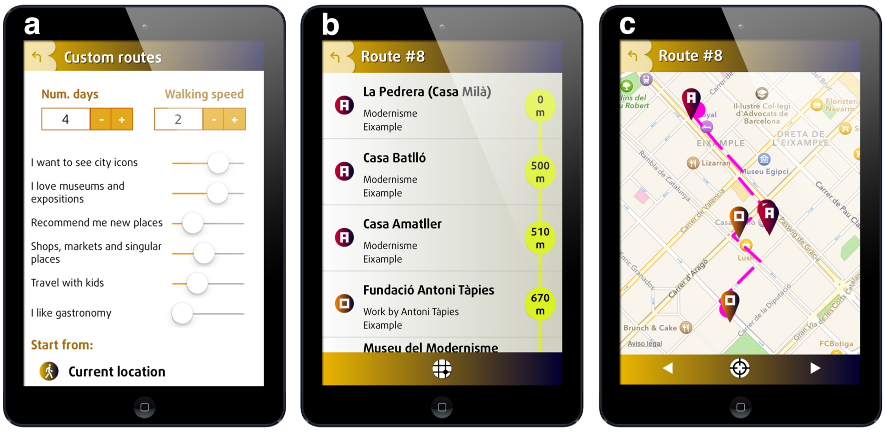
In this experiment we illustrate how the proposed TTDP and MCS paradigm can be used to achieve city administrators' mobility objectives. We simulate a real-world use case where a certain urban area is promoted by tuning the city scores assigned to corresponding POIs and routes between them. The reasons for promoting a certain area can be various, e.g. the area is not popular among tourists, traffic in city needs to be balanced, festivals are organized in the area, local shopping stores need advertisement, etc.. In this example, we want to promote the area of Montjuic within Sant-Montjuic district in Barcelona (Figure 59a). We first simulate a crowd of tourists with different interests by generating 1,000 new problem instances using random user profiles. In this case the tourist scores in the range [0,100] (%) correspond to interests of each simulated tourist. On the other hand, we assign city scores at random in the range [40,60] (%). This way, we assure that the algorithm respects the tourist's interest to a greater extent. Thereafter, for each POI located in the target area and for each route passing through the area, we increase the city score by 50%. Notice that these scores are in real world crowdsensed from volunteer participants and city administrators. Additionally, the parameter α introduced in Section 5.1.2 is set to 0.5, meaning that we equally respect city's and tourist's interests. The start and finish locations of an itinerary in each instance are chosen at random within Barcelona's city boundaries. Using the algorithm under evaluation we constructed 1,000 6-hour itineraries. The results are presented in Figure 59.

Figure 59b shows all the POIs selected in at least one of the 1,000 generated itineraries. Since the start and finish locations are chosen at random for each itinerary, these POIs are distributed over the entire city's area. However, the colour of each marker corresponds to total number of POI occurrences in the generated itineraries. In other words, the darker the colour, the more times a POI has been selected. Obviously, the most popular POIs are located within the target area of Montjuic. In Figure 59c we assume that tourists follow recommended itineraries, and using a heatmap we show the accumulated amount of time spent by tourists in different areas around the city. The plot confirms the efficiency of the proposed TTDP model. The most visited areas (in red) are located in the target area, where 1,000 tourists spent 2,606 hours in total. This means that, in average, a tourist spends 2.6 hours of her 6-hour trip (43.43 %) within the target area of Montjuic. Finally, notice that the coloured areas are usually located around the popular POIs where tourists usually spend most of their time.



a) b) c)

Figure 59: Simulation of a real-world use case where a certain urban area is promoted. In Figure (a) target area is highlighted. Figure (b) shows all the POIs selected in itineraries. Figure (c) shows the accumulated amount of time spent by simulated tourists in different areas around the city.



Custom routes

Num. days 4 - +

Walking speed 2 - +

I want to see city icons

I love museums and expositions

Recommend me new places

Shops, markets and singular places

Travel with kids

I like gastronomy

Start from:

Current location

Route #8

La Pedrera (Casa Milà) Modernisme Eixample 0 m

Casa Batlló Modernisme Eixample 500 m

Casa Amatller Modernisme Eixample 510 m

Fundació Antoni Tàpies Work by Antoni Tàpies Eixample 670 m

Museu del Modernisme

Route #8

Figure 510: PET example: Barcelona Official Guide mobile application

5.4. System pilot

To investigate the usability of the proposed TTDP we developed a MCS-driven mobility management platform which allows city governments to formulate mobility policies, and thus promote or reduce the traffic of people flows in certain areas. However, instead of pushing the final users, i.e. tourists and citizens, to follow the created policies, the policies are injected into Personalized Electronic Tourist Guides (PET) linked to the MCS platform. Advantages of such approach are threefold: city governments can easily implement their mobility goals, PET providers can upgrade their logistic and mobility services with respect to the present mobility in the city, and final users can move in a less crowded and balanced city.

In our pilot project, we upgraded the existing mobile application, Barcelona Official Guide provided by *Turisme de Barcelona* and *Triangle Postals S.L.*, and connected it to the developed MCS-driven mobility management platform (Figure 510). We integrated

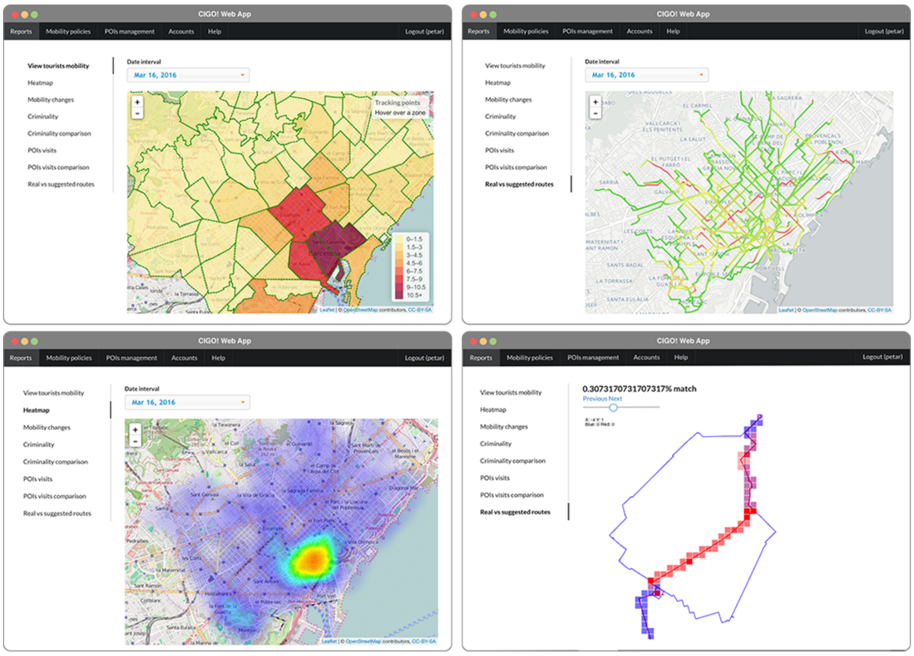


Figure 511: Client Web application

*Sparksee[2] mobile graph database locally into the application, containing POIs (represented as nodes) and different travel routes between them (represented as edges). The graph database can be easily updated remotely with new POIs, routes and mobility policies, while its out-of-the-box graph algorithms make it suitable for routing and navigation purposes. The mobility policies are represented as weights, i.e. city scores, assigned both to POIs and routes between them, i.e. nodes and edges in the graph database. The policies are updated by simply injecting new city scores to the graph database. Moreover, we also allowed tourist scores to be assigned to POIs and routes between them, representing estimated degree of tourist's satisfaction with a corresponding visit or travel. The tourist scores are computed and assigned dynamically after the user expresses her interest in several POI categories (Figure 510a) in crowdsensing fashion. Finally, the introduced city and tourist scores are used by the locally implemented algorithm to generate personalized tourist itineraries (Figure 510b-c).

In addition to the client mobile application, we developed a Web application which provides insight into present mobility in the city using the data collected from several sources (crowdsensed mobility data such as check-ins, GPS trajectories, or text reports, Open Data, sensors, government data, etc.). This helps city administrators in making important mobility decisions. The data is visualized using different interactive maps, also called *mobility reports*, which suggest how the mobility policies should be formulated. Different maps report on the people concentrations and mobility changes in different city neighbourhoods (Figure 511). Additionally, the criminality records from external data sources are displayed on the maps which allows city administrators to analyse street-level and neighbourhood-level crimes in the city. The mobility tracking modules implemented within the Barcelona Official Guide allow us to report tourist mobility and compare it with the recommended

[2] `http://www.sparsity-technologies.com/#sparksee`

itineraries. By understanding the current mobility trends in the city, city administrators can more easily create and update the mobility policies. We employ interactive maps and allow users to create the mobility policies by choosing a POI, street or neighbourhood from the map and assigning weights to promote or to reduce the corresponding visits. The system then automatically assigns introduced weights to nodes and edges of the underlying graph database. The updated database is then pushed to the linked mobile applications, i.e., MCS agents.

5.5. Related work

Recent technological advances in mobile and wireless communication, as well as the rapid evolution of web technologies and social networking, have opened up new possibilities for mobile tourist recommendation systems and personal electronic tourist guides. The field has been gathering extensive attention from the research community, especially in the last five years. However, similar systems are often proposed under different names, e.g. mobile tourist guides (MTG) [114], personal navigation systems for tourism (PNS) [123], personalised electronic tourist guides (PET) [124], etc. The recent surveys by Gavalas et al. [125] and Borras et al. [126] manage to provide detailed and up-to-date overview of the field. The surveys outline main functionalities offered by mobile recommender systems in tourism and provide taxonomies relying on different classification criteria.

Personalized tour planning has been recognized as one of the key features offered by recommender systems in tourism. The problem consists of identifying the most interesting POIs and planning daily itineraries without violating tourist's time and/or budget restrictions. In the literature, this problem is known as the *Tourist Trip Design Problem* (TTDP). Thanks to its algorithmical complexity, TTDP has been in the focus of many research works. A recent survey of TTDP can be found in [115].

In the last decade, a large number of combinatorial optimization problems in operational research has been modelled as the Orienteering Problem (OP) [121]. In the OP, a set of nodes is given, each with a score. The goal is to generate a path, limited in length, that visits some of the nodes and maximises the total sum of the collected scores. Vansteenwegen and Van Oudheusden [114] successfully formulated TTDP using the OP. Ever since, researchers try to solve new TTDPs with additional restrictions and functionalities, while using extensions of OP as modelling tools. The Team Orienteering Problem (TOP) [127] extends the traditional OP and is used to model multiple tour TTDPs. Different algorithmic approaches for solving TOP have been proposed in literature. Archetti et al. [128] proposed two variants of a generalized tabu search algorithm and a variable neighbourhood search (VNS) algorithm to solve the TOP. On the other hand, Ke et al. [129] proposed Ant Colony Optimization (ACO) approach to find near-optimal solutions for TOP. Finally, Vansteenwgen et al. focus on improving the computational time needed to solve TOP instances. They first implemented a Guided Local Search (GLS) framework [130] and later a Skewed Variable Neighbourhood Search (SVNS) framework [131]. The proposed frameworks are able to obtain good TOP solutions in only a few seconds. Another extension of (T)OP with time windows (TOPTW), is used to model more realistic itinerary planning scenarios where POIs' working hours are considered using predefined time windows. Vansteenwegen et al. [132] proposed fast and effective Iterated Local Search (ILS) meta-heuristic to solve the TOPTW. Later, the time dependent (T)OPTW is proposed in [133] to model multi-modal public transportation by taking into account time dependency in calculating travel times between POIs. Gavalas et al. [134] tackle the same problem but with no assumptions on periodic service schedules. Further extensions of TOP model more specific scenarios by assigning additional attributes to POIs such as entrance fee or type of visit [135, 136].

Obviously, orienteering problems proved to be a valuable tool in modelling a wide range of realistic TTDP. However, to the best of our knowledge, the attempt to formulate and solve TTDP with respect to global city needs has not yet been addressed in research communities. In this work we propose to extend the OP by assigning scores both to POIs end edges between them. This extended formulation has been described in [121] as a combination of OP and Arc Orienteering Problem (AOP) where scores are associated only to edges (arcs). Therefore, the formulation has been named Mixed Orienteering Problem (MOP). In [137] Gavalas et al. present approximation algorithms for the AOP and show that MOP can be reduced to AOP. However, in our model of TTDP we also allow parallel edges between POIs in order to represent alternative routes between them. Additionally, we separate the city's and tourists' goals by introducing bi-objective scores. Nevertheless, we manage to reduce the proposed model to AOP and therefore we compare our solution approach with two recent metaheuristic algorithms for AOP: Greedy Randomized Adaptive Search Procedure-based algorithm (GRASP) [118] and Iterated Local Search (ILS) [119]. We point out the impracticality of these state-of-the-art AOP approaches for solving proposed model, as they can only be efficiently applied to low degree graphs usually employed to represent traffic road networks. However, we adopt the conceptual representation of solutions from [118] and build new heuristic approach for our problem by using variable neighbourhood search.

5.6. Conclusion

In this chapter we demonstrated that the tourist trip planning problem can be used by the city government to manage the urban environment and achieve a balanced and sustainable growth. The introduced problem was formulated as a variant of MOP where each POI and route, i.e. node and edge, is associated with two profit scores representing the importance of a visit or walk for the tourist and the city government. In this way we allowed city administrators to control the planning process by simply assigning their own scores to POIs and routes between them.

In order to find a good trade-off between helping tourists in planning their trips and city governments in achieving mobility objectives, we proposed an algorithmic approach based on the VNS framework. The algorithm is evaluated on a large real-life dataset related to the city of Barcelona, one of the most visited cities in Europe. The proposed algorithm generates an average error of 0.47% within 30 seconds of computational time, and thus outperforms baseline AOP algorithms applied to the same dataset. The proposed algorithm can also find high-quality solutions ($\sim 97\%$ accuracy) in fewer iterations (< 10) in only a couple of milliseconds, which makes it suitable for online applications. Finally, developed pilot MCS system demonstrates how city administrators can easily make informed decisions based on travel preferences and mobility data crowdsensed from citizens and visitors.

Chapter 6

Multi-vehicle route planning approach for balancing urban freight transport

This chapter is based on materials from the following peer-reviewed paper:

> P. Mrazovic, E. Eser, H. Ferhatosmanoglu, and J. L. Larriba-Pey, "Multi-vehicle Route Planning for Efficient Urban Freight Transport", in *Intelligent Systems (IS), 2018 9th IEEE International Conference on*, IEEE, 2018.

In the previous chapter we focused on finding individual routes which are in line with the city's global mobility objectives. However, the proposed solution does not consider batch queries, and as such it suffers from lack of coordinated planning. In other words, different travellers can be in parallel guided to the same parts of the city, and thus collapse them by creating congestions, which is completely opposite to the city's global mobility objectives. Therefore, in this chapter, as an extension to the previous problem, we aim to balance the use of public traffic infrastructures by developing multi-query route planning algorithm. As a case study we return back to our MCS system introduced in Chapter 4 and propose an application to improve the circulation of freight vehicles and to avoid congestions in urban freight transport systems. The accompanying routing algorithm will take into consideration capacity constraints imposed by dense urban environments and respectively coordinate a fleet of freight vehicles by avoiding traffic jams and congestions.

6.1. Introduction

As we previously mentioned in Chapter 4, provision of loading areas designated for the freight vehicles has been often recognized as one of the most effective measures for organizing last-mile delivery operations. However, we also explained that nowadays these areas can hardly absorb rapidly increasing urban transport demands. They are becoming over-occupied, and there is not much room left to improve the urban transport and parking infrastructure. This requires new technologies for better managing shared loading areas and planning urban freight transport activities. In this chapter we investigate computational solutions for developing an efficient urban freight transport system without the need for large investments or sophisticated sensor technologies.

In this chapter we want to address the problem of collective route optimization and consider the time spent in the loading areas. We introduce a novel multi-vehicle delivery route planner as a part of existing MCS systems in order to improve the circulation of vehicles and avoid congestions at the loading areas. As opposed to traditional solutions,

Table 61: Table of notations

Notation	Description
$G(V, E)$	graph model of an urban freight transport network
V	node set representing delivery areas
E	edge set representing shortest paths between loading areas
v_i	node representing a loading area, $v_i \in V$
l_i	capacity of node v_i
e_{ij}	edge representing a shortest path between v_i and v_j, $e_{ij} \in E$
c_{ij}	travel time needed to traverse e_{ij}
H	fleet of vehicles
h_k	delivery vehicle, $h_k \in H$
Q	set of route queries
q_k	route query for vehicle h_k, $q_k \in Q$
D_k	delivery location list, i.e., list of nodes to be visited by vehicle h_k, $D_k \subseteq V$
v_s^k	initial position (depot) of vehicle h_k, $v_s^k \in D_k$
t_s^k	delivery start time for vehicle h_k
d_i^k	estimated duration of delivery performed by h_k at v_i
p_k	solution route for vehicle h_k
w_i^k	amount of time vehicle h_k waits at congested delivery area v_i
$cost(p_k)$	total duration (time cost) of route p_k
t_i^k	arrival time, i.e., time at which vehicle h_k is scheduled to arrive at v_i in route p_k
z_i^k	departure time, i.e., time at which vehicle h_k is scheduled to depart from v_i in route p_k

which merely compute individually optimal routes for each driver, the proposed system focuses on the global requirements and identifies a set of routes which would collectively achieve better overall mobility in the city. While traditional approaches focus only on individual routes and look for minimizing individual travel costs, the waiting times at parking/loading areas need to be considered as a first-class citizen in any traffic system. A large number of vehicles in urban freight transport networks cause congestions both in the edges (roads) and in the nodes (loading areas). Our solutions trades individual sub-optimal routes when they together minimize the overall aggregate travel costs, and avoids cascades of delays due to the limited capacities of loading areas. We formally define a new routing problem as a variant of the Hamiltonian Path Problem (HPP) where each traveller, i.e., deliverer, needs to visit a set of predefined loading areas while her route choice affects the delay of other deliverers in the system due to the limited capacity of the areas.

We will analyse freight transport data collected from citizens and deliverers, and illustrate our system within the urban freight transportation in the city of Barcelona. The dataset contains 3,704,034 parking information for 49,172 deliverers. As we will see later, the results confirm that the multi-vehicle delivery planner improves the circulation of vehicles and avoids congestions at the loading areas. Therefore, our work makes the case and show the need for a collective planning of urban freight transportation with a global optimization task and fairness to all deliverers in the system.

6.2. Problem Statement

In this section, we summarize the main requirements and architecture of the routing system, and define the underlying multi-route optimization problem. We first present the used notations and introduce definitions that will be needed to formalize the problem under study. Table 71 summarizes the notation of concepts and terms used throughout this chapter.

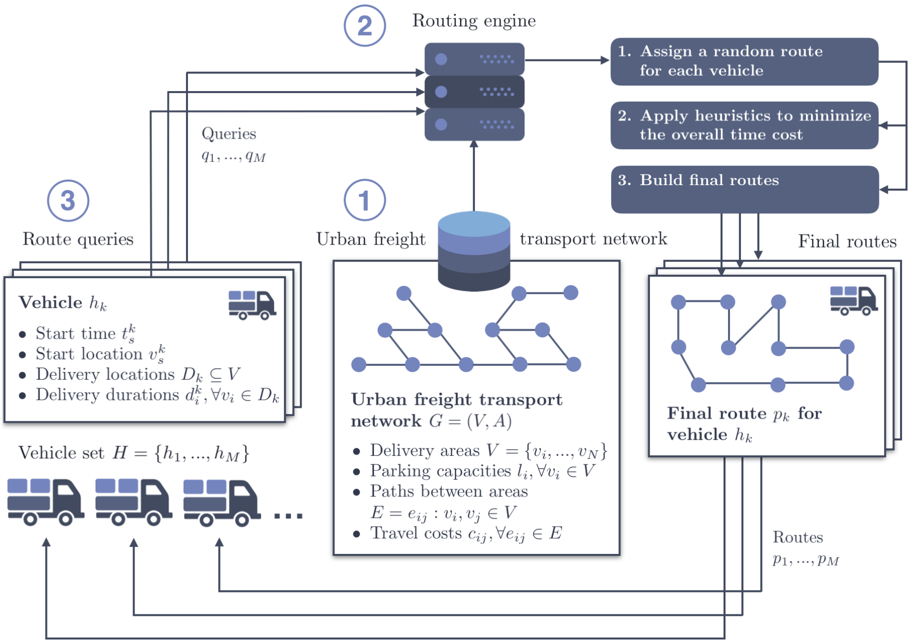

2
Routing engine
1. Assign a random route for each vehicle
2. Apply heuristics to minimize the overall time cost
3. Build final routes
Queries
$q_1, ..., q_M$
3
Route queries
1
Urban freight transport network
Final routes
Vehicle h_k
Start time t_s^k
Start location v_s^k
Delivery locations $D_k \subseteq V$
Delivery durations $d_i^k, \forall v_i \in D_k$
Urban freight transport network $G = (V, A)$
Delivery areas $V = \{v_i, ..., v_N\}$
Parking capacities $l_i, \forall v_i \in V$
Paths between areas
$E = e_{ij} : v_i, v_j \in V$
Travel costs $c_{ij}, \forall e_{ij} \in E$
Final route p_k for vehicle h_k
Vehicle set $H = \{h_1, ..., h_M\}$
Routes
$p_1, ..., p_M$

Figure 61: System components of the proposed multi-vehicle route planner

As illustrated in Figure 61, the developed multi-route planning system consists of (1) urban freight transport network, (2) routing engine, and (3) route query set. The transport network represents the underlying physical traffic network composed of loading areas and roads between them. Naturally, the network can be formalized with the aid of directed graph $G(V, E)$ whose node set $V = \{v_1, ..., v_N\}$ represent loading areas and edge set $E = \{e_{ij} | v_i, v_j \in V\}$ shortest path between them. Furthermore, vehicle capacity l_i and travel cost c_{ij} are assigned to each node $v_i \in V$ and edge $e_{ij} \in E$, respectively. Urban freight transport network, i.e., its graph model, is stored in a graph database and accessed by the routing engine.

Definition 6.2.1 (Graph model) *Let $G(V, E)$ be a directed graph where $V = \{v_1, ..., v_N\}$ is the set of nodes representing loading areas, and $E = \{e_{ij} | v_i, v_j \in V\}$ is the set of edges representing the shortest paths between them. Each node $v_i \in V$ is assigned with the capacity l_i representing the number of parking spaces at the loading area v_i. Each edge $e_{ij} \in E$ is assigned with the travel cost c_{ij}.*

The routing engine is the central component that consists of our solutions for near optimal solving complex route optimization problems. It takes as input a set of route queries $Q = \{q_1, ..., q_M\}$ assigned to delivery vehicles $H = \{h_1, ..., h_M\}$. Each query q_k consists of the vehicle h_k's initial location v_s^k, delivery start time t_s^k, set of delivery locations (i.e., loading areas) to be visited along the planned route D_k, and estimated delivery durations $d_i^k \ \forall v_i \in D_k$.

Definition 6.2.2 (Route query) *Vehicle h_k's route query q_k is composed of its initial position $v_s^k \in V$, delivery start time t_s^k, set of predefined delivery locations $D_k \subseteq V$, and estimated delivery durations $\{d_i^k | v_i \in D_k\}$.*

Route queries $Q = \{q_1, ..., q_M\}$ are concurrent and their delivery location sets D_k $\forall q_k \in Q$ can intersect. The loading areas are limited in capacity, and thus can only accommodate limited number of vehicles at the same time. This means that the route choice for a particular vehicle affects the duration of routes planned for other vehicles in the system. In other words, if a delivery area is at its full capacity, other vehicles need to wait in a queue based on *first-come-first-served* policy before being parked at the congested area (Figure 62). Therefore, the goal is to build an optimal set of routes $P = \{p_1, ..., p_M\}$ (one for each query $q_k \in Q$) which collectively minimizes the total time cost in the transport system expressed as the sum of the durations of all the routes $p_k \in P$.

Definition 6.2.3 (Route duration) *The duration of a single route p_k is computed as the sum of the travel times along p_k, delivery durations d_i^k $\forall v_i \in D_k$, and waiting times w_i^k $\forall v_i \in D_k$. Formally,*

$$cost(p_k) = \sum_{v_i \in D_k} \Big(d_i^k + w_i^k + \sum_{v_j \in D_k} x_{ij}^k c_{ij} \Big), \tag{6.1}$$

where $x_{ij}^k \in \{0, 1\}$ is a decision variable defined to be equal to 1 if edge e_{ij} is traversed in route p_k, and equal to 0, otherwise.

Recall that for a particular route p_k, the travel costs c_{ij} and estimated delivery durations d_i^k $\forall v_i \in D_k$ are known from the graph model $G(V, E)$ and route query q_k, respectively. On the other hand, the queue waiting time w_i^k at each loading area $v_i \in D_k$ needs to be explicitly computed. However, such computation is not straightforward since it depends on other routes in the system, while the delivery durations differ from route to route. Therefore, in the next section we propose an efficient algorithm to compute the total waiting time in $O(n\ log(n))$ time.

We formally define the goal of the proposed multi-vehicle optimization problem as follows:

Definition 6.2.4 (Problem goal) *Given an urban freight transport network $G(V, E)$ and a fleet of vehicles $H = \{h_1, ..., h_M\}$ with the corresponding set of route queries $Q = \{q_1, ..., q_M\}$, the goal of the proposed problem is to determine the set of M routes $P = \{p_1, ..., p_M\}$ that collectively minimize the total time cost, while visiting all of the planned*

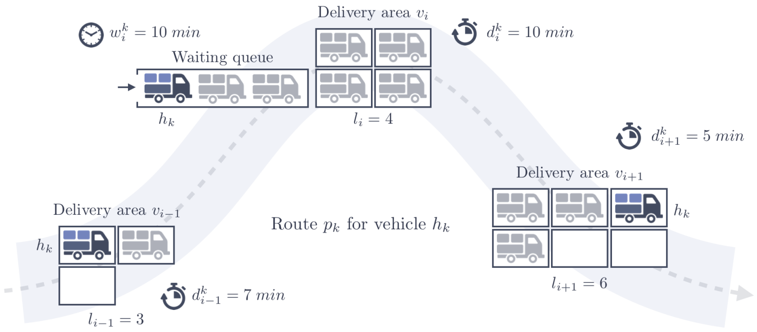


Figure 62: An example of a route with a congested loading area. At the time vehicle h_k arrived at v_i, the area was at its full capacity and thus h_k was placed in a waiting queue.

nodes $v_i \in D_k$ $\forall q_k \in Q$. We further formalize it as follows,

$$\begin{aligned} min\ f(P) &= \sum_{p_k \in P} cost(p_k) \\ &= \sum_{p_k \in P} \sum_{v_i \in D_k} \Big(d_i^k + w_i^k + \sum_{v_j \in D_k} x_{ij}^k c_{ij} \Big), \end{aligned} \tag{6.2}$$

where x_{ij}^k is a decision variable introduced in Definition 6.2.3, and each route $p_k \in P$ visits all of the planned nodes $v_i \in D_k$ starting at node v_s^k and time t_s^k.

Lemma 6.2.1 (NP-completeness) *The proposed multi-vehicle routing problem is NP-complete.*

If the loading areas had unlimited capacities (and consequently no waiting queues), a relaxed variant of the proposed optimization problem can be reduced to a set of separate NP-complete HPPs [138]. The optimal solution, in this case, consists of a set of the shortest travel routes computed individually for each route query. In real life, deliverers tend to seek such routes in order to minimize their total travel time. However, in Section 7.6 we will show that such approach in busy freight networks often leads to congestions and waitings at the delivery areas which additionally increases duration of delivery routes. Therefore, in this work we show that in dense urban environments delivery routes should be planned collectively in advance with fair respect to all deliverers in the system.

Remark 6.2.1 (Size of a solution space) *In order to further illustrate the complexity of the proposed problem, we consider a case with only* 10 *route queries, i.e., $Q = \{q_1, ..., q_{10}\}$, each with* 10 *delivery locations that need to be visited along the solution route, i.e. $|D_k| = 10, \forall q_k \in Q$. Since the first node v_s^k in each route $p_k \in P$ is predefined, there is* 9! *possible ways to arrange the visits in a single route. Therefore, in this particular case, there are* $(9!)^{10} = 3.96 \cdot 10^{55}$ *possible solutions. More generally, the size of a solution space can be computed as follows,*

$$|S(Q)| = \prod_{q_k \in Q} (|D_k| - 1)!\ . \tag{6.3}$$

6.3. The proposed solution

We first introduce the concept of solution representation and propose an efficient algorithm to compute the objective function values (i.e., the total time cost of a solution). We then present the construction and improvement/search phases of the proposed solution.

6.3.1. Solution representation and objective function computation

A solution in our algorithm is represented as a combination of permutations of nodes in each path $p_k \in P$. Therefore, a single path $p_k \in P$ is represented as an ordered set (i.e., tuple) of nodes from delivery location list D_k. For example, a solution $P = \{p_1, p_2, p_3\}$ where $D_1 = \{v_1, v_2, v_3, v_4, v_5\}$, $D_2 = \{v_2, v_4, v_6, v_8\}$, $D_3 = \{v_1, v_5, v_6, v_7, v_9\}$, $v_s^1 = v_s^3 = v_1$, and $v_s^2 = v_2$, is represented as a combination of ordered sets, i.e.,

$$P = \left\{ \begin{array}{l} p_1 = (v_1, v_5, v_3, v_4, v_2), \\ p_2 = (v_2, v_6, v_8, v_4), \\ p_3 = (v_1, v_7, v_9, v_5, v_6) \end{array} \right\}. \tag{6.4}$$

From the introduced representation we can compute the objective function value. As duration of delivery differs from vehicle to vehicle, and from node to node, finding the order in which vehicles leave the waiting queues is not obvious. Therefore, we introduce the concept of *events* and develop a computationally inexpensive algorithm to compute the objective function values.

The proposed algorithm tracks arrivals and departures at/from loading areas. In Algorithm 10, each such event is modelled as a tuple containing the type of the event (arrival or departure), corresponding node and route, and the scheduled time. For example, tuple $(Arrival, p_k, v_i, t_i^k)$ represents arrival at node v_i in route p_k scheduled at time t_i^k. Similarly, tuple $(Departure, p_k, v_i, z_i^k)$ represents departure from node v_i in route p_k scheduled at time z_i^k. Notice that we use a different notation for a scheduled arrival time (t_i^k) and departure time (z_i^k). This is due to the fact that these times are correlated, meaning that one can be computed as a result of the other by knowing estimated waiting time, delivery duration and travel cost. We formally define the arrival and departure times, and discuss their relationship in Remark 6.3.1.

Definition 6.3.1 (Arrival/Departure time) *Arrival time t_i^k is the time at which vehicle h_k is scheduled to arrive at node v_i in path p_k. Departure time z_i^k is the time at which vehicle h_k is scheduled to departure from node v_i in path p_k.*

Remark 6.3.1 (Correlation between arrival and departure time) *Since vehicles depart from loading areas after finishing their deliveries, departure time z_i^k can be computed from arrival time t_i^k by adding waiting time w_i^k and delivery duration d_i^k. Formally,*

$$z_i^k = t_i^k + w_i^k + d_i^k. \tag{6.5}$$

On the other hand, arrival time t_i^k can be computed from departure time z_j^k from v_i's predecessor v_j, by adding travel cost c_{ji}, i.e.,

$$t_i^k = z_j^k + c_{ji}. \tag{6.6}$$

Before we discuss Algorithm 10 in details, we need to introduce the concept of *event queue*, which is a priority queue to store and process arrival and departure events.

Definition 6.3.2 (Event queue) *Event queue T is a priority queue containing arrival and departure events sorted by their scheduled times in ascending order. The queue is implemented as an array indexed by priority, where each array cell contains an event with a scheduled time as priority. By convention, we call the queue insert operation* `enqueue` *and the remove operation* `dequeue`*.*

Algorithm 10 uses the event queue as a buffer where scheduled events wait to be processed one by one. In other words, until event queue T is emptied, at each iteration, the algorithm will remove and process the earliest scheduled event from T (lines 8-34). If the currently being processed event (e) is an arrival at node v_i in route p_k, the algorithm computes departure time z_i^k using (6.5), and inserts a new departure event into event queue T (lines 13-23). However, in order to compute a new departure time z_i^k we need to know the exact time at which delivery area v_i becomes available for vehicle h_k. For that purpose, for each node $v_i \in V$ we keep a list of departure times scheduled for the events contained in T that still need to be processed (`departures[`v_i`]` $\forall v_i \in V$). If the size of list `departures[`v_i`]` is larger than node v_i's capacity l_i, then vehicle h_k will be placed in the waiting queue. In this case, we compute the size of the waiting queue b (line 17) and retrieve the b-th earliest departure time z_i^b at which loading area v_i will become available for vehicle h_k from

Algorithm 10: Objective function computation

```
Input: Solution $P = \{p_1, ..., p_M\}$
Output: Objective function value, i.e., total time cost
Procedure ComputeCost(P)
    departures[$v_i$] ← [∅], $\forall v_i \in V$
    cost[$p_k$] ← 0, $\forall p_k \in P$
    $T$ ← [∅]
    forall $p_k \in P$ do
        $T$.enqueue($Arrival, p_k, v_s^k, t_s^k$)
    end
    while $T$ not empty do
        $e$ ← $T$.dequeue()
        Let $v_i$ represent the node assigned to event $e$
        Let $p_k$ represent the route assigned to event $e$
        if event $e$ is an arrival then
            Let $t_i^k$ represent the scheduled time of event $e$
            if departures[$v_i$].length $< l_i$ then
                $z_i^k \leftarrow t_i^k + d_i^k$
            else
                $b$ ← departures[$v_i$].length - $l_i$
                $z_i^b$ ← departures[$v_i$].get($b$)
                $z_i^k \leftarrow z_i^b + d_i^k$
                cost[$p_k$] ← cost[$p_k$]+$z_i^k - t_i^k$
            end
            departures[$v_i$].add($z_i^k$)
            $T$.enqueue($Departure, p_k, v_i^k, z_i^k$)
        else if event $e$ is a departure then
            Let $z_i^k$ represent the scheduled time of event $e$
            departures[$v_i$].remove($z_i^k$)
            if $v_i$ is not the last node in $p_k$ then
                Let $v_j$ represent $v_i$'s successor in $p_k$
                $t_j^k \leftarrow z_i^k + c_{ij}$
                cost[$p_k$] ← cost[$p_k$]+$c_{ij}$
                $T$.enqueue($Arrival, p_k, v_j, t_j^k$)
            end
        end
    end
    return $\sum_{p_k \in P}$ cost[$p_k$]
```

`departures[`v_i`]` (line 18). z_i^k is then computed from z_i^b by adding delivery duration d_i^k (line 19). In line 20, time cost `cost[`p_k`]` is increased by waiting time w_i^k and delivery duration d_i^k computed as the time difference between t_i^k and z_i^k. However, if event e is a departure from node v_i in route p_k and v_i is not the last node in p_k, the algorithm computes arrival time t_j^k at v_i's successor v_j using (6.6), and inserts a new arrival event into event queue T (lines 25-32). Time cost `cost[`p_k`]` is then increased by travel cost c_{ij} in line 30. Finally, the total time cost of solution P is computed in line 35 by the summarizing time costs across all the routes $p_k \in P$. The event queue is initialized at the beginning of Algorithm 10 in lines 4-7. For each route $p_k \in P$, the arrival at the initial node v_s^k is scheduled at delivery start time t_s^k and added to event queue T.

Remark 6.3.2 (Time complexity of Algorithm 10) *We assume that the event queue is implemented using a heap and the departure lists using self-balancing binary search trees. This means that an event can be enqueued and dequeued in $O(log(n))$, with n being the total number of events, and, similarly, a departure time can be inserted, removed and searched for in $O(log(n))$. Since Algorithm 10 needs to process two events (i.e., arrival and departure) for each node visited in P, its total time complexity is $O(n\ log(n))$.*

6.3.2. Construction phase

In the construction phase of the proposed solution, we build an initial solution P_0 in a greedy fashion. Here, we focus only on travel time and aim to minimize it for each individual route $p_k \in P$. In other words, we seek for near-optimal solutions to a set of HPPs. Greedy algorithms have the advantage of being fast and easy to implement. However, since the purpose of the initial solution is to position the search in a promising area of the solution space, a completely greedy algorithm can be expected to miss some potentially valuable areas. Therefore, in the construction phase we combine greediness and randomness using GRASP [63].

To obtain variability in the candidate set of greedy solutions, GRASP keeps the best-ranked solution elements in a restricted candidate list (RCL) from where some of them are randomly chosen when building up the solution. Usually, the size of RCL, α, is used as a parameter to control the balance between greediness and randomness. In other words, when α is small, the algorithm becomes greedier, but when α is large, the algorithm becomes more random. In the extreme cases, restricting the size of RCL to only one greedy solution element, i.e., $\alpha = 1$, leads to a completely deterministic greedy search, whereas eliminating the RCL size restriction altogether, leads to a completely random search. We refer the interested reader to [139] for a more extensive description of GRASP.

The GRASP's concept of RCL can be translated into many combinatorial optimization problems. In order to obtain semi-greedy solutions to HPPs, we implement RCL using the distance matrix containing pre-calculated travel costs between each pair of loading areas $(v_i, v_j) \in V \times V$. For each path $p_k \in P$, starting from initial node v_s^k, RCL is populated with the α closest loading areas, i.e. nodes, using the pre-computed distance matrix. Afterwards, a random node v_r is chosen from the current RCL. Now, a new RCL is populated with the α nodes closest to v_r, omitting any node that has been already selected in the previous iterations. If the number of unselected nodes is fewer than α then the size of current RCL is decreased. Therefore, instead of using the fixed size of RCL, we introduce a greediness factor α_g which represents the percentage of unselected nodes that can populate RCL. We then dynamically compute the size of RCL as $\alpha = \lceil \alpha_g \cdot |V_u| \rceil$ where V_u represents a set of unselected nodes. Finally, the complete GRASP-based construction of the initial solution is detailed in the Algorithm 11.

Algorithm 11: The construction of initial solution

```
Input: Route query set $Q = \{q_1, ..., q_M\}$
Output: Initial solution $P_0 = \{p_1, ..., p_M\}$
Procedure ConstructRandomSolution($Q$)
    $P_0 \leftarrow \varnothing$
    forall $q_k \in Q$ do
        $p_k \leftarrow (v_s^k)$
        $v_r \leftarrow v_s^k$
        $V_u \leftarrow D_k \setminus v_r$
        while $V_u$ not empty do
            $\alpha \leftarrow \lceil \alpha_g \cdot |V_u| \rceil$
            $RCL \leftarrow \alpha$ nodes from $V_u$ closest to $v_r$
            $v_r \leftarrow$ random node from $RCL$
            $p_k$.add($v_r$)
            $V_u \leftarrow V_u \setminus v_r$
        end
        $P_0 \leftarrow P_0 \cup \{p_k\}$
    end
    return $P_0$
```

6.3.3. Improvement/Search phase

In the improvement/search phase, we adapt a Variable Neighbourhood Search (VNS) based approach [62, 140] to further improve the previously generated initial solution. We search the solution space with a systematic change of *neighbourhood*, by both descending to local optima and escaping from valleys which contain them.

The basic scheme of VNS is given in Algorithm 12. At the initialization of the algorithm, a set of neighbourhood structures N_k $(k = 1, ..., k_{max})$ has to be defined for the solution

Algorithm 12: The basic VNS metaheuristic

```
Define neighbourhood structures $N_k$ $(k = 1, ..., k_{max})$
Generate initial solution $s \in S$
Choose a stopping condition
while stopping condition is not met do
    $k \leftarrow 1$
    while $k \leq k_{max}$ do
        $s' \leftarrow$ Perturbate($s$); $s' \in N_k(s)$
        $s'' \leftarrow$ LocalSearch($s'$); $s'' \in S$
        if $Score(s'') > Score(s)$ then
            $s \leftarrow s''$
            $k \leftarrow 1$
        else
            $k \leftarrow k + 1$
        end
    end
end
```

space S (line 1). In other words, for each solution, $s \in S$, $N_k(s)$ represents a set of neighbour solutions. In most implementations of VNS, successive neighbourhoods N_k are nested, i.e., $N_1(s) \subset N_2(s) \subset ... \subset N_{k_{max}}(s)$. After the initialization, starting from the initial solution s and the first neighbourhood structure $N_1(s)$, the algorithm systematically explores solution space S by moving within chosen neighbourhood structures. At each iteration, the *Perturbate* procedure (sometimes called *Shake* procedure) randomly selects a new solution s' within the current neighbourhood $N_k(s)$ (line 7). Then, a descent from s' is done by the *LocalSearch* routine which leads to a new local optimum s'' (line 8). At this point, the new solution s'' is compared with the current solution s (line 9). If s'' is better than s, the algorithm re-centers the search around s'' by replacing s with s'' (line 10), and starts all over from the first neighbourhood structure $N_1(s)$ (line 11). If s'' is not better than s, the algorithm moves to the next neighbourhood structure (line 13) and selects a new random solution from $N_{k+1}(s)$. The algorithm iterates until a predefined termination condition is met. An example of the termination condition could be a number of iterations until the solution has not been improved. The choice of neighbourhood structures, as well as the implementation of local search and perturbation, affect the performance of VNS. Therefore, we introduce our design of VNS. We first define the local search moves employed to build solution neighbourhoods.

Search moves

We use two local search moves with combining: *shift* and *2-opt*. We aim to minimize waiting cost with one of them and try to minimize travel cost with the other. First, the *shift* move generates a neighbour solution by relocating (i.e., rescheduling) a single visit within one of the solution's routes. For example, in Figure 63a, a new route p'_k is created from route p_k by shifting visit to node v_i two positions to the right (after visit to node v_{i+2}). Here, the newly created route p'_k is part of a new neighbour solution P'. Notice that we denote by (v_i, p_k) a visit to node v_i in route p_k. The *shift* move can be efficiently applied to reduce waiting times at congested nodes (i.e., loading areas). We can reschedule costly visits to less busy times by simply shifting them using the introduced move. The *2-opt* move, on the other hand, aims at reducing the travel time between selected nodes. *2-opt* was originally proposed for solving the traveling salesman problem (TSP) [141], but has been successfully applied in many other route optimization problems. The main idea behind the *2-opt* move in our solution is to reduce the total travel time by detecting and then reordering non-optimal parts of a solution where its routes cross over themselves. *2-opt* move removes two edges from a route, and reconnects the two new created sub-routes. For example, in Figure 63b, *2-opt* removes edges $e_{(i-2)(i-1)}$ and $e_{(i+1)(i+2)}$ and reconnects the route with a pair of new edges $e_{(i-2)(i+1)}$ and $e_{(i-1)(i+2)}$. Note also that *2-opt* reverses a segment of the route between the exchanged edges.

Neighbourhood structures and local search

Using the two-layer search moves together, we can reduce both waiting and travel time of a solution. Therefore, we use them both to move from one solution to another in the search for local optimum. This means that the neighbourhood of a solution s is composed of all solutions that can be obtained by applying one *shift* and one *2-opt* move to solution s. However, such neighbourhood is far too large to be completely explored at each iteration of VNS. For example, for a single route with n nodes, we can reschedule (i.e., shift) visits to $n-1$ nodes at $n-2$ different positions in route s, yielding the total of $(n-1)(n-2)$ possible *shift* moves. On the other hand, we can exchange $n(n-3)/2$ different pairs of

Algorithm 13: Local search

```
Input: Random solution $P_r$, VNS depth $k$
Output: Improved solution $P$
Procedure LocalSearch($P_r$,$k$)
    $P \leftarrow P_r$
    while $P$ is improved do
        $P_c \leftarrow$ bestShift($P$,$k$)
        $P_c \leftarrow$ best2Opt($P_c$,$k$)
        if $f(P_c) < f(P)$ then
            $P \leftarrow P_c$
        end
    end
    return $P$
Input: Input solution $P_r$, VNS depth $k$
Output: Improved solution $P$
Procedure bestShift($P_r$,$k$)
    $\alpha \leftarrow$ random size of RCL
    $RCL \leftarrow \alpha$ visits with the highest waiting time
    $RCL_k \leftarrow k$ random visits from RCL
    $P \leftarrow P_r$
    foreach $(v_i, p_k) \in RCL_k$ do
        foreach possible shift length $m$ do
            $P_c \leftarrow$ shift($v_i$,$p_k$,$m$)
            if $f(P_c) < f(P)$ then
                $P \leftarrow P_c$
            end
        end
    end
    return $P$
Input: Input solution $P_r$, VNS depth $k$
Output: Improved solution $P$
Procedure best2Opt($P_r$,$k$)
    $\alpha \leftarrow$ random size of RCL
    $RCL \leftarrow \alpha$ traversed edge pairs with the highest travel cost
    $RCL_k \leftarrow k$ random edge pairs from RCL
    $P \leftarrow P_r$
    foreach (edge pair $(e_{ij}, e_{mn}), p_k) \in RCL_k$ do
        $P_c \leftarrow$ 2opt($p_k$,$e_{ij}$,$e_{mn}$)
        if $f(P_c) < f(P)$ then
            $P \leftarrow P_c$
        end
    end
    return $P$
```

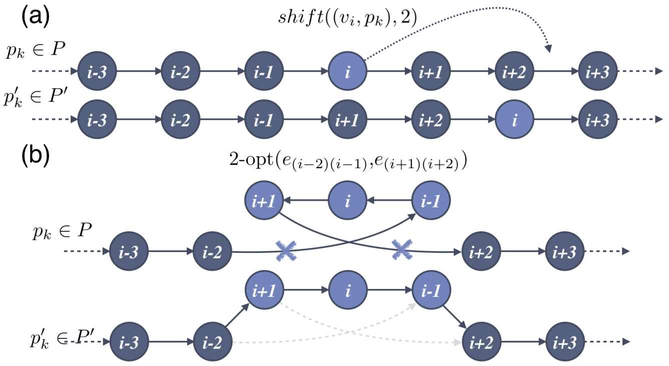


Figure 63: Local search moves: (a) *shift* and (b) *2opt*

edges in the *2-opt* move. Finally, this means that by applying the search moves to only one of the solution's route, we can obtain a total of $n(n-1)(n-2)(n-3)/2$ different neighbour solutions. This is why in the local search procedure we choose to restrict the size of neighbourhood in the following way. Since the goal of *shift* move is to reduce the total waiting time of a solution, we select random k visits with the longest waiting times, and consider only *shift* moves that reschedule the selected visits. Here, we again employ the concept of RCL and randomly select k visits from the list of visits sorted by waiting times. From the reduced neighbourhood, we then choose a solution with the lowest total time cost (i.e., objective function value). This approach is outlined in the `bestShift` procedure of Algorithm 13.

Since the goal of *2-opt* is to reduce the total travel time of a solution, we randomly select k edge pairs from RCL among the traversed edges with the highest relative travel cost. The relative cost is computed by the relative distance between the current edge cost and the cost of the shortest incoming edge to the directed node. We avoid to select the edges with the largest cost directly because for a node, all incoming edges may have large costs. In that case, the incoming edge with a large cost is less likely to indicate it is suboptimal. After selecting, we construct new solutions by applying *2-opt* moves to the selected edge pairs only in the selected routes. Again, a solution with the lowest time cost is chosen from the reduced neighbourhood. The approach is outlined in the `best2Opt` procedure of Algorithm 13. Also in Algorithm 13, the local search procedure is executed until a local optimum of the current neighbourhood is encountered, i.e., until current solution P cannot be improved with the employed local search moves. Finally, notice that the size of a neighbourhood depends on the current depth of VNS, k. This necessarily means that employed neighbourhood structures N_k are nested, which helps VNS to avoid being trapped in local optima by a systematic change of neighbourhood.

Perturbation

As we previously discussed, it is important for VNS implementations to balance intensification (i.e., local search) and diversification (i.e., perturbation) during the search process to allow escaping from local optima. In our implementation of VNS, we perturbate solutions by shuffling a certain percentage of their routes, based on the current depth of VNS, k. More precisely, we randomly select $\lceil M/k \rceil$ routes to be shuffled, where M represents the total number of routes in the solution. Thereafter, the selected routes are shuffled by randomly rescheduling the visits using the *shift* move. Since our implementation of perturbation is straightforward, we omit the algorithm pseudo code to save space.

6.3.4. Complete algorithm

Algorithm 14: Overall Approach

```
Input: Route query set $Q$, maximum number of successive iterations without
       improvement $maxIter$, maximum number of neighbourhood degree $k_{max}$
Output: Optimal solution $P$
Procedure FindSolution($Q$,$maxIter$,$k_{max}$)
    $iter \leftarrow 0$
    $best \leftarrow$ ConstructRandomSolution($Q$)
    while $iter < maxIter$ do
        $iter \leftarrow iter + 1$
        $P \leftarrow$ ConstructRandomSolution($Q$)     } Construction
        $k \leftarrow 1$                                 } phase (GRASP)
        while $k \leq k_{max}$ do                        } Improvement
            $P_r \leftarrow$ Perturbate($P$,$k$)          } phase (GRASP+VNS)
            $P_c \leftarrow$ LocalSearch($P_r$,$k$)
            if $f(P_c) < f(P)$ then
                $P \leftarrow P_c$
                $k \leftarrow 1$
                $iter \leftarrow 0$
            else
                $k \leftarrow k + 1$
            end
        end
        if $f(P) < f(best)$ then
            $best \leftarrow P$
        end
    end
    return $best$
```

The overall approach which combines the construction and improvement phases is outlined in Algorithm 14. Both phases are repeated until solution P could not be improved in $maxIter$ successive iterations.

6.4. Experimental evaluation

We conducted extensive experiments on real-world data collected across Barcelona's urban freight transport network using MCS infrastructure introduced in Chapter 4. We first present experiments on synthetic data to illustrate the trade-offs and parameters involved in the accuracy of the proposed solution. We then present the analysis and results on Barcelona data that provide useful insights and show practical applicability of our solution.

6.4.1. Experiments on Synthetic Data

Experimental Setup

We randomly generated 50 test instances, each composed of $5, 10, 15, 20$ or 25 concurrent route queries (see Definition 6.2.2). In order to increase solving complexity, we provoked more congestions by limiting the capacity of each area (l_i) to only one parking space. While at first this dataset may seem insufficiently large, notice that the formulated test instances are NP-hard problems and as such are computationally extremely expensive. Nevertheless, we need to solve them to optimality in order to evaluate the accuracy of our approach. Therefore, we developed an integer programming model for the problem under study and solved them using IBM's commercial solver CPLEX Optimizer. The computation was limited to 6 hours and 220 GB of memory per test instance, and thus computing the optimal solutions for all 50 test instances lasted more than 12 days in total on a machine with 2.40 GHz Intel® Xeon® E5-2630 processor and 265 GB of RAM.

Finally, we assess the quality of our approach with the accuracy measure defined as follows,

$$accuracy(P, Q) = \frac{\sum_{p \in P_{opt}} cost(p)}{\sum_{p_k \in P} cost(p_k)}, \tag{6.7}$$

where P and P_{opt} are sets of solution routes for the route queries in Q, obtained by our algorithm and CPLEX solver, respectively.

In addition to the above comparison, we also compute the accuracy of the solutions composed of individually optimal routes with the shortest travel times. The aim of such comparison is to validate our hypothesis that, in urban transport networks with limited parking facilities, delivery routes should be planned collectively, rather than individually, and thus to efficiently balance waiting and travel cost. In the rest of this section, we refer to the individually optimal solutions as HPP, and to the solutions obtained with the proposed Multi-Vehicle Route Planner as MVRP.

Results

Figure 64 plots (a) the accuracy and (b) running time of MVRP iterations for different number of nested neighbourhood structures k_{max} (see section 6.3.3). Figure 64a shows that the proposed algorithm is able to obtain high quality solutions ($> 90\%$ accuracy) in less than 50 iterations. It is also interesting to notice that with the smaller number of neighbourhood structures (e.g., $k_{max} = 5$), MVRP is able to quickly find good solutions, but it is unable to find the optimal one in 300 iterations for particular example. This is due to very frequent change of search neighbourhoods, which are sometimes left insufficiently examined. In Figure 64a, this can be observed as successive steps (i.e., jumps) in accuracy line plot. On the other hand, larger number of neighbourhood structures (e.g., $k_{max} = 20$), allows MRVP deeper to explore promising search areas and thus to find the optimal solution (in this particular case). However, as shown in Figure 64b this comes at an expense of longer computation time. For example, running 300 iterations took 748 ms with $k_{max} = 5$, while with $k_{max} = 20$ took twice as much (1,667 ms). Obviously, as we increase the maximum number of nested neighbourhood structures, the iterations become longer as the algorithm needs to descend into deeper search areas. Finally, notice from Figure 64 that we are in general able to find high quality solutions in less than 1.5 seconds for synthetic test instances.

In Figure 65 we plot the accuracy of MVRP and HPP approach for test instances of different sizes (i.e., number of route queries). As shown in the figure, the accuracy of the solutions with individually optimal routes (HPP) is significantly lower than the accuracy

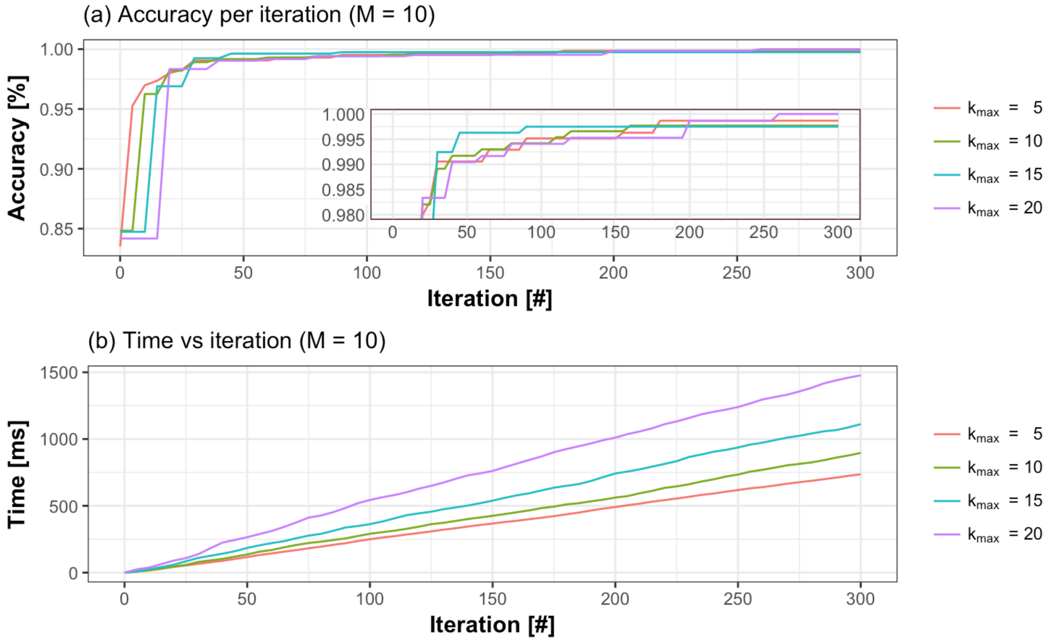


Figure 64: Accuracy improvement with respect to number of iterations (a) and computation time (b)

of the solutions obtained by MVRP. For smaller test instances with 5 route queries, HPP approach yields relatively good solutions with $\sim$ 96% accuracy. However, as we increase the number of route queries, the accuracy drops to $\sim$ 88%. Notice here that as we increase the number of concurrent route queries, we also increase the possibility for congestions and longer waiting times at loading areas. Obviously, for larger test instances, HPP approach fails to avoid congestions and reduce waiting times. On the other hand, MVRP can find high quality solutions, even for more complex test instances with 10 and more route queries. Further, as we increase the search depth (k_{max}) and iteration limit `maxIter` (see Algorithm 14), we can obtain even better solutions, however, at an expense of longer computation time, as previously explained.

6.4.2. Experiments on Real-world Data

Experimental setup

We used MCS system introduced in Chapter 4 to collect real-world data for *areaDUM* urban freight infrastructure [142]. The data is crowdsensed from citizens and deliverers that reflects the usage of the urban freight infrastructure. It consists of 3,704,034 parking check-ins by 49,172 users, i.e., deliverers, in 2,038 loading areas with the total of 8,707 parking spaces, over the period between January 1st, 2016 and July 15th, 2016. Each check-in consists of the ID of the deliverer who made the check-in, her vehicle's plate number (anonymized), the loading area to which the check-in was made, geographic coordinates, and a timestamp. Using the *areaDUM* dataset, one can discover the actual routes taken by deliverers during the course of their work. We inferred the sequences in which the loading areas were visited by grouping check-ins made in the same day by the same deliverer.

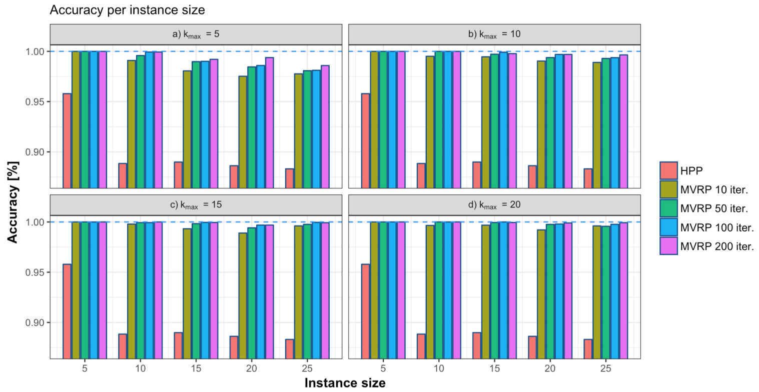


Figure 65: Accuracy improvement over whole synthetic dataset with different setups.

However, the loading areas are often occupied by construction operators and casual deliverers who usually perform only few stops per day [143]. Therefore, we filtered out shorter routes and focused only on those with four or more stops. This way, we obtained the total of 181,454 routes. Further, in order to estimate durations of the deliveries performed at each stop, we needed to find typical travel times between each pair of loading areas in Barcelona's urban transport network. Here, we assumed that deliverers act economically and usually take the shortest path between loading areas. Therefore, we employed Open Source Routing Machine (OSRM)[1] with Barcelona's road network to compute the shortest travel times between each pair of loading areas. Thereafter, we estimated delivery durations as $d_i^k = (t_{i+1}^k - t_i^k) - c_{i,i+1}$ where t_i^k and t_{i+1}^k are arrival times at loading area v_i and v_{i+1}, respectively, and $c_{i,i+1}$ is the travel time at the shortest path between them.

Finally, we generated 10,000 test instances by sub-setting the discovered delivery routes. The complexity of the instances varies in terms of the number of routes, i.e., route queries, and similarities between them. 1,000 test instances were generated for 10 different problem sizes (with 10, 20, 30, ..., and 100 route queries). Obviously, in instances with highly similar route queries, i.e., with frequent mutual stops, congestions at the delivery areas occur more often and, thus, are harder to avoid. We solved all of the instances using the proposed solution, and then compared the obtained solution routes with the real-world routes traversed by actual deliverers in the city of Barcelona, as well as with the individually optimal routes computed with CPLEX using the HPP formulation.

Results

Table 62 summarizes the pairwise comparison between solution routes obtained by our algorithm (MVRP), actual routes derived from the *areaDUM* dataset (REAL), and individually optimal routes (HPP). The comparison was made in terms of the percentage of test cases in which one route set is more efficient (i.e., less time consuming) than another. For instance, for the smallest test instances (10 route queries), MVRP routes are better than HPP in 7.1% of the cases, while HPP routes are more efficient in only 0.4% of the cases. In the rest of the cases, MVRP and HPP routes were equally efficient.

In general, both HPP and MVRP routes were almost always less time consuming than REAL routes, regardless of the instance size, i.e., number of route queries. However, in rare

[1] http://project-osrm.org/

Table 62: Route improvement comparison

size	REAL vs		HPP vs		MVRP vs	
	HPP	MVRP	REAL	MVRP	REAL	HPP
10	4.1%	4.1%	95.9%	0.4%	**95.9%**	**7.1%**
20	1%	0%	99%	4.1%	**100%**	**60.4%**
30	0.7%	0%	99.3%	13%	**100%**	**66.7%**
40	3.5%	0%	96.5%	15.9%	**100%**	**83.5%**
50	1%	0%	99%	9.8%	**100%**	**90.2%**
60	5.4%	0%	94.6%	9%	**100%**	**91%**
70	0.7%	0%	99.3%	15.4%	**100%**	**84.6%**
80	1.1%	0%	98.9%	4.3%	**100%**	**95.7%**
90	2.1%	0%	97.9%	16.9%	**100%**	**83.1%**
100	3.91%	0%	96.09%	10.03%	**100%**	**89.97%**

occasions, REAL routes are still better than HPP routes. This was expected, since in HPP we assume that deliverers act opportunistically and aim at minimizing the total travel time, which can occasionally lead to congestions and waiting times at loading areas. On the other hand, REAL routes were more efficient than MVRP routes in only 4.1% of the times for the smallest test instance with 10 route queries. It is also worthwhile to note that MVRP routes were in most cases more or equally efficient than HPP routes. This can be clearly observed by comparing the percentages in columns HPP-vs-MVRP and MVRP-vs-HPP. However, such similarity in performance is only observable for smaller test instances with fewer route queries, where there is a low probability for congestions and waitings at loading areas. In these cases, the most efficient route arrangement is composed of individually optimal (i.e., shortest in length) routes. For larger test instances with more than 30 route queries, MVRP routes are in 90% of test cases more efficient than REAL and HPP routes.

While both arrangements improve over actual routes taken, we want to observe the extent to which route arrangement provides superior improvements. We plot the average time cost improvements compared to actual route arrangements inferred from the *areaDUM* dataset (REAL routes) in Figure 66. In Figure 66a and d, we first plot the average overall time cost reduction per route for different instance sizes in minutes and percentages, respectively. At the beginning of the curve, HPP and MVRP perform similarly for smaller test instances with up to 30 route queries. In such cases, the two methods can reduce the total durations of real-world routes for up to 4% (5 minutes) in average. However, for larger test instances, the difference between the performance of HPP and MVRP becomes more observable. This can be seen in the plot as a separation of the curves for test instances with more than 40 route queries. For example, for the largest test instances with 100 route queries, MVRP reduced the average time cost per route by $\sim$ 11.5% (18.4 minutes) while HPP performed at the same level as for smaller instances reduced the cost only for close to 3 % (4 minutes). Such behaviour can be explained by the fact that most of the loading areas in Barcelona contain around 4 parking spaces, and as such, they are usually large enough to accommodate vehicles scheduled in smaller test instances. On the other hand, congestions and waitings at loading areas are more common in larger test instances, and thus the HPP approach, which aims only at minimizing the travel costs, performed significantly worse than MVRP.

Figure 66b-c and e-f separate travel and waiting cost improvements to further investigate the performance of HPP and MVRP. Figure 66b and c suggest that both methods performed

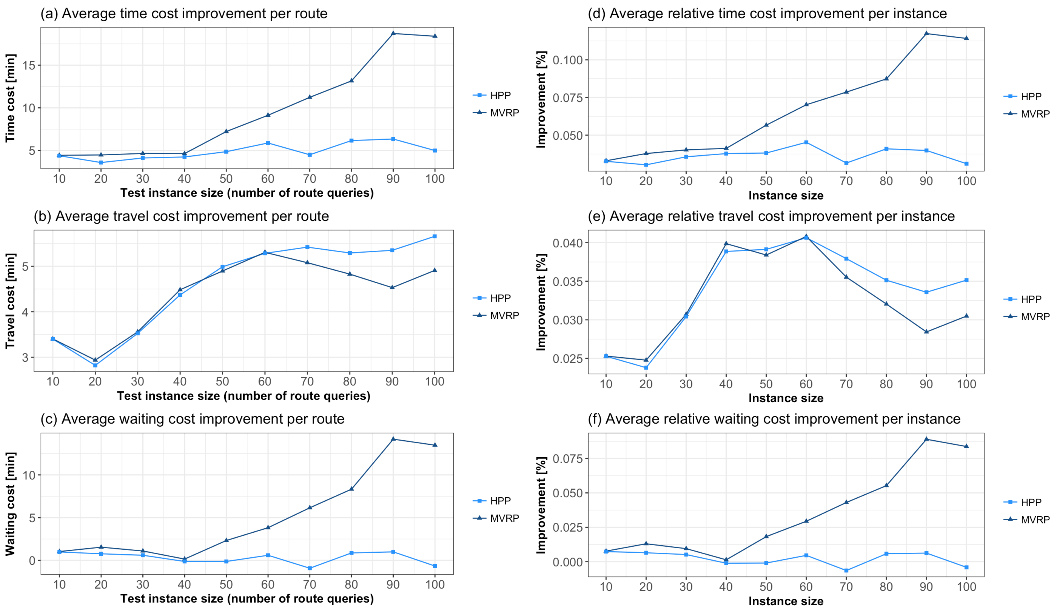


Figure 66: Route improvement in terms of average (a-c) and relative (d-f) total, travel, and waiting costs.

comparably in terms of travel time reduction for test instances with up to 60 route queries. For larger test instances this reduction is less significant for MVRP, since it aims to rearrange routes to minimize waiting times. This can be clearly observed in Figure 66c and d. In the same plot, we see that HPP is unable to reduce waiting costs, regardless of the instance size, which was expected. The two plots show that MVRP is able to strike a balance between travel and waiting costs, which directly confirms the effectiveness of the employed *shift* and *2-opt* search moves. Furthermore, the effectiveness of the proposed solution grows with the number of concurrent route queries, which makes it especially suitable for routing applications in dense urban environments.

6.5. Related work

To the best of our knowledge, the new multi-vehicle routing problem and its urban freight delivery application have not been addressed before both in research and practice. On the theoretical side, a minimum-weight Hamiltonian path problem is used to compute individually optimal (i.e., shortest in length) delivery routes, by assuming that deliverers act economically and try to minimize the total travel time. Travelling salesman problem (TSP) is a special case of Hamiltonian cycle problem defined on edge-weighted graphs. A variation of TSP, where the initial point does not have to be revisited at the end of the cycle, is known as Travelling Salesman Path Problem (TSPP). A related combinatorial graph problem is multiple TSP (mTSP) [144]. In mTSP, a set of routes needs to be determined as to visit all of the planned nodes and minimize the total cost. Here, each node can be visited exactly once in only one route, while each route needs to start and end at the same depot. The problem is composed of both dividing the set of planned visits among multiple travellers and solving TSP for each of them. Even though local search methods [145] can be employed to solve mTSP, various metaheuristics such as genetic algorithms (GA) [146] and ant colony optimization [147] have been proposed. The proposed problem assigns each route with its own set of nodes that need to be visited. We also introduce the time dependencies between the routes by assigning capacities constraints to nodes, which, to the best of our knowledge, have not been considered in the literature.

In a special case of TSP with time windows (TSPTW) [148], each node needs to be visited within the given time interval $[a, b]$. Although arriving after b is not acceptable, a vehicle can arrive to a node before a but it needs to wait until a. Kara et. al. formalize the problem of minimizing the total tour cost including the waiting times [149]. Here, the arrival to the next node is computed by summing the travel costs of previously traversed edges and and waiting times. We also include delivery durations which provides a more realistic model for the problem.

An VNS method that employs Greedy Randomized Adaptive Search Procedure (GRASP) was proposed for Travelling Repairman Problem (TRP), to build initial solutions and thus speed up the search convergence [150]. The GRASP's concept of restricted candidate list is utilized in our approach to combine randomization and greediness when selecting the promising search moves in the local search phase. To build solution neighbourhoods at each iteration, we use two local search moves: *shift* and *2-opt*. *2-opt* was originally proposed for TSP [141], but has been successfully in other route optimization problems. We used *2-opt* to minimize the travel time of routes. On the other hand, the *shift* move was introduced to reschedule visits to busy nodes, and thus to avoid congestions and reduce the waiting times.

All of the aforementioned problems are NP-complete, and thus different heuristic algorithms have been proposed in the literature. They can be categorized into three groups: tour construction (e.g., nearest-neighbourhood, convex-hull), tour improvement (e.g., k-opt, Or-opt), and metaheuristic-based algorithms (e.g., simulated annealing [151], variable neighbourhood search (VNS) [150, 152, 153], tabu search [154, 155], genetic algorithms [156, 157]). We develop a greedy randomized constructive approach to build an initial solution before the actual search process, and improve the search process by positioning the search in the promising solution space areas. Also our approach for selecting the right node as well as the shift position have not been addressed in research nor practice before.

6.6. Conclusion

A new problem formulation and practical solution are presented to solve the congestion problem in real-world urban freight transport networks. The analysis and experiments on data collected across Barcelona's urban freight transport network confirmed their effectiveness and ability to significantly reduce the average length of delivery routes. For example, for moderately-sized problems with only 100 concurrent delivery routes, the proposed system saves up to 19 minutes per route, which sums up to 1,900 minutes of idling[2] time across all 100 routes. According to [158], 1,900 minutes of heavy-duty vehicle idling can produce 146.8 kg of CO_2 which can be a significant contribution to overall emission reduction. The developed solution achieves a more effective urban freight mobility for 90% of cases compared to both the individually optimal and real life traces of delivery. The improvement achieved by the proposed solution grows with the number of concurrent route queries, which makes it especially suitable in dense urban environments.

Our solution opens a new exciting direction of collective planning of traffic and urban freight transport, and makes the specific case of improving transportation with a global

[2] Idling refers to running a vehicle's engine when the vehicle is not in motion.

optimization task and fairness to all vehicles in the system. On the other hand, MCS paradigm naturally arises as a means to seamlessly collect mobility data to act upon.

Chapter 7

An online framework for adaptive rebalancing of public bicycle sharing systems

This chapter is based on materials from the following peer-reviewed paper:

> P. Mrazovic, J. L. Larriba-Pey, and M. Matskin, “A Deep Learning Approach for Estimating Inventory Rebalancing Demand in Bicycle Sharing Systems”, in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, IEEE, 2018, pp. 110–115.

While in the previous chapter we focused on balancing the use of stationary traffic infrastructure, i.e., public parking areas, in this chapter we focus on balancing rental stations in shared transport systems. This problem may at first seem equivalent to the one studied in the previous section, however, it requires a more advanced online approach which would take in consideration the real-time information in order to balance supply and demand in shared transport systems. In this chapter we will first study how to forecast user demand in shared transport and estimate inventory target levels of rental stations, and then we will use the results to route a fleet of freight vehicles to economically re-distribute mobile assets among the stations. As a case study, we will investigate bicycle sharing systems as the most popular form of shared transport, and make use of crowdsensed data containing information about bicycle rentals and returns.

7.1. Introduction

Over the last decade, many cities around the world have introduced public Bicycle Sharing Systems (BSSs) as a more sustainable and less dangerous alternative to motorized forms of transport. BSSs offer a "green" solution to the first-and-last mile connection problem and provide a mobility service where other modes of transport are not available. In addition, BSSs contribute to public health by encouraging people to increase their physical activity and, at the same time, to avoid congestions in motorized traffic. However, despite the significant benefits from BSSs, their exploitation implies various problems among which the most prominent one is to ensure high bicycle availability in order to satisfy customers’ needs and meet contractual service level agreements.

BSSs consist of a collection of bicycle rental stations strategically distributed over the service area. Each station is equipped with a self-service terminal and a number of bicycle stands. Using the self-service terminal, registered user can easily pick up a bicycle from one of the station's stands, and return it (usually within a certain time frame) to any other station with an empty stand. However, over time, user behaviour results in spatial imbalance of the bicycle inventory. In other words, various factors, such as daily commuting patterns or topographical conditions, can lead to an unbalanced state where the numbers of rented and returned bicycles differ significantly among the stations. Such state becomes critical when stations run completely empty or full, and thus prevent users from renting or returning bicycles, respectively.

In order to avoid described service disruptions, operators need to dynamically rebalance BSSs by redistributing bicycles between stations using a fleet of maintenance vehicles. This problem is known in the literature as the *Bicycle sharing Rebalancing/Repositioning Problem* (BRP). BRP consist of (1) estimating the stations' inventory target levels (i.e., redistribution needs) and (2) computing optimal routes for a fleet of capacitated vehicles. Even though BRP has been in focus of many research works over the past several years (e.g., [159–162]), the online (adaptive) solutions to BRP have not been well studied. As opposed to traditional approaches for solving BRP, which rely on an offline training phase during which new inventory target levels are estimated, we propose to use an online approach which would take in consideration the real-time information from BSSs to re-estimate inventory target levels and accordingly to update (i.e., re-optimise) routes that were already assigned to rebalancing vehicles. This way, we aim to find a solution which can efficiently adapt to unplanned events (e.g., weather changes, traffic jams, system failures, etc.), and track repositioning progress of each vehicle in order to balance their workload.

7.2. Problem Formulation

In this section, we summarize the main requirements for the envisioned rebalancing framework, discuss its architecture, and formally define the underlying problem of multiple capacitated vehicles route optimisation. However, before doing so, we first need to get familiar with the notations used throughout this chapter and introduce several new definitions that will be needed to formalise the problem under study. Table 71 summarises the notation of concepts and terms used throughout this chapter. Keep in mind that not all of the presented notations will be used in this section, but will appear later in the chapter.

7.2.1. Preliminaries

As illustrated in Figure 71, the proposed rebalancing framework consists of a station network (1), a demand forecasting model (2), a redistribution model (3), and a routing engine (4). The station network represents the underlying physical bike sharing network composed of bike stations and connections between them. Naturally, the network can be formalized with the aid of a directed graph $G(V, E)$ whose node set $V = \{v_1, ..., v_N\}$ represents bike stations and edge set $E = \{e_{ij} | v_i, v_j \in V\}$ the shortest connections between them. Furthermore, maximum bike capacity c_i^{max} and travel cost c_{ij} are assigned to each node $v_i \in V$ and edge $e_{ij} \in E$, respectively. The station network, i.e., its graph model, is stored in a graph database and accessed by the routing engine.

Definition 7.2.1 (Graph model) *Let $G(V, E)$ be a directed graph where $V = \{v_1, ..., v_N\}$ is the set of nodes representing bike stations, and $E = \{e_{ij} | v_i, v_j \in V\}$ is the set of edges representing the shortest (or fastest) connections between them. Each node $v_i \in V$ is assigned with the capacity c_i^{max} representing the total number of bike stands (both occupied and empty) at station v_i. Each edge $e_{ij} \in E$ is assigned with the travel cost c_{ij}.*

Table 71: Table of notations

Notation	Description
$G(V, E)$	graph model of a station network
V	node set representing bike stations
E	edge set representing shortest connections between bike stations
v_i	node representing a station, $v_i \in V$
c_i	current capacity, i.e., inventory level, of station v_i (total number of bikes at v_i)
c_i^{max}	maximum capacity of station v_i (total number of bike stands at v_i)
e_{ij}	edge representing a shortest connection between v_i and v_j, $e_{ij} \in E$
c_{ij}	travel time needed to traverse e_{ij}
P	bicycle trip data
p	bicycle trip record, $p \in P$
$\mu_i(t)$	pick-up demand at station v_i and time t
$\lambda_i(t)$	drop-off demand at station v_i and time t
d_i	rebalancing target of station v_i, i.e., total number of bikes that need to be delivered to or removed from v_i
V_R	set of station that need to be re-balanced, $V_R \subseteq V$
H	fleet of rebalancing vehicles
h_k	rebalancing vehicle, $h_k \in H$
v_0^k	current position of rebalancing vehicle h_k
l_i	number of bikes carried by a vehicle after visiting v_i
l_{max}	maximum vehicle capacity (total number of bike stands)
T_{update}	re-optimisation period
R	set of solution routes
r_k	route of vehicle v_i

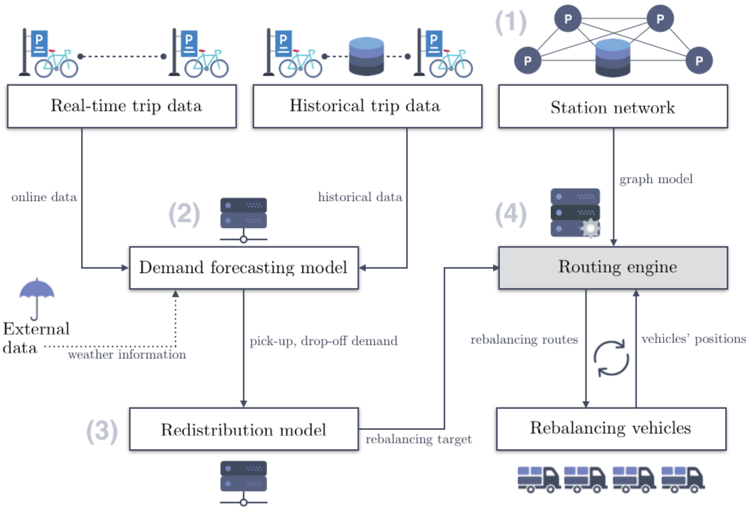


Figure 71: System components of the proposed rebalancing framework

Definition 7.2.2 (Bicycle trip data) *The bicycle trip data P consist of a set of bicycle usage records $p = (v_o, \tau_o, v_d, \tau_d)$ where v_o and v_d represent pick-up and drop-off station, and τ_o and τ_d represent pick-up and drop-off time, respectively.*

Historical and real-time trip data is fed into the demand forecasting model and the redistribution model. The goal of the demand forecasting model is to predict the future user demand. Since traditional bicycle-sharing scheme allows users to rent a bicycle from any particular station and to return it back at any other station within some time frame, the user demand at each station in BSS can be naturally expressed in terms of the total number of pick-ups and drop-offs per time interval. Therefore, we define station pick-up and drop-off demands as follows.

Definition 7.2.3 (Station pick-up and drop-off demand) *Station pick-up demand $\mu_i(t)$ is the total number of bicycle rentals (pick-ups) at station v_i during time interval t. Similarly, station drop-off demand $\lambda_i(t)$ is the total number of bicycle returns (drop-offs) at station v_i during time interval t.*

Notice from the above definitions that the station demands at past time intervals can be easily computed from historical bicycle trip data. For example, station pick-up demand $\mu_i(t)$ can be computed as the total number of trip records $(v_o, \tau_o, v_d, \tau_d)$ where $v_o = v_i$ and $\tau_o \in t$. On the other hand, station pick-up demand $\mu_i(t)$ can be computed as the total number of trip records where $v_d = v_i$ and $\tau_d \in t$. We can formalise this computation as follows,

$$\mu_i(t) = |\{p \mid p \in P \wedge v_o = v_i \wedge \tau_o \in t\}| \tag{7.1}$$

$$\lambda_i(t) = |\{p \mid p \in P \wedge v_d = v_i \wedge \tau_d \in t\}| \tag{7.2}$$

Using the above equations (7.1) and (7.2), demand forecasting model is able to construct time series of station demands and exploit them to forecast the future demands. Additionally, the time series can be augmented with external data such as weather reports $w(t)$, as to introduce a context which can improve forecasting accuracy (we discuss the weather data in more details in the next section).

Based on forecast pick-up and drop-off demands and stations' current inventory levels (i.e., number of available bikes), redistribution model computes the number of bicycles that need to be removed from or delivered to each station, in order to balance the stations and thus meet the predicted service demands. Similar to the definition proposed by Liu et al. [160], we define the optimal inventory level as the one which maximises the time duration within which the station remains balanced, i.e., with enough bicycles and empty stands to meet the future demand.

Definition 7.2.4 (Station rebalancing target) *Let c_i be the current inventory level, and c_i^{max} the capacity of station v_i. The station rebalancing target d_i, i.e., the total number of bikes that need to be removed from or delivered to v_i, is decided by the following calculation which aims to maximise the total number of consecutive time intervals within which v_i remains balanced,*

$$\underset{d_i}{\operatorname{argmax}}\{j \mid 0 \leq c_i + d_i + \sum_{t=t_0}^{t_{0+j}} (\lambda_i(t) - \mu_i(t)) \leq c_i^{max}\} \tag{7.3}$$

Notice that rebalancing target d_i takes negative values when vehicles need to be removed from v_i, and positive values when they need to me delivered to v_i.

Notice again from Definition 7.2.4 that in order to maximize the total time during which a station remains balanced, demand forecasting model needs to be able to forecast pick-up and drop-off demands multiple steps ahead of time.

The vehicle route optimisation engine is the central component of the proposed framework. Based on the computed rebalancing targets, the engine finds near optimal routes for a fleet of rebalancing vehicles. However, at this point, we still treat the engine as a black box and discuss it only in terms of its inputs and outputs. Later in this chapter, we will formally define the underlying route optimisation problem and analyse the proposed solution. The routing engine takes as input a set of stations to be rebalanced $V_R \subseteq V$, along with the corresponding rebalancing targets d_i, $\forall v_i \in V_R$, and a fleet of vehicles $H = \{h_1, ..., h_M\}$ of capacity l_{max} along with their current position in the bike station network $v_0^k \in V$. It is important to emphasize that the route engine takes the real-time vehicle information and periodically (or on a request) re-optimises the routes through a feedback loop. This makes the proposed framework highly adaptive to unplanned events (e.g., weather changes, traffic jams, vehicle failures, etc.) and driver's performance pace.

To better explain the introduced workflow, let us consider a toy example illustrated by Figure 72. The figure shows a snapshot of route plan at some time point τ_x. Two routes have been planned for a small fleet of vehicles $H = \{h_1, h_2\}$. Each line represents the route of one vehicle, where at each stop (represented as horizontal line parts) the rebalancing targets d_i are indicated. Journeys which have been already conducted by the time point τ_x are indicated by a solid line, whereas journeys which should start after τ_x are indicated by a dashed line, meaning they may be subject to change. As previously mentioned, the routes are re-optimised periodically or on a request based on vehicles' progress and updated predictions from demand and redistribution models. Here, the next re-optimisation period is denoted by T_{update}. At T_{update} the routing engine pulls the progress data (i.e., current positions in the station graph) from the vehicles, and feeds them with updated routes. Let us now assume that the snapshot shown in Figure 72 is taken at the re-optimisation period T_{update}, i.e., $\tau_x = T_{update}$. In this case, only part of the planned journeys have been conducted ($\{v_3, v_{10}, v_{13}, v_{19}\}$ by vehicle h_1, and $\{v_8, v_4, v_7\}$ by vehicle h_2). The routing engine reads the vehicles' current positions $v_0^1 = v_{19}$ and $v_0^2 = v_7$ and, based on updated rebalancing targets, computes a new set of routes. This way, the route planning seamlessly adapts to driver's performance pace and unplanned traffic events.

7.2.2. Problem definition

In the rest of this section, we separately define three main goals, i.e., research problems, of the proposed online bike sharing rebalancing problem. Notice below that each of the problems corresponds to one of the previously introduced framework components - demand forecasting model, redistribution model, and routing engine.

Problem 1 (Multi-step-ahead demand forecasting) *Given the pick-up demands $\mu_i(t-k+1), ..., \mu_i(t)$, drop-off demands $\lambda_i(t-k+1), ..., \lambda_i(t)$, and weather reports $w(t-k), w(t-k+1), ..., w(t)$ for the past k time intervals, the goal of the first problem is to forecast the future demands $\mu_i(t+1), ..., \mu_i(t+n)$ and $\lambda_i(t+1), ..., \lambda_i(t+n)$ for each station v_i in the next n time intervals.*

Problem 2 (Rebalancing target estimation) *Given the forecast demands for n future time intervals, the goal of the follow-up problem is to compute rebalancing targets d_i for each station v_i using calculation (7.3) introduced by Definition 7.2.4.*

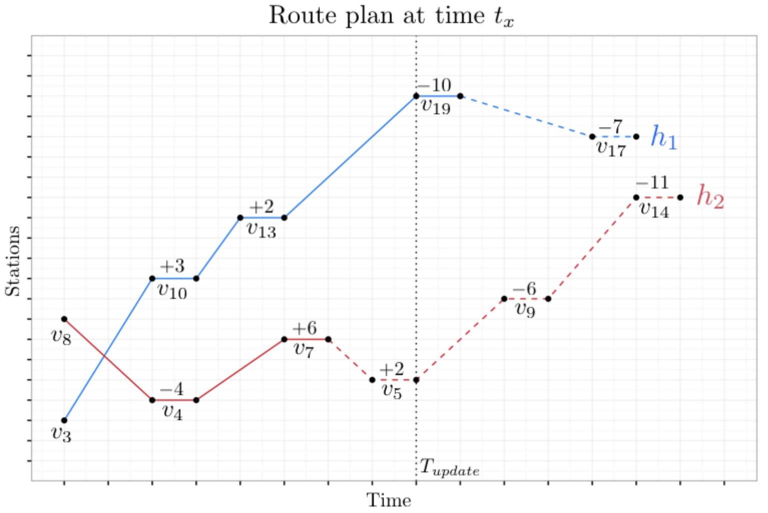


Figure 72: An example snapshot of route plan at time point τ_x.

Problem 3 (Multi-capacitated vehicle routing problem) *Given a bike station network $G(V, E)$, a fleet of located vehicles $H = \{h_1, ..., h_M\}$ with known current load (i.e., number of carried bikes), and rebalancing targets d_i $\forall v_i \in V_R \subseteq V$, the goal of the proposed routing problem is to determine a set of M routes $R = \{r_1, ..., r_M\}$, i.e., one for each vehicle, that collectively minimise the total travel cost while visiting (i.e., rebalancing) all of the unbalanced stations V_R. We further formalise it as follows,*

Minimize

$$\mathcal{F}(\mathrm{x}) = \sum_{v_i \in V_R} \sum_{v_j \in V_R} \sum_{r_k \in R} x_{ij}^k c_{ij} \tag{7.4}$$

s.t.

$$\sum_{r_k \in R} \sum_{v_i \neq v_j} x_{ij}^k = \sum_{r_k \in R} \sum_{v_i \neq v_j} x_{ji}^k = 1, \quad \forall v_j \in V_R \tag{7.5}$$

$$\sum_{v_j \in V_R} x_{0j}^k = 1, \quad \forall r_k \in R \tag{7.6}$$

$$\sum_{v_i \in V_R} x_{i0}^k = 0, \quad \forall r_k \in R \tag{7.7}$$

$$l_i + \sum_{r_k \in R} \sum_{v_j \neq v_i} x_{ij}^k d_j = \sum_{r_k \in R} \sum_{v_j \neq v_i} x_{ij}^k l_j \quad \forall v_i \in V_R \tag{7.8}$$

$$\sum_{v_i \in V_R} \sum_{v_j \in V_R} x_{ij}^k \leq |S| - 1 \quad \forall r_k \in R, \forall S \subseteq V_R, S \neq \varnothing \tag{7.9}$$

$$x_{ij}^k \in \{0, 1\} \quad \forall v_i \neq v_j \in V_R, \forall r_k \in R \tag{7.10}$$

where the decision variable x_{ij}^k is defined to be equal to 1 if edge e_{ij} is traversed by the vehicle h_k (i.e., h_k carries bikes from v_i to v_j), and equal to 0, otherwise.

Objective equation (7.4) represents the travel time over the entire set of solution routes. Constraint (7.5) ensures that vehicles enter and leave each of the unbalanced stations exactly once. Constraints (7.6) and (7.7) indicate that each of the routes $r_k \in R$ starts at the vehicle's current position v_0^k and does not revisit it. Equation (7.8) is employed to conserve bike flow in the network. Here, we introduce a new variable l_i which represents a vehicle's load, i.e., the total number of carried bikes, after visiting station v_i. Notice that initial loads at the vehicles' current positions $l_i \; \forall v_i \in \{v_0^k | h_k \in H\}$ are known beforehand from the feedback loop introduced in the previous section. Constraints (7.9) are classical subtour elimination constraints (GSECs) which impose the connectivity of a solution (see, e.g., [163]). However, they increase exponentially with the number of stations, making the problem practically unsolvable for even moderate number of bike stations. Finally, constraint (7.10) ensures that employed decision variable x_{ij}^k only takes values 0 or 1.

7.3. Demand forecasting

Given that the optimal inventory levels aim at maximising the time within which the stations remain balanced (i.e., with enough bicycles and empty stands to meet the future user demand), we propose to forecast user demand multiple time steps into the future. Such approach allows for more accurate long-term inventory planning which can lower the number of redistribution runs and thus significantly reduce the operating cost. However, accurate multi-step-ahead time series forecasting is a very challenging problem which faces growing amount of uncertainties such as the accumulation of errors, reduced accuracy, and lack of information. While traditional strategies to multi-step-ahead forecasting rely on recursive or direct use of one-step-ahead models [164] and thus suffer from the aforementioned drawbacks, we model our task as a machine learning problem and propose a deep learning approach to solve it efficiently. We introduce multi-input multi-output (MIMO) prediction model based on Long Short-Term Memory networks (LSTMs).

7.3.1. Input modelling

Typical system data provided by BSS operators contains details about the trips between bicycle stations taken by individual users. For example, the data can for each trip include start and finish time, origin and destination stations (ID, name, and geolocation), total duration, bicycle ID, and anonymised user information (gender, year of birth, and customer type). However, since for many BSSs trip data is not this detailed, in this work, we restrict ourselves to minimal feature set containing only trip's start and finish time, and IDs of origin and destination stations. This way, we can represent each trip with a 4-tuple (record) just like described in Definition 7.2.2.

Given that each trip record represents one pick-up and one drop-off event at some points in continuous time domain, we need to discretise time into fixed-size intervals and count the total number of such events for each station. We choose 30-minute discretisation interval, since in most BSSs users are allowed to rent bicycles in 30-minute increments. After applying equations (7.1) and (7.2) over the entire trip data and all 30-minute intervals within continuous data collection period, we obtain time series of the pick-up and drop-off demands, $\mu_i(t)$ and $\lambda_i(t)$, for each station s_i.

In order to apply deep learning to our forecasting problem, we transform the time series data into a set of overlapping time-lagged sequences using time-delay embedding. To illustrate this process let us consider time series of the pick-up user demand at station s_i,

$$... \mu_i(t-2), \mu_i(t-1), \mu_i(t), \mu_i(t+1), \mu_i(t+2), ...$$

where the values up to time interval t are known. Assuming that the next values of the time series depend on at most k past values, and since we want to predict n steps into the future, we transform the time series into pairs of input and output vectors (sequences) $\boldsymbol{x}_t(\mu_i)$ and $\boldsymbol{y}_t(\mu_i)$ composed of k and n consecutive measurements of the original time series,

$$\boldsymbol{x}_t(\mu_i) = [\mu_i(t-k+1), ..., \mu_i(t-1), \mu_i(t)]$$
$$\boldsymbol{y}_t(\mu_i) = [\mu_i(t+1), \mu_i(t+2), ..., \mu_i(t+n)].$$

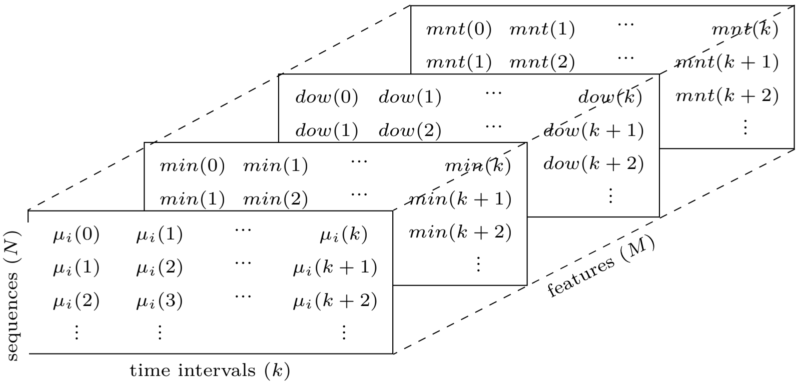
We can now construct input and output matrices $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{Y}(\mu_i)$ for each station s_i simply by stacking all of the sequences as rows, i.e.,

$$\boldsymbol{X}(\mu_i) = \begin{bmatrix} \mu_i(0) & \mu_i(1) & ... & \mu_i(k) \\ \mu_i(1) & \mu_i(2) & ... & \mu_i(k+1) \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$$\boldsymbol{Y}(\mu_i) = \begin{bmatrix} \mu_i(k+1) & \mu_i(k+2) & ... & \mu_i(k+n) \\ \mu_i(k+2) & \mu_i(k+3) & ... & \mu_i(k+n+1) \\ \vdots & \vdots & & \vdots \end{bmatrix}.$$

In the same way we construct input and output matrices $\boldsymbol{X}(\lambda_i)$ and $\boldsymbol{Y}(\lambda_i)$ for each station s_i using the time series of the drop-off demand. Later in the chapter we will see how different choices for embedding dimensions k and n affect the performance of the forecasting model.

Notice that for now our input data $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{X}(\lambda_i)$ is univariate, i.e., it depends on past values of a single input variable – number of pick-ups/drop-offs. However, given that the user demand varies strongly by time of day and day of week, by giving additional temporal context to the input data, we can help our model learn such dependencies and improve forecasting accuracy. Therefore, we augment the input matrices with the additional temporal variables (features). Variable $min(t)$ is introduced to represent time of day as a number of minutes from midnight to the beginning of time interval t. Days of week are represented as weekday numbers $dow(t) \in [1, 7]$, where 1 encodes Monday, and 7 encodes Sunday. Further, we also introduce variable $mnt(t) \in [1, 12]$ to represent months of year. In order to accommodate these new temporal features and still preserve sequential nature of input matrices $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{X}(\lambda_i)$, we reshape the matrices into third-order tensors and stack the temporal features along its third dimension. In other words, we transform $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{X}(\lambda_i)$ into $N \times k \times M$ tensors where N is the total number of input sequences, k is embedding dimension (length of input sequence), and M is the total number of input features. The following representation of input tensor $\boldsymbol{X}(\mu_i)$ further illustrates this transformation.


$mnt(0)$ $mnt(1)$ $\cdots$ $mnt(k)$
$mnt(1)$ $mnt(2)$ $\cdots$ $mnt(k+1)$
$mnt(k+2)$
$dow(0)$ $dow(1)$ $\cdots$ $dow(k)$
$dow(1)$ $dow(2)$ $\cdots$ $dow(k+1)$
$dow(k+2)$
$min(0)$ $min(1)$ $\cdots$ $min(k)$
$min(1)$ $min(2)$ $\cdots$ $min(k+1)$
$min(k+2)$
$\mu_i(0)$ $\mu_i(1)$ $\cdots$ $\mu_i(k)$
$\mu_i(1)$ $\mu_i(2)$ $\cdots$ $\mu_i(k+1)$
$\mu_i(2)$ $\mu_i(3)$ $\cdots$ $\mu_i(k+2)$
sequences (N)
time intervals (k)
features (M)

Above input representation allows us to conveniently introduce new features by expanding the third dimension (i.e., depth) of the tensor. In order to demonstrate this, let us consider weather information as an additional input to the forecast. We assume that a weather report $w(t)$ is available for each time interval t. The report consists of typical weather variables such as sky condition, temperature (°C), humidity (%), visibility (km), and wind speed (km/h), i.e., $w(t) = \big(sky(t), temp(t), hum(t), vis(t), wind(t)\big)$. Here it is important to emphasize that all of the above variables are numerical except the sky condition $sky(t)$ which is usually represented as a short phrase (e.g., *sunny*, *mostly cloudy*, *light rain*, *clear*, etc.). To make it numeric-valued, we can use one-hot encoding, i.e., introduce new variables for each possible condition phrase, setting them all to zero except for the one which corresponds to current sky condition, which is set to one. However, the number of possible sky conditions can be large, while most of them rarely take place. For example, the weather provider that we use in this work (Weather Channel) specifies 133 different condition phrase[1] while only 16 of them appear among the weather reports collected in this work. Therefore, in order to reduce the number of new condition variables, and thus prevent the forecasting model from overfitting, we analysed the correlation between number of pick-ups/drop-offs and sky conditions and then appropriately grouped possible condition phrases into three groups – *good*, *bad*, *moderate*. Using this grouping, we can now replace $sky(t)$ with only three new binary variables $sky\_good(t)$, $sky\_bad(t)$, and $sky\_moderate(t)$.

Finally, we incorporate weather information by expanding the third dimension of the input tensors $\boldsymbol{X}(\lambda_i)$ and $\boldsymbol{Y}(\lambda_i)$ for each weather variable in $w(t)$. This leaves us with tensors containing N sequences of k intervals and $M = 11$ features.

7.3.2. Deep learning model

Over the last several years, recurrent neural networks (RNNs) with hidden Long Short-Term Memory (LSTM) units [165] have become the model of choice for learning from data with sequential features. LSTM overcomes the problem of vanishing gradients [166], which makes very difficult for traditional RNNs to learn to correlate temporally distant events. This is achieved with carefully designed memory cells which protect their hidden activation using special gating mechanism to control the information flow through the cell. We refer the interested reader to [165] and [167] for a detailed description of LSTM units and memory cell implementation.

In this work, we harness the sequential nature of RNNs and propose an LSTM-based approach for user demand prediction in BSSs. In Figure 73 we illustrate an example of two-layered LSTM architecture unrolled through time, which assists us in introducing sequence-to-sequence (seq2seq) configuration [168] that allows us to forecast over multiple time steps. Notice here that input $\boldsymbol{x}$ and output $\boldsymbol{y}$ are sequences whose time steps are revealed in unrolled representation of the network. The illustrated network is composed of two LSTM layers named *encoder* and *decoder*. The purpose of the encoder is to convert input sequences of variable length and encode them in a fixed length representations, which is then used as the input state (i.e., context) for the decoder. In our example, the encoder processes all inputs in chronological order from $t-k+1$ to t and passes its hidden state to the decoder at the final sequence step of t, whereupon the decoder starts to generate an output sequence. The main advantage of this architecture is that it allows us to use inputs of arbitrary length to predict for an arbitrary number of future time steps. The encoder-decoder LSTM was developed for natural language processing problems where it demonstrated state-of-the-art performance, specifically in the area of text translation. However, observe that

[1] https://www.wunderground.com/weather/api/d/docs?d=resources/phrase-glossary

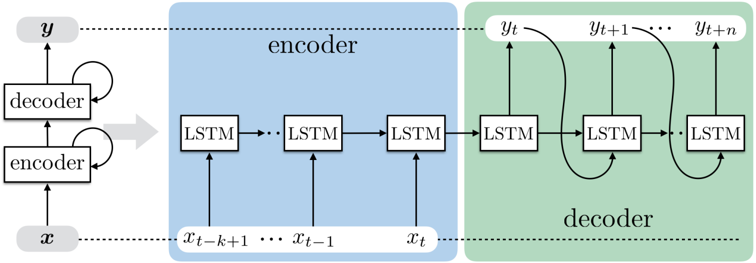


Figure 73: Sequence-to-sequence architecture

this architecture is particularly appropriate in our case where user demands for n future time steps is predicted based on their values in k preceding time steps, where $n \neq k$. For more details about sequence-to-sequence prediction we refer the reader to [168] where the architecture was first introduced.

The encoder and decoder in the network in Figure 73 could easily have more layers to empower the capability to learn more complex dependencies. A naïve approach to our forecasting problem would be to train such multi-layered network separately over $\boldsymbol{X}(\mu_i), \boldsymbol{Y}(\mu_i)$ and $\boldsymbol{X}(\lambda_i), \boldsymbol{Y}(\lambda_i)$ for each station s_i. While this approach at first may seem as a good choice given that the networks are trained for specific stations, we will see in Section 7.6 that with the separate models we are essentially ignoring any correlation that might exist between the stations. In addition, maintaining multiple models requires additional computation and memory. Therefore, we propose to train a more advance network over data aggregated over all stations in BSS. We construct new input and output tensors by concatenating $\boldsymbol{X}(\mu_i)$ and $\boldsymbol{Y}(\mu_i)$ for each station s_i, i.e.,

$$\boldsymbol{X}(\mu) = \begin{bmatrix} \boldsymbol{X}(\mu_0) \\ \boldsymbol{X}(\mu_1) \\ \vdots \\ \boldsymbol{X}(\mu_S) \end{bmatrix}, \boldsymbol{Y}(\mu) = \begin{bmatrix} \boldsymbol{Y}(\mu_0) \\ \boldsymbol{Y}(\mu_1) \\ \vdots \\ \boldsymbol{Y}(\mu_S) \end{bmatrix}, \forall s_i$$

where S is the total number of stations in BSS. In the same way we aggregate tensors $\boldsymbol{X}(\lambda_i)$ and matrices $\boldsymbol{Y}(\lambda_i)$, $\forall s_i$, into $\boldsymbol{X}(\lambda)$ and $\boldsymbol{Y}(\lambda)$. Now, in order to track which sequences belong to which station, we introduce an auxiliary input matrix $\boldsymbol{Q}$ which in each row t contains non-sequential features, such as station ID, of the corresponding sequence (i.e., row) t in $\boldsymbol{X}(\mu)$ and $\boldsymbol{Y}(\mu)$ (and $\boldsymbol{X}(\lambda)$ and $\boldsymbol{Y}(\lambda)$). In our case, in addition to station ID, matrix $\boldsymbol{Q}$ will also contain information about holidays. For example, row $\boldsymbol{q_t} = [123\ 1]$ of matrix $\boldsymbol{Q}$ suggests that sequence t is recorded at station with $ID = 123$ during a holiday (1).

Finally, in order to handle both sequential and non-sequential inputs we propose the architecture illustrated in Figure 74. The architecture contains two input layers, one for sequential inputs $\boldsymbol{X}(\mu)$ and $\boldsymbol{X}(\lambda)$, and one for non-sequential inputs $\boldsymbol{Q}$. Sequential inputs are fed directly to encoder-decoder LSTM network configured as shown in Figure 73. To obtain sequences of n future time steps, output of the decoder at each time step from t to $t+n$ is separately processed by fully-connected (inner product) layer. Given its temporal dimension, we refer to this layer as time-distributed fully connected layer (TD-FC). At this point, we introduce non-sequential data for each of the output sequences from the TD-FC layer. In other words, we row-wise concatenate $N \times P$ matrix $\boldsymbol{Q}$ (where P is the total number of non-sequential features) and $N \times n$ matrix $\boldsymbol{R}$ which is the output of TD-FC

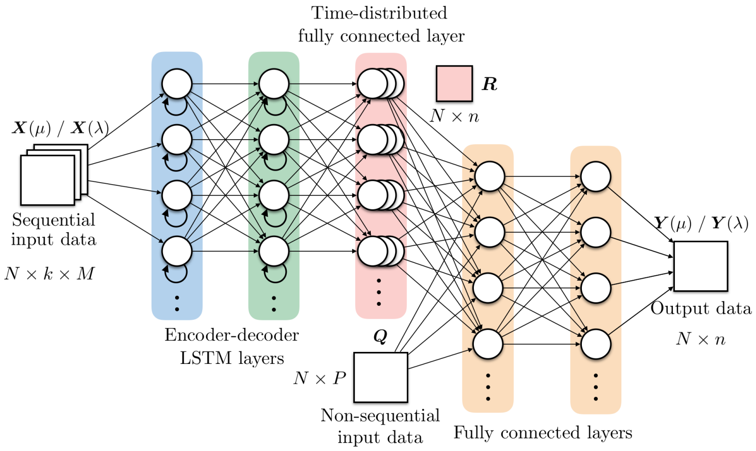


Figure 74: The proposed network architecture

layer. Finally, the merged data is fed to two-layered fully connected network to produce final sequences of n future time steps.

7.4. Rebalancing target estimation

In the final step of the proposed framework, we use predicted user demands to compute rebalancing targets for each of the stations in BSS. Recall from Definition 7.2.4 that this computation consists of finding a rebalancing target d_i which would maximise the total number of consecutive time intervals during which the station remains balanced. In order to find such rebalancing target, Algorithm 15 iterates over all future horizons, i.e., time intervals, and at each iteration it looks for a set of possible values for d_i ($\widehat{D}$) with respect to time intervals considered in previous iterations. $\widehat{D}$ is determined in line 6 based on the current inventory level c_i, the station's capacity l_i, and the total bicycle net flow (net) represented as the difference between expected number of returned and rented bicycles up until time interval of the current iteration (line 5). Thereafter, the algorithm restricts set $\widehat{D}$ by selecting subset of rebalancing targets d_i (D) which keep the station balanced over the current time interval, but also over all of the previous time intervals considered in past iterations (line 8). If such subset does not exist, i.e., D is empty, the algorithm stops the iteration (line 10). Finally, if there are several rebalancing target values left in D, the algorithm takes one with the minimal absolute value (line 13) in order to minimize the cost of rebalancing operation.

Notice that sets $\widehat{D}$ and D are connected, i.e., they are both intervals, which means that the intersection between them (lines 7-8) can be found in constant time. This means that the computation time of Algorithm 15 depends only on the number of future horizons n, i.e., its time complexity is linear ($O(n)$).

7.5. Multi-capacitated vehicle routing

In Section 7.2.2 we introduced a very complex vehicle routing task (see Problem 3) which calls for a more advanced optimisation paradigm with large, bold, search moves to escape local optima. In Chapter 5 and 6 we were dealing with simpler-structured problems

Algorithm 15: Rebalancing target estimation

```
Input: Predicted pick-up and drop-off demands $\mu_i(t+j)$, $\lambda_i(t+j)$ $\forall j \in [1..n]$,
       current inventory status $c_i$, capacity of the station $l_i$
Output: Rebalancing target $s_i$
Procedure EstimateRebalancingTarget($s_i$)
    $net \leftarrow 0$
    $D \leftarrow \varnothing$
    for $j \in [1..n]$ do
        $net \leftarrow net + \lambda_i(t+j) - \mu_i(t+j)$
        $D_j \leftarrow \{d_i \mid 0 \leq c_i + d_i + net \leq l_i\}$
        if $D \cap D_j \neq \varnothing$ then
            $D \leftarrow D \cap D_j$
        else
            break
        end
    end
    return $\min\{|d_i| | d_i \in D\}$
```

where small search moves employed within VNS metaheuristic could deliver near-optimal solutions. However, solutions to complex problems have to meet many constraints, and often finding any kind of valid solution can be already very hard. Therefore, applying small search moves may not always lead to valid neighbouring solutions. We approach this "discontinuity problem" by using a very simple optimisation principle called *Ruin and Recreate* (R&R). We ruin a large fraction of the solution which then gives as freedom to restore it as best as we can. Hopefully, the new solution will be better than the previous one. R&R principle is proved to be powerful all-purpose-heuristic which is able to deliver high quality solutions to many well-known NP-hard problems. For example, [169] reports on record braking results for related VRP using R&R principle. Therefore, in this section we adopt R&R principle and propose slightly modified implementations of *ruin* and *recreate* procedures proposed in [169] and [170]. For more information about the adopted methods, we refer the interested reader to aforementioned works.

7.5.1. Solution representation

A solution to the proposed variant of VRP is represented as a combination of permutations of nodes in each route $r_k \in R$. Therefore, a single route $r_k \in R$ is represented as an ordered set (i.e., tuple) of nodes with the corresponding rebalancing targets computed in the previous section. For example, a solution $R = \{r_1, r_2, r_3\}$ with rebalancing targets d_i $\forall v_i \in V_R = \{v_1, ..., v_{11}\}$ can be represented as a combination of ordered sets, i.e.,

$$R = \left\{ \begin{array}{l} r_1 = (v_0^1, v_5, v_3, v_4, v_2), \\ r_2 = (v_0^2, v_{10}, v_6, v_8), \\ r_3 = (v_0^3, v_1, v_{11}, v_7, v_9) \end{array} \right\}. \tag{7.11}$$

Notice here that nodes v_0^1, v_0^2, and v_0^3 represent current locations of vehicles h_1, h_2, and h_3, respectively. The feasibility of the above solution is defined by the following definition.

Definition 7.5.1 (Solution validity) *Solution R with the above representation (7.11) is valid if the following three conditions are satisfied:*

1. *Each route $r_k \in R$ starts with the current location v_0^k of the corresponding vehicle h_k.*
2. *Each node $v_i \in V_R$ is visited in exactly one route.*
3. *Vehicle's load l_i after visiting each node $v_i \in V_R$ is below vehicle's capacity l_{max}.*

7.5.2. Ruin procedure

Schrimpf et al. [169] introduced many different *ruin* strategies which can successfully introduce search diversification into metaheuristic algorithms for VRP (e.g., radial ruin, sequential ruin, random ruin). In this work we employ the *random ruin* strategy which will remove A randomly selected nodes from V_R and put them into the bag B. At this point *ruin* procedure will not check the feasibility of the ruined solution as it can be significantly changed after *recreate* procedure. Notice here that any node can be removed except the starting ones which represent vehicles' current locations. A random number of removed nodes A is usually selected with $A \leq |V_R| \cdot f$, where f is a fraction, i.e., $f \in [0, 1]$, usually determined experimentally. After A nodes have been removed and put into the bag B, *recreate* procedure will re-insert one by one node from B with a specific strategy to maximise the probability of improvement.

7.5.3. Recreate procedure

After *ruin* procedure we are left with a partial solution where not every node is visited. To restore validity (see Definition 7.5.1), we apply *repair* procedure which will insert non-visited nodes from B into exiting routes. There are many different ways to insert non-visited nodes, but we will use *best insertion* strategy which will successively add all non-visited nodes in the best possible way to the system. In other words, *recreate* procedure will take one by one node from B and consider all possible insert positions (between each pair of successive nodes) among all the routes in R. Finally, it will choose an insert position which will create a new valid solution with the minimum cost. Of course, such greedy approach may not always lead to optimal solution, however, re-applying *ruin* and *recreate* procedures can increase the probability of obtaining an optimal or near-optimal solution. In addition to *ruin* and *recreate* procedures, Dell'Amico et al. [170] define a set of local search moves to further improve the solution after *recreate* procedure. Given that we have already introduced some of this moves in previous chapters, we refer the interested reader to [170] for more information. Finally, the overall framework of the algorithm is reported in Algorithm 16.

7.6. Experimental evaluation

Our experimental study is based on real-world bicycle trip data collected in New York's BSS known as CitiBike. We used a subset of this data containing 12,246,450 trips with bicycle pick-ups/drop-offs recorded at 750 stations over the period between January 1$^{\text{st}}$, 2017 and October. 1$^{\text{st}}$, 2017. Each pick-up and drop-off event is represented as a timestamp and ID of the station where the event was recorded. We augmented this data with weather information for each timestamp by obtaining historical weather data using Weather Channel's API. This data consists of typical weather variables that we introduced previously in Section 7.3.1. We constructed input sequences composed of $k = \{4, 8, 16, 24\}$ past and $n = 8$ future 30-minute intervals, and then randomly split them into a training set (80%) and a test set (20%). Notice that 8 future time intervals corresponds to 4 hours, i.e., half of a common 8-hour working shift. In order to measure the performance of the proposed

Algorithm 16: Ruin & Recreate algorithm

```
Input: Set of nodes to be rebalanced $V_R$, rebalancing targets $d_i\ \forall v_i \in V_R$, fleet of
       vehicles $H$, vehicle capacity $l_{max}$, vehicles' current locations $v_0^k\ \forall h_k \in H$,
       ruin factor $f$
Output: Solution routes $r_k \in bestSolution$
Procedure RuinAndRecreate()
    Choose a random route $r_k\ \forall h_k \in H$ with $v_0^k$ as a starting node
    $bestSolution \leftarrow R$
    $A \leftarrow \lfloor |V_R| \cdot f \rfloor$
    while $!$ stopping condition do
        $B \leftarrow \varnothing$
        $B \leftarrow$ randomly remove $A$ nodes from $R$                          // Ruin
        for $v_i \in B$ do
            $bestPos \leftarrow \varnothing$
            $bestCost \leftarrow \varnothing$
            for $r_k \in R$, insert position $p$ do
                Temporally insert $v_i$ at position $p$ in route $r_k$
                $newCost \leftarrow$ compute new cost
                if $newCost < bestCost$ then
                    $bestCost \leftarrow newCost$
                    $bestPos \leftarrow (r_k, p)$
            end
            Insert $v_i$ at $bestPos$                                        // Recreate
        end
        if $cost(R) < cost(bestSolution)$ then
            $bestSolution \leftarrow R$
    end
    return $bestSolution$
```

approach we use the mean absolute error (MAE) of user demands at each of the $n = 8$ predicted time intervals. In addition, we compare our approach with traditional neural network, i.e., multilayer perceptron, simple linear regression, and two ensemble methods that demonstrated competitive performance in previous related works [162, 171] – random forest and gradient boosting machines.

We developed the proposed deep learning model using Keras neural networks API, written in Python and running on top of TensorFlow machine learning library. The model's parameters were tuned manually by trial and error. The number of units contained in encoder and decoder LSTM layers and two fully connected layer were 128, 256, 256, 256, respectively. Given the very large number of training sequences (> 9 million) we trained our model in batches of 1024 sequences over 100 training epochs. In Figure 75a we plot MAE of pick-up and drop-off demand at each time horizon for different number of lagged time intervals k. As expected, the prediction error decreases as we increase the number of lagged observations, i.e. the model learns more complex dependencies. In the particular experiment, average MAE over 8 time horizons decreases from 0.799 pick-ups for $k = 4$ to 0.713 pick-ups for $k = 24$ (and from 0.754 to 0.681 drop-offs). Also notice that prediction error is lower at closer time horizons. For example, MAE of predicted pick-up demand in the next 30 minutes (one step ahead) is only 0.667 for $k = 24$, while MAE of the same prediction 4 hours (8 time steps) ahead of time is 0.741. This performance drop is more

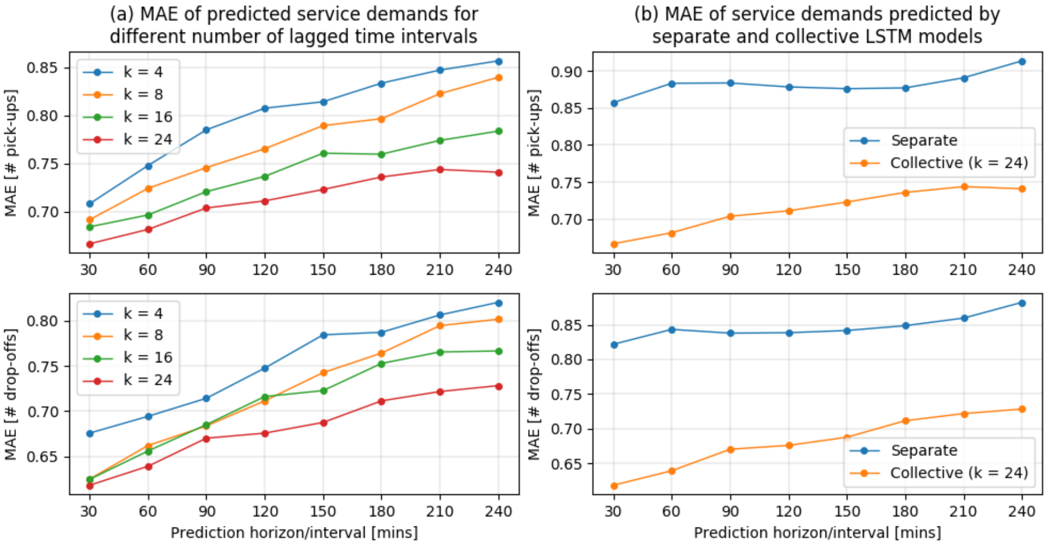


Figure 75: Effects of different parameters on the prediction accuracy

significant for smaller k values. For example, for $k = 4$, MAE increases 1.731% from first to last time horizon, and 1.001% for $k = 24$. Similar can be observed in case of drop-off prediction.

The guiding design principle behind our model was to separate sequential and non-sequential data in order to train a single model for all stations in a BSS. We have previously argued that this way we are not ignoring correlations that might exist between the stations, which can significantly improve the overall prediction accuracy. To justify our design choice, in Figure 75b we plot MAE of predicted pick-up and drop-off demands when each station is trained on a separate model. Notice that these models are identical to the one proposed in Figure 74 but without non-sequential input data and fully connected layers. The plot clearly demonstrates the advantage of our approach. MAE averaged over 8 time horizons is 0.882 pick-ups and 0.846 drop-offs in case of separate model training, and 0.713 pick-ups and 0.681 drop-offs in case of collective model training.

In Figure 76 we compare the performance of our model against the baselines. Given that the employed baseline models cannot handle sequential inputs, we represented each time step as a set of additional features (i.e., columns) in 2D input. As expected, simple linear regression model (LR) was not able to learn more complex nonlinear dependencies, hence the largest MAE of 1.218 pick-ups and 1.192 drop-offs across all 8 time horizons. Here we can also clearly observe the effect of error accumulation where MAE grows from 0.98 pick-ups and 0.95 drop-offs at the first time horizon to 1.32 pick-ups and 1.31 drop-offs at the last time horizon. This effect is least noticeable in case of multilayer perceptron (MLP). The employed MLP model was composed of 3 layers with 256 neurons each, and it was trained in the same way as our deep learning model (in batches of 1024 samples over 100 training epochs). MLP performs significantly better than LR, and comparably to ensemble models – random forest (RF) and gradient boosting machines (GBM). Our experiments confirm the efficiency of the ensemble models reported in previous works. They manage to successfully average out biases and reduce the variance. Finally, as can been seen in the plot, our method outperforms tested baselines. In average, it performs 41% better than LR,

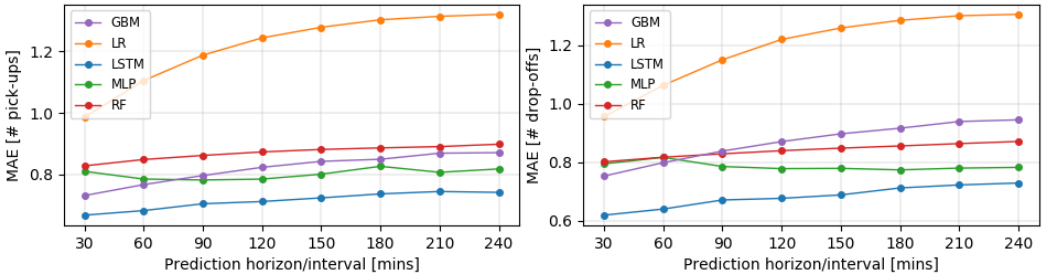


Figure 76: MAE of (a) pick-up and (b) drop-off demand at each time horizon for different baselines.

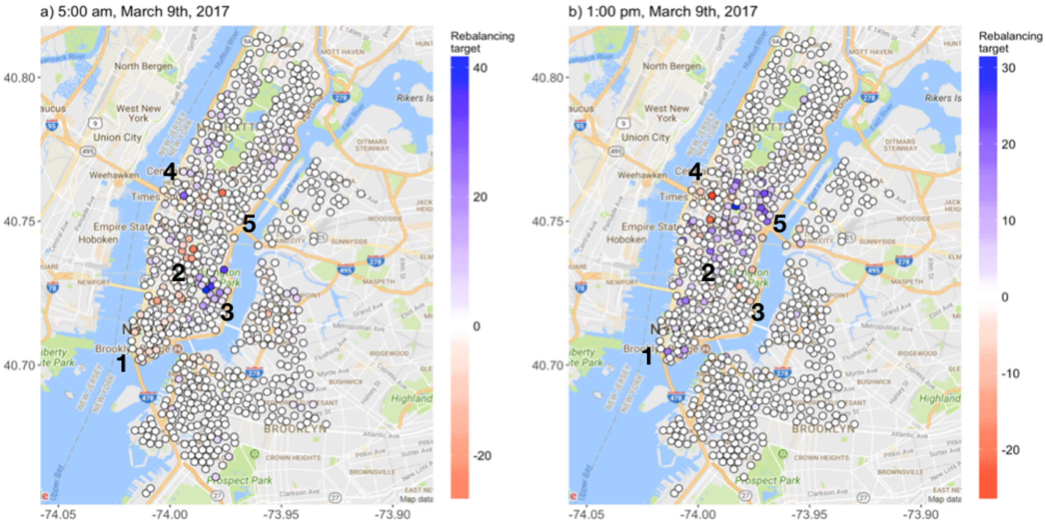


Figure 77: Rebalancing targets in New York's BSS

11% better than MLP, 18% better than RT, and 13% better than GBM.

In order to demonstrate the applicability of our framework, in Figure 77 we visualize rebalancing targets for the stations in New York's BSS. Each rebalancing target is illustrated as circle whose color represents the amount and type of redistribution, i.e., shade of red for bicycle removals and shade of blue for bicycle restocking. In this particular example, we computed the rebalancing targets (Algorithm 15) based on inventory snapshot recorded on August 9$^{\text{th}}$, 2017 at 5am (a) and 1pm (b), i.e., before morning and afternoon rush hours. In Figure 77a stations located in business areas in Lower (1) and Midtown (2) Manhattan are coloured red, meaning some number of bicycles need to be removed due to increased drop-off demand in morning hours. On the other hand, stations in residential areas, for example in East Village (3) and Upper East Side (4), are coloured blue, meaning increased pick-up demand is expected in morning hours and additional bicycles need to be brought to these stations. In Figure 77b, the color are reversed. Some stations in business dominant areas (e.g., 1, 2) does not have enough bicycles to meet high pick-up demand in afternoon hours (coloured blue), while some stations in residential areas (e.g., 3, 4) have to many bicycle and cannot meet upcoming drop-off demand.

Chapter 8

Conclusion and future work

Mobility management of the future – without environmental sensors or hardware

Some of the conclusions in this chapter were previously discussed in the following position paper:

> P. Mrazovic, I. De La Rubia, J. Urmeneta, C. Balufo, R. Tapias, M. Matskin, and J. L. Larriba-Pey, "CIGO! Mobility management platform for growing efficient and balanced smart city ecosystem", in *Smart Cities Conference (ISC2), 2016 IEEE International*, IEEE, 2016, pp. 1–4.

Throughout this thesis we learned that Smart Mobility can be achieved in many different ways, from infrastructural projects which change urban landscape to Intelligent Transport Systems (ITS) which heavily rely on Information and Communications Technologies (ICT). One of the main goals of the thesis was to promote the extensive use of ICT in achieving Smart Mobility. In particular, we aimed to demonstrate how to leverage MCS paradigm to collect user-generated mobility data, process it, and finally, use it to route and manage people flows in urban environments. We argued that such approach reduces the need for large investments or sophisticated sensor technologies, while achieving common Smart Mobility objectives such as reducing traffic congestions and noise pollution, improving transfer speeds and costs, and increasing people safety.

In the second part of the thesis we introduced algorithmically challenging mobility problems which rely on simple mobility data crowdsensed from volunteer participants. We showed that with the adequate data processing mechanisms and algorithm design we can solve some common routing problems in MCS settings (e.g., multi-objective planning, reducing congestions, balancing the use of public infrastructure, meeting user demands, etc.). However, we also learned that mobility management problems are application-specific and as such they require distinct algorithmic approaches. This naturally opens some interesting technical and philosophical questions. Can we design and develop a general mobility management system that can easily accommodate any application and completely rely on MCS? How would such system attract so many volunteer participants and data contributors? Who are the main actors in such system and how do they form a value chain? Is this a mobility management of the future, i.e., without environmental sensors or hardware? As a conclusion to this thesis, we will try to answer some of these questions by presenting our own attempt to develop such system.

As a part of our research, we designed an affordable MCS-driven mobility management platform which can attract numerous mobility actors and still be orchestrated by the city

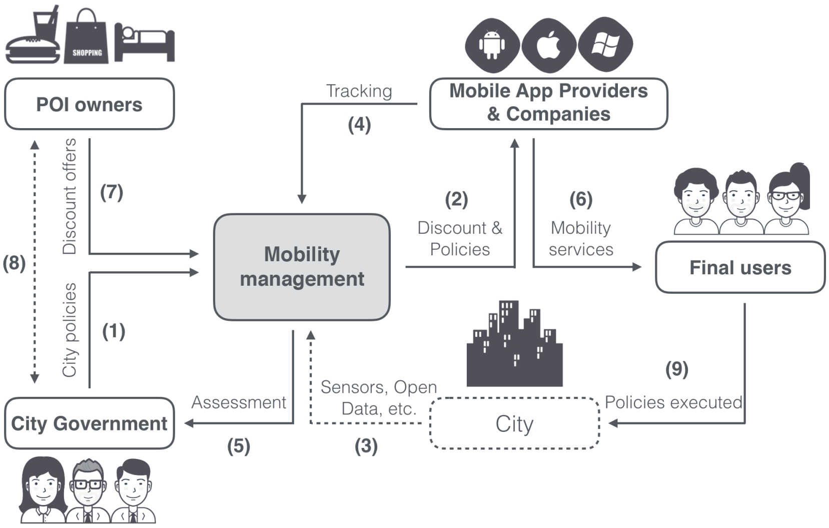


Figure 81: Mobility actors in the proposed general mobility management platform

administrators. Our design allows city administrators (i.e., city governments) to formulate mobility policies (to promote or reduce the traffic of people flows in certain areas) and to motivate different actors to execute them while tracking how those policies are being executed. However, instead of pushing the final users to follow the created policies, the policies are injected into the specific third party mobile applications linked to the platform. This way, logistic companies and providers of mobile applications, who provide services closely tied to the city resources, become aware of how the city government would like to organize the mobility. This makes their services conformant with the city mobility policies. Obviously, advantages of such approach are threefold: city governments can easily implement their mobility goals, companies and mobile application providers can upgrade their logistic and mobility services with respect to the present mobility in the city, final users can freely move in a less crowded and balanced city. Additionally, we include the POI owners and allow them to promote their businesses through the adapted mobility policies. In Figure 81 the proposed platform is represented as a black box in order to emphasize the involved actors and to show how they interact using different information flows. Using the scheme we can explain the role of each actor in the proposed model.

City governments – The city governments are the mobility supervisors and policies makers, and thus are represented as main actors in the system. They need to alleviate the number of people in certain areas of the city in order to reduce agglomerations and to promote alternative urban areas to bring well economic growth. With the proposed platform the city governments are able to formulate their mobility policies (1) and motivate the final users, i.e. citizens, tourists and employees of the companies that utilize the platform, to execute them through the linked mobile applications (2). Recall from Chapter 5 that, depending on an application, mobility policies can be formulated as restriction values associated to specific urban areas, traffic routes or POIs around the city. As shown in Figure 81 the platform also integrates the data from various sources such as Open Data, mobile applications (MCS), and government data (3) and use it to ease the mobility policies creation, e.g., by visualizing the current mobility on the city map. Additionally, the data from the associated mobile applications provides feedback on effectiveness of injected mobility policies (4), and thus creates a virtuous cycle of creating, tracking and improving the policies (1,5).

Mobile application providers and companies – Logistic companies and providers of the third party mobile applications which offer services related to the city resources (e.g. tourist applications, public transport applications, parking applications, navigation systems, etc.) are hardly aware about what the mobility in the city or the city government's mobility objectives are. Therefore, the employed logistic and mobility services are non-conformant with the strategies applied by the city to improve its mobility. By connecting mobile applications to the proposed platform, companies and application providers are able to adapt their services to injected mobility policies (2), and thus make their final users conscious of the present mobility in the city and provide better quality of service (6).

POI owners – POIs (e.g. shops, bars, restaurants, museums, etc.) exist in different economically developed areas of the city and they need to generate a growing business to evolve. However, some street businesses are placed in parts of the city which do not have the necessary turnover of people to grow and need to have a showcase to users. Our vision of a modern mobility management platform can give shops and restaurants a better exposure to the customers, either citizens or visitors, at the cost of making small discounts or business deals for them (7). Using an arbitrary business model between POI owners and the city government (8), the platform can allow city governments to inject new mobility policies adapted to the advertised POIs to a specific group of users (2). POI owners, on the other hand, attract new customers by offering discount coupons, and thus additionally motivate the users to execute injected mobility policies (9).

Final users – The final users, i.e. citizens, visitors and employees of logistic companies, are able to exploit less crowded, balanced and more liveable city at its maximum. Additionally, they can discover different POIs and new areas that were under cover before by following discount offers tailored to their tastes and needs.

Even though the above value chain and business model may seem very optimistic given that they gather numerous actors which should be properly coordinated and motivated for participation, we still argue that the proposed design can be on track of achieving a general mobility management system which can accommodate many different applications. Therefore, in order to further investigate our view of such system, we developed a platform prototype CIGO![1] which brings together aforementioned actors and tries to orchestrate mobility in the city. We refer the interested reader to our position paper [172] where we presented CIGO! platform and its logical architecture. As a study case we integrated into CIGO! our bi-objective routing framework from Chapter 5. As a future work, we plan to accommodate rest of the algorithms and frameworks proposed in this thesis. Obviously, this calls for a modular and easily extendable design which we plan to study in the future. Further, the proposed model of a general mobility management platform which mainly relies on user-generated data opens up additional issues such as trust, reputation, and privacy of volunteer participants, which all need to be addressed eventually. Another challenging task is to develop quality control mechanisms which would assure high reliability of crowdsensed data. As we can see, designing and developing a powerful mobility management platform without environmental sensors or hardware calls for very ambitious, challenging, and time consuming study which we are looking forward to conduct in the future.

[1] www.smart-cigo.com

Bibliography

[1] United Nations (UN), Department of Economic and Social Affairs, *World Urbanization Prospects: The 2014 Revision*. New York, 2014.

[2] United Nations Human Settlements Programme (UN-Habitat), *Urbanization and Development: Emerging Futures - World Cities Report 2016*. Nairobi, 2016.

[3] T. Dixon, “Sustainable urban development to 2050: Complex transitions in the built environment of cities”, *Oxford Institute for Sustainable Development, Oxford Brookes University, Retrofit*, vol. 2050, 2011.

[4] E. Commission *et al.*, “Green paper, towards a new culture for urban mobility”, *European Union, Brussels*, 2007.

[5] C. Benevolo, R. P. Dameri, and B. D’Auria, “Smart mobility in smart city”, in *Empowering Organizations*, Springer, 2016, pp. 13–28.

[6] B. Jiang, J. Yin, and S. Zhao, “Characterizing the human mobility pattern in a large street network”, *Physical Review E*, vol. 80, no. 2, p. 021 136, 2009.

[7] L. Liu, C. Andris, and C. Ratti, “Uncovering cabdrivers’ behavior patterns from their digital traces”, *Computers, Environment and Urban Systems*, vol. 34, no. 6, pp. 541–548, 2010.

[8] F. Calabrese, M. Colonna, P. Lovisolo, D. Parata, and C. Ratti, “Real-time urban monitoring using cell phones: A case study in Rome”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 1, pp. 141–151, 2011.

[9] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W.-Y. Ma, “Understanding transportation modes based on GPS data for web applications”, *ACM Transactions on the Web (TWEB)*, vol. 4, no. 1, p. 1, 2010.

[10] E. Kanoulas, Y. Du, T. Xia, and D. Zhang, “Finding fastest paths on a road network with speed patterns”, in *Data Engineering, 2006. ICDE’06. Proceedings of the 22nd International Conference on*, IEEE, 2006, pp. 10–10.

[11] P. S. Castro, D. Zhang, and S. Li, “Urban traffic modelling and prediction using large scale taxi GPS traces”, in *International Conference on Pervasive Computing*, Springer, 2012, pp. 57–72.

[12] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, “Adaptive fastest path computation on a road network: a traffic mining approach”, in *Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 2007, pp. 794–805.

[13] C. Yin, Z. Xiong, H. Chen, J. Wang, D. Cooper, and B. David, “A literature survey on smart cities”, *Science China Information Sciences*, vol. 58, no. 10, pp. 1–18, 2015.

[14] O. Vermesan, P. Friess, P. Guillemin, S. Gusmeroli, H. Sundmaeker, A. Bassi, I. S. Jubert, M. Mazura, M. Harrison, M. Eisenhauer, *et al.*, “Internet of things strategic research roadmap”, *Internet of Things-Global Technological and Societal Trends*, vol. 1, pp. 9–52, 2011.

[15] H. Kopetz, *Real-time systems: design principles for distributed embedded applications.* Springer Science & Business Media, 2011.

[16] G. Suciu, A. Vulpe, S. Halunga, O. Fratu, G. Todoran, and V. Suciu, “Smart cities built on resilient cloud computing and secure internet of things”, in *Control Systems and Computer Science (CSCS), 2013 19th International Conference on*, IEEE, 2013, pp. 513–518.

[17] R. Giffinger, C. Fertner, H. Kramar, E. Meijers, *et al.*, “City-ranking of European medium-sized cities”, *Cent. Reg. Sci. Vienna UT*, 2007.

[18] M. Deakin and H. Al Waer, “From intelligent to smart cities”, *Intelligent Buildings International*, vol. 3, no. 3, pp. 140–152, 2011.

[19] United Nations (UN), *Smart Cities: Regional Perspectives.* Dubai, 2015.

[20] K. N. Qureshi and A. H. Abdullah, “A survey on intelligent transportation systems”, *Middle-East Journal of Scientific Research*, vol. 15, no. 5, pp. 629–642, 2013.

[21] M. Stiglic, N. Agatz, M. Savelsbergh, and M. Gradisar, “The benefits of meeting points in ride-sharing systems”, *Transportation Research Part B: Methodological*, vol. 82, pp. 36–53, 2015.

[22] A. Amey, J. Attanucci, and R. Mishalani, “Real-time ridesharing: opportunities and challenges in using mobile phone technology to improve rideshare services”, *Transportation Research Record: Journal of the Transportation Research Board*, no. 2217, pp. 103–110, 2011.

[23] P. Midgley, “Bicycle-sharing schemes: enhancing sustainable mobility in urban areas”, *United Nations, Department of Economic and Social Affairs*, pp. 1–12, 2011.

[24] G. H. de Almeida Correia and A. P. Antunes, “Optimization approach to depot location and trip selection in one-way carsharing systems”, *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 233–247, 2012.

[25] J. A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, S. Reddy, and M. B. Srivastava, “Participatory sensing”, *Center for Embedded Network Sensing*, 2006.

[26] B. Guo, Z. Yu, X. Zhou, and D. Zhang, “From participatory sensing to mobile crowd sensing”, in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*, IEEE, 2014, pp. 593–598.

[27] P. Mrazovic, M. Matskin, and N. Dokoohaki, “Trajectory-Based Task Allocation for Reliable Mobile Crowd Sensing Systems”, in *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*, IEEE, 2015, pp. 398–406.

[28] R. K. Ganti, F. Ye, and H. Lei, “Mobile crowdsensing: current state and future challenges”, *IEEE Communications Magazine*, vol. 49, no. 11, 2011.

[29] B. Guo, Z. Wang, Z. Yu, Y. Wang, N. Y. Yen, R. Huang, and X. Zhou, “Mobile crowd sensing and computing: The review of an emerging human-powered sensing paradigm”, *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 7, 2015.

[30] J. Liu, H. Shen, and X. Zhang, "A survey of mobile crowdsensing techniques: A critical component for the internet of things", in *Computer Communication and Networks (ICCCN), 2016 25th International Conference on*, IEEE, 2016, pp. 1–6.

[31] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "CarTel: a distributed mobile sensor computing system", in *Proceedings of the 4th international conference on Embedded networked sensor systems*, ACM, 2006, pp. 125–138.

[32] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones", in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ACM, 2008, pp. 323–336.

[33] F. Calabrese and C. Ratti, "Real time rome", *Networks and Communication studies*, vol. 20, no. 3-4, pp. 247–258, 2006.

[34] L. Liu, A. Biderman, and C. Ratti, "Urban mobility landscape: Real time monitoring of urban mobility patterns", in *Proceedings of the 11th International Conference on Computers in Urban Planning and Urban Management*, 2009, pp. 1–16.

[35] P. Zhou, Y. Zheng, and M. Li, "How long to wait?: predicting bus arrival time with mobile phone based participatory sensing", in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ACM, 2012, pp. 379–392.

[36] C. Chen, D. Zhang, Z.-H. Zhou, N. Li, T. Atmaca, and S. Li, "B-Planner: Night bus route planning using large-scale taxi GPS traces", in *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*, IEEE, 2013, pp. 225–233.

[37] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "VTrack: accurate, energy-aware road traffic delay estimation using mobile phones", in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ACM, 2009, pp. 85–98.

[38] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service", in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, IEEE, 2013, pp. 410–421.

[39] C. Costa, C. Laoudias, D. Zeinalipour-Yazti, and D. Gunopulos, "SmartTrace: Finding similar trajectories in smartphone networks without disclosing the traces", in *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, IEEE, 2011, pp. 1288–1291.

[40] D. Zeinalipour-Yazti, C. Laoudias, C. Costa, M. Vlachos, M. I. Andreou, and D. Gunopulos, "Crowdsourced trace similarity with smartphones", *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 6, pp. 1240–1253, 2013.

[41] S. M. Kumari and N. Geethanjali, "A survey on shortest path routing algorithms for public transport travel", *Global Journal of Computer Science and Technology*, vol. 9, no. 5, pp. 73–75, 2010.

[42] S. N. Kumar and R. Panneerselvam, "A survey on the vehicle routing problem and its variants", *Intelligent Information Management*, vol. 4, no. 03, p. 66, 2012.

[43] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review", *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016.

[44] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems", *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.

[45] M. Padberg and G. Rinaldi, “A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems”, *SIAM review*, vol. 33, no. 1, pp. 60–100, 1991.

[46] R. Gomory, “An algorithm for the mixed integer problem”, RAND CORP SANTA MONICA CA, Tech. Rep., 1960.

[47] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. Savelsbergh, and P. H. Vance, “Branch-and-price: Column generation for solving huge integer programs”, *Operations research*, vol. 46, no. 3, pp. 316–329, 1998.

[48] R. Fukasawa, H. Longo, J. Lysgaard, M. P. de Aragão, M. Reis, E. Uchoa, and R. F. Werneck, “Robust branch-and-cut-and-price for the capacitated vehicle routing problem”, *Mathematical programming*, vol. 106, no. 3, pp. 491–511, 2006.

[49] G. Clarke and J. W. Wright, “Scheduling of vehicles from a central depot to a number of delivery points”, *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[50] M. M. Solomon, “Algorithms for the vehicle routing and scheduling problems with time window constraints”, *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[51] W. Dullaert *et al.*, “A sequential insertion heuristic for the vehicle routing problem with time windows with relatively few customers per route”, Tech. Rep., 2000.

[52] G. Ioannou, M. Kritikos, and G. Prastacos, “A greedy look-ahead heuristic for the vehicle routing problem with time windows”, *Journal of the Operational Research Society*, vol. 52, no. 5, pp. 523–537, 2001.

[53] J. Bramel and D. Simchi-Levi, “Probabilistic analyses and practical algorithms for the vehicle routing problem with time windows”, *Operations Research*, vol. 44, no. 3, pp. 501–509, 1996.

[54] J.-Y. Potvin and J.-M. Rousseau, “A parallel route building algorithm for the vehicle routing and scheduling problem with time windows”, *European Journal of Operational Research*, vol. 66, no. 3, pp. 331–340, 1993.

[55] C. Foisy and J.-Y. Potvin, “Implementing an insertion heuristic for vehicle routing on parallel hardware”, *Computers & operations research*, vol. 20, no. 7, pp. 737–745, 1993.

[56] B. E. Gillett and L. R. Miller, “A heuristic algorithm for the vehicle-dispatch problem”, *Operations research*, vol. 22, no. 2, pp. 340–349, 1974.

[57] K. Sörensen and F. W. Glover, “Metaheuristics”, in *Encyclopedia of operations research and management science*, Springer, 2013, pp. 960–970.

[58] D. Whitley, “A genetic algorithm tutorial”, *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[59] F. Glover, “Tabu search—part I”, *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

[60] ——, “Tabu search—part II”, *ORSA Journal on computing*, vol. 2, no. 1, pp. 4–32, 1990.

[61] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *et al.*, “Optimization by simulated annealing”, *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[62] N. Mladenović and P. Hansen, “Variable neighborhood search”, *Computers & operations research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[63] T. A. Feo and M. G. Resende, “Greedy randomized adaptive search procedures”, *Journal of global optimization*, vol. 6, no. 2, pp. 109–133, 1995.

[64] M. Dorigo and G. Di Caro, “Ant colony optimization: a new meta-heuristic”, in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, IEEE, vol. 2, 1999, pp. 1470–1477.

[65] G. Chatzimilioudis, A. Konstantinidis, C. Laoudias, and D. Zeinalipour-Yazti, “Crowdsourcing with smartphones”, *IEEE Internet Computing*, vol. 16, no. 5, pp. 36–44, 2012.

[66] G. Cardone, L. Foschini, P. Bellavista, A. Corradi, C. Borcea, M. Talasila, and R. Curtmola, “Fostering participaction in smart cities: a geo-social crowdsensing platform”, *IEEE Communications Magazine*, vol. 51, no. 6, pp. 112–119, 2013.

[67] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, “Mining interesting locations and travel sequences from GPS trajectories”, in *Proceedings of the 18th international conference on World wide web*, ACM, 2009, pp. 791–800.

[68] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, “Understanding mobility based on GPS data”, in *Proceedings of the 10th international conference on Ubiquitous computing*, ACM, 2008, pp. 312–321.

[69] Y. Zheng, X. Xie, and W.-Y. Ma, “GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory.”, *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.

[70] P. Mrazovic and M. Matskin, “Mobics: Mobile platform for combining crowdsourcing and participatory sensing”, in *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, IEEE, vol. 2, 2015, pp. 553–562.

[71] T. Roopa, A. N. Iyer, and S. Rangaswamy, “CroTIS-Crowdsourcing Based Traffic Information System”, in *Big Data (BigData Congress), 2013 IEEE International Congress on*, IEEE, 2013, pp. 271–277.

[72] A. Tamilin, I. Carreras, E. Ssebaggala, A. Opira, and N. Conci, “Context-aware mobile crowdsourcing.”, in *UbiComp*, 2012, pp. 717–720.

[73] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda, “PEIR, the personal environmental impact report, as a platform for participatory sensing systems research”, in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, ACM, 2009, pp. 55–68.

[74] A. Korthaus and W. Dai, “Crowdsourcing in Heterogeneous Networked Environments-Opportunities and Challenges”, in *Network-Based Information Systems (NBiS), 2012 15th International Conference on*, IEEE, 2012, pp. 483–488.

[75] S. S. Kanhere, “Participatory sensing: Crowdsourcing data from mobile smartphones in urban spaces”, in *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, IEEE, vol. 2, 2011, pp. 3–6.

[76] M. Allahbakhsh, A. Ignjatovic, B. Benatallah, S.-M.-R. Beheshti, N. Foo, and E. Bertino, “An Analytic Approach to People Evaluation in Crowdsourcing Systems”, *arXiv preprint arXiv:1211.3200*, 2012.

[77] H. Amintoosi, M. Allahbakhsh, S. Kanhere, and M. Niazi Torshiz, “Trust assessment in social participatory networks”, in *Computer and Knowledge Engineering (ICCKE), 2013 3th International eConference on*, IEEE, 2013, pp. 437–442.

[78] K. L. Huang, S. S. Kanhere, and W. Hu, "Are you contributing trustworthy data?: the case for a reputation system in participatory sensing", in *Proceedings of the 13th ACM international conference on Modeling, analysis, and simulation of wireless and mobile systems*, ACM, 2010, pp. 14–22.

[79] H. Yang, J. Zhang, and P. Roe, "Using reputation management in participatory sensing for data classification", *Procedia Computer Science*, vol. 5, pp. 190–197, 2011.

[80] E. Chang, F. Hussain, and T. Dillon, *Trust and reputation for service-oriented environments: technologies for building business intelligence and consumer confidence.* John Wiley & Sons, 2006.

[81] S.-W. Huang and W.-T. Fu, "Enhancing reliability using peer consistency evaluation in human computation", in *Proceedings of the 2013 conference on Computer supported cooperative work*, ACM, 2013, pp. 639–648.

[82] P. G. Ipeirotis, F. Provost, and J. Wang, "Quality management on amazon mechanical turk", in *Proceedings of the ACM SIGKDD workshop on human computation*, ACM, 2010, pp. 64–67.

[83] A. P. Dawid and A. M. Skene, "Maximum likelihood estimation of observer error-rates using the EM algorithm", *Applied statistics*, pp. 20–28, 1979.

[84] S. Reddy, D. Estrin, and M. Srivastava, "Recruitment framework for participatory sensing data collections", in *Pervasive Computing*, Springer, 2010, pp. 138–155.

[85] F. Hao, M. Jiao, G. Min, and L. T. Yang, "A trajectory-based recruitment strategy of social sensors for participatory sensing", *Communications Magazine, IEEE*, vol. 52, no. 12, pp. 41–47, 2014.

[86] G. Cardone, A. Cirri, A. Corradi, and L. Foschini, "The participact mobile crowd sensing living lab: The testbed for smart cities", *Communications Magazine, IEEE*, vol. 52, no. 10, pp. 78–85, 2014.

[87] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma, "Mining user similarity based on location history", in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, ACM, 2008, p. 34.

[88] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: ordering points to identify the clustering structure", in *ACM Sigmod Record*, ACM, vol. 28, 1999, pp. 49–60.

[89] M. Lin and W.-J. Hsu, "Mining GPS data for mobility patterns: A survey", *Pervasive and Mobile Computing*, vol. 12, pp. 1–16, 2014.

[90] D. Birant and A. Kut, "ST-DBSCAN: An algorithm for clustering spatial–temporal data", *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 208–221, 2007.

[91] J. Sander, X. Qin, Z. Lu, N. Niu, and A. Kovarsky, "Automatic extraction of clusters from hierarchical clustering representations", in *Advances in knowledge discovery and data mining*, Springer, 2003, pp. 75–87.

[92] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann, "Time-aware point-of-interest recommendation", in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2013, pp. 363–372.

[93] Y. Zheng, L. Wang, R. Zhang, X. Xie, and W.-Y. Ma, "GeoLife: Managing and understanding your past life over maps", in *Mobile Data Management, 2008. MDM'08. 9th International Conference on*, IEEE, 2008, pp. 211–212.

[94] European Commission, *Urban Freight Transport and Logistics: An Overview of the European Research and Policy.* European Commission Publications, 2006.

[95] CIVITAS Initiative WIKI consortium, *Smart choices for cities: Making urban freight logistics more sustainable.* 2015.

[96] C. Jaffeux and P. Wieser, *Essentials of logistics and management.* CRC Press, 2012.

[97] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice.* OTexts, 2014.

[98] P. Pluvinet, J. Gonzalez-Feliu, and C. Ambrosini, "GPS data analysis for understanding urban goods movement", *Procedia-Social and Behavioral Sciences*, vol. 39, pp. 450–462, 2012.

[99] X. Yang, Z. Sun, X. Ban, and J. Holguin-Veras, "Urban Freight Delivery Stop Identification with GPS Data", *Transportation Research Record: Journal of the Transportation Research Board*, no. 2411, pp. 55–61, 2014.

[100] M. Tozzi, M. V. Corazza, and A. Musso, "Recurring patterns of commercial vehicles movements in urban areas: the Parma case study", *Procedia-Social and Behavioral Sciences*, vol. 87, pp. 306–320, 2013.

[101] J. Munuzuri, P. Cortes, L. Onieva, and J. Guadix, "Modelling peak-hour urban freight movements with limited data availability", *Computers & Industrial Engineering*, vol. 59, no. 1, pp. 34–44, 2010.

[102] ——, "Estimation of daily vehicle flows for urban freight deliveries", *Journal of Urban Planning and Development*, vol. 138, no. 1, pp. 43–52, 2011.

[103] T. Cherrett, J. Allen, F. McLeod, S. Maynard, A. Hickford, and M. Browne, "Understanding urban freight activity–key issues for freight planning", *Journal of Transport Geography*, vol. 24, pp. 22–32, 2012.

[104] J. Froehlich, J. Neumann, and N. Oliver, "Sensing and Predicting the Pulse of the City through Shared Bicycling.", in *IJCAI*, vol. 9, 2009, pp. 1420–1426.

[105] A. Kaltenbrunner, R. Meza, J. Grivolla, J. Codina, and R. Banchs, "Urban cycles and mobility patterns: Exploring and predicting trends in a bicycle-based public transport system", *Pervasive and Mobile Computing*, vol. 6, no. 4, pp. 455–466, 2010.

[106] B. Xu, O. Wolfson, J. Yang, L. Stenneth, S. Y. Philip, and P. C. Nelson, "Real-time street parking availability estimation", in *2013 IEEE 14th International Conference on Mobile Data Management*, IEEE, vol. 1, 2013, pp. 16–25.

[107] M. Caliskan, A. Barthels, B. Scheuermann, and M. Mauve, "Predicting parking lot occupancy in vehicular ad hoc networks", in *2007 IEEE 65th Vehicular Technology Conference-VTC2007-Spring*, IEEE, 2007, pp. 277–281.

[108] A. Klappenecker, H. Lee, and J. L. Welch, "Finding available parking spaces made easy", *Ad Hoc Networks*, vol. 12, pp. 243–249, 2014.

[109] E. I. Vlahogianni, K. Kepaptsoglou, V. Tsetsos, and M. G. Karlaftis, "A real-time parking prediction system for smart cities", *Journal of Intelligent Transportation Systems*, vol. 20, no. 2, pp. 192–204, 2016.

[110] Y. Zheng, S. Rajasegarar, and C. Leckie, "Parking availability prediction for sensor-enabled car parks in smart cities", in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on*, IEEE, 2015, pp. 1–6.

[111] B. Chen, F. Pinelli, M. Sinn, A. Botea, and F. Calabrese, "Uncertainty in urban mobility: Predicting waiting times for shared bicycles and parking lots", in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, IEEE, 2013, pp. 53–58.

[112] F. Yu, J. Guo, X. Zhu, and G. Shi, "Real time prediction of unoccupied parking space using time series model", in *Transportation Information and Safety (ICTIS), 2015 International Conference on*, IEEE, 2015, pp. 370–374.

[113] World Tourism Organization (UNWTO), *Global Report on City Tourism - Cities 2012 Project.* Madrid, 2012.

[114] P. Vansteenwegen and D. Van Oudheusden, "The mobile tourist guide: an OR opportunity", *OR Insight*, vol. 20, no. 3, pp. 21–27, 2007.

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

[116] I.-M. Chao, B. L. Golden, and E. A. Wasil, "A fast and effective heuristic for the orienteering problem", *European Journal of Operational Research*, vol. 88, no. 3, pp. 475–489, 1996.

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

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

[119] C. Verbeeck, P. Vansteenwegen, and E.-H. Aghezzaf, "An extension of the arc orienteering problem and its application to cycle trip planning", *Transportation research part E: logistics and transportation review*, vol. 68, pp. 64–78, 2014.

[120] Y. Lu and C. Shahabi, "An arc orienteering algorithm to find the most scenic path on a large-scale road network", in *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ACM, 2015.

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

[122] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer programming formulation of traveling salesman problems", *Journal of the ACM (JACM)*, vol. 7, no. 4, pp. 326–329, 1960.

[123] A. Maruyama, N. Shibata, Y. Murata, K. Yasumoto, and M. Ito, "A personal tourism navigation system to support traveling multiple destinations with time restrictions", in *Advanced Information Networking and Applications, 2004. AINA 2004. 18th International Conference on*, IEEE, vol. 2, 2004, pp. 18–21.

[124] A. Garcia, M. T. Linaza, O. Arbelaitz, and P. Vansteenwegen, "Intelligent routing system for a personalised electronic tourist guide", *Information and Communication Technologies in Tourism 2009*, pp. 185–197, 2009.

[125] D. Gavalas, C. Konstantopoulos, K. Mastakas, and G. Pantziou, "Mobile recommender systems in tourism", *Journal of Network and Computer Applications*, vol. 39, pp. 319–333, 2014.

[126] J. Borràs, A. Moreno, and A. Valls, “Intelligent tourism recommender systems: A survey”, *Expert Systems with Applications*, vol. 41, no. 16, pp. 7370–7389, 2014.

[127] I.-M. Chao, B. L. Golden, and E. A. Wasil, “The team orienteering problem”, *European journal of operational research*, vol. 88, no. 3, pp. 464–474, 1996.

[128] C. Archetti, A. Hertz, and M. G. Speranza, “Metaheuristics for the team orienteering problem”, *Journal of Heuristics*, vol. 13, no. 1, pp. 49–76, 2007.

[129] L. Ke, C. Archetti, and Z. Feng, “Ants can solve the team orienteering problem”, *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 648–665, 2008.

[130] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. Van Oudheusden, “A guided local search metaheuristic for the team orienteering problem”, *European Journal of Operational Research*, vol. 196, no. 1, pp. 118–127, 2009.

[131] ——, “Metaheuristics for tourist trip planning”, in *Metaheuristics in the service industry*, Springer, 2009, pp. 15–31.

[132] ——, “Iterated local search for the team orienteering problem with time windows”, *Computers & Operations Research*, vol. 36, no. 12, pp. 3281–3290, 2009.

[133] A. Garcia, P. Vansteenwegen, O. Arbelaitz, W. Souffriau, and M. T. Linaza, “Integrating public transportation in personalised electronic tourist guides”, *Computers & Operations Research*, vol. 40, no. 3, pp. 758–774, 2013.

[134] D. Gavalas, C. Konstantopoulos, K. Mastakas, G. Pantziou, and N. Vathis, “Efficient heuristics for the time dependent team orienteering problem with time windows”, in *Applied Algorithms*, Springer, 2014, pp. 152–163.

[135] K. Sylejmani, J. Dorn, and N. Musliu, “A Tabu Search approach for Multi Constrained Team Orienteering Problem and its application in touristic trip planning”, in *Hybrid Intelligent Systems (HIS), 2012 12th International Conference on*, IEEE, 2012, pp. 300–305.

[136] A. Garcia, P. Vansteenwegen, W. Souffriau, O. Arbelaitz, and M. Linaza, “Solving multi constrained team orienteering problems to generate tourist routes”, 2009.

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

[138] M. R. Garey, D. S. Johnson, and L. Stockmeyer, “Some simplified NP-complete graph problems”, *Theoretical computer science*, vol. 1, no. 3, pp. 237–267, 1976.

[139] M. G. Resende and C. C. Ribeiro, “GRASP: Greedy randomized adaptive search procedures”, in *Search methodologies*, Springer, 2014, pp. 287–312.

[140] P. Hansen, N. Mladenović, and J. A. M. Pérez, “Variable neighbourhood search: methods and applications”, *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, 2010.

[141] G. A. Croes, “A method for solving traveling-salesman problems”, *Operations research*, vol. 6, no. 6, pp. 791–812, 1958.

[142] P. Mrazovic, B. Eravci, J. L. Larriba-Pey, H. Ferhatosmanoglu, and M. Matskin, “Understanding and Predicting Trends in Urban Freight Transport”, in *Mobile Data Management (MDM), 2017 18th IEEE International Conference on*, IEEE, 2017, pp. 124–133.

[143] B. Kolbay, P. Mrazovic, and J. L. Larriba-Pey, “Analyzing Last Mile Delivery Operations in Barcelona’s Urban Freight Transport Network”, in *Cloud Infrastructures, Services, and IoT Systems for Smart Cities*, Springer, 2017, pp. 13–22.

[144] T. Bektas, “The multiple traveling salesman problem: an overview of formulations and solution procedures”, *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[145] L. Bianchi, J. Knowles, and N. Bowler, “Local search for the probabilistic traveling salesman problem: Correction to the 2-p-opt and 1-shift algorithms”, *European Journal of Operational Research*, vol. 162, no. 1, pp. 206–219, 2 005.

[146] M. Sedighpour, M. Yousefikhoshbakht, and N. Mahmoodi Darani, “An effective genetic algorithm for solving the multiple traveling salesman problem”, *Journal of Optimization in Industrial Engineering*, no. 8, pp. 73–79, 2012.

[147] W. Liu, S. Li, F. Zhao, and A. Zheng, “An ant colony optimization algorithm for the multiple traveling salesmen problem”, in *4th IEEE Conference on Industrial Electronics and Applications*, IEEE, 2009, pp. 1533–1537.

[148] Y. Dumas, J. Desrosiers, E. Gelinas, and M. M. Solomon, “An optimal algorithm for the traveling salesman problem with time windows”, *Operations research*, vol. 43, no. 2, pp. 367–371, 1995.

[149] I. Kara and T. Derya, “Formulations for Minimizing tour duration of the traveling salesman problem with time Windows”, *Procedia Economics and Finance*, vol. 26, pp. 1026–1034, 2015.

[150] A. Salehipour, K. Sörensen, P. Goos, and O. Bräysy, “Efficient GRASP+VND and GRASP+VNS metaheuristics for the traveling repairman problem”, *4OR: A Quarterly Journal of Operations Research*, vol. 9, no. 2, pp. 189–209, 2011.

[151] M. Malek, M. Guruswamy, M. Pandya, and H. Owens, “Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem”, *Annals of Operations Research*, vol. 21, no. 1, pp. 59–84, 1989.

[152] K. Fleszar, I. H. Osman, and K. S. Hindi, “A variable neighbourhood search algorithm for the open vehicle routing problem”, *European Journal of Operational Research*, vol. 195, no. 3, pp. 803–809, 2009.

[153] F. Carrabs, J.-F. Cordeau, and G. Laporte, “Variable neighborhood search for the pickup and delivery traveling salesman problem with LIFO loading”, *INFORMS Journal on Computing*, vol. 19, no. 4, pp. 618–632, 2007.

[154] J. Brandão, “A tabu search algorithm for the open vehicle routing problem”, *European Journal of Operational Research*, vol. 157, no. 3, pp. 552–564, 2004.

[155] M. Gendreau, G. Laporte, and F. Semet, “A tabu search heuristic for the undirected selective travelling salesman problem”, *European Journal of Operational Research*, vol. 106, no. 2-3, pp. 539–545, 1998.

[156] Z. H. Ahmed, “Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator”, *International Journal of Biometrics & Bioinformatics (IJBB)*, vol. 3, no. 6, p. 96, 2010.

[157] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, “Genetic algorithms for the traveling salesman problem”, in *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, 1985, pp. 160–165.

[158] A. S. Khan et. al., “Idle emissions from heavy-duty diesel vehicles: review and recent data”, *Journal of the Air & Waste Management Association*, vol. 56, no. 10, pp. 1404–1419, 2006.

[159] J. Schuijbroek, R. C. Hampshire, and W.-J. Van Hoeve, “Inventory rebalancing and vehicle routing in bike sharing systems”, *European Journal of Operational Research*, vol. 257, no. 3, pp. 992–1004, 2017.

[160] J. Liu, L. Sun, W. Chen, and H. Xiong, “Rebalancing Bike Sharing Systems: A Multi-source Data Smart Optimization”, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2016, pp. 1005–1014.

[161] J. Pfrommer, J. Warrington, G. Schildbach, and M. Morari, “Dynamic vehicle redistribution and online price incentives in shared mobility systems”, *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 4, pp. 1567–1578, 2014.

[162] R. Regue and W. Recker, “Proactive vehicle routing with inferred demand to solve the bikesharing rebalancing problem”, *Transportation Research Part E: Logistics and Transportation Review*, vol. 72, pp. 192–209, 2014.

[163] G. Laporte, “The traveling salesman problem: An overview of exact and approximate algorithms”, *European Journal of Operational Research*, vol. 59, no. 2, pp. 231–247, 1992.

[164] S. B. Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, “A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition”, *Expert systems with applications*, vol. 39, no. 8, pp. 7067–7083, 2012.

[165] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[166] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks”, in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[167] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with LSTM”, 1999.

[168] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks”, in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[169] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt, and G. Dueck, “Record breaking optimization results using the ruin and recreate principle”, *Journal of Computational Physics*, vol. 159, no. 2, pp. 139–171, 2000.

[170] M. Dell, M. Iori, S. Novellani, T. Stützle, *et al.*, “A destroy and repair algorithm for the bike sharing rebalancing problem”, *Computers & Operations Research*, vol. 71, pp. 149–162, 2016.

[171] J. Malani, N. Sinha, N. Prasad, and V. Lokesh, *Forecasting Bike Sharing Demand*.

[172] P. Mrazovic, I. De La Rubia, J. Urmeneta, C. Balufo, R. Tapias, M. Matskin, and J. L. Larriba-Pey, “CIGO! Mobility management platform for growing efficient and balanced smart city ecosystem”, in *Smart Cities Conference (ISC2), 2016 IEEE International*, IEEE, 2016, pp. 1–4.