Multi-Network integration for an Intelligent Mobility

Ali Masri

Multi-Network Integration for an Intelligent Mobility

Thèse de doctorat de l'Université Paris-Saclay préparée à l'Université de Versailles Saint-Quentin-En-Yvelines

École doctorale n°556 sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Versailles, le 28 Novembre 2017, par

Ali MASRI

Titre : Intégration Multi-Réseaux pour la Mobilité Intelligente

Mots clés : Données de transport ouvertes, intégration de données spatiales, interconnexion de schémas, planification de trajets multimodaux.

Les systèmes de transport sont un des leviers puissants du progrès de toute société. Récemment, les modes de déplacements ont évolué significativement et se diversifient. Les distances quotidiennement parcourues par les citoyens ne cessent d'augmenter au cours de ces dernières années. Cette évolution impacte l'attractivité et la compétitivité mais aussi la qualité de vie grandement dépendante de l'évolution des mobilités des personnes et des marchandises. Les gouvernements et les collectivités territoriales développent de plus en plus des politiques d'incitation à l'éco-mobilité.

Dans cette thèse, nous nous concentrons sur les systèmes de transport public. Ces derniers évoluent continuellement et offrent de nouveaux services couvrant différents modes de transport pour répondre à tous les besoins des passagers. Outre les systèmes de transport en commun, prévus pour le transport de masse, de nouveaux services de mobilité ont vu le jour, tels que le transport à la demande, le covoiturage planifié ou dynamique et l'autopartage ou les vélos en libre-service. Ils offrent des solutions alternatives de mobilité et pourraient être complémentaires aux services traditionnels. Cependant, ces services sont à l'heure actuelle isolés du reste des modes de transport et des solutions multimodales. Ils sont proposés comme une alternative mais sans intégration réelle aux plans proposés par les outils existants. Pour permettre la multi-modalité, le principal challenge de cette thèse est l'intégration de données et/ou de services provenant de systèmes de transport hétérogènes.

Par ailleurs, le concept de données ouvertes est aujourd'hui adopté par de nombreuses organisations publiques et privées, leur permettant de publier leurs sources de données sur le Web et de gagner ainsi en visibilité. On se place dans le contexte des données ouvertes et des méthodes et outils du web sémantique pour réaliser cette intégration, en offrant une vue unifiée des réseaux et des services de transport. Les verrous scientifiques auxquels s'intéresse cette thèse sont liés aux problèmes d'intégration à la fois des données et des services informatiques des systèmes de transport sous-jacents.

Nos principales contributions sont les suivantes :

i) Une approche d'appariement de schémas pour les données géospatiales :

L'intégration des données au niveau du schéma est importante pour faciliter l'accès aux différentes représentations de données. En l'absence de normes et de format auto-décrit, il devient nécessaire d'intégrer à l'aveugle (c'est à dire, sans connaissance préalable du schéma, mais uniquement en se basant sur les instances) des données de transport. La difficulté réside ici dans le fait que ces entités sont géo-localisées est que l'information de localisation spatiale est représentée de manières très différentes selon les sources de données (coordonnées géographiques sous différente formes, adresse, etc.). La méthode que nous avons proposée utilise des services web géographiques pour identifier et d'apparier les informations de localisation dans des sources hétérogènes.

ii) Une méthode de découverte de connexions entre des réseaux de transport hétérogènes :

Cette contribution concerne l'intégration des données au niveau des instances de différents réseaux de transport. Mais contrairement à l'alignement d'entités identiques comme c'est généralement le cas des bases de données ou de connaissances, on vise à connecter des réseaux en fonction de la possibilité de connecter leurs nœuds selon des critères complexes, géographiques, temporels et sémantiques. L'approche élaborée dans la thèse permet de générer des connexions selon les conditions souhaitées et de les enrichir par diverses propriétés également définies par l'utilisateur. Elle a été appliquée efficacement à l'intégration de réseaux de transport collectif et partagé.

iii) une solution pour la planification de trajet multimodale qui intègre pleinement les nouveaux services de mobilité avec celles existantes du transport public :

Les applications de planification de trajet actuelles fonctionnent en vase clos car elles se basent sur des systèmes d'information fermés. Le covoiturage dynamique est un service émergent de transport qui devient attractif, en particulier en agglomération. L'intégration de ces services dans les solutions de planification de voyage existantes peut grandement améliorer la qualité des voyages et servir de plan de secours en cas de retard ou d'imprévu. S'il est assez simple de calculer les trajets des transports en commun connaissant leurs horaires, ce n'est pas le cas dans les services de covoiturage où tout est dynamique : les offres et les demandes ainsi que les lieux. De plus, l'ajout de la connexion avec d'autres modes augmente considérablement l'espace de recherche des solutions et donc la complexité des calculs. La solution proposée dans cette thèse propose une intégration au niveau de l'appel des services entre un planificateur dédié au transport en commun et un ou des planificateurs de covoiturage dynamique. Elle utilise des heuristiques pour élaguer l'espace de recherche et propose différentes techniques d'optimisation permettant un bon compromis entre le gain en temps de trajet et les performances de calculs.

Toutes ces contributions ont été mises en œuvre et regroupées dans un framework. Elles ont été évaluées avec des données ouvertes et des services liés au transport.

Title: Multi-Network Integration for an Intelligent Mobility

Keywords: Open transportation data, Spatial data integration, schema matching interlinking, multimodal trip planning.

Multimodality requires the integration of heterogeneous transportation data and services to construct a broad view of the transportation network. Many new transportation services (e.g. ridesharing, car sharing, bike-sharing) are emerging and gaining a lot of popularity since in some cases they provide better trip solutions. However, these services are still isolated from the existing multimodal solutions and are proposed as alternative plans without being really integrated in the suggested plans. The concept of open data is raising and being adopted by many companies where they publish their data sources to the web in order to gain visibility. The goal of this thesis is to use these data to enable multimodality by constructing an extended transportation network that links these new services to existing ones. The challenges we face mainly arise from the integration problem in both transportation services and transportation data. Our main contributions are: i) an automatic schema matching approach for geospatial datasets. It uses geospatial web services as mediators to help in automatically matching geospatial properties in geospatial datasets, ii) an approach that enables rich semantic connection generation and allows users to define custom relations between transportation data entities, iii) a multimodal trip planning approach that fully integrates ridesharing solutions within public transportation trip planners.

Declaration of Authorship

I, Ali MASRI, declare that this thesis titled, "Multi-Network Integration for an Intelligent Mobility" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: [illegible]

Date: 28/11/2017

"Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle. As with all matters of the heart, you'll know when you find it."

Steve Jobs

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisors Prof. Karine ZEITOUNI (UVSQ), Prof. Zoubida KEDAD (UVSQ) and Mr. Bertrand LEROY (VEDECOM) for the continuous support of my Ph.D study and related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better advisors and mentors for my Ph.D study.

Besides my advisors, I would like to thank the rest of my thesis committee: Prof. Jose Antonio F. DE MACEDO, Prof. Thomas DEVOGELE, Prof. Bruno DEFUDE, Prof. Dimitris KOTZINOS and Prof. Omar BOUCELMA. It is an honor having you as my jury members and having my work validated by you.

I thank my fellow lab-mates for the stimulating discussions and for all the fun we have had in the last three years. I thank my friends, Raef, Mohammad, Hanan, Ticiana, Ragheb, Tarek, Mary and Alexandros. We were not only able to support each other by deliberating over our problems and findings, but also happily by talking about things other than just our papers. In particular, I am grateful to Dr. Yehia TAHER for enlightening me the first glance of research.

Apart from the lab and the research world, I am grateful for having my friends Hasan, Ahmad, Mohammad Ali, Sandy, Rana, Ranin, Tourin, Manal, Dina and Fatima. We shared unforgettable experiences together and will share even much more in the upcoming years.

Last but not the least, I would like to thank my family: my parents Husein and Sanaa, my brothers Mohammad and Hasan, and finally my little sister Fatima for supporting me spiritually throughout writing this thesis and my life in general. Everything I have reached is because of your continuous support and love.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

Among the critical advances in human technological evolution, transportation stands as one of the most important technologies that enables us to move people and goods to far places in a short amount of time. Transportation modes vary to cover different infrastructures such as highway, rail, water, pipeline and air, while each infrastructure has its own variety of vehicle types. Transportation modes are either public and available as services for different users, or private for personal use, such as personal cars. In this thesis we focus on public transportation systems which are systems organized by the government or private organizations and meant to provide transportation services for the public in return of some fees.

Public transportation systems are continuously evolving with new services covering different modes of transportation to suit all passenger needs. Besides the public transportation systems for the mass which are scheduled in advance, new mobility services have emerged, such as transport on demand, car sharing, and ride sharing. However, each system has its own units and data which differ from the others. In fact, today planning applications work in silo since they utilize data related to network infrastructure, timetable and transportation units in one system.

Having multiple transportation services is a benefit for passengers. It gives them more options that fit different profiles and trip needs. Meanwhile, since transportation systems cover different areas, often passengers end up with combining multiple services with different modes to come up with more optimized trips. The problem is that manually doing this combination is not easy. It requires passengers to be aware of all available services and their schedule in addition to finding out the best combination of services that produces the optimal trip.

In order to optimize transportation services, an automatic way of integrating different transportation modes is required. This is known as multimodality which significantly reduces the effort on the passengers' side, since it gives them the ability to easily plan optimized trips across different services. Achieving multimodality is not a simple task. It requires coping with the data heterogeneity problem in transportation data sources. There is yet no unique standard followed by transportation systems where each one uses different data representation for modeling its time-schedule, units, stops and other information. Some approaches have moved into creating a public repository to integrate public transportation data (Google Transit[1], Syndicat des transports d'Ile-de-France (STIF)[2]). Such solutions require understanding and translating every single relevant data source in a supported area. Even though this task is a complicated one, it becomes even more complex when considering the evolution of the integrated data and the necessity of maintaining and keeping them up to date. Moreover, they are not taking into account the new highly evolving datasets such as car sharing, bike sharing, car pooling and so on. These datasets have more complex characteristics other than their dynamic nature. For example, they do not have the notion of a fixed transportation stop (i.e. no fixed location for pickup/drop-off) or any previously-known schedule, which makes their integration more complicated than scheduled transportation services.

Our goal is to enable multimodality while taking into account the new dynamic transportation services. To do so, we take advantage of the open data principle [44, 11] that companies are adopting to gain themselves better market visibility. We aim to use these public open data to provide a connection portal that represents the transportation connections between the different services and the means to discover them in a flexible and customized manner.

[1]http://maps.google.com/landing/transit/index.html
[2]http://www.stif.info

1.2 Objectives and Challenges

Despite the fact that the sources we are dealing with are open, integrating them requires careful study. In transportation data, the connections between data sources have a different meaning: rather than entity resolution, links between different transportation networks capture their physical connections in the real world, and these connections, as we will see, are complex. They mostly depend on spatial and temporal constraints. The second characteristic is that the sources are rarely documented, and the geospatial data, in particular, is hard to identify due to the diversity of geolocation representation in general. Therefore, we need to take into account both instance (for the connection) and schema levels integration. In addition, the new dynamic transportation services have different characteristics than the regular public transportation systems. They have no notion of fixed stops and time-schedules, thus making the integration, at the service level, a challenging task.

Next we define the three main problems we target in this thesis on multiple levels of integration including: instance, schema and service levels.

1.2.1 Schema Level Integration

Automatic schema matching/mapping aims at proposing an automated way of discovering matching rules between datasets. However, the domain of transportation has some specific characteristics that existing approaches cannot handle. Transportation data contain geospatial properties that are represented in various formats and structures. In order to detect a mapping between different representations, existing approaches use individual or combined matchers that work on schema and/or instance levels using various techniques, e.g.,linguistic, constraint-based, data-type based, etc. However, the mathematical-based operators used to define the similarity between relations are not suitable alone to detect the complex relations in transportation data. For instance, there is no way to find out that a combination of street1, street2, zip-code, city and country is the same as a combination of latitude and longitude between two datasets by using only some mathematical functions. This problem raises the question of how we can be able to automatically identify and map different representations of geospatial characteristics between two schemas. The fact that each transportation dataset may contain different instances is a challenge since we cannot rely on the basic instance matching techniques to know the schema mappings. Moreover, relying only on other properties, such as column names or value types, may not be sufficient by themselves.

1.2.2 Instance Level Integration

Many solutions took benefit from open data to provide rich data for smart city applications. They use linked data techniques and data interlinking tools to provide extended information relevant to both transportation and passenger profile queries [83, 20]. These techniques address equivalence detection between entities to establish links between data sources. This may help in enriching the information about entities. However, this is not always enough in transportation data. Further complex relations are required to reflect the nature of transportation connections. Beyond equivalence or sameAs links, we are interested in finding connections between transportation data sources based on the geospatial characteristics of the data, which capture the reachability between different transportation networks. Furthermore, using the given tools, we face two main limitations. The first is the restriction to a predefined set of functions for composing linking rules, due to the lack of flexibility of existing systems in defining custom functions. For instance, to calculate information such as the closeness of two transportation points of transfer (bus stop, train station, etc.), we cannot define custom functions to calculate walking distances, driving distances, etc. The user is forced to dig into the code (if available) and modify it directly. The second limitation is the representation of the generated output. Supporting complex relations requires more complex output patterns. As an example, let us suppose that a link is established between two transportation points of transfers. Existing tools can provide the output (BusStop1 nextTo TrainStation132) which does not give information about the occurrence of this relation. They are next to each others, but how close are they, and what are the modes of transportation that we can use, etc.?

1.2.3 Service Level Integration

Ridesharing is a transportation service where individual travelers share a vehicle for a trip and split travel costs with other travelers who have similar itineraries and time schedule [39]. These services are getting a lot of attention in the recent years as they are beneficial for both travelers and drivers, and friendly to the environment [39]. The main problem is that these services are still isolated from existing public transportation networks. They are proposed as alternative plans if no public transportation plan is found or simply not preferred. Integrating these services within the existing trip planning solutions may vastly improve the quality of the trips and serve as a backup plan in case some delays or unexpected events occur. The main cause behind the isolation is that it is simpler to calculate trips with scheduled networks due to the fact that we know in advance the departure and arrival times of each transportation unit. This is not the case in ridesharing services where a driver may at any time notify an intention of sharing a ride with others. This complicates the problem because these requests will appear/disappear dynamically on the network. In addition, the ridesharing problem is complex on its own [99, 39] and adding the connection with other modes will vastly increase the size of the search graph, and will result in more complex calculations.

1.3 Contributions

In this context, this thesis deals with the multi-network integration problem for an intelligent mobility. It provides different approaches for the effective and efficient integration of heterogeneous sources and services of transportation. The main contributions can be devised at three levels: the schema level, the instance level, and the service level.
On the schema level, we target the problem of the heterogeneous representations of geospatial data in transportation datasets and the lack of efficient state of art solutions to automatically detect and match them. To do so, we propose an instance-based statistical approach that uses geospatial web services as mediators to guide the matching task. The solution uses instances from the given datasets and constructs queries that are later matched with the web services results to conclude the position of the geospatial information. Using this technique and given the web service structure we are able to

detect and match n*n matching rules between the given schemas.
On the instance level, we extend existing interlinking approaches to go beyond equivalence detection by enabling the discovery of customized complex relations between dataset instances. This solves the problem of integrating open transportation datasets with providing the necessary solution to represent transportation connections between transportation instances.

Finally, on the service level, we deal with the problem of isolating the new mobility services such as ridesharing from existing trip planning solutions. These solutions propose trip plans while neglecting the integration of new mobility services as part of their plans, leaving a possibility of potentially more optimized trips for passengers. Integrating those services is complex due to their dynamic nature. There is no fixed pickup/drop-off position nor a previously-known schedule. The intuition behind our solution is to use existing trip planners as base solution and try to improve it via smart service injection algorithm. Our approach results in better trips that are faster, more comfortable with a small additional cost.

All these contributions have been implemented, and grouped in a framework we call FORTIfy. FORTIfy stands for "A Framework for Transportation Data Integration", it is a framework that alongside its components stands as a solution for enabling multimodality in transportation networks. The main goal of FORTIfy is to provide a solid foundation for building multimodal solutions by allowing easy processing, matching and integration of transportation data and services. In this thesis we use trip planning as our use-case since it is the most natural and trivial use-case on transportation data. As seen in Figure 1.1, the framework is composed of two main modules: the integration module and the services module. Our main solutions are represented in blue rectangles that are: AMiGO the automatic schema matching approach, Link++ the customized flexible interlinking approach and finally RETRy the real multimodal trip planning and its querying interface. From trip planning point of view, the framework allows travelers to plan a trip that integrates multiple transportation data sources and services, then monitor and update the suggested plan in real-time to adapt to changes. In the following, we will introduce the role of each module, its different components and how they all collaborate to form the desired solution.

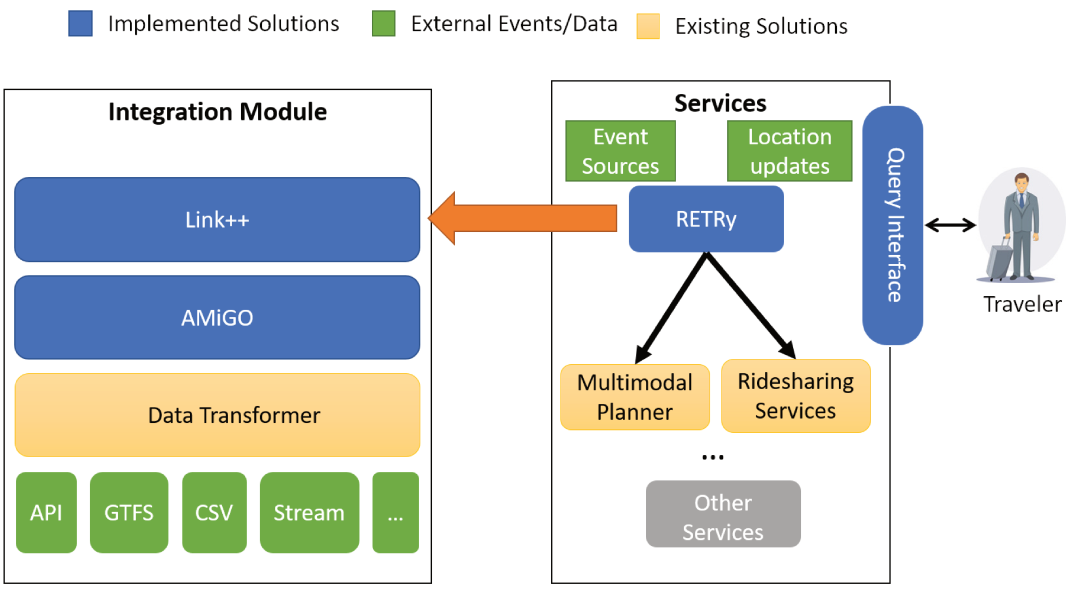


FIGURE 1.1: Overall Framework

1.3.1 Integration Module

The objective of the integration module is to gather and integrate data from different data sources in different representation and formats. Data may be obtained from APIs, GTFS or CSV files, web streams, or any other source of data. The gathered data are transformed into a unified data representation in order to make it easier to work with.

After the data being transformed, here comes the role of AMiGO, our automatic schema matcher that matches different geographical data representations. This allows us to deal with only one geospatial data representation that will be used later to link the transportation data entities.

The linking process is handled by Link++. It is an approach that enables the creation of rich semantic links between any given data sources. Link++ generates connections between transportation data entities that are then stored in a connection store to be used by other modules/services.

1.3.2 Services Module

The services module is a collection of services that uses the integrated data to serve as a foundation to transportation applications. It contains the necessary set of external event sources to keep the data up-to-date and offer trip planning services to travelers. Our multimodal trip planner - RETRy - is an example of these services that integrates multiple modes of transportation including ridesharing services to offer optimized trips for travelers. RETRy uses existing trip planning solutions to form a baseline solution that is later optimized with the integration of ridesharing services.

1.4 Thesis Structure

This thesis consists of five chapters apart from the introductory Chapter 1.

In Chapter 2 we discuss different approaches that falls in the category of transportation data integration and data modeling. We start by a introducing the standardizations efforts and their limitations. This leads us to move towards the domain of schema and instance matching were we survey the different approaches in each domain and the reason why they are still insufficient to be applied for transportation datasets. Finally, we move towards transportation services integration and the different trip planning approaches. We target the isolation of the on-demand transportation services from the existing multimodal trip planners.

In Chapter 3, we target the schema matching problem in transportation datasets. We state why existing approaches are insufficient to be applied on transportation schemas and the specific requirements needed for this domain. We introduce our instance-based automatic schema matching approach that is able to detect the heterogeneous geospatial patterns in transportation datasets. The approach is evaluated by matching the schemas of two different transportation sources and showing how the approach improved the matching results compared to the state of art approaches.

In Chapter 4 we tackle the problem of data interlinking in transportation datasets. We state why existing interlinking methods are not enough for transportation data interlinking. We introduce our approach that extends the existing solutions by defining a framework that allows users to fully customize the interlinking output and the methods to identify the relations between instances. The approach is tested by integrating two different transportation modes and showing how it improved the overall trips.

In Chapter 5 we move the focus towards the integration of transportation services. More specifically, we target the on-demand transportation services and the problem of isolating them from existing multimodal trip planners. We introduce an algorithm that integrates both services to introduce a real multimodal trip planning solution that results in more optimized trips while taking into consideration both trip duration and cost. The approach is evaluated by comparing different trips with different modes of transportation according to trip duration, cost and execution time.

We end with Chapter 6 where we sum up our contributions and show how their combination can propose a reliable solution for an improved transportation multimodality. We discuss the open problems in each solution and how they can be approaches for future improvements.

Chapter 2

State of the Art

2.1 Introduction

As mentioned in the introduction, the transportation domain is flooding with many transportation systems allowing the existence of a vast amount of services and options for travelers. Having more services is definitely better for travelers, since it enables them to select the best services that suit their needs. Despite the advantages of having this variety of services, the work-in-isolation strategy stands a restriction against what a collaboration may offer. Companies tend to provide separate services to their customers. In most cases, it is hard for customers to know all the available nearby services and find a good combination that forms the optimal trip. Obviously, connecting more sources together would improve our view of the transportation system and enable broader queries with better and more optimized plans. However, the heterogeneity of transportation data stands a challenge behind reaching the connection we desire. Transportation data is represented in different formats for each transportation category (public transportation, car sharing services, road networks, etc.). Heterogeneity makes the communication between these systems very complex, thus requiring a lot of time and efforts.

In order to find a solution for connecting multiple transportation systems, many research and standardization projects were conducted. Some think of the problem as a standardization problem e.g. OPTICITIES project[1]. Thus, they focus on creating or updating standards to cover all areas of transportation. In this way we will have a unified model that represents all aspects

[1] http://www.opticities.com/

of transportation data. This model will enable the unified access we aim to achieve multimodality. However, this solution suffers from a lot of problems as we are going to see in details in the upcoming section. But in short, the existence of many standardization organizations lead to the existence of many proposed standards that in some cases cover the same area or overlap with others. This affects the adoption of standards since what was supposed to be unified turned out to be heterogeneous. Due to this, operators moved into easier and cheaper ways to represent their data using other models [82].

Due to the standardization problems, the integration approaches targeted the problem from another level. The focus was on finding an easy way to integrate the different schemas that represent each transportation system's data [13]. Existing approaches proposed many techniques to solve this problem. In the upcoming section we will have a deeper look on the latest approaches and the techniques used in each.

Integrating the data on the schema level is important for an easier access to the different data representations. However, the main goal is to integrate the different transportation units/entities across the datasets. Here comes the role of instance level integration. More specifically, finding physical connections between transportation entities [2, 95]. The difficulty here lies in the data representation problem of how entities are represented differently across various datasets. We are going to find how instances are integrated in the state of art approaches and their limitations concerting connecting transportation data.

On the other hand, new transportation services have other issues when it comes to integration. For example, ridesharing services nowadays are still proposed as isolated services and not really integrated as sub-trips in an overall plan. The main cause behind the isolation is that it is simpler to calculate trips with scheduled networks due to the fact that we know in advance the departure and arrival times of each transportation unit. This is not the case in ridesharing services where a driver may at any time notify an intention of sharing a ride with others. This complicates the problem because these requests will appear/disappear dynamically on the network. In addition, the ridesharing problem is complex on its own [99, 39] and adding the connection with other modes will vastly increase the size of the search graph, and will result in more complex calculations.

Summing up, we classify the approaches that targets transportation data integration into four main fields: data standardization, schema matching, data interlinking and service integration. In this chapter we provide the state of the art in the above fields, stating the latest advances in each in addition to their limitations to support out ultimate goal of enabling multimodality.

2.2 Standards

Transportation data have many facets and could be abstract or represented in different ways. This has motivated the development of diverse standards. The goal of these standards is to find a unified way to represent either a specific aspect or the overall view of the transportation system. The standards we have surveyed are diverse and supported from European organizations (CEN), international organizations (ISO) and other national (French) and general ones.

The European Committee for standardization (CEN)[2] and the International Standards Organization (ISO)[3] provided many efforts to produce well developed and harmonized standards over Europe and the world. Regarding CEN, our interest lies on the two technical committees TC278 and TC287[4], which are responsible for (Road Transport and Traffic Telematics) and (Geographic Information) respectively. In working group WG3[5] of TC278, four interesting standards were produced: Transmodel [85], IFOPT [54], NeTEx [84] and SIRI [86]. We will have a look as well on INSPIRE [42] the European directive 2007/2/EC. On the international level, we are interested in the geographic data model - the GFD standard [53] - for representing geographic data files and other exchange formats such as GML [41], RDS-TMC [55] and TPEG [106]. In the following we will detail each of the mentioned standards and state their role in the field of transportation data.

[2]https://www.cen.eu/Pages/default.aspx
[3]https://www.iso.org/home.html
[4]http://www.itsstandards.eu/wgs
[5]http://www.itsstandards.eu/pt

2.2.1 Data Models

First we begin with the standards that target the data model of transportation data. Transmodel [85] is a conceptual data model for public transport. It presents the main structures of data to be used as a basis for information systems in the areas of network description, tactical planning, vehicle scheduling, fare collection, multimodal operation, schedules, personal disposition, operation monitoring, rostering, passenger information and management information. Transmodel version 6 will also include the support for stop places, accessibility, topography and point of interest. Transmodel has no official implementation. IFOPT [54] can be considered as an extension of Transmodel to support the physical components of the main transport-related fixed objects such as: boarding positions, grouping of components being a sop place, walking path, entrances, equipment and facilities, accessibility, etc. IFOPT will be included in Transmodel version 6, so no further versions are to be developed. Finally, GDF [53] (Geographic Data Files) provides a structured description of the road network and related static data. It is mainly used in: car navigation systems, fleet management, dispatch management, traffic analysis, traffic management and automatic vehicle location. Example of objects covered by GDF: road, road element, intersection, address area, building, elevation, bridge, tunnel, traffic light, traffic sign, public transport line and scope, etc.

2.2.2 Exchange formats

In order to exchange transportation system information, some standards were proposed to model these information. NETEX [84] is an exchange format based on Transmodel and IFOPT. It is divided into three parts: Network topology, timing information and description of the tariff offer. SIRI is another project launched in 2004 to provide operators with a standard framework for exchanging real time information of transport data. The concepts underlying SIRI [86] are based on Transmodel. SIRI does not provide a full description of the planned transport offer, otherwise it supports the changes to this offer. And so on the scope for a SIRI message is limited to a single day. Accessing information is done using a set of SOAP web services. Some of the services offered by SIRI: general messaging service, vehicle monitoring service, situation exchange, facility monitoring, production timetable service, estimated timetable service, stop timetable service, connection time table service and connection monitoring service.

2.2.3 Events and Traffic Information

Regarding road data events and traffic information, DATEX [52] covers six main categories: road and traffic related events (abnormal traffic, accidents, etc.), operator actions (network management, traffic control, road works, etc.), non-road event information (transit service information, car parks, etc.), measured data (travel times, traffic status, weather values, etc.), elaborated data (predicted data based on the measured ones), variable message sign messages(textual messages, pictogram, etc.). RDS-TMC [55] stands for Radio Data System - Traffic Message Channel using ALERT-C. Is an exchange format for delivering traffic and travel information using the Radio Data System based on FM radio broadcasts. RDS-TMC is moving towards the end of its life due to the merging of TMC-Forum (The manager for RDS-TMS) and TPEG-Forum under the association named TISA (Traveler Information Services). TPEG [106] stands for Traffic and Travel Information via Transport Protocol Experts Group. Managed by TISA association as an extension for TMC. TPEG covers real-time data and is dedicated to the communication with end user devices. Services covered by TPEG: service and road network information, road traffic messages, public transport information, location referencing, parking information, packing information, congestion and travel time, conditional access information. TPEG can be transmitted by any digital communication mean, e.g. DAB digital radio, DMB, internet. Currently two variants exist: a binary data format for transmission over DAB and BMB, and tpegML which is an XML implementation for Internet delivery. UTMC [107] was launched in 1997 by the UK Department for Transport. It covers real time road traffic data such as: traffic control, variable message signs, closed circuit television (CCTV), car park guidance, automatic number plate recognition (ANPR), air quality sensors, etc. The main focus of UTMS is to provide standards and protocols for the communication between roadside units and the control center, applications within the control center and between control centers. UTMC Data can be published using DATEX-II standards.

2.2.4 Geospatial-specific Standards

Concerning geospatial information, another set of standards were proposed to cover the variety of shared geospatial data. INSPIRE aims to create a Ëuropean union spatial data infrastructure, which will enable sharing geospatial information among organizations and facilitate public access across Europe. To support environmental applications 34 spatial data themes with their meta data are addressed by INSPIRE, along with technical implementation rules. The main scope covers geographic features, maps related information, roads, bridges, lakes, rivers, cadastral parcels, buildings, and protected sites. GML (Geographic Markup Language) is an XML-based exchange format designed for the description and exchange of geographical features such as: points, lines and polygons. GML provides extensions to express business specific applications through profiles and application schema. As an example, users can refer to "roads" instead of "line" or "polygon". The transportation infrastructure is one of the themes dealt with in the general directive INSPIRE [42]. In this context the GML exchange format [41] is widely used for geospatial data in all fields.

2.2.5 De facto Standards

Some projects can not be categorized as standards, however, they launched their own initiative for representing geospatial and transportation information e.g. OpenStreetMap and GTFS. OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world. Created by Steve Coast in the UK in 2004 who was inspired by the success of Wikipedia. OSM is a database generated by more than one million volunteers who perform systematic ground surveys. There are more than 43 categories of the data collected by OSM as: roads, railways, paths, waterways, bicycle routes, etc.

GTFS The General Transit Feed Specification define a common format for public transportation schedules and associated geographic information. Transit agency can publish their transit data in a GTFS feed allowing developers to write applications to consume that data in an interoperable way. In a nutshell, a GTFS feed is composed of a series of text files collected in a ZIP file. Each file models a particular aspect of transit information: stops, routes, trips, and other schedule data. GTFS can be used to produce trip planners, time table publishers, and a variety of applications. Since GTFS lacks the support for the real-time transport data, a new extension named GTFS real-time was introduced. GTFS real-time allows public transportation agencies to provide real-time updates about their fleet to application developers The real-time updates supports: Trip updates (delays, cancellations and changed routes), Service alerts (stop moved, unforeseen events affecting a station, route or the entire network), Vehicle positions (information about the vehicles including location and congestion level). The ease of implementation and use of GTFS made is widely popular by transportation companies, where many of them are publishing their data in this format.

2.2.6 Tools

Due to the large number of different formats, some tools were developed to convert the data from one format to another. CHOUETTE [72] is an open source software mainly financed by the French Ministry of Ecology, Sustainable Development, Transport and Housing. Its main purpose is to capture and exchange data, describing the scheduled timetables of public transport networks, in accordance with a standard exchange formats and profiles (NeTEx, NEPTUNE and GTFS). CHOUETTE is complemented by IRYS for SIRI based real-time information. GeomRDF [46] is another interesting tool that targets transforming geospatial data into RDF (see Section 4.3). GeomRDF takes only geospatial representation as an input, this it is not able to take other file formats such as CSV for example, which makes it hard to be adopted for transforming transportation data formats.

2.2.7 Conclusion

Despite the large efforts in the field of standardization, there still exist numerous problems that need reconsideration before adopting standards as a solution. First of all, the number of standards changes rapidly, and each time new standards cover aspects of existing standards (old ones are withdrawn in case of overlapping paradigm shifts). Moreover, the market visibility of standards is somehow slow and this gives long time for them to be supported and adopted. In addition, some standards have a limited lifetime (3 to 5 years) where the same market problems apply here too. Another problem is related

to diversity, we have seen that standards come from different organizations and origins producing some overlapping and diverse standards, and each supporting the same or different scope of data. This problem shows that the standards solution to unify data diverges us into a new heterogeneity problem of unifying standards themselves.
The latter raised the adoption of defacto standards and open data models and techniques [82] to represent transportation datasets. However, these models are non-documented as in standards and the representations again vary from one operator to another. To overcome this issue, approaches in schema and instance level integration may be considered as good candidates. They are able to identify matching rules for schemas and entities in different datasets without the need of creating a new unified model. In the next two sections we will dive into these fields and target the problems in each.

2.3 Schema Level Integration of Transportation Datasets

Due to the large number of heterogeneous formats and schemas for representing transportation data, the matching of these schemas became necessary for connecting the different services and enabling multimodality. Usually, the matching operation is done manually. However, as schemas grow larger, manual matching becomes tedious, time and effort consuming and very hard to maintain. These consequences lead to many efforts on automatizing the matching task. Many interesting surveys where provided through the past years such as [88, 101, 59, 109, 9, 102].

Automatic schema matching is one of the approaches to solve schema heterogeneity. It provides the means and the necessary techniques to a uniform access to the data. Roughly speaking, matching is the action of finding correspondences between elements of two schemas.

2.3.1 The matching problem

A mapping element defines a mapping relation between entities of two different data sources. Based on [35, 36, 101], a mapping element is a 5-uple: (id, e, e', n, R) where:

- *id* is a unique identifier of the given mapping element
- *e and e′* are the entities of the first and second schema respectively
- *n* is a confidence measure holding for the correspondence between the entities *e* and *e′*
- *R* is a relation (e.g. equivalence, more general, disjointness, overlapping) holding between the entities *e* and *e′*

The matching operation determines an alignment (a set of mapping elements) for a pair of schemas/ontologies, with additional optional parameters such as: an input alignment, matching parameters (weights, thresholds) and external resources (e.g. thesauri) see Figure 2.1.

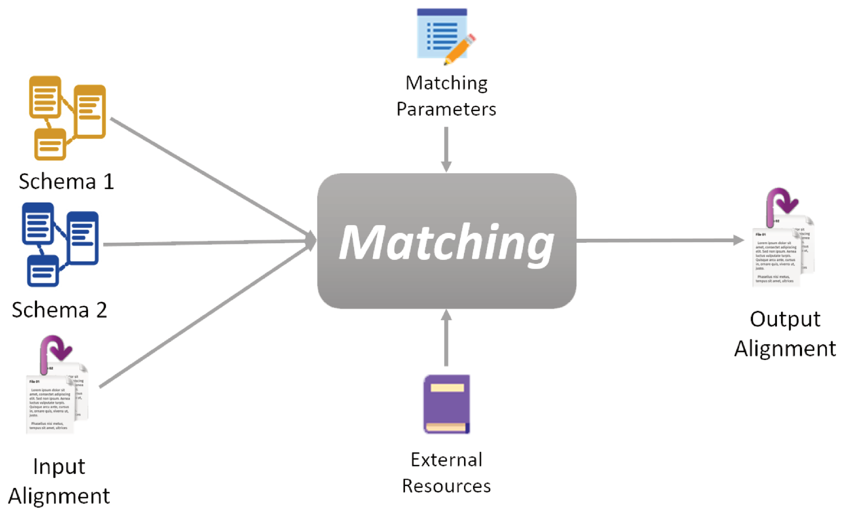


FIGURE 2.1: The matching operation

2.3.2 Classification of matching approaches

There are many ways to approach the problem of automatic schema matching. The approaches vary in the input they take, the process and techniques they follow and the output they generate [32, 31, 114]. It is important to know how these approaches are classified, and what techniques they use, in order to check how they conform with our main problem of matching transportation datasets. In this section we will present the most used classifications from [101] and [88] and dig deeper into the used techniques.

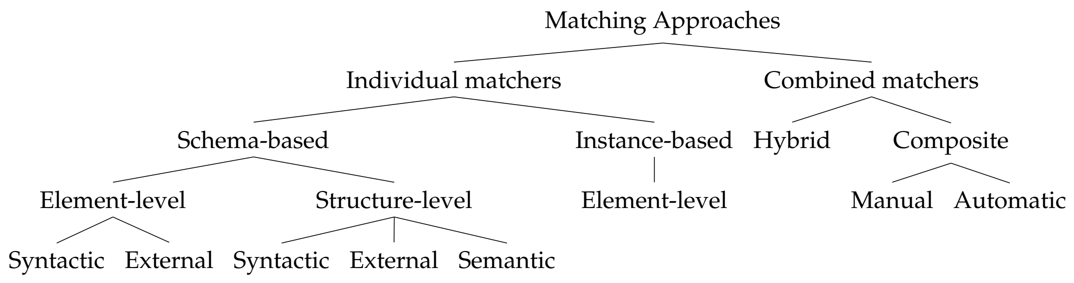
Matching dimensions

In [101] the authors introduced three matching dimensions to classify matching algorithms.

- **Input dimension** which is related to the algorithm's input. Algorithms are classified based on the used data model or on the kind of the data exploited e.g. schema level information vs instance level or both.
- **Process dimension** classifies matching algorithms based in their nature. Matching algorithms may be exact or approximate. Exact algorithms compute the precise solution of a problem, while approximate algorithms sacrifice exactness for performance.
- **Output dimensions** concerns the form of the results an algorithm produce. This is related to the cardinality of the mappings, whether the solution is final and the suitability of the relations.

Matching techniques

Another classification can be considered based on the matching techniques used. In the following we combine both classifications of [88, 101].



On the first level we may classify matchers by whether they are individual matchers or combined ones. Individual matchers are independent matchers that tend to solve the matching task based on some criteria e.g. matching by string similarity. However, these individual matchers often miss some important aspect which could significantly damage the results. As an example: Product.name matches Person.name using simple string similarity but does not match using other techniques such as path similarity. Combined matchers tackles this problem by taking advantage of each supported aspect of existing matchers and combining them to provider better results. Matchers can be combined into one algorithm (Hybrid matchers) or can be composed independently from another (Composite matchers). The advantage of Hybrid matchers is that they are faster since they require only on pass over the data.

While composite matchers require each matcher to traverse independently from another. Selecting the type of matcher to be used is dependent on the matching task. Even-though hybrid matchers are faster, composite matchers are much more flexible.
On the other hand, matchers can be classified based on their input interpretation as shown next.

- **Schema vs instance** Schema level techniques consider only schema information without the instance data, i.e. element names, attributes, relations, cardinalities, etc. Instance level techniques take benefit of instance data to guide the matching task. These techniques are used when the given schema information is limited or to improve the matching result by using instances to validate pre-computed correspondences.

- **Element-level vs structure-level** Element-level techniques consider schema elements on their own in the matching process. They try to match each element with its corresponding element in the second schema. e.g. $S_1.phone = S_2.mobile$. The most popular techniques in this field are: String-based [17], Language-based [43, 69], constraint-based [62], Linguistic resources [73, 12, 43, 69], Alignment reuse [88, 4, 89] and upper levels formal ontologies [76, 40]. On the other hand, structure-level techniques proceed with matching a combination of elements within a structure to their corresponding element or structure. e.g. $S_1.firstname + S_1.lastname = S_2.name$. The most popular techniques are: graph-based [97, 70, 37], taxonomy-based [77, 34], repository of structures [89], and model-based [71, 12, 43].

- **Syntactic vs external vs semantic** Syntactic matchers consider only the sole structure of an input, and apply a given similarity algorithm, e.g. name based matching. External information can be very helpful for a matching algorithm. External matchers make use of such information such as dictionaries, thesaurus, background knowledge, previous matching results, etc.This improves the accuracy and sometimes decrease the matching time in case of using a previous matching result. Semantic matchers use formal semantic for interpreting the input and justifying results.

2.3.3 Schema matching approaches

As we have seen, there are multiple techniques to address the problem of schema matching. However, researchers found that combining multiple techniques yield in better results than using just one technique. Based on this, schema matching approaches are either hybrid (package multiple techniques in one algorithm) or composite (techniques are combined independently from each another). Even-though hybrid matchers are faster, they lack the flexibility achieved by composite matchers. In this section, we survey the most known schema matching algorithms. We decompose the section into four main parts. The first two parts target generic hybrid and composite matchers, the third part targets ontology matching systems since they share the same goal as in schema matching systems. Finally, we introduce some matchers that are specific to geospatial data and end by discussing the limitations of the presented approaches to our main objective.

Hybrid Schema Matchers

Similarity Flooding [70] is the first hybrid matching algorithm we start with. It is semi-automatic and based on the idea of similarity propagation. It handles input schemas as directed labeled graphs and match them following the intuition: *Elements of two distinct model are similar when their adjacent elements are similar*. Example: Two elements *Employee* and *Personnel* are more likely to be equal when their attributes *pName* and *eName* matches. Their technique starts with string-based comparison of the graphs' labels which produces an initial alignment that is later refined using iterative fix-point computation. The idea in general is that similarity floods from similar nodes to their adjacent ones through propagating coefficients. The depth is increase from an iteration to another until the fix-point is reached. Results are later filtered to produce an optimized alignment. Cupid [69] is another hybrid matching algorithm that combines linguistic and structural matching techniques. It uses domain specific thesauri to guide the matching task. The algorithm encodes input schemas as graphs that undergoes three matching phases. The first phase computes linguistic similarities between schema elements. The second phase calculates structural similarity coefficients. In the final phase, the algorithm calculates weighted similarity coefficients and a final alignment is generated by selecting elements with coefficients passing the defined threshold. We end up with S-Match [43], that takes two schemas (tree-like structure) as an input and determines semantic relationships between the concepts assign to nodes (equivalence, general than, etc). The algorithm is divided into four steps based on two key notions: *concept of a label* and *concept of a node*. Step one calculates the concept between both trees' labels. Step two calculates the concept between both trees' nodes. Step three calculates the relations among the concepts of all label pairs. Step four calculates the relations among the concepts of all node pairs. The first two steps are executed in the pre-processing phase once and for all trees while the other two are executed at run time. S-Match uses 16 matchers with 13 element-level and 3 structure level ones. It uses WordNet [73] as extensional information guiding the matching.

Composite Schema Matchers

COMA [30, 4] is a composite schema matching tool that provides: an extensible library of matching algorithms, a framework for combining obtained results and an evaluation platform. COMA introduced as well the idea of reuse-oriented matching which take benefit of previously generated alignments. Compared to other tools, COMA is more flexible and allows users to perform full and fragment schema matching. COMA accepts XSD, XDR, OWL, CSV, SQL as input schemas that are later encoded as DAGs. NOM [34] differs from other approaches by determining similarity through rules manually formulated by ontology experts. In their approach they define 17 rules. Example of these rules are: **R1** *if labels are the same, then entities are probably also the same*, **R2** *if two entities have the same identifier they are identical*. They present many functions to integrate the results of these rules for an overall alignment. QOM [33] is the descendant of [34]. Its goal is to present an efficient matching algorithm. QOM has lower run-time complexity with the possibility for producing high-quality results. It uses the same pre-defined rules as its predecessor and follows six steps in their process: *Feature Engineering*, *Search Step Selection*, *Similarity Computation*, *Similarity Aggregation*, *Interpretation* and *Iteration*. In the worst-case run-time behavior QOM achieves an $O(n.log(n))$ complexity measure.

Ontology Matchers

Starting with ontology matching frameworks, we introduce RiMOM [67], a dynamic multi strategy ontology alignment. It uses the textual and structural characteristics of ontologies to their feature factors. These factors help later in dynamically selecting and combining the most relevant matching strategies. A similarity propagation strategy is followed in order to refine the results, and discover new alignments that can not be found using other strategies. AgreementMaker [23] is another framework that performs, evaluates and compares a wide range of matching methods. It supports large scale ontologies in the formats of XML, RDFS, OWL and N3 ontologies and outputs 1:1, 1:m, n:1 and n:m alignments. It provides a rich user interface allowing users to select a manual or automatic matching configuration with the capability of merging multiple match results. The automatic matchers combinations is done on three layers: compare concept features (labels, comments, annotations and instances), compare structural ontology properties and finally combine results via a linear weighted combination based on thresholds and the desired output cardinality.

Anchor-PROMPT [77] is a tool dedicated to aligning and merging ontologies by finding semantically similar terms between their entities. It is a hybrid matching algorithm that takes an input of two ontologies and a set of anchors defined by the user or automatically detected (via lexical matching). The algorithm proceeds by analyzing the paths of the input ontologies limited by the anchors in favor of determining frequently appearing terms in similar positions or paths. The algorithm terminates by determining the matching candidates based on the frequency and the user feedback.

Falcon [50] operates on RDFS and OWL and mainly targets large ontologies. It uses divide-and-conquer strategy operating in three phases: partitioning, matching blocks, and aligning blocks. Falcon uses a structure based partitioning for the first phase. In the first phase, Falcon uses a structure based partitioning to separate entities of each ontologies into clusters that are later handled as blocks. In the second phase, the generated blocks are matched based on anchors. Anchors are discovered using I-SUB string comparison technique [105]. The selected blocks is determined by a cutoff threshold based on the number of common anchors found. In the final phase, the results are combined by a sequential composition to discover alignments between block pairs.

Some approaches focus on enriching the mappings using external knowledge. BLOOMS [57] is one of those approaches that targets the fact that linked open data are mostly matched at the instance level ignoring schema-level information. The approach is based on using Wikipedia as a background knowledge to detect semantic relationships between linked open data classes. The matching process proceeds as follows. First the input ontologies are pre-processed by removing property restrictions, individuals and properties, then by tokenizing class names and removing stop words. In a second step, constructs what is called the BLOOMS forest, which is a set of trees describing the hierarchical related concepts for each class. Later the generated trees are compared to derive the relevant semantic relationships. Discovered relations can be: subClassOf and equivalentClass. The final step post-process the results using the Alignment API [35] and a reasoner. STORMA [1] is another approach towards providing richer semantic ontology mappings. They extend the existing *is-a* and *related* correspondences provided by most of the tools, to new *part-of* relationship. The approach proceeds by using a state-of-art matching tool, then taking the results and applying linguistic techniques enriched with a background knowledge to detect the semantic relatedness between elements. The advantage is that this approach reduces the search space and can be much faster on bigger ontologies. However, it is dependent on how well the first matcher behave and the assumption that the first matcher will discover all the required links is somehow weak.

ASMOV [58] is an algorithm for automatic matching and verification of ontologies. It operates on OWL ontologies and their approach is two step fold. In the first step the similarity is calculated using lexical, structural and extensional matchers. The results are then aggregated using a weighted average. Step two derives an alignment and verifies it by checking for semantic inconsistencies. The process is repeated until no new correspondences are found. OLA [37] proposed a family of distance measures for ontology matching that encompasses all OWL-Lite features. The system looks for a matching between the ontologies that minimizes the computed distances. The algorithm starts with a basics distance measure comparing labels and their data types, and improves with a fix-point algorithm.

The large support for ontology matchers is promising. However, it is rarely the case of having transportation data backed-up with such ontological representations of its entities, making the use of such approaches infeasible.

Geographic Matchers

As we have mentioned in the introduction, new data representation are being adopted to represent transportation information. The Open Data's Resource Description Framework [82] (more details in the next section) is one of those representations that models data as entities and relations between them. For this reason, some tools were proposed to match relations between these entities. For instance, Taxomap [47] is match tool developed for detecting such relations in the geographic domain. It uses linguistic and structural techniques accompanied with background resources to detect rich 1:m correspondences between concepts. The relations Taxomap can detect are: *Equivalence (isEq), subclass (isA) and semantically related (isClose) relationships*. Taxomap uses ontologies partitioning as well in order to support large scale ontologies [45]. The problem is as stated in the previous section, the ontological representation is rarely the case in the published transportation datasets. They are often represented in defacto standards or simple open data representations [82] that do not have such complex back-support of meta-data representation.

Brauner et al. [13] propose an instance-based approach for schema matching. The main goal is to match export schemas of geographical database Web services. They assume the webservice to be well described so that their input and output is known. The matching process proceeds as follows. A query formulator queries the webservices *WS1 and WS2* based on a set of global instances defined based on a global schema. The results conforming and output schemas *OS1 and OS2* are then collected and store in local tables. An occurrence matrix is constructed showing the number of elements found in common between the global schema and each input schema instances. Based on it, the system assumes that the columns with the largest common number of occurrences match. Finally, by using the transitivity characteristic the input schemas are matched based on their correspondence with the global schema. This approach is simple and effective in case the databases shares the same instances. Otherwise, it does not consider the possible different data type or structural representation between the input schemas. Considering our domain, applying this technique can not be valid since we can not have a set of global instances for all transportation operators units if they are not already integrated in the first place. This renders the use of this approach to be infeasible in our focus of study.

Conclusion

Transportation datasets contain many complex properties such as geospatial information and time schedules that can significantly vary in representations between one data source and another. These representations differ in data types, geographical scale, schema composition and many other complex properties that are mainly present as many-to-many mappings between the different schemas. An example of such properties is the geographical location of a transportation unit. This location can be represented in many different ways such as a street address (street, zipcode, city, country), a geographical point (latitude, longitude) or a more complex geographical shape as a polygon.

Analyzing existing automatic schema matching approaches shows that they are insufficient for detecting the different geospatial representations in transportation data sources. These systems mainly use mathematical similarity metrics to detect linguistic or structural similarities between entities. These metrics are not sufficient to detect the complex relations between transportation schemas. There is no mathematical way that alone can detect a relation between two schemas where the first represents a location by a latitude and longitude while the second represents it via a normal textual street address. In addition, there is no much support for many-to-many alignments, otherwise systems mostly focus on one-to-one matching rules. Considering approaches that use background knowledge (e.g. [13, 47]) such as a global knowledge base, these techniques are not applicable to transportation systems since each system has its own data and units that is not common with others. The evaluation in [57] shows the limitations of existing tools when matching geospatial datasets. The results show very low numbers while matching geospatial properties between DBpedia and Geonames schemas.

The transportation field is still not targeted specifically by any approach. It requires more powerful matchers that are able to detect the complex geospatial properties represented within it schemas. Different geospatial patterns are still not detected in current systems and the existing matchers lack the ability to match them. As an example: a schema $S1$ may consider an address based on a set of zip code, street name and a city name, while another schema $S2$ represents addresses as lat,lon couples. Matching transportation systems requires techniques that are automatically able to detect this many-to-many mapping rules with different scales, data types and representations, with no prior knowledge about the geospatial properties. We focus on the localization which can have completely different type and structure.

2.4 Instance Level Integration of Transportation Datasets

Enabling multimodality requires integrating more and more services together. In transportation systems, this can be viewed from two dimensions, linking the same entity across different datasets in order to enrich the knowledge about a single entity or linking different entities across different datasets to enable wider access to the different provided services. The first dimension is useful for linking data represented in different scopes. This is called data conflation [27, 26, 92] that is popular in geospatial data where a spatial object might have multiple geometric representations due to the precision or to the scale, e.g., a building is represented as a polygon in the first dataset and a simple point in the other. Our focus in this study is on the second dimension, that means connecting together different units from different services. In order to achieve this, data from all different services is needed. Fortunately, data is becoming more and more easily accessible and available with the introduction of the Open Linked Data concept [10]. In short, it means representing data from different sources in a unified model that can be linked together to resemble a huge graph that describes the semantic relations between its entities. With the introduction of this concept, many approaches proposed the means and methods of exposing their data in order to join this huge knowledge representation. In the category of transportation data and smart cities, the authors in [3, 83, 20] did this integration by following the linked data principles [10, 11] and they succeeded in connecting data from different sources to produce applications with wider-scope services. In [61], the authors tackled the problem of cataloging, exploring, integrating, understanding, processing and transforming urban information. They proposed an approach for incremental and continuous integration of static and streaming data, based on Semantic Web technologies, while they tested their system to a traffic diagnosis scenario.

The GeoKnow [64] project and DataLift [95] platform came as a solution to help transforming data from isolated silos into linked data. They provide the necessary tools to transform data into linked data compatible format (the resource description framework RDF [82]), link data from multiple sources using interlinking frameworks [75, 108], publish the new links and query them [87]. The result is a unified access with a rich querying framework (SPARQL) [87] over data extracted from multiple different sources with different formats.

2.4.1 Geometric Based Data Integration

As mentioned in the section's introduction, an interesting domain in geometric data integration is conflation. It targets detecting the same geographic instances described in different datasets. Knowing that the are many different ways of representing geographical entities, it is important to know the identical entities for the purpose of integration. For an instance, a building can be represented by a circle on the map, a point, a line, a polygon or any other shape representation.
Having these different representations describing the same entity in different datasets is what defines the conflation problem. Conflation aims to identify the identical entities no matter how they are represented. From an application point of view, conflation is important for combining maps to enable complex query-ability for solving spatial queries and developing a spatial analysis.
There are many approaches that target this problem. Basically, the approaches use geometric, topologic and/or semantic characteristics in the data to indicate the similarity of two entities. Geometric matching is based on the detection of corresponding objects in different datasets by comparing their geometric characteristics [92, 15]. Topologic matching uses composition or topologic relationships between different objects to match a given object. If two relationships correspond, then this correspondence can be used to find homologous objects linked by this relationship [26]. Semantic matching works according to the proximity degree of the semantic attributes between two objects from different datasets [26, 18]. It is used to find corresponding objects from different datasets that share some common or comparable attributes [112]. The simplest case is that two datasets have the same attributes whose

meanings or value ranges are defined in the same way. The semantic similarity can be also identified even when the objects from various datasets have significant representational differences [100].
Going back to our main problem, we see that conflation targets a different aspect of integrating geospatial data. The approaches aim to detect similar geospatial entities in different datasets. However, our aim is to detect relationships between different geospatial entities in transportation datasets. That is why we move in the next section to the domain of data interlinking, to study the proposed approaches and see how they can fit to our problem.

2.4.2 Data Interlinking

Links are created for open data datasets are through data interlinking/link discovery approaches [75, 108]. Data interlinking in general is a way to identify entities that represent the same real world object across different RDF data sources in a semi-automatic fashion. The process of linking requires two datasets a source and a target, a distance measure and a threshold. A link between two entities of a dataset is successfully assigned if a distance measure between them exceeds a selected threshold. The main goal is to link similar instances – that are scattered between different data sources – in order to expand the knowledge graph. The MeLinDa survey [93] described data interlinking in more details and highlighted the characteristics of the most popular approaches.

We can divide the link discovery frameworks into two categories: domain specific and generic ones. The domain specific frameworks aim to discover the links between knowledge bases of a particular domain. The second category is designed to consider the linking tasks regarding the knowledge base domain. Based on [93], table 2.1 shows the most popular interlinking tools with their properties. In the domain specific category, RKB-CRS short for co-reference resolution system [56], proposed an architecture for managing URI equivalences on the Web of Data by using Consistent Reference Services. Their approach requires a JAVA program to be written for each pair of datasets to integrate. In each program, the resources are selected along with their comparison functions. Those functions are defined by string similarity metrics on property values. GNAT [90] is an automatic interlinking tool that works on music datasets described within the Music Ontology [91].

	Similarity Techniques	**Output**	**Domain**
RKB-CRS [56]	String	owl:sameAs	Publications
GNAT [90]	String, similarity-propagation	owl:sameAs	Music
ODD-Linker [49]	String	link set	Independent
RDF-AI [94]	String, WordNet	alignment format	Independent
Silk [108]	String, numerical, date	owl:sameAs, user-specified	Independent
LIMES [75]	String, geographical, numerical, date	owl:sameAs, user-specified	Independent

TABLE 2.1: Comparison between different interlinking tools

It is implemented in prolog and based on similarity aggregation algorithm to detect relations based on resource's neighbors in a graph.

Regarding approaches with limiting output representations, ODD-Linker [49] proposed an extensible framework for interlinking relational data with high quality links. Linking rules are expressed in LinQL that is later translated to SQL queries in order to compare and identify links. LinQL supports many string matching algorithms, synonyms, hyponyms and other conditions on attributes. RDF-AI [94] is a dataset matching and fusion architecture. It takes two files, the datasets to be linked and a set of XML files describing the linking process (pre-processing, matching configuration, dataset structure, merge configuration). A local copy of the datasets is needed, and the matching is based on string similarity with an external resource (WordNet).

Moving to the more flexible systems with independent domains and user specific outputs, Silk [108] input datasets are inserted via a SPARQL endpoint URI, a local copy, or a database access. Matching configuration can be done either with a GUI toolbox or the Silk Link Specification Language (Silk LSL). User specifies the properties to be matched, the pre-processing functions and the matching technique to be used. Matching function are combined via aggregation functions (MAX, MIN, AVG). Silk provides a load of pre-processing function on Literals and numeric data types. Many comparison functions are defined including string similarity, numerical distance, date-time. The only distance function available for matching geospatial datasets, is the geometric distance — based on the Euclidean distance. It takes the latitude and the longitude of both entities and matches them according to a given threshold. The good thing in this distance function is that the threshold is well formatted, the user can define the minimal distance in meters or kilo-meters. LIMES [75] is one of the tools provided by the GeoKnow [2] project, it handles the matching in a very fast speed compared to other link discovery frameworks. LIMES provide better distance functions for geospatial data, thus we have more options to match. It supports basic string metrics, numeric vectors such as Euclidean and Orthodromic distance metrics and many other similarity metrics e.g. Hausdorff, Sum of minimum, Fréchet, etc. Writing a linkage rule in LIMES is done via XML, and no GUI is available to support the process. Although many distance metrics are supported, these functions are only good for geographical data. Geographical data is a subset of transportation data, so more powerful functions are needed, those who enable richer linking between transportation units or objects.

2.4.3 Conclusion

Existing link discovery approaches aim at detecting same instances scattered between different datasets, thus making them targeted towards equivalence detection. To do so, they provide functions and aggregations to detect owl:sameAs, part-of, subClass and other similar relationships as seen in the previous section.

The presented approaches may be suitable in some use-cases for detecting similar geographical entities in geospatial datasets such as the GeoKnow [2] and LinkedGeoData [104] projects. However, these characteristics are not enough for the case of transportation data. In transportation data, the links we aim to create represent physical connections between transportation entities. This adds additional requirements to the existing solutions to go beyond equivalence detection. More complex link types are required which in turn need different ways for detecting and representing them. A transportation link requires more suitable functions to be detected between one entity and another. Such functions must consider timetable information, geographical and road network constraints, cost information and many other characteristics. In addition, the classical owl:sameAs output is not enough to represent what a transportation connection really is. A transportation connection must represent many information in order to allow further post-processing by applications such as trip planners.

Consider that we want to connect two transportation data sources with the intention of discovering how we can reach one stop from another. The links we want to detect are the physical links from one stop to the second. What we really expect is a rich linking rule that links stops based on timetable schedules and the reaching feasibility based on road network constraint. The output we seek is a rich link that represents how much time is required to reach one stop from another, the time the link is available at, the fees or using this link, the transportation mode and many other characteristics that may be helpful for later decision making and planning. The problem is that doing so with existing tools limits us to equivalence detection due to the provided functions and output format. This results in an unsuitable way of defining the linking rule which may lead to false results in addition to insufficient link representation that does not suffice for later processing. What is required is a more representative and semantic way to connect these sources [6] showing how they can be connected from a transportation point of view.

As a conclusion, the output of an interlinking process mainly focuses on detecting a set of owl:sameAs links. However, we need to have more information in the generated links to enable better post-processing and analysis and to reduce re-calculation costs (e.g., include information about a connection status and the distance between two connected entities in transportation links).

2.5 Service Level Integration of Transportation Datasets

Up till now, we have seen studied approaches of integrating data on the schema level and on the instance level. Relating them to transportation data, the schema level integration may help in matching different data representations adopted by different transportation operators. This indeed help with the heterogeneity problem and makes it easier to work with all the different datasets. The instance level integration is important to find relations between the entities of the transportation operators. This is necessary to find physical real-world connections between the transportation units or stops for enabling the planning of multimodal solutions. This leaves us with one more interesting integration problem, the service level integration.

It is trivial that transportation trip planners are by far the most popular applications on transportation data. They provide calculations of trip that best fit a traveler's needs. Current trip planners are mostly focused around the standard scheduled transportation services. However, there are many new transportation services that differ from these services e.g. Ridesharing, Carpooling, etc. These new services are being highly adopted by people but unfortunately not by integration approaches. Integrating such services is important to allow real multimodality and more customized trip plans. For this purpose, this section shows a study of trip planning approaches on the timetable networks and on the new transportation services.

Regarding the new transportation services, we focus on ridesharing. Ridesharing is a transportation service where individual travelers share a vehicle for a trip and split travel costs with other travelers who have similar itineraries and time schedule. These services are getting a lot of attention in the recent years as they are beneficial for both travelers and drivers, and friendly to the environment [39]. We will study how ridesharing applications plan their trips, see the main problem ridesharing approaches target and finally conclude what makes integrating ridesharing with existing trip planning a problem.

Many trip planning algorithms were developed along the years. They can be decomposed based on the way they approach the problem and the solutions they provide. Some algorithms only target the problem as a routing problem on road networks. These algorithms are suitable for walking or driving passengers who seek a guide on how to traverse their routes efficiently. Other approaches target trips including public transportation services such as trains, buses, etc. They use timetable and infrastructure data to plan optimized trips for passengers. Passengers may use these trips to reach their destinations using a combination of public transportation services. On the other hand, with the appearance of on demand transportation services such as ridesharing, new algorithms were developed. The goal of these algorithms is to match service requests with service offers.

In this section we introduce the latest approaches in the fields of route planning, timetable trip planning, and ridesharing matching algorithms.

2.5.1 Route Planners

Route planning is an extensively studied field with a lot of contributions that are aimed at building fast and optimized algorithms for navigating routes. The shortest path problem is one of the most popular problems in this field, especially in trip planning since the shortest trip is generally the most requested query in such applications. In the following we will present some of the most used approaches for solving the shortest path problem.

Dijkstra's algorithm [29] is a well-known shortest path algorithm. It works by maintaining a priority queue of vertices ordered by distance from a source point. The algorithms begins with initializing all distances to infinity except the distance to the source which is initialized to zero and added to the queue. After the initializing phase the algorithm stars the iteration phase. On each iteration the algorithm extracts from the queue the vertex with the minimum distance and scans it. It looks at all of its neighbors and determines the distance to them. If the value improves the distance it is then updated and the edge is relaxed. Dijkstra is a label setting algorithm meaning that labels are only scanned once and the algorithm terminates once reaching the target.

Bellman-Ford [7] is another well-known shortest path algorithm. It does not use priority queues as in Dijkstra. Instead, it works in rounds by scanning all vertices whose distance labels have improved. The vertices to be scanned are stored in a simple FIFO queue. Bellman-Ford is a label correcting algorithm meaning that labels can be scanned more than once. It is often much faster than Dijkstra and has an advantage of working on graphs with negative edge weights.

Another algorithm is Floyed-Warshall algorithm [38]. It works by computing the distances between all pairs of vertices. Even though it may sound expensive, it is efficient for use on dense graphs.

A* algorithm [48] is a well-known algorithm that uses goal oriented techniques. In short, these techniques aim to target the search towards the goal to avoid the scan of unnecessary vertices. It uses a potential function on the vertices then runs a modified version of Dijkstra. In addition, it can be run bidirectionally to speed up the process.

Many other contributions were presented in the field of route planning. They are classified based on: i) basic techniques that work by scanning all vertices as in the top three mentioned algorithms, ii) goal oriented techniques as seen in A*, iii) separator techniques that uses separators to decompose the graphs into sub graphs, iv) hierarchical techniques that exploits the hierarchy of road networks, v) Bounded-Hop techniques that use pre-computation to add virtual shortcuts to the graph speeding up the search process and finally vi) approaches that use combinations of the above techniques. Readers are advised to look at [5] for deeper insights on the mentioned techniques and approaches.

2.5.2 Timetable-based Planners

Timetable planners [28, 25, 8, 110] aim to find trips from a source to a destination using a combination of nearby services using transit stops. In general, the timetable information require remodeling the network graph to be able to represent timetable information. Two main approaches have been proposed for modeling the shortest path problems in timetable systems: the time expanded approach and the time dependent approach.

In the time expanded approach, multiple nodes are constructed that correspond to a specific time event (departure or arrival) at a station while the edges are the connections between two events in the network. The result is a very large graph that contains all possible connections at different times according to the timetable. The authors in [96] adopted the time expanded approach to optimally solve the shortest path problem on a static graph.

Furthermore, the time expanded approach was extended in to solve multi-criteria problems in [74].

The time dependent approach avoids the existence of multiple nodes per station. It models the network as one node per station with multiple edges representing the possible different times for events between two stations. The authors in [21] were the first to target the shortest path problem on time-dependent graphs. Later, the approach was generalized to support multi-criteria by the authors in [60]. An important study was presented in [79, 78] to investigate the complexity of the shortest path problems and give the required algorithms.

The connection scan algorithm (CSA) [28] is a shortest path algorithm that uses the time expanded model. It works by receiving a stream of connections ordered by departure time and chooses the fastest way to reach one stop from another. Due to the fact that the connections are pre-sorted and can be accessed one by one in a single iteration, CSA is faster and more scalable than other existing algorithms. RAPTOR [25] is an another interesting algorithm that optimizes the number of transfers in the Pareto-sense in addition to the arrival time. SUBITO [8] is an accelerated version of Dijkstra applied to the time dependent graph model. Instead of scanning all the nodes, SUBITO uses lower bounds on the travel time to prune the search space. The authors in [110] proposed a preprocessing-based algorithm to this problem. It computes all possible transfers between trains in order to speed up the query time. However, the preprocessing time is large which leads to an extension of the approach in [111] to achieve higher speedups.

2.5.3 Ridesharing Matching Algorithms

A ridesharing request consists of two points and two constraints. The points specify the pick-up and drop-off positions and the constraints specify the waiting time and a service constraint. The waiting time constraint is the maximal time the rider can wait after making the request. The service constraint is the acceptable extra detour time from the shortest possible trip duration. The main problem in ridesharing is: given a set of cars on the road network, we need to match a rider's request to a car that can satisfy all the constraints we previously mentioned.

The main challenge in dynamic sharing systems is the ability to handle large number of trip request and cars in real-time. This is due to the dynamic movement of cars and riders and the requirements of handling requests in matter of seconds. The Filter and Refine framework [99] facilitates the problem by splitting it into two main problems and allowing us to conquer the problem on a smaller scale. In the Filter phase, the framework filters all the drivers that do not match the request criteria. While in the Refine phase, an appropriate algorithm is applied to get the matching pairs taking into account the constraints.

An example of a filtering approach can be viewed in [68], where the authors use a grid-based index to partition the map. A grid distance matrix is constructed to fill-out all the distances between the grids. Each grid index contains three main sets: a spatial set indicating a set of grids from nearest to furthest from a geospatial point of view, a temporal set indicating the set of grids from the nearest to the furthest from a temporal point of view and finally a set of vehicles that will enter the grid in the future with a time window of two hours. These indexes are then used to quickly filter-out cars that can not be matched to requests. The limitation of this approach is that the pre-computations are costly are not suitable for large scale ridesharing.

In SHAREK [14], the authors proposed three main pruning techniques to minimize the need for shortest path calculations. The used pruning techniques are: Euclidean temporal pruning that performs a range query to filter far cars, Euclidean cost pruning that filters-out cars that do not match the given cost constraints and finally a Semi-Euclidean skyline-aware pruning that selects candidates by balancing the cost vs time constraints.

Some approaches use techniques such as Branch and Bound [81] or Integer Programming [19], however, the problem with these approaches is the fact that each request requires the rescheduling of all previously computed schedules. Therefore, there is no use of previous computations.

The authors in [51] propose a tree structure to preserve previous computations and handle new requests as an insertion to the tree. The resulting solution may not be the optimal. However, it is very fast to compute. This reduces the complexity of the matching algorithms from O(k!) to O(k^2) where k is the number of requests. The limitation of this approach is that once the tree is computed, it is not updated in real time which makes the computation to be outdated.

2.5.4 Discussion

The domain of route planning has been a target of extensive research to provide passengers with optimal trips as fast as possible. Approaches in this domain differ to support different kinds of trips starting from the classical shortest path problem to more complex ones that handle public transportation networks and their timetable information. Timetable planners solve the timetable problem by introducing time-dependent graphs and the required algorithms to optimally traverse and plan trips on top of them. These algorithms are used in various trip planning applications that integrate other public transportation systems in order to provide more optimal trip plans.

On the other hand, planners that target the new on-demand transportation services are still offered as isolated solutions without being integrated within existing public transportation plans. This lead to non-optimal trips that may be improved when integrated with other services. In addition to finding a route plan, these approaches target the problem of matching a candidate driver with each trip request while satisfying the specified constraints [99, 68, 14, 51]. This problem is challenging and considered as NP-hard problem.

The main advantage of ridesharing services is their flexibility and availability which makes them perfect backup plans in case of delays or cancellations in public transportation systems. However, they are more expensive which makes them less preferable. Integrating both ridesharing services and timetable planners may lead to better trips that are faster, more reliable and comfortable with a little additional cost that is cheaper than using ridesharing alone.

Many challenges arise from this goal. In ridesharing services there is no fixed notion of stop place and time schedule where we can easily know the time and position of each car in the future. In addition, when considering both services, everything will be moving including both the passenger and the driver, which makes it difficult to detect the best place and time to issue a request. Moreover, unexpected cancellations and delays by the ridesharing service may result in a poorly planned trips that will result in an increased traveling time. Finally, more services means more search space which may greatly affect the execution time of the algorithms.

What is required is a system that is able to integrate ridesharing services and existing public transportation trip planners with a feasible execution time and reliable plans. This integration may greatly improve the trip plans and result in better trips cover more areas and satisfy different travelers profiles.

2.6 Conclusion

In this chapter we discussed the different approaches that targets transportation multimodality. We have seen the standards approaches that mainly focus on finding a unified model to represent the different aspects in transportation systems. The main limitation in such approaches is the various number of organizations that propose standards leaving another problem of standardizing standards themselves. In addition, standards have short life cycle and take a lot of time to be integrated in the market
On the other hand, we have seen another approach that targets automatic matching of different transportation schemas. In these approaches they try to match the existing transportation schemas in order to find a way of accessing different information scattered across different sources. The problem with existing approaches is their limitations in handling geospatial data that is one of the most important characteristics in transportation data. This is mainly due to the used techniques that rely on mathematical similarity metrics to find a match between on representation and another which is not the case in geospatial data. This leaves us with a need for a smart approach that is able to automatically detect different representation of geospatial data in different schemas.

Digging deeper into the instance level, we found a limitation in linking different data instances from different transportation services using the existing open data interlinking approaches. Existing approaches are more suitable to instance matching which is finding the same instance across different datasets and not finding links between different entities in different datasets. Moreover, we noticed that the interlinking output is not mature enough to represent transportation data links. What is required is a more rich way to discover and represent transportation links.

Finally, we targeted the service level integration. More specifically, we noticed the isolation of new mobility services such as ridesharing in existing multimodal journey planners. This isolation was mainly due to the different characteristics these services hold e.g. dynamicity, no notion of fixed stops or schedules. This makes the integration very complex and requiring more search space an time expensive computations. Therefore, a new approach is needed to integrate these mobility services with existing ones allowing more flexible trips that suit different passengers preferences.

As a conclusion, this work mainly targets three different aspects of transportation data integration. The first is on the schema level to enable an automatic way of matching transportation data schemas, the second is on the instance level to create links between different transportation units across different services and finally on the service level by enabling the integration of new mobility services within existing journey planners. In the upcoming chapters we will dig forward into each problem and the proposed solutions to solve them.

Chapter 3

Automatic Schema Integration for Geospatial Datasets

3.1 Introduction

Enabling multimodal solutions requires access to a large set of transportation datasets for all the considered transportation modes. The information in these datasets is large and complex. A transportation dataset must contain information for describing its entities (buses, trains, stops, etc.), fares, schedules, real-time information and much more depending on the services they offer. As we have seen in Chapter 2, there exist a large amount of transportation standards and exchange formats that are continuously evolving, making the adoption of only one universal standard an impossible objective. As a result, there is no way of uniquely representing transportation information across different datasets. This leads each company to adopt different data representations for modeling their data. Therefore, transportation datasets are highly heterogeneous, standing a big challenge against enabling multimodal solutions.

In order to integrate the data, companies need to find matching rules that map their data representation to other representations in the datasets that needs to be integrated. These matching rules are relations that describe how one or more elements from one schema can be matched with one or more elements from the second schema. Doing this task manually is error prone and time consuming. Meanwhile, the field of automatic schema matching may serve a good solution for our problem by proposing an automated way of discovering the desired matching rules. Chapter 2 highlights the existing approaches in this domain and how the problem is targeted via multiple

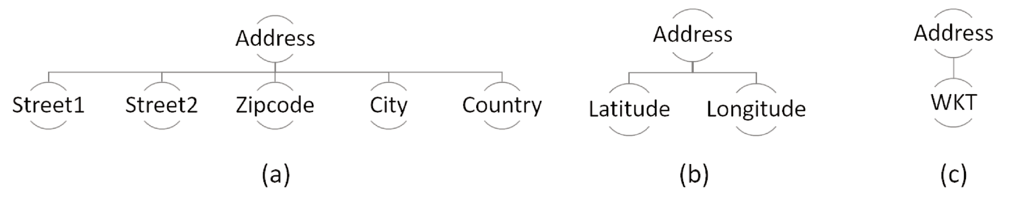


FIGURE 3.1: Different address representations

techniques. However, applying the existing matching approaches to transportation datasets does not yet produce reliable results. This is due to the specific nature of geospatial properties in transportation datasets.

Geospatial data can be represented in various formats and structures. We will discuss this further by explaining through a simple example how an address can be represented in different systems. An address can take various formats and representations in different datasets see Figure 3.1. It can be a text representation combining the attributes street1, street2, zip-code, city and a country. It can be represented by a position as a combination of latitude and longitude values, a WKT[1] (Well-known text) representation, or any other format that depends on each data source.

The differences may grow to be much more complex than what precedes. Some geospatial systems use different reference systems [63] for their data points or even different scales. For example, a stop may be represented as a single point with x and y coordinates, and it can be represented in a more complex representation such as a polygon of multiple connected points.

In order to detect matching rules, existing approaches [101] use individual or combined matchers that work on schema or instance levels using various techniques, e.g., linguistic, constraint-based, data-type based, etc.However, the proposed similarity operators are not sufficient to detect the complex relations in transportation data. For instance, we can not know that a combination of street1, street2, zip-code, city and country is the same as a combination of latitude and longitude between two datasets using only some mathematical or string equality functions. Some geospatial matching approaches [13,

[1]http://www.opengeospatial.org/standards/wkt-crs

65] use reference knowledge. Others require either identical values or representations, and will therefore fail in identifying matching rules between WKT and address for example - where the instances are different. This makes the typical instance based matching inapplicable. Moreover, relying only on other properties such as column names or value types may not be sufficient as well. In order to overcome these limitations, automatic identification and matching of geospatial characteristics is required.

The main problem we target in this chapter is the problem of automatically detecting different representations of geospatial data in transportation datasets. The challenges lie in the limitation of existing techniques in detecting the matching rules due to the problems seen above.

Our main goal is to find a solution that helps in detecting these complex relations in an automated manner. To do so, we propose AMiGO, an instance based approach to identify and match geospatial properties of transportation datasets by the use of geospatial web services. The intuition behind our solution is to use existing geospatial web services as a base knowledge. Then, query this base knowledge with the dataset instances in order to get a hint on how the data is represented in each of them. After knowing the representation in each dataset, we will be able to conclude the matching rules automatically.

In this chapter we introduce AMiGO in more details, giving more insight on the matching process and its different phases. We later evaluate the approach in a real-case scenario that walks-through each step and how it is executed.

3.2 Problem Statement

Transportation data sources use different representations to describe their data. Knowing that integrating those sources is essential to support multimodality, the integration task remains a complicated task due to the complex properties held by transportation data sources. Being more specific, the geospatial information in transportation data are highly heterogeneous, they can be represented in various formats and structures. Manually integrating

these data is doable, however it is very time consuming and error prone due to the large sets of data present. What is required is an automatic way of discovering those complex geospatial characteristics across the various data sources.
State of art matching tools rely mostly on similarity-based functions that fail detecting the different structures of geospatial data. Therefore, our main problem is to enable an automatic way for detecting the different representations of geospatial properties in transportation data sources. Consider two transportation data sources **A** and **B** with each schema consisting of **n** and **m** elements respectively. Each schema contains a set of elements $\mathbf{e}_i$ that represent different transportation properties which differ in data type, representation, scale or any other characteristics. Knowing that there is no direct mathematical similarity function that detects the relationship between the elements, the problem is to find a set of mapping rules $\mathbf{m}_i$ that indicate how elements from schema A refer to element from schema B. In addition, it is required to find a mapping function $\mathbf{f}_i$ that accompany each mapping rule and indicates how we can reach one representation from another in order to automatically switch from one representation to another.
The problem can be summarized as the following: Given two transportation data sources A and B that are implicitly geo-referenced i.e. with no information about the geo-location representation. Knowing that there exist some geospatial information in both data sources' schemas, it is required to detect those information, know what they represent and finally how they can be matched with those of the second dataset and vice-versa.

3.3 An Instance-based Approach for Automatic Schema Matching

Transportation data instances always refer to real-world objects, e.g., bike stations, bus or train stops, etc. These data are characterized by the description of an object's geographical location, represented by properties, such as coordinates, shapes, addresses, etc. The problem we are faced with is the different representations of these information. We aim at investigating a way to automatically identify and match geospatial information in transportation datasets despite their heterogeneity.

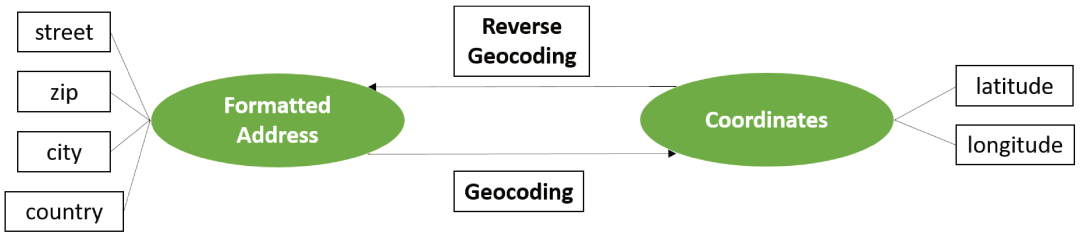


FIGURE 3.2: Geocoding vs reverse-geocoding

We propose AMiGO (Automatic Matcher for Geospatial Objects) which allows both automatic identification and matching of multiple geospatial representations. In the following, we briefly introduce the general idea, then its detailed description. Later we describe the implementation, before showing its application to the use case of transportation network schema matching.

The multi-representation feature of geospatial data is well-known, and their exist methods and tools to transform one representation to another. As an example, switching from an address to map coordinates is called Geocoding e.g. "Place de l'Etoile, Paris, France" place address to (48.8733575,2.2925186) latitude and longitude coordinates [98]. Geocoding services[2][3][4][5][6] provide this kind of service by transforming a description of a location (name of a place, coordinates, etc.) to a location on the Earth's surface via geocoding and reverse geocoding functions. They work as a search engine where the output contains all possible information regarding the location of the queried data. Geocoding is the process of taking a coded location information (such as addresses) and turning it into explicit location information e.g. "Place de l'Etoile" to (48.8733575,2.2925186) latitude and longitude coordinates. Reverse geocoding works the other way around by taking coordinates data and locating the exact or nearest address, grid, etc. The process is visually described in Figure 3.2.

In fact, these services provide more than conversion between the two representations, by creating a canonic output containing both representations.

[2] https://developers.google.com/maps/documentation/geocoding/intr
[3] http://dev.virtualearth.net/REST/v1/Locations/
[4] http://cloudmade.com/documentation/geocoding/
[5] http://www.mapquestapi.com/geocoding/
[6] https://developer.yahoo.com/boss/placefinder/

Based on the schema description of input and output, we exploit these services to guide the matching to automatically identify geospatial characteristics in the datasets. The idea is to use an instance information to query a web service which will in turn return more detailed information about the instance. These information will contain richer details about the instance and more specifically more detailed geospatial information. In other words, we get more information on how the queried instance can be represented in other geospatial formats from the web service and the goal is to match the different formats together. Since we know already the format of the web service result, we try to match the web service result with the queried instance and try to figure out what each characteristic in the data means. By doing so, we will know what each column represent, hence, we will be able to represent it in other formats all because of the web service result. Here is the general plan of the approach: First we query a geocoding/reverse-geocoding web service with existing instances in order to get richer information about the given instance. Using those information we can find matching rules between the queried instances and the web service response. The schema of the web service must be known in advance, so a match between the queried instance and the web service instance will give us some information about the schema of the queried instance. This enables us to detect complex relations between two different representations by using the web service as a mediator. Data sources are mapped to the mediator at first, then by previously-known information about the structure of the mediator, we can detect the required matching rules. Due to the fact that we know how a web service is defined, we can detect n to m relations between the schemas.

To materialize this idea, we need to target three main issues. The first is how to construct queries from the given instances to the selected web service, the second is how to match the query results with the original instances to figure out an intermediate matches, the last one is how to generate the final matchings between the given sources. For this sake we came up with the AMiGO framework which consists of multiple components that tackle the mentioned problems. The AMiGO framework is composed of the three main components: web service selector and query formulator, co-occurrence matrix constructor and, finally, the matching rules generator. In short, the goal of the first component is to define the web service to be used and how its schema is structured. Then, the query formulator takes the instances from the datasets and issues a query to the webservices. Later the second component takes the web service results and construct a co-occurrence matrix that defines the statistical relation of the dataset schema and the web service schema. Finally, the final component uses the co-occurrence matrices from both datasets to conclude the mapping between them.

It is important to note that a pre-processing step precedes our approach in order to unify the structural representations in each data source and to do some filtering and/or modifications on the data. The main reason for pre-processing is that some input data may contain irrelevant information while querying leading to false results or irrelevant data. Moreover, since some columns on their own cannot provide meaningful input for a web service query, pre-processing can perform some random combination/split of columns as additional data that may improve the web service query results, e.g., combine street name with city name to get more precise results from the web service. Since automation is the whole concern of this approach, the combination is done automatically and blindly without any prior information about the dataset schema. In this approach, we transform the instance data to CSV format since it allows easier query formulation. In addition, we note that even after the unification of the format, the representation may be totally different which is the main purpose of our approach. For example, both files are in CSV format, but each represents addresses differently. Figure 3.3 shows a global view of our framework.
In the following we discuss in details each component and the steps required to reach the final matching process.

3.4 Web Service-Based Query Formulation

A web service acts as the mediator that maps the schemes of our data sources. The web service takes one input and generates one output that is a rich representation of the given query e.g. The web service takes as input the string "Place de l'étoile" then outputs in a semi-structured format rich geospatial information about the queried instance such as location in longitude and latitude pairs, street name, postal code, city, country, etc. In this approach, a well-defined geospatial web service is required since it will form the base knowledge that we map our schemes with. The idea of using a web service as a mediator could apply in different domains where several representations exist for the same entity, but its effectiveness of depends on the way

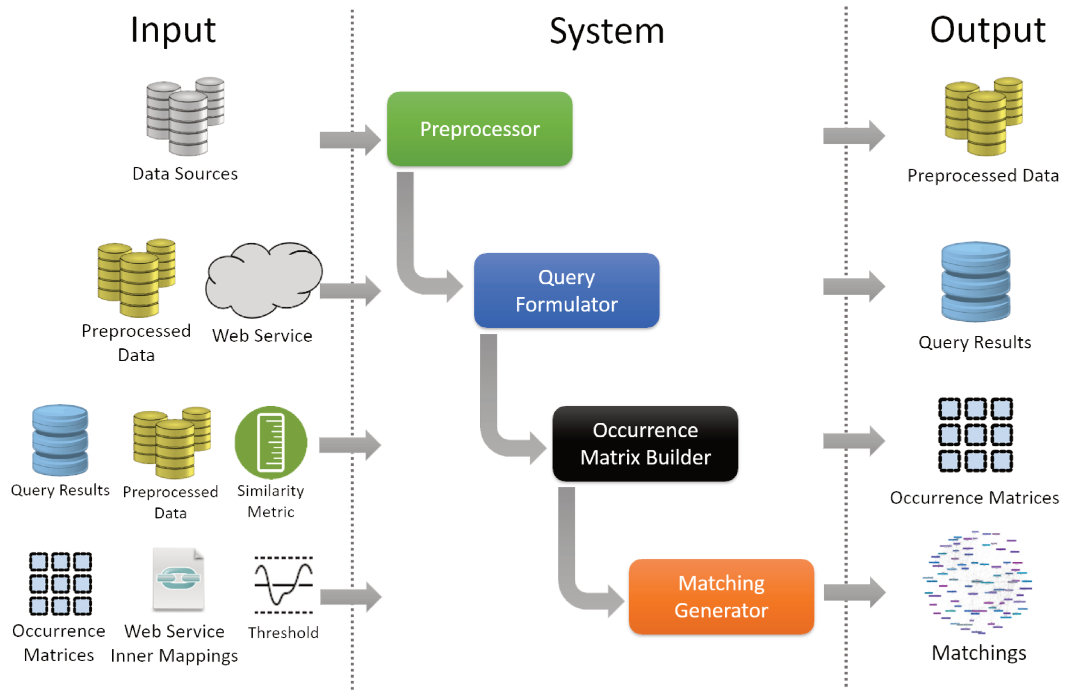


FIGURE 3.3: A system for the automatic detection of geospatial information.

it covers different types of representations. In addition to the service itself, the process requires specifying the relationships between the different representations in the web service itself. For example, if a web service contains longitude, latitude and a WKT representation of an address, we must specify how a combination of latitude and longitude can be represented in WKT and vice versa. These information are saved as *inner mapping rules* of a web service that are used later in the matching task. The main goal of the inner mapping rules is to be able to match data sources describing similar concepts but represented in different structures, scales, reference systems, etc.

After service identification, we need a component to construct and issue the queries from the given instances to the defined web service. This is the objective of the query formulator which aims to query the web service with the instances information and get information about their geospatial characteristics. Consider a simple example of an instance having an unknown property with the value "avenue des etats unis", querying the web service with this data will result in more information regarding the geospatial details of it. This includes precise location, a street name, etc. The idea is that the result will contain, somewhere in its schema, a value that matches our query, this will allow us to know the meaning of our property and how it can be represented in other geospatial formats based on the structure of the web service result.

The query formulator constructs a query for each instance, calls the web service and stores the results in order to be matched later by another component. In case the web service successfully returned a result, we will end-up with new information about the instance including various data about its geospatial characteristics (more details about the matching between the queries and the instances are found in the next component). Here comes the importance of having a web service that contains multiple geospatial representations.

More specifically, each dataset instances are queried separately. The query formulator creates a query from each instance and sends it to the web service. Separate requests are issued for each column in a row as shown in Figure 3.4 or by a random split/combination of columns - previously done in a pre-processing phase, e.g., the fourth column in Figure 3.4 is the result of pre-processing the file by combining Columns 1 and 3. Note that Col1, Col2, Col3 represent any column names while v1, v2, v3 represent any possible value. The web service results are grouped by the queried columns and stored in a repository for later tasks. Algorithm 1 shows the query formulation process.

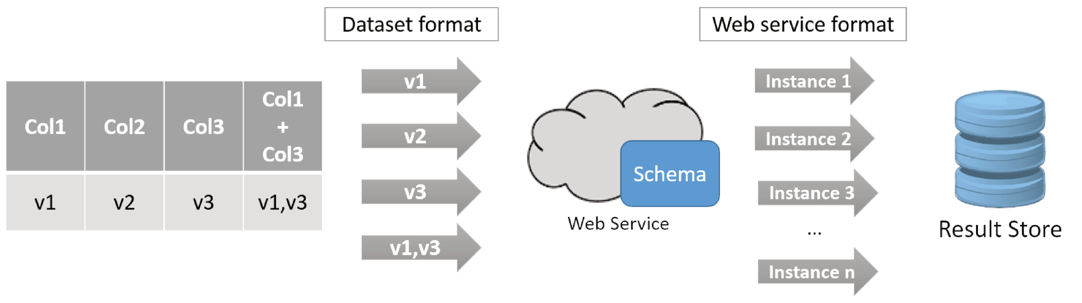


FIGURE 3.4: Querying web services to obtain instances with richer data.

On line 2 we start by creating an empty map data type for holding the results. Then we iterate over each column of each row in the data (lines 3 and 4) then we query the web service with the value of each column and store the query result in the results variable (line 6).

```
Input: Datasource D, webservice W
Result: A set of query responses containing rich information about the dataset
        instances
1 // Initialize an empty map
2 $results = empty$;
3 for $i \leftarrow 0$ to $D.nbRows$ do
4     for $j \leftarrow 0$ to $D.nbColumns$ do
5         // Query the web service with each column and add the
              response to the results map given the key as the column
              value
6         $results[j].add(W.query(D[i][j]))$;
7     end
8 end
9 return results;
```

Algorithm 1: Query formulation algorithm

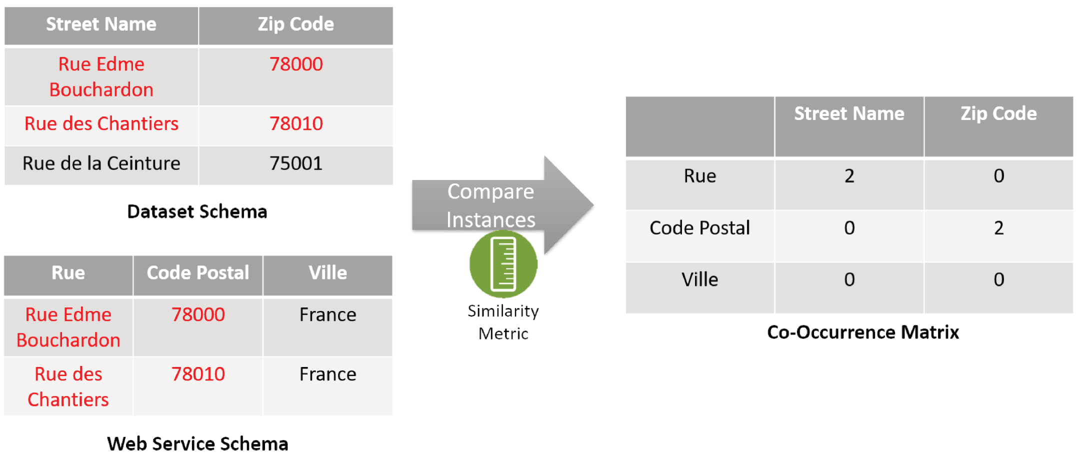
3.5 Co-Occurrence Matrix Construction

The following step is to use the web service results and the dataset instances in order to construct a co-occurrence matrix. A co-occurrence matrix signifies statistically how much elements from one schema corresponds to another.

Definition 1. *A co-occurrence matrix **M** is a matrix of n * m rows, where n and m are respectively the number of columns in the dataset and the web service schema. Each entity in this matrix corresponds to the number of times an element $a_{i\ j}$ appears at the same time in the column i of the dataset schema and the column j of the web service result schema.*

The element comparison is done via a similarity metric [16, 66] in which each time a similarity is detected, the corresponding value in the matrix is incremented by one. The higher the value, the higher the probability that these two columns map to each other. An example of what precedes is shown in Figure 3.5 with the elements in red representing common occurrences. We see two schemes, one representing a dataset schema and the second representing the web service schema. In the dataset schema, a street is represented by its name and zip-code written in English words, while it is in the web service schema represented by the set {Rue, Code Postal and Ville} that stands for {Street, Postal code and City} in French. The co-occurrence matrix lists the columns of both schemes as rows and columns of the array, and each element in the matrix represents the number of times the same value appears in row/column combination. For example, we see that the columns

"Rue" and "Street Name" have two values in common, which are "Rue Edme Bouchardon" and "Rue des Chantiers". To calculate the matrix, we first iterate over each row in the dataset and compare the value of each column with the column values of each row in the web service results. If the similarity between the values exceeds a threshold, the value at the specific row/column index in the matrix is incremented, e.g., if the value at Street Name is similar to the value at Rue, then the cell corresponding to column Street Name and row Rue is incremented.



Street Name	Zip Code
Rue Edme Bouchardon	78000
Rue des Chantiers	78010
Rue de la Ceinture	75001

Dataset Schema

Rue	Code Postal	Ville
Rue Edme Bouchardon	78000	France
Rue des Chantiers	78010	France

Web Service Schema

	Street Name	Zip Code
Rue	2	0
Code Postal	0	2
Ville	0	0

Co-Occurrence Matrix

FIGURE 3.5: Example of co-occurrence matrix calculation.

Algorithm 2 shows the co-occurrence matrix construction process. It starts by initializing an empty co-occurrence matrix with all values set to zero (line 1). Then the algorithm iterates over each value in the data source (lines 2 and 3) in order to compare it with the values obtained from the web service (line 5). The comparison is done using a user defined similarity metric that can use any technique such as string similarity or other similarity functions. If the similarity value is above the similarity threshold, then the related row/column value in the co-occurrence matrix is incremented by 1 (line 6). Finally, the algorithm terminates by returning the calculated co-occurrence matrix (line 11).

3.6 Matching Rules Generation

The calculated co-occurrence matrices capture the possible matching rules between the data sources and the web service and in turn will help with generating the matching rules between their schemas. Here, we iterate over each row and select the highest value. Then, we generate a matching between the

```
Input: Datasource - D, webservice results - results, similarity metric - SM,
       similarity metric threshold - SMThreshold
Result: Co-occurrence matrix between the data source and the web service results
1  $cooccurrenceMatrix = empty$;
2  for $i \leftarrow$ 0 to D.nbRows do
3      for $j \leftarrow$ 0 to D.nbColumns do
4          for $value in results[j]$ do
5              if $SM.compare(D[i][j], results[j]) < SMThreshold$ then
6                  $cooccurrenceMatrix[i][j] \leftarrow cooccurrenceMatrix[i][j] + 1$;
7              end
8          end
9      end
10 end
11 return $cooccurrenceMatrix$;
```

Algorithm 2: Co-Occurrence matrix construction algorithm

corresponding row/column if the number of co-occurrences is higher than some pre-defined threshold.

After having the matching between each dataset and the web service, we use the web service inner mappings to detect how elements from each dataset can be matched together. To illustrate this, let us consider two datasets, DS1, DS2, and a web service, WS. Suppose that DS1 contains the columns a1 and b1, the WS schema contains ws1,ws2 and ws3 and DS2 contains a2. Knowing that the column ws3 is the combination of ws1 and ws2, a1 and b1 map to ws1 and ws2, respectively, and a2 maps to ws3, we can conclude that we can map DS1 elements to DS2 elements by the property "a1 combined with b1 is equivalent to a2". The global picture is shown in Figure 3.6.

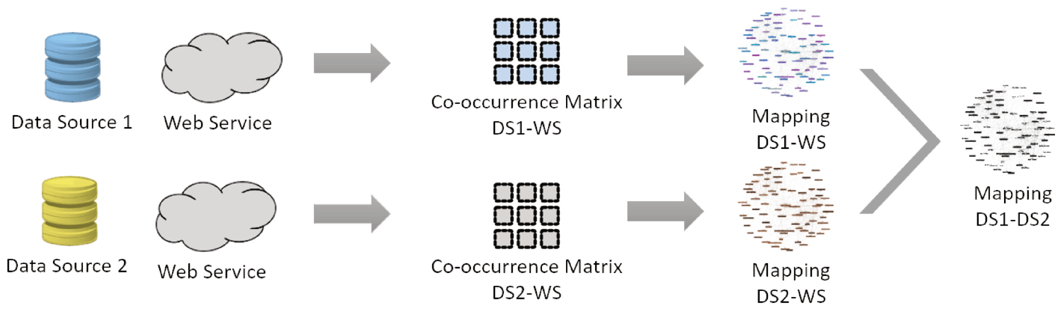


FIGURE 3.6: Discovering matching rules between datasets using a web service as a mediator. DS, dataset; WS, web service.

Algorithm 3 shows the matching rules generation process between the data source schema and the web service schema. It starts with an empty initialization of the matching array (line 1). Then it iterates over each co-occurrence matrix value (lines 2 and 3) in order to compare the value with the predefined matching threshold (line 4). If the value is higher than the threshold then the algorithm concludes a matching between the related columns of the web service and data source schemas (line 6). The algorithm terminates by returning the matches array on line 10.

```
Input: Data source - D, Web service schema - W, Co-Occurrence matrix -
       cooccurrenceMatrix, matching threshold - mThreshold
Result: A set of mappings between the data source and the and web service
        result schema
1  matches = empty;
2  for i ← 0 to cooccurrenceMatrix.nbRows do
3      for j ← 0 to cooccurrenceMatrix.nbColumns do
4          if cooccurrenceMatrix[i][j] > mThreshold then
5              // mappings is the list of matched elements between D
                  ans W
6              matches.add(D.getColumn(i),W.getColumn(j));
7          end
8      end
9  end
10 return matches;
```

Algorithm 3: Matching rules generation algorithm

Summing up, the idea is to query each dataset element with a web service that has a known schema and inner mapping rules. We then use the resulting instances to create co-occurrence matrices for each dataset. The matrices are then used to define a matching between each dataset and the web service schema. Finally, using the inner mapping rules of the web service we are able to create matching rules between the input datasets.

This process is done twice for both datasets. Using the matching rules from D1 to WS and from D2 to WS in addition to the inner mapping rules of WS, the process terminates by showing the matching between D1 and D2.

3.7 Evaluation

We evaluate our approach using two datasets representing transportation services in the Paris area, RATP and Autolib, a public transportation services group and a car sharing service respectively. The goal is to automatically match RATP and Autolib schemas.

Input data are collected from the open data portals for RATP[7] and Autolib[8] in CSV representations. The number of instances in each of the RATP and Autolib datasets is 1067 and 869, respectively. Figure 3.7 shows the original schema of the datasets.

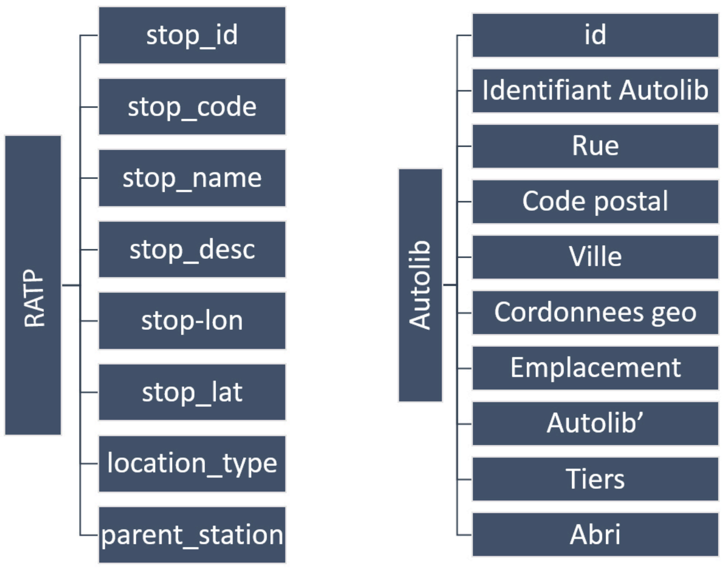


FIGURE 3.7: The original schemas of RATP and Autolib datasets.

Following the steps described in the previous section, we evaluate our approach on the input data. In a pre-processing phase, we pass the data to an automatic simple pre-processor that splits columns containing special characters (commas, semi-colons) into two or more columns named by the original column's name with an incremented value concatenated to its end. For example, in the given dataset the Autolib's column "Cordonnees geo" is split into two columns "Cordonnees geo 0" and "Cordonnees geo 1".

For the web service selection, we chose Google's geocoding web service [9] with one function on top implemented by us to filter out the results in a simple schema that consists of three columns: formatted-address (representing

[7]http://gtfs.s3.amazonaws.com/transilien-archiver_20160202_0115.zip
[8]http://opendata.paris.fr/explore/dataset/stations_et_espaces_autolib_de_la_metropole_parisienne/
[9]https://developers.google.com/maps/documentation/geocoding

a textual address representation), lng (longitude) and lat (latitude).

The query formulator queries the web service with each column's value for all of the existing rows, then groups the results by column names and saves them into a repository, as presented in the previous section. The total number of issued queries is 20,185 divided into 8536 and 11,649 for RATP and Autolib, respectively.

One co-occurrence matrix is constructed for each column ignoring columns that gave no results from the web service. The used similarity metric is the Levenshtein distance [113] in order to show how a basic similarity metric can give us good results. However, more complex metrics can be used to increase the precision of the similarity calculation. The resulting matrices for RATP and Autolib are shown in Tables 3.1 and 3.2. In order to generate the

TABLE 3.1: RATP's co-occurrence matrix.

-	Formatted_Address	Lat	Lng
stop_id	0	35	39
stop_code	0	0	0
stop_name	11	0	0
stop_desc	1	0	0
stop_lat	0	296	5
stop_lon	0	0	14
location_type	0	0	0
parent_station	0	0	0

TABLE 3.2: Autolib's co-occurrence matrix.

-	Formatted_Address	Lat	Lng
ID	0	12	0
Identifiant Autolib'	18	0	0
Rue	2	0	0
Code postal	0	0	0
Ville	27	0	0
Coordonnees geo_0	0	5876	523
Coordonnees geo_1	0	545	916
Emplacement	0	0	0
Autolib'	1397	0	0
Tiers	0	0	0
Abri	0	0	0

matching rules, we iterate over each row, get the maximum value and assign a matching between the corresponding row/column pair. Using Tables 3.1 and 3.2, we obtain the following matching rules between each of them and the web service; for RATP: (stop-id, lng), (stop-name, formatted-address), (stop-desc, formatted-address), (stop-lat, lat) and (stop-lon, lng); for Autolib: (ID, lat), (Identifiant Autolib', formatted-address), (Rue, formatted-address), (Ville, formatted-address), (Cordonnees geo-0, lat), (Cordonnees geo-1, lng) and (Autolib', formatted-address). The execution time took around 3.5 min on the given datasets, including a one-second cool-down per each ten queries to comply with the restrictions of the web service.

Analyzing the results for RATP, our system correctly obtained matching of the latitude and longitude properties. Moreover, since the stop-name and stop-dec are normally names of the corresponding area, they were detected as geospatial properties, as well. Regarding the stop id, this false positive matching rule can be solved by combining the results with some constraint-based approaches.

Regarding Autolib, the matching rules detected correct relations between the columns "rue" and "formatted-address" and the same for the "latitude" and "longitude" with "cordonnees geo 0" and "cordonees geo 1". The false positive matches were: (ville, formatted-address), (ID, lat), (Identifiant Autolib', formatted-address) and, finally, (Autolib', formatted-address). The false negatives' matching rules can also be discarded using constraint-based approaches, for example by removing matching from repeated column values or id columns, etc.

The results show a 100% precision and 80% recall for RATP and 100% precision 42% and recall for Autolib. Matching results could be improved in different ways: (i) choosing richer web services; (ii) refining the preprocessing of the output; or (iii) using alternative similarity metrics. Combining both matching rules, we can deduce the following valid rules between RATP and Autolib: "Cordonnees geo" from Autolib maps to the combination of (stop-lat,stop-lon) in SCNF; "Rue" from Autolib maps to "stop-desc" in RATP.

We tested the algorithm on other datasets to validate it. The chosen datasets are hospital locations in the U.K. and points of interests (POI) in Paris, in addition to the previous train and car stations. The idea here is that this ap-

TABLE 3.3: Evaluation of the matching algorithm.

	Precision	Recall	F-Measure
RATP [1]	1	0.8	0.88
Autolib [2]	1	0.42	0.59
Hospitals [3]	1	0.8	0.88
POI [4]	1	1	1

[1] http://gtfs.s3.amazonaws.com/transilien-archiver_20160202_0115.zip
[2] http://opendata.paris.fr/explore/dataset/stations_et_espaces_autolib_de_la_metropole_parisienne/
[3] https://data.gov.uk/dataset/hospitals
[4] http://opendata.paris.fr/explore/dataset/zones-touristiques-internationales/export/

proach can help in checking if the datasets contain geospatial information in addition to the ability to identify them and the relation to other datasets. This can be used in uses cases such as finding the nearest hospital from an accident location or finding some POIs near a hotel, etc. The results are shown in Table 3.3.

3.8 Conclusion

The problem of schema heterogeneity is essential when handling transportation data integration. Since operators use different data representation to describe their content, it is crucial to find a way of matching those different representations together. Manually executing such task is error prone and highly time consuming, especially when the amount of data to be integrated is large. Due to this, we investigated the automatic schema matching domain for a solution. However, our studies showed that existing automatic schema matching approaches fail to detect the complex characteristics in transportation schemas. More specifically, the diverse and complex representations of geospatial characteristics which play a main role in representing transportation information and units.

In this chapter we introduced an instance based approach for automatic detection and matching of geospatial information. Our approach uses web services as mediators to help guiding the matching task. In short, the web service is queried with the dataset instances and the results are then matched to the queried instances. By doing so, we are able to know what each column represents and the way to match it with the others. This enables us to discover complex n * m matching rules that can not be detected using existing techniques.

Our approach is suitable for the current open data transportation representations, since they use defacto standards for their exchange formats. However, more complex formats such as in standards requires combining our matcher with more complex ones. It is important to improve the query formulation approach in order to decrease the number of web service requests and more importantly optimize the query itself to make sure we get correct query results.

Chapter 4

Instance Level Integration of Transportation Datasets

4.1 Introduction

In order to enable multimodality, we are now aware of the need to integrate multiple transportation modes and services. The integration will allow us to form links between different services in order to provide richer and more optimized planning solutions. What is interesting about transportation data is that these links are translated to real physical links between the transportation entities. For example, when we say that there is a relation between a railway stop and a bus stop - a link from a railway stop S1 and a bus stop S2 - we mean that there is a physical path at a specific moment that connects these two stops, i.e. there is a way for the passenger to drop off at the railway station S1 then take the bus service at station S2. Therefore, we give passengers the ability to use this path as a transit service to switch from one mode to another. This switch may be very important in many cases. This combination of services may be better than using only one mode, since it allows for improvement in trip time, maybe the cost and for sure in extending trip planning possibilities and options.

In general, existing approaches tend to solve the integration problem by mapping the data they need into a unified model, then storing the unified data into a repository supported by an API e.g. Google Transit[1], STIF[2]. However, besides the huge amount of work this integration requires, existing

[1]http://maps.google.com/landing/transit/index.html
[2]http://www.stif.info

approaches still do not take into consideration the new transportation services such as car sharing, bike sharing, car pooling, etc. These services have different characteristics than the public transportation systems. They are dynamic and often do not have the notion of a fixed transportation stop. While a public transportation service has a pre-defined schedule and known geographical stops, some of the new services operate in a complete different manner. They have neither fixed stops nor pre-defined schedule and their units are dynamic (new offers from time to time). In turn, the need to handle these new characteristics in the integration task makes the integration problem more challenging than before.

Our goal is to find a simple way that identifies and links nearby transportation services to enable multimodality. We aim for a solution that enables the identification of links between the entities of transportation data sources. What is required is a homogeneous light-weighted representation of transportation links (transfer points from one stop to another) and the means to discover them in a flexible and customized manner. With this representation we can connect different types of transportation services regardless the mode or service they offer. All what transportation solution providers need to know is just how to handle these light links and use them to plan optimized multimodal trips, which is much simpler than handling heterogeneous data and maintaining them.

Enabling such solution requires access to transportation sources which can be obtained from open data [44, 11]. Open data is gaining a great deal of popularity and numerous transportation operators are using it to publish their data to the web[3,4,5]. The main cause behind publishing the data is to increase the market visibility for each service by allowing others to refer to or include their services in their solutions. Many solutions took benefit from this to enrich the data for smart cities applications by linking data from one or more sources about a specific topic, e.g. collecting data about city events or points of interest. They use linked data techniques and data interlinking tools to provide extended information relevant to both transportation and

[3]http://opendata.paris.fr/page/home/

[4]http://www.strasbourg.eu/ma-situation/professionnel/open-data/donnees/mobilite-transport-open-data

[5]http://www.uitp.org/tags/open-data

passenger profile queries [83, 20]. These techniques address equivalence detection between entities to establish links between data sources. This may help in enriching data about entities. However, this is not always enough for linking transportation data. Further complex relations are required to reflect the nature of transportation connections. Indeed, beyond equivalence or sameAs links as in existing approaches, we are interested in finding other types of links between transportation data sources based on the geospatial characteristics of the data which capture, for example, the reachability between different transportation networks. Furthermore, using the given tools we face two main limitations. The first is the limitation in defining the linking rule that triggers link generation between entities. Existing approaches allow rule creation by a simple composition of a set of predefined functions. For an instance, to calculate information such as the closeness of two transportation points of transfer (bus stop, train station, etc), we are bounded to use some predefined geospatial distance metrics which may be infeasible in transportation networks due to infrastructure constraints and network status. As an example, we may define a rule that measures the geometric distance between two stops and assign a link if the distance is feasible for walking. However, due to the network infrastructure, stops maybe very close in geometry but very far according to a road network e.g. two stops on the opposite banks of a river are very near, however unreachable on foot. The second limitation is the representation of the generated link between entities. Supporting complex relations requires more complex link representations. As an example, let us suppose that a link is established between two transportation points of transfer. Existing tools can provide the link *BusStop1 nextTo TrainStation132* which does not provide enough information about this relation. They are next to each others but how close are they? and what are the modes of transportation that we can use? How much will it cost? etc.

Based on what precedes, there is no way of creating links that are suitable to many complex interlinking tasks other than the standard equivalence interlinking tools. Our main goal is to provide a system that is flexible and rich enough to allow users to define their own way of linking data sources. Users must be given the power to use rich interlinking rules and define any form of output needed in their tasks.

In this chapter we introduce our framework for enabling richer and customizable links for Open Data instances. These links can be applied to fill the connections gap between transportation operators and services. With this approach, we can generate rich semantic links between entities published as open data in the sense of improving re-usability and reducing the need for re-calculation. We evaluate our approach using a real use case on connecting two transportation modes and checking how this affects the time of trips. We also extend an existing trip planning algorithm called CSA which exploits the resulting inter-connected transportation network, mixing scheduled (e.g., trains) and unscheduled (e.g., bikes) modes.

4.2 Problem Statement

In order to enable multimodality and allow more advanced and customized usage of transportation services, it is required to integrate data from multiple transportation operators. By transportation data integration we mean forming links between different transportation entities such as transportation stop points. These links will extend the view of the transportation network and the various services present in each area. This integration will allow better trip planning and monitoring of the overall transportation network. For example, planning trips using one transportation service will limit the trip plans to the scope of the covered area, therefore, limiting passengers to reach their desired destinations efficiently. By increasing the number of services and combining them together, passengers will have the power to form the best trips that suit their needs.

We aim at using the recently popular open data concept [11] and the published transportation datasets to provide the view we imagine. However, this integration is not simple due to data heterogeneity between the datasets. Transportation datasets represent their information in different formats, making it difficult to easily detect relations between different instances.

Link discovery [93] techniques are approaches that target discovering links between open data sources. They work by first taking a linking rule as an input then applying it on different datasets instances in order to generate the required links [108, 75, 56, 49, 94]. However, the problem is that the scope of existing link discovery techniques is targeted for equivalence detection, that is, detecting similar instances represented differently across datasets e.g. detect an event that is represented differently in two different datasets. Therefore, the supported link identification rules and the final representation of the output are suitable for equivalence detection and thus are simple and not suited to transportation data integration. This is because a transportation link is more complex than a simple equivalence link. A transportation link represents transportation/travel information between transportation entities e.g. the reachability between one station and another. As an example, consider two data sources representing a bus and a train service operators. Using the current tools, the only way to link both sources is to define a rule that calculates the geometric distance between the entities and generate a link if the distance is feasible for walking. By this, we say that we can use the other stop as a transit stop in the trip. The generate link is a simple link that may be represented as *BusStop1 nextTo TrainStation132.*, which is very simple and does not show any information such as the required walking time for example. In addition, the defined linking rule is not sufficient to reflect the existence of a transportation connection between the stops. The reason is that a close geometric distance does not mean that the stops are reachable simply due to the nonexistence of a path according to the transportation infrastructure (e.g. road network).

The problem we target is how to extend the scope of interlinking tools to go beyond equivalence detection. The goal is the following: Given two transportation data sources each containing a number of instances representing real physical stops or transportation units, find links between these datasets' instances that reflect possible relations between them from a transportation point of view. The links must be suitable to represent the necessary information describing the nature of the connection from transportation point of view.

4.3 Resource Description Framework

Before going in details into our model, we have first to understand one important model used to represent entities and relations in the open data concept. This model is known as the Resource Description Framework (RDF). RDF is based on a model representing triples made of resource, property and

value. A resource is an entity that is accessible by an URI on the Web. A resource is described by RDF statements (set of triples). A property defines a binary relation between resources and/or atomic values. By properties we are able to attach information or descriptions to resources. A value can be either a simple character string or a resource. An RDF statement specifies a value for a property of a resource. RDF has an XML syntax and can be seen as an object-oriented formalism for meta-data statements. These meta-data can rely on common ontologies represented using RDF Schema (RDFS). RDF statements can be considered as triples (resource, property, value). The vocabulary used in these triples can be defined using RDFS, by a hierarchy of classes and a hierarchy of properties. Contrary to object-oriented or frame-based representations, RDF relies on a property-centric approach. Anyone can define properties about Web resources, in order to offer descriptions for these resources [22].
Through the chapter, we are going to use RDF as our base model for our data representation with the problem of detecting and representing complex relations between them.

4.4 Model Definition

As indicated in Chapter 2, discovering links between transportation points of transfer can not be done using existing interlinking tools. A more complex link discovery and generation process is needed to enable richer and more flexible link representation. Therefore, in order to cope with the limitations of existing approaches, we introduce a new model that is able to form a rich and flexible solution for creating transportation data links. Our model consists of the following concepts: a link that is the connection between the dataset instances, a linking rule that represents when to create a connection and finally an output pattern that is the final representation of the generated connection. These concepts serve a unique task named the Linking Task which we will see later. In the following, we define in details the different concepts and their role in the overall approach.

4.4.1 Link Definition

A link is a relationship between two instances in two different datasets. This relation goes beyond the sameAs relationship, by allowing wider semantics and assigning richer information to the generated link. A link is represented by a relation name and relation attributes. A relation name represents the semantics or the concept behind a specific link instance e.g. reachable-from, walkable-from, next-to. It can be used for better understanding of the link type between entities. On the other hand, the relation attributes can be considered as a meta-data to a specific link. They describe the characteristics of a link and the details required to use it. For example, we consider the link "reachable-from" between a bus station A and a train station B that has the following attributes: "duration": "10 minutes", "cost": "4 euros", "departure-time": "14:00". The relation name "reachable-from" indicates the semantic meaning of the relation which represents the reachability of one station from another. This may be very important for later understanding of relationships and their meanings. The attributes "duration", "cost" and "departure-time" give more insights about the link instances and more details on how it can be used. This is very helpful for later post-processing in trip-planning applications for example.

Definition 2. *We say that there is a link/relation between two entities e_1 and e_2 of two datasets D1 and D2, if there exist a link l_i such that:*
$l_i = (e_1, e_2, n, p_i) | e_1 \in D1, e_2 \in D2, n \in String, p_i \in P$
Where, e_1 and e_2 are instances, n is the relation name, and p_i is a set of relation attributes.

Definition 3. *A set of relation attributes is a set of key/value pairs corresponding to a given link l_i to describe the characteristics of the link.*
$p_i = [(k, v) | k \in String, v \in Any]$
Where, k is the attribute name and v is the attribute value.

4.4.2 Linking Rule

A linking rule specifies the conditions required to generate a link between a pair of instances. It is a combination of boolean expressions packed in a way to describe a specific condition that decides whether to form a link between a pair of entities or not. An example of a rule: If the distance between Stop A and Stop B is less than 2 kilo-meters and there are no road works, then form a link between Stops A and B. The main goal is to apply this rule to each instance pair and generate a link if the rule holds.

A linking rule is defined by a chain of functions that can either be preprocessing or metric functions. Preprocessing functions are simple functions that alter an input to reformat or convert them into a specific format e.g. convert letters from lowercase to uppercase. A metric is a function that evaluates the relationship between instances. It can be mathematical, linguistic or of any other form that returns a real value (from the range zero to one, where zero indicates no similarity while one indicate high similarity) e.g. calculate the walking distance between stops A and B and compare it to a predefined threshold.

4.4.3 Output Pattern

An output pattern is the answer to how the final links are represented and what are the properties they should hold. Therefore, it is important to be precise when defining an output pattern. A pattern specifies the structure of the generated links and the required information they must contain. In other words, it represents a template that will be filled when a link is instantiated.

An output pattern is composed of a set of properties, where each property is defined by a function that calculates it. Function parameters can be the inputs from the data sources or predefined by the rule composer e.g. The output link should contain the distance attribute that is calculated by the walkingDistance function given the parameters l1 from the entity A and l2 from the entity B. An output pattern is freely chosen by a user according to the interlinking task and the post-processing needs.

Definition 4. *Given D_1 and D_2 two data sources with instances e_i, e_j respectively. We define* Pr *as the set of properties that describe the relation between e_i and e_j where each property is represented by a property name n and a corresponding function f from a set of functions F. The functions f is represented by its name and the set of parameters it needs.*
$Pr = \{(n,f) \mid n \in String,\ f \in F\}$
Definition 5. *An output pattern is the set of these properties between a pair of instances. Therefore it is represented as the following:*
$O = (d_1,d_2,pr) \mid d_1 \in D_1,\ d_2 \in D_2,\ pr \subseteq Pr$

4.4.4 Transportation Link

We have introduced the general definition of a link. Here we move to a more specific type of links that is the transportation link our main focus in this study. A transportation link can be described as an accessible path from one transportation point of transfer to another. A point of transfer is any stop that allows users to change a transportation unit or mode. A transportation link contains properties describing both the departure and arrival stops in addition to other properties. There are two types of transportation links: a scheduled timetable link and a non-scheduled link.

- Scheduled transportation links have specific departure and arrival times defined by the network operator. As an example they can have the following properties: departure-time, arrival-time, departure-stop and arrival-stop.
- Non scheduled transportation links are other links that have no schedule information and for which availability is not restricted by timing constraints. A an example these links have the following properties: departure-stop, arrival-stop and distance.

To better define our transportation links for being used in open data. We we propose a new ontology named the "Multimodal Transportation Link ontology" compliant with the definition above. Our ontology extends existing ones [24] by adding more properties to the link. It is mainly focused on the connection between transportation datasets, taking into consideration the temporal and geospatial aspects. Figure 4.1 shows a summarized graphical view of the ontology (we omit the data properties of classes for space purpose, e.g. time). Each link must specify two nodes (source and destination node) as well as a set of properties which explicit its semantics. A node represents a stop which is a geospatial entity that can be linked to larger pre-exiting ontologies for further description. The multimodal transportation link ontology includes the following properties: transportation mode, distance between the nodes, required cost, accessibility and the availability of the connection (represented by the schedule). The distance and the transportation mode are important for analysis purposes. The cost can be used as a filter for users queries. The accessibility describes the services available for handicap or people require special care. And finally, the availability describes the schedule (day and time) a link is available on, which can be

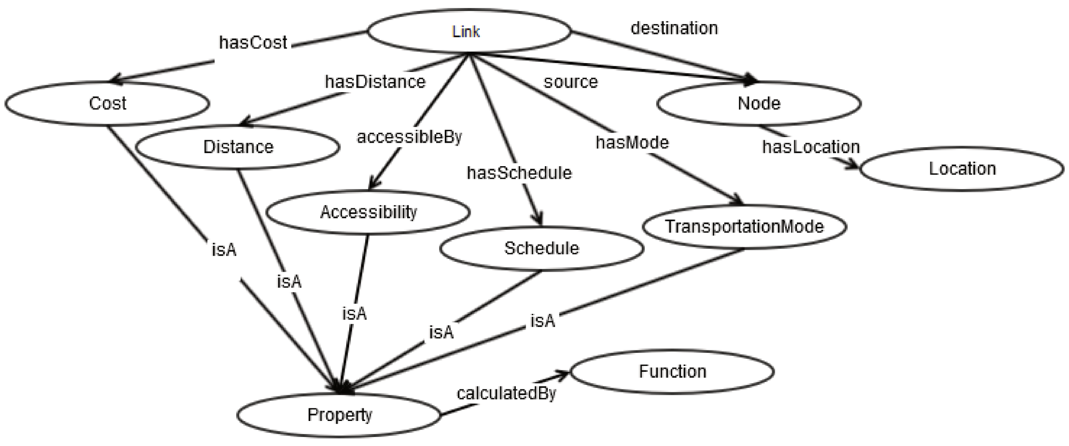


FIGURE 4.1: Multimodal Transportation Link Ontology

analyzed given the transportation systems' schedules. The defined properties are expandable where defining new properties requires the function that specifies how the property can be calculated. A function represents an algorithm, a service, or just a data value taken from the data source. For example, you can add a new property just by specifying how this property is calculated and where to get the inputs from.

4.5 Link++ Framework

A linking task is a process of creating links between instances of two different datasets. The linking task takes as an input two datasets and outputs a set of custom defined links between their entities. The way of how these links are discovered is specified in the linking rule we have seen above while the final representation of the generated link is specified in the output pattern.

Definition 6. *A linking task is defined as a tuple that contains the datasets, the linking rule, the output pattern and the functions required to form the rule and instantiate the link.*
$T = (D_1, D_2, R, O, F)$.

The above model extends existing approaches by a new definition of links and the way to discover and represent them. The model's components form together a powerful interlinking architecture we named Link++. Link++ enables rich and customized way of discovering an generating links for transportation datasets. The framework is introduced in figure 4.2 and shows how the components are grouped into two main phases: the link design phase and the link generation phase (in the rule execution engine).

- The link design phase is the first phase in the link discovery process. It allows users define the data sources, the output patterns, the required functions and linking rules.
- The link generation phase starts at the execution time such that: 1) the configurations are taken and applied to the datasets 2) the rule is applied to the entities 3) and when valid, a connection is created and stored in a repository.

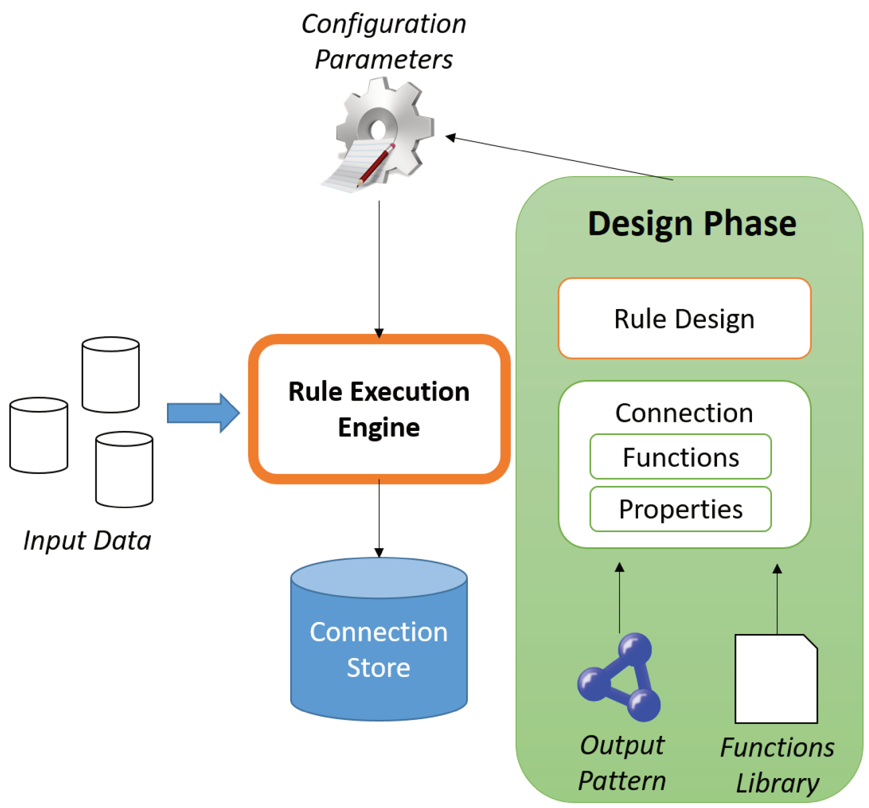


FIGURE 4.2: Link++: A framework for flexible and customizable connection generation.

The following sections explain in detail the interlinking process and the components interaction within the framework. We begin with the design phase that is responsible for designing required functions, linking rule and output pattern. Later we move to the link generation phase where we introduce the connection discovery algorithm.

4.6 Design Phase

Enabling users to define their own functions is crucial for a complete system that supports all required matching tasks. Thus, the first task is to enable users to write any functions to be used in their linking rules or similarity calculations. A user can simply do it with Link++ by implementing his own code, using external libraries or a combination of both.

The user specified functions play an important goal in the matching tasks since they can form a linking rule, a metric, a transformation/preprocessing operations or any other function based on users' needs. The functions are gathered in a file accompanied by the used libraries, then they are compiled at run time and used when needed. To reference each method the user simply address the function as follows: "class-name"."method-name". For example: MyClass.walkingDistance.

4.7 Defining a Linking Rule

As mentioned in the model definition, the linking rule states whether the system should create a link between a pair of entities or not. To recall, the rule can be visualized as a chain of functions that returns a boolean expression stating whether a link should be formed or not between two entities. As an example: consider an interlinking task of linking entities from two transportation stations the first being a bus station and the second being a metro station. The rule can be defined as a travel distance function that calculates how much does it take to go from one station to the other. However, since data is collected from different datasets we may use some pre-processing functions to convert or combine some values from one format to another e.g. transform a combined value f latitude/longitude into separate variables.

In Link++, we define the structure of interlinking rule as a Data Type Definition (DTD) that is shown below. The rule is the root node that has either an aggregation or a comparison function. An aggregation is composed of another aggregation function (to allow chaining) or a comparison function. A comparison function takes its parameters either from a property or another transformation operator. A property is a value directly taken from the dataset or specified by the user. It has a name, a data source and a value. The transformation function is either a function applied on a property or a function applied over other transformation functions. To define a rule, the user needs to define a file that conforms with the specified DTD. A real example on how a linking rule is defined is shown in the evaluation section in Figure 4.3.

```
<!DOCTYPE rule [
<!ELEMENT rule (aggregation|comparison)?>
<!ELEMENT aggregation
  (aggregation|comparison)+>
<!ELEMENT comparison
  (property|transformation)*>
<!ELEMENT transformation
  (property|transformation)*>
<!ELEMENT property (#PCDATA)>
<!ATTLIST aggregation function
  CDATA #REQUIRED>
<!ATTLIST aggregation threshold
  CDATA #REQUIRED>
<!ATTLIST comparison function
  CDATA #REQUIRED>
<!ATTLIST comparison threshold
  CDATA #REQUIRED>
<!ATTLIST transformation function
  CDATA #REQUIRED>
<!ATTLIST property name
  CDATA #REQUIRED>
<!ATTLIST property datasource
  CDATA #REQUIRED>
<!ATTLIST property value CDATA #IMPLIED>
]>
```

4.8 Defining an Output Pattern

To recall, the output pattern defines the final representation of the the link discovery output and the properties each link should contain. To define an output pattern, the user must define all the properties a link must have and the functions required to calculate their values.

In Link++, the structure of an output pattern is defined in a DTD that is shown below. The output pattern is a set of links where each link is a set of properties. A property is function that has a name and parameters. Function's parameters have a name which is their name in the dataset, the data source they are taken from and the corresponding value that is either specified as a constant or filled later after the link is instantiated. In the evaluation section we will see a real example of how to define an output pattern and the instantiated links at the end of the discovery process (Figure 4.4).

```
<!DOCTYPE output-pattern [
<!ELEMENT link (properties?)>
<!ELEMENT properties (property*)>
<!ELEMENT property (function)>
<!ATTLIST property name CDATA #REQUIRED>
<!ELEMENT function (params*)>
<!ATTLIST function name CDATA #REQUIRED>
<!ELEMENT params (param*)>
<!ELEMENT param (#PCDATA)>
<!ATTLIST param name CDATA #REQUIRED>
<!ATTLIST param value CDATA #IMPLIED>
<!ATTLIST param datasource CDATA #REQUIRED>
]>
```

4.9 Link Discovery Process

Once the design phase is completed, the link discovery process starts as described in the sequel. Algorithm 4 represents the pseudo-code of the implemented linking process. The algorithm iterates over each pair of entities (lines 2 and 4) in the two data sources and evaluates the linking rule between them (line 6). Based on the rule evaluation, the algorithm decides if a link must be created or not. If a rule is triggered, a new link is generated by evaluating the output pattern and applying the corresponding function of each property (line 8). The values are calculated by the specified functions in the output pattern, and their parameters are filled from the currently-compared entities (lines 10-14). Here, we instantiate the connection and fill in its information from the return values of the functions. The link is stored in a specified repository (line 16), and the algorithm continues on the remaining pairs until all are treated.

```
Data: D1, D2, O, R, F
Result: Discover the list of link and add them to the connections store
 1 /* iterate over the elements of D1                                  */
 2 foreach e1 in D1 do
 3     /* iterate over the elements of D2                              */
 4     foreach e2 in D2 do
 5         /* evaluate the linking rule                                */
 6         if evaluateRule(e1, e2, R) is true then
 7             /* if the rule holds, create new connection based on the
                  output pattern                                       */
 8             c ← createLink(e1, e2, O);
 9             /* calculate the value of each property in the pattern
                  based on the specified function                      */
10             foreach p in c.properties do
11                 f ← F.getFunction(p.getFunction);
12                 value ← f.calculate(p.getProperties);
13                 c.addProperty(p.name, value);
14             end
15             /* the link is instantiated and ready to be added to the
                  connection store                                     */
16             add c to connections store;
17         end
18     end
19 end
```

Algorithm 4: Link discovery algorithm.

4.10 Algorithm Complexity

In the worst cases, the time complexity of the algorithm is O(n * m * k) where n and m are the sizes of the input datasets and k is the complexity of the user's custom function, which is constant, and Since k is constant, the cost is equivalent to O(n * m). The storage complexity (in terms of data pages) is the same as a nested loop join in databases that is equal to the size of the smallest dataset in addition to one page, which usually fits in memory [80]. This complexity may be reduced by using some pre-filtering techniques that the system may offer in a future version; for instance, using a spatial index to replace the inner loop by a search in an index, which reduces the cost to O(m * Log(n)). Then, the specific rules and function defined by the user will be applied in a refinement phase automatically by the system.

4.11 Experimentation

Link++ framework is implemented as a JAVA project that can be downloaded and executed from the GitHub link [6]. To recall, in transportation networks, a link can be described as an accessible path from one transportation point of transfer to another. It contains properties describing both the departure and arrival stops in addition to other properties. The goal in this evaluation is to integrate, using our framework, two different transportation network by creating rich links between them.

As in the previous chapter, we evaluate the approach using the datasets from RATP[7] and Autolib[8] companies in GTFS and CSV formats respectively. The number of instances in each of dataset is 1067 for RATP and 869 for Autolib.

The goal is to create links between every stop in the transportation network. To do so, we need to create links between RATP and Autolib as different systems in addition to internal links between the stops of each system (RATP and Autolib). The internal links are required to build a global repository of links that contains every connection between all the stops in the datasets. In the following, we describe the evaluation phases from preparing the data, configuring the required parameters and visualizing the generated output.

4.11.1 Data Preparation

Since RATP's data is described in timetables, we can extract links between RATP stops from the given timetable information. The first task is to reformat the data from timetable representation into link representation. To this

[6] https://github.com/alimasri/link-plus-plus.git
[7] http://gtfs.s3.amazonaws.com/transilien-archiver_20160202_0115.zip
[8] http://opendata.paris.fr/explore/dataset/stations_et_espaces_autolib_de_la_metropole_parisienne/

end, we have proposed an algorithm that transforms timetable data from GTFS files into scheduled links. The algorithm iterates over the timetable information for each stop and creates a link that starts from a departure stop at a departure time and ends with an arrival stop with the specified time.

In case of Autolib, we do not have timetable information, so we need a way to discover the links between its stops. Using our approach, we can match Autolib's dataset with itself (in order to know when a Autolib station is reachable from an another) to discover these unscheduled connections between. Since the configuration task is common and independent, the following section describes how to use our approach to discover the unscheduled connections for Autolib-Autolib and Autolib-RATP.

Two tasks are required one for Autolib-Autolib connections and one for Autolib-RATP connections. In this example, unscheduled links are driving or walking links between Autolib-Autolib and Autolib-RATP, respectively. We use our approach to search for links that match a predefined criteria. Since our approach works on RDF data, we have used the DataLift [95] platform to transform both RATP stops and Autolib CSV files into RDF turtle formats. In the sequel, we describe in detail all of the required tasks to achieve the needed integration.

4.11.2 Defining custom functions

Our system is flexible as it allows users to create any custom function or external library to be used in the linking task. This is important to extend the existing interlinking solutions and allow better interlinking options. In our example, we define the functions getWalkingDistance, getWalkingTime, getDrivingDistance and getDrivingTime. In a real scenario, we get this information from a web service, such as Google's distance matrix API[9]. However, due to the query limit, we have chosen to implement them by local functions based on mathematical calculations[10].

[9] https://developers.google.com/maps/documentation/distance-matrix/
[10] http://www.movable-type.co.uk/scripts/latlong.html

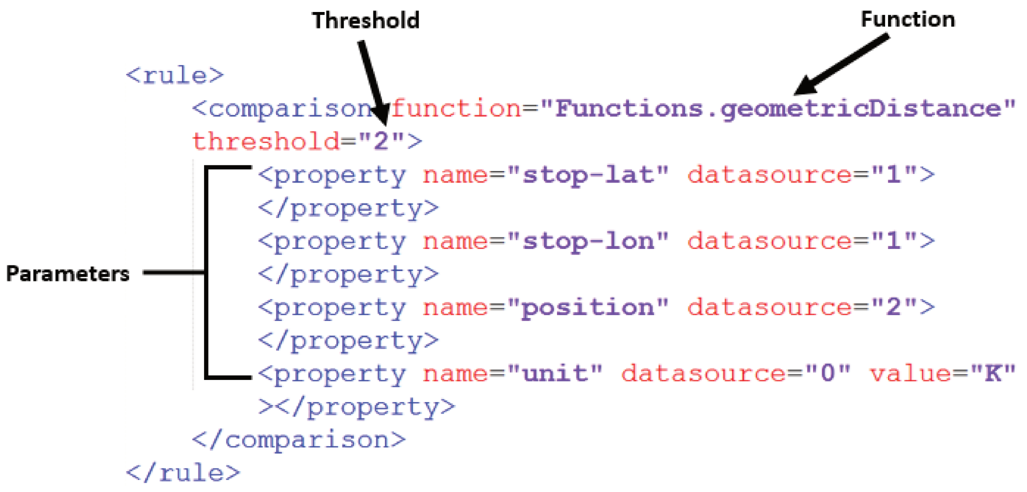
4.11.3 Defining the linking rule

Recall that the linking rule describes the condition that triggers the creation of a connection. Two rules are required, one for Autolib-Autolib and the other for Autolib-RATP. For the first one, the condition of the defined rule is the following: ″If a driving path exists within 200 km (the time before the battery is totally discharged), create a connection″. For Autolib-RATP connections, the rule is: ″If a walking path exists from one stop to another within one kilometer, create a connection″.

Rules are written in XML format in compliance with the DTD we have seen in the model definition. The functions that calculate the walking distance and time are referenced from the custom functions file. The parameters of the max driving and walking distances are customized by the user responsible of the configuration and according to his/her preferences. In this example we have chosen ″200 km″ and ″1 km″ for driving and walking time respectively. We set these parameters as the maximum feasible scope for a person to ride the car or walk from one station to another. Figure 4.3 shows an example of how a rule can be defined. We see that the rule starts with a comparison operator being the root. The comparison operator is a custom function called geometricDistance. The threshold of the comparison function is specified by the value 2. This means that the rule is valid if the returned value from the function is less than 2. geometricDistance takes five parameters. The first two parameters represent the latitude and longitude information from the first data source. The third parameter is the position parameter from the second data source. Finally, the last parameter represents returned unit of comparison which is chosen here in Kilometers.

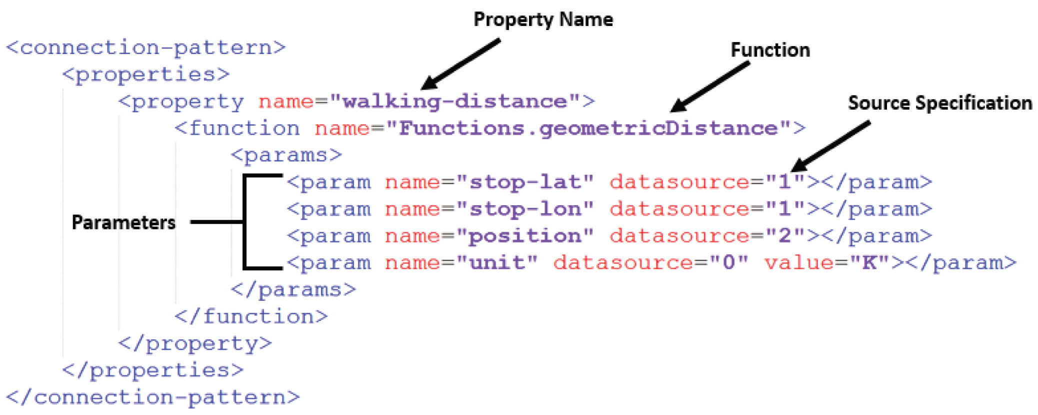
4.11.4 Defining the connection pattern

We define the output generated by the system at each valid rule. We have chosen the following properties to be represented in a connection pattern: source-id, target-id, walking/driving distance and walking/driving time. This pattern is the same for both tasks, and an example is shown in Figure 4.4.



```
<rule>
    <comparison function="Functions.geometricDistance"
    threshold="2">
        <property name="stop-lat" datasource="1">
        </property>
        <property name="stop-lon" datasource="1">
        </property>
        <property name="position" datasource="2">
        </property>
        <property name="unit" datasource="0" value="K"
        ></property>
    </comparison>
</rule>
```

FIGURE 4.3: An example of rule definition in XML



```
<connection-pattern>
    <properties>
        <property name="walking-distance">
            <function name="Functions.geometricDistance">
                <params>
                    <param name="stop-lat" datasource="1"></param>
                    <param name="stop-lon" datasource="1"></param>
                    <param name="position" datasource="2"></param>
                    <param name="unit" datasource="0" value="K"></param>
                </params>
            </function>
        </property>
    </properties>
</connection-pattern>
```

FIGURE 4.4: An example of a connection pattern in XML

4.11.5 Link Generation

After the definition phase, the data and the definitions are taken by the system to start the link generation process. Due to the flexibility in designing the linking rule and the output pattern, our approach succeeded in enriching the transportation network by generating new links between the transportation stops. The final results were a total discovery of 535,966 internal links between Autolib car stations and 272 new links between the two different transportation modes RATP and Autolib. These new links are very important and can be used to bridge the missing links between the two separate transportation modes. Figure 4.5 shows a sample of the generated links between the two stops. The link and its properties are shown in Grey color. It shows some information regarding the relation between the two stops that can be very useful for post-processing. We notice how our link is much richer

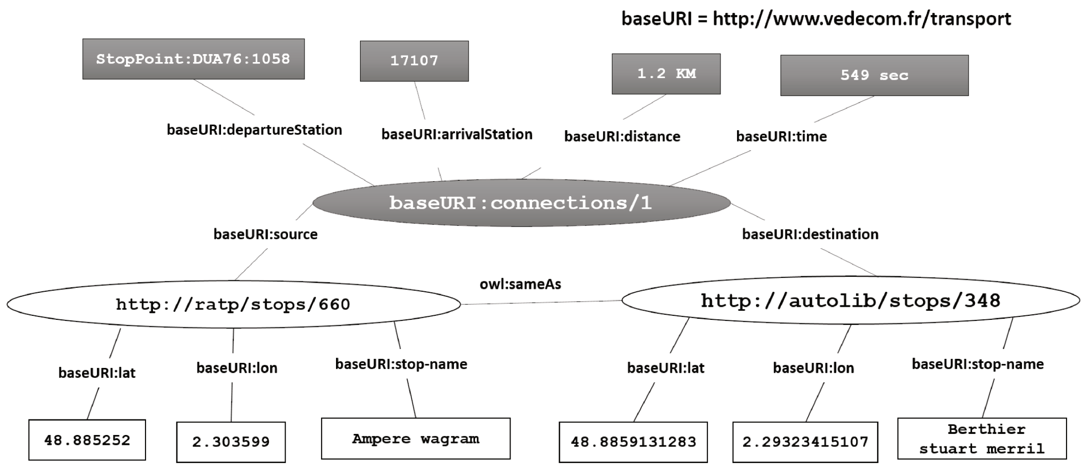


FIGURE 4.5: An example of a transportation link between two entities

than the shown owl:sameAs link in the figure.

In the next section we will see how we can make use of the generated links in a routing algorithm. We will see how the new links improved the travel time while using the integrated network instead of using each system separately.

4.12 Using the Discovered Links

We will illustrate hereafter how to exploit the generated links to calculate the earliest arrival time (EAT) [28]. EAT is the earliest time we can reach all stops in a transportation network from a given departure stop and time. We have chosen this approach to get a broad view on how the newly-introduced connections can massively affect a large network. We have used the connection scan algorithm (CSA) [28] as an EAT implementation, since it matches with our notion of transportation link.

4.12.1 Connection Scan Algorithm

In short, CSA works by receiving a stream of connections ordered by departure time and chooses the fastest way to reach one stop from another. Due to the fact that the connections are pre-sorted and can be accessed one by one in a single iteration, CSA is faster and more scalable than other existing algorithms. For more details about CSA kindly refer to Chapter 2 route planning section.

However, CSA has some limitations in our case. Firstly, it only supports timetable networks, which makes it unable to compute trips, including other services. Secondly, it does not support unscheduled connections. It only supports one footpath transition between two points of transfers. It is therefore not possible to combine scheduled connections, unscheduled connections and footpaths to create a more optimized trip.

4.12.2 CSA Extension

CSA handles only public transportation networks and footpaths. In order to support multimodality, we have introduced unscheduled links beside the ones based on timetables. We have also enabled multiple unscheduled links between multiple points of transfer. The unscheduled links are created when a link is reached. For each iteration, all of the available unscheduled links from an arrival stop are checked to create scheduled links by setting the departure time to be equal to the arrival time at the station; to this is added the minimum transfer duration and the arrival time for the unscheduled link.
The process is shown in Algorithm 5. In line 1 the algorithm sets up the labels variable which will contain the final results. It is an object that contains the stop id and its respective estimated arrival time (eta) starting from the departure stop. It is initialized as an empty array (line 1). This array is filled with the first loop on line 2. This loops iterates over the stops and sets up their eta to infinity except the initial departure stop which is equal to the departure time subtracted from the minimum change time at the stop. The minimum change time is the time needed to take the corresponding train/bus at the stop(more in [28]). After that the algorithm generates the initial set of scheduled links from the unscheduled ones. This is done on line 13 where the generateScheduedLinks function takes the departureStop, lists all the nearby unscheduled links, then generate scheduled links out of them. The generated scheduled link has the departure time equals to the arrival time to the stop. On line 14 all the generated links are added to the beginning of existing links collection (since they have the closest departure time). On line 15, the algorithms starts iterating over the links. It checks if the link is reachable or not (line 16). In short it checks whether we can reach the link starting from the departure time plus the link's trip duration (For more details refer to [28]).

In case the link is reachable, we compare the eta of the existing link with the existing eta of the arrival stop (line 19). If the eta is improved we replace the existing label with the new one (line 23). After that, since we are sure that we will use this link to reach the arrival stop, we generate the scheduled links from arrival stop starting from the eta (lines 25 and 25). The algorithm continues with the rest of the links and finally returns the labels array that contains the final results (line 28).

```
Data: departureStop, departureTime, stops, links
Result: A list of the estimated arrival time (eta) from an origin stop to all the
        other stops in the network
1  labels ← empty;
2  foreach stop in Stops do
3      label ← new label;
4      label.stop = stop;
5      if stop == departureStop then
6          label.eta ← departureTime - stop.changeTime
7      end
8      else
9          label.eta ← Infinity;
10     end
11     labels.add(label);
12 end
13 nearbyLinks ← generateScheduledLinks(departureStop);
14 links.add(nearByLinks);
15 for link in links do
16     if isReachable(link) then
17         arrivalStop ← link.arrivalStop;
18         eta = labels.get(arrivalStop).eta;
19         if connection.eta < eta then
20             label ← new label;
21             label.eta ← eta;
22             label.stopId ← connection.departureStop;
23             labels.add(label);
24             nearbyLinks ← generateScheduledLinks(departureStop);
25             links.add(nearbyLinks);
26         end
27     end
28 end
29 return labels;
```

Algorithm 5: Extended CSA Algorithm

4.12.3 Evaluation

We fed our new algorithm with both scheduled and unscheduled links and tested the estimated arrival time for each stop. To check the effects of introducing generated links, we have calculated the estimated arrival times with and without them, and we have compared the results. Figure 4.6 represents the estimated arrival time for every stop starting from the RATP departure stop "Sainte-Colombe Septveilles". The intuition is that the lower the value, the earlier a passenger can reach a stop point starting from a departure station. Figure 4.6 shows the estimated arrival time from the departure stop

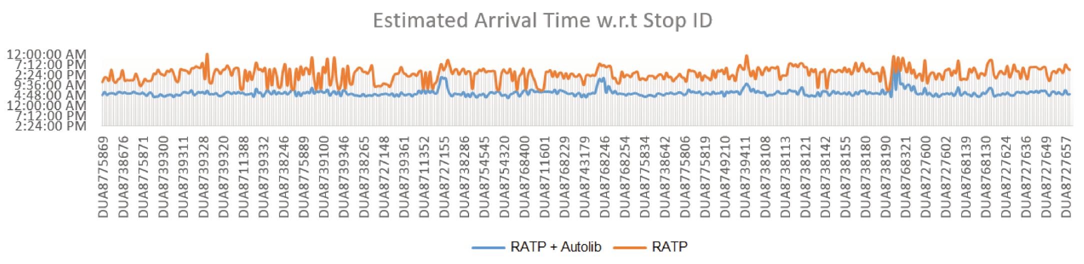


FIGURE 4.6: The estimated arrival time for each stop with and without our discovered connections.

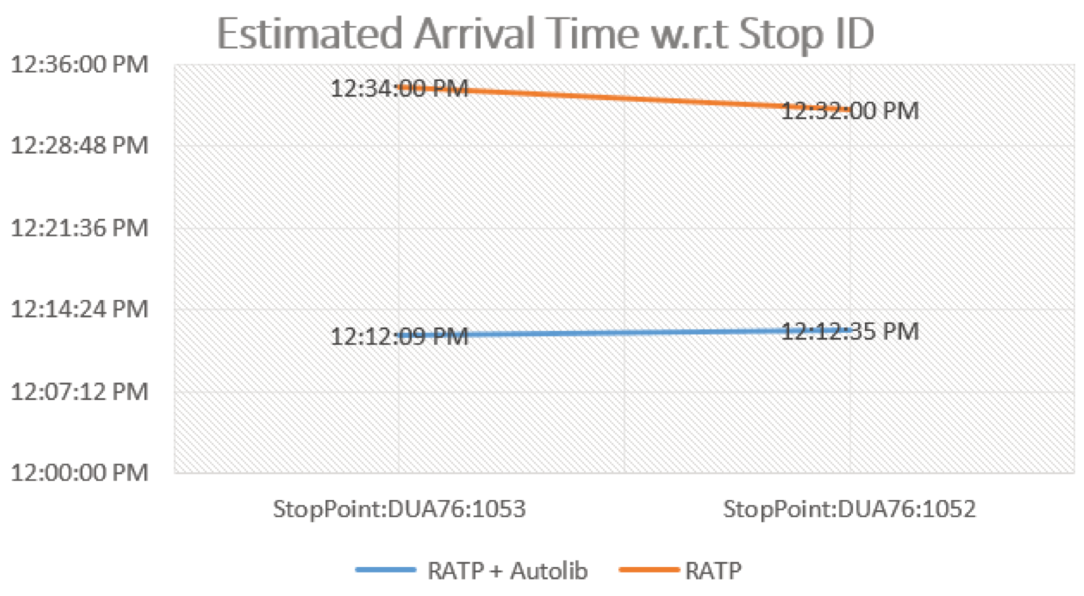


FIGURE 4.7: A Zoomed-in version of the evaluation figure

"Sainte-Colombe Septveilles"to all the other stops in the network. The orange plot represents the time using only RATP connections, while the blue one shows the time when both networks are integrated. Analyzing the figure shows that using the generated connections and integrating them in the transportation network can reduce the estimated arrival time. The zoomed-in version in Figure 4.7 gives a clearer view of the amount of time gained for reaching two different stops after using the integrated approach. On average around 25 minutes are won using the integrated approach. On the other hand, we notice sometimes in the graph a huge gain of time in some periods. This is mainly due to the operation time of the network. In case there are limited buses or trains due to the defined service hours, the integrated approach will provides a much faster arrival time due to the availability of Autolib services. Figure 4.8 shows how the gain varies according to different stops due to the variation of availability of the RATP services.For example reaching the first stop is not available before 12 AM before since night buses are not available, however, with Autolib combines in the network you can reach the stop at 08:24 PM which is obviously much more preferable.

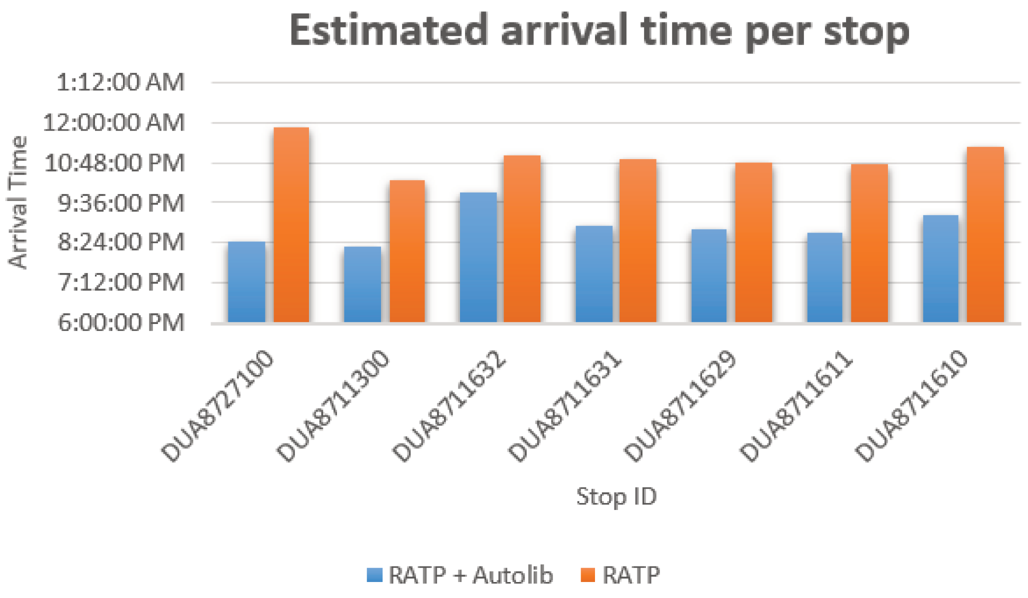


FIGURE 4.8: The different gain in time due to service hours

Therefore, introducing these connections decreases the waiting time for passengers and results in more optimized trips. We can now consider new types of mobility that were not previously taken into account (bike sharing, car sharing, etc.). This can be used to fit to passengers profiles by combining the appropriate connections while planning trips. Passengers will be able to define connection types, modes and find the best trip type.

Compared to the existing link discovery frameworks, our approach succeeded in discovering links with richer representations and extended properties that can be used for numerous tasks (Earliest Arrival Time in our example).

4.13 Conclusion

Enabling multimodality requires forming relations between different transportation services and units. To do so, creating connections between transportation data instances is essential to reach our desired goal. Many operators are embracing the use of open data concept and are pushing their data to the web. Taking advantage of these published data is a chance to reach the connected dataset we aim for and enable multimodal solutions. However, existing open data techniques for linking datasets are suitable and focused on equivalence detection or entity resolution. Linking transportation datasets required another type of links. The links we aim for represent real physical links between transportation units and the rules to define their availability and characteristics.

In this chapter, we have introduced the means and methods to represent homogeneous connections between transportation networks. We have proposed an approach named Link++ that enables a flexible and customizable way of generating connections between data sources. This enables a better way for transportation systems to access information about nearby services and integrate them with their own network. We have described the process of defining custom connections, and used these connections to expand a transportation network containing trains and bike networks. We have evaluated the effect of the newly introduced connections by providing an extension of the CSA algorithm for the estimated arrival time problem.

In the future we will focus on the dynamic part of the connections since they are not always static and may be affected by some external events. On the other hand, the rules that decides when to generate connections can vary according to the user profile. Taking into account these profiles while generating rules is essential for our approach to be more customizable. Finally, the introduction of the new unscheduled connections to the CSA algorithm has decreased its performance which should be taken into account to be better optimized for multimodal purposes.

Chapter 5

Service Level Integration of Transportation Operators

5.1 Introduction

Trip planning applications are becoming one of the essential applications we regularly use in our daily lives. They help us plan better trips that are faster, cheaper and more secure. Many research works were conducted over the years to provide better trip planning algorithms. Up to this point, the algorithms gave good results on the data especially when they are bounded to public transportation networks. However, with the need of integrating new transportation data and services, existing algorithms became insufficient to support the new requirements. Many new services are proving to be very efficient for travelers. Ridesharing is an example of these services and the scope of this work.

Ridesharing is a transportation service where individual travelers share a vehicle for a trip and split travel costs with other travelers who have similar itineraries and time schedule. These services are getting a lot of attention in the recent years as they are beneficial for both travelers and drivers, and friendly to the environment [39].

Ridesharing can be a solution for areas not covered by public transport or a backup plan in case of some perturbations. It has many advantages that benefit travelers, drivers and the environment. Some of ridesharing advantages include cost saving, lower parking demand, lower emissions, better urban design, less congestion and less vehicle travel.

The main problem is that these services are not yet fully integrated in existing trip planning algorithms. They are proposed as alternative plans if no public transportation plan is found or simply not preferred. For example, consider a traveler who plans to travel from a source to a destination. Existing solutions may suggest the following plans. The first is a sequence of multimodal trips e.g. walk from source to tramway station A → take the tramway from tramway station A to train station A → take train from train station A to bus stop B → take bus from bus stop B to your final destination. The second is a car sharing solution that simply suggests: walk from source to pickup location → take the car sharing from the pickup location to the drop-off location → walk from the drop-off location to destination. In Figure 5.1 we see graphically the problem of isolated solutions between the timetable networks and the car sharing service. We notice here that the car sharing solution is proposed in isolation of the multimodal solution and not really as part of the trip.

Integrating these services within the existing trip planning solutions may

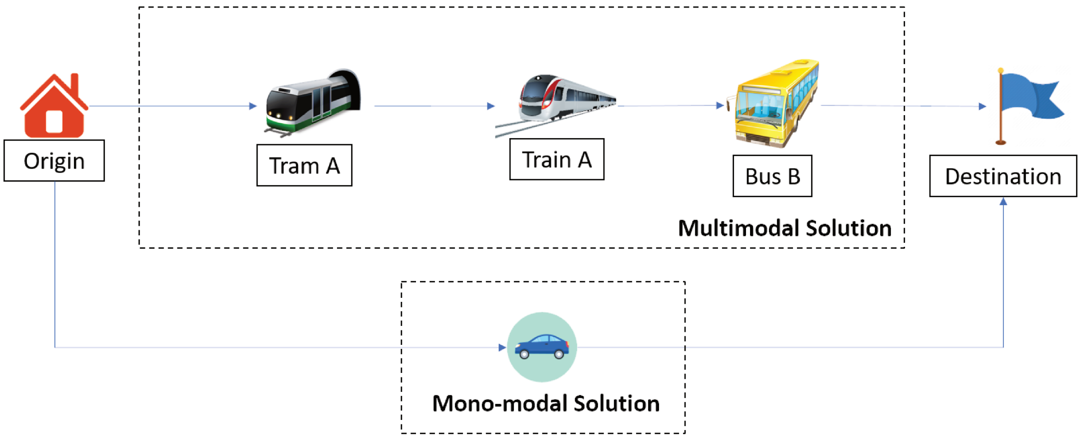


FIGURE 5.1: Isolation of Ridesharing solution from existing trip planners

vastly improve the quality of the trips. In addition it may serve as a backup plan in case of some delays or unexpected events - which may happen in public transportation services.

The main cause behind the isolation is that it is simpler to calculate trips with scheduled networks due to the fact that we know in advance the departure and arrival times of each transportation unit. It is not possible in ridesharing services where a driver may at any time notify an intention of sharing a ride with others. This complicates the problem because these requests will appear/disappear dynamically on the network. In addition, the ridesharing problem is complex on its own [99, 39] and adding the connection with other modes will vastly increase the size of the search space, and will result in more complex time consuming calculations.

In this work, we seek for a way that integrates the complex characteristics of ridesharing services with existing trip planning algorithms. Our goal is to provide an algorithm that is able to handle efficiently the dynamic nature of these services. Many challenges arise from this goal which we summarize as follows: 1) adding the new services will enormously enlarge the search space, 2) there is no fixed pickup and drop-off stops for ridesharing services as in public transportation services, 3) there is no information about the future position of the ridesharing cars, 4) the additional cost added by integrating the new services should be taken into account and finally 5) there is no clear way on where we should integrate the services on the trip timeline.

This chapter introduces RETRy, an approach enabling the integration of ridesharing transportation services with existing multimodal public transportation networks while reducing the search space. The idea in general is to calculate a trip - named *reference trips* - using existing trip planners and use it as a baseline. We then use the passenger's location, passenger's destination and the location of public transportation stops in the *reference trip* as possible pickup and drop-off parameters for ridesharing services, trying to optimize the trip by introducing more optimal sub-trips by using ridesharing mode. The resulting trip forms the initial plan for a passenger to follow. During the trip execution the trip can be re-planned as a response to unplanned events, which makes it adaptive and reliable. The feasibility of the trip is calculated by the time both drivers and passengers may wait for the pickup. Two approaches are proposed for reference trip generation and are discussed later in the chapter. The first uses only the base plan generated by trip planners, while the second uses the K-nearest trips as base plans. We have evaluated the approach by issuing random trips with different modes and comparing the gain we have got in each mode. The evaluated modes are: public transportation only, public transportation with ridesharing and ridesharing only. The comparison between the different modes shows that integrating ridesharing with public transportation networks results in faster trips with little additional cost especially with the K-nearest trips approach.

5.2 Problem Statement

Enabling more optimal transportation trips requires the integration of all the available transportation services in a given area. This integration is difficult when it comes to the variety of transportation modes and the characteristics each one holds. Up to the moment, transportation trip planners work on timetable networks that are driven with a pre-known schedule and fixed transportation departure/drop-off locations allowing easy trip planning. This becomes harder when considering the new transportation services, such as ridesharing, which do not have this notion of fixed stops or fixed schedule. This type of services is dynamic and unpredictable. We can not know in advance the availability nor the location of a service which makes it very difficult to consider them in our plans. The main focus in ridesharing solutions is on matching a passengers request with a driver. This problem is complex and considered NP-hard [99, 39] due to the dynamic nature of the service, the big search space and the complex constraints it follows. Therefore, integrating ridesharing services leads to complex time consuming calculations due to the huge search space resulted from both types of transportation services.

The problem is as follows: consider a passenger traveling from an origin **O** to a destination **D** at a specified departure time **t**. Given a set of public transportation and ridesharing services $\mathbf{S}_i$, the goal is to effectively generate a multimodal trip **T** that utilizes the different transportation modes. **T** is a set of sub trips $\mathbf{l}_i$ where each can be served by a transportation service $\mathbf{S}_i$ or a simple walking trip. The generated trip must be optimized in terms of both speed and cost. Since ridesharing services have no notion of fixed stops, the challenge is to integrate them within the trip plan while taking into account their unique characteristics.

5.3 Integrating Ridesharing into Trip Planning

Integrating new transportation services with current trip planning solutions will add many challenges and performance issues to the standard trip planning problem. The search space will be a huge and dynamic with services appearing/disappearing at random. There will be no clear location to issue a service from especially when the stops are not fixed. In order to solve this problem we introduce RETRy. RETRy is a framework enabling the integration of ridesharing and public transportation services. The general idea is to use existing public transportation trip planning algorithms [5] to generate a base plan that we call a reference trip. Then we try to improve the reference trip by injecting ridesharing services when possible. By doing so, the search space is reduced a lot and the positions of the requests is bounded to the transit transportation stops. In addition, RETRy makes it possible to adapt to changes by a repeated execution of the process over the course of the trip.

This process is not straightforward for many reasons. First of all, reference trips do not always exist. Furthermore, the future positions of the ridesharing cars are not known in advance since a ridesharing service is dynamic. Finally, optimizing the selection of pick-up and drop-off positions for the ridesharing request is a combinatory problem which is complex in its nature.

To cope with these problems, our proposal uses a heuristic approach. In the following, we present the approach in details. The first section discusses the first part of the algorithm, which is generating the reference trip. The second section targets the optimization of the generated reference trip with the injection of ridesharing services.

5.3.1 Bootstrapping

We define reference trips as those that can be calculated by existing trip planning algorithms [5] are already optimized for public transportation routing problems.

Definition 7. *A reference trip* $\boldsymbol{T}$ *is a set of sub-trips* $\boldsymbol{l}_i$*, where each sub-trip is assigned to different transportation services* $\boldsymbol{S}_i$*.*

Definition 8. *A sub-trip is composed of: departure stop* ***dStop****, departure time* ***dTime****, arrival stop* ***aStop****, arrival time* ***aTime*** *and trip cost.*

Given a user's location and destination, our planner asks the existing trip planner to generate a reference trip combining a set of sub-trips to match the user's query.

However, it is not always possible to find a reference trip simply due to lack of existence of nearby services. To cope with the problem, we provide an alternative method. We propose to calculate a sub-reference trip, which is a trip enabling the traveler to reach the closest point of his destination from the closest point of his origin. A sub-reference trip is intended to utilize public transportation as much as possible, thus reducing the walking time for passengers.

To generate a sub-reference trip we propose two different approaches. In the first one, the planner performs two range queries. The first query selects all source's nearby public transportation stops as starting points. The second range query selects the destination's nearby public transportation stops to be considered as destination points. After having both source and destination points, we issue trip requests from the combination of both points until a trip is found. The nearby stops can be obtained from a dataset of stop locations. Here we use the Google Places API[1].
The second approach is to choose the midpoint of the origin and destination, then issue two queries from the starting point to the midpoint then from the midpoint to the destination. This process is done recursively until a solution is found. The midpoint can be obtained using two main methods. The first is via a simple mathematical calculation - which is fast - using the positions of the two stops. In this case, the midpoint can appear anywhere in space, even outside a road network. The second is using a web service (http://geomidpoint.com/) that automatically calculates the road midpoint between two stops. In this approach, we used the mathematical one since it is faster especially for the evaluation since we issued many queries over time. In some cases, the midpoint does not fall on a road network, and this is not a problem for our algorithm. The reason is that the trip planner will plan the trips automatically based on the nearest road network from the given source location (exactly like when we plan a trip using our mobile phones from a location far from roads). Our approach is inspired by the idea of selecting

[1]https://developers.google.com/places/documentation

nearby grids in [68] and the time-dependent range query introduced in [14].

We note that, at this step, these trips are not optimized and may contain a lot of walking gaps that are infeasible for the traveler. for this reason, we introduce the next step of the core planner which tries to optimize this reference trip using the ridesharing component.

Algorithm 6 shows the trip generation algorithm according to the midpoint method described above. The algorithm starts by calculating a reference trip (line 1). The caclulateTrip function calls a public transportation trip planner with the source and destination points in addition to the departure time. If the reference trip was successfully found, the algorithm terminates by returning the calculated trip. Otherwise, the algorithm gets the midpoint between the source and destination (line 5). Two more trips are issued on lines 6 and 7 that represent a trip from the source to the midpoint (left trip) starting from the departure time and a trip from the midpoint to the destination (right trip) starting from the left trip's arrival time. It is important to note the recursive calls for these trips. This is due to the possibility of also not finding trips for these two parts, thus the algorithm works recursively by computing the midpoint in each part and calculating left and right trips. Finally, the algorithm combines both left and right trips and return the results (lines 8 and 9).

```
Input: source: src, destination: dst, time: departure time
Result: a trips from a source to a destination
1  /* Calculate a trip from source to destination using existing
      multimodal planner                                          */
2  $trip \leftarrow calculateTrip(src, dst, time)$;
3  if trip not empty then
4  |   return trip;
5  end
6  /* If no trips were found, get the midpoint of the trip and issue
      the queries from source to midpoint and from midpoint to
      destination recursively                                     */
7  $midpoint \leftarrow getMidpoint(src, destination)$;
8  $lefttrip \leftarrow calculateTripRecursive(src, midpoint, time)$;
9  $righttrip \leftarrow calculateTripRecursive(midpoint, dst, lefttrip.arrivaltime)$;
10 /* Combine both sub-trips to get the overall one                */
11 $overalltrip \leftarrow combinetrips(lefttrip, righttrip)$;
12 return $overalltrip$;
```

Algorithm 6: Trip Generation Algorithm

5.3.2 Extended Trip Generation Algorithm

The reference trips generated by public transportation trip planners are highly dependent on the state and the infrastructure of transportation networks. The reference trip only takes into account the first nearest public transportation trip neglecting the fact that alternative trips may perform better when integrated with other trips.

For this reason, an alternative method is proposed to take advantage of this possibility. Instead of querying just the nearby stop, we choose the K-nearest neighboring stops from the source and issue K queries to the destination. Even though some stops may be far away from the starting point, there is a possibility that reaching the stop with ridesharing will result in a faster overall trip to the destination. This modification results in multiple reference trips to be optimized later with RETRy by injecting the ridesharing services. In addition, instead of querying directly to the destination point, we may use the same technique and query the K-nearest stops closest to the destination. This will enable more flexibility to the trip and allow more choices. Finally, we add the walking path from the source to the initial stops and from the final stops to the destination. The resulting trips are then passed to the optimization phase. Figures 5.2 and 5.3 show graphically the difference between the single trip planning and the K-nearest planning. We see in the single trip planning only one reference trip is taken from the stop nearest to the starting position, while in the second, the K-nearest stops are taken and then used to generated the K-reference trips. Note that the dashed lines represent the walking path from one position to another.

Algorithm 7 shows the extended approach. It first starts by initializing an

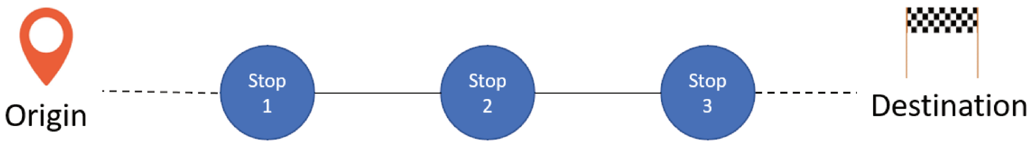


FIGURE 5.2: Single Reference Trip Planning Method

empty trips array (line 1). Later, the K-nearest stops to the source point are queried (line 2). The algorithm loops on the list of stops (line 3) and calls algorithm 6 to generate a reference trip from the each nearby stop to the destinations (line 5). Note that the departure time is the initial departure time given by the passenger in addition to the walking time to reach the stop.

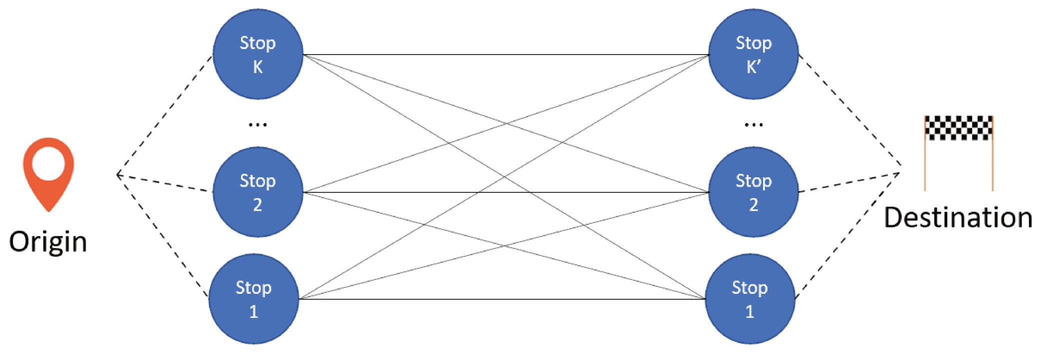


FIGURE 5.3: K-Nearest Trip Planning Method

Thus, we calculate the walking time from the source to the stop and query based on the new time (line 4). All the generated trips are appended to the final trips array. Before the algorithm terminates, it adds a walking path from the source point to each nearby stop (line 6). This path will be later optimized by a ridesharing service if necessary. Finally, we return the appended trips to the caller (line 7).

```
Input: source: src, destination: dst, time: departure time, number of nearest
       stops: K
Result: a set of trips from a source to a destination
1  /* Initialize the trips array                                          */
2  $trips = []$;
3  /* Get the nearest K stops from the source                             */
4  $nearby \leftarrow getNearbyStops(src, K)$;
5  /* Issue a query to destination starting from each stop                */
6  for stop in nearby do
7      /* Calculate the walking from src to the stop                      */
8      $newTime \leftarrow time + walkingTime(src, stop)$;
9      /* Calculate the reference trip from stop to destination
           starting at the new time                                       */
10     $trips.append(generateReferenceTrip(stop, dst, newTime))$;
11 end
12 /* Append a walking distance from the source to each nearby stop
       */
13 $trips \leftarrow appendSrc(trips, src)$;
14 return $trips$;
```

Algorithm 7: Extended Trip Generation Algorithm

5.3.3 Injecting Ridesharing Services

In this section, we focus on the problem of adding the ridesharing services to the reference trip. The idea here is to iterate over the proposed sub-trips and check if ridesharing can be a better alternative. Since it is complicated to find a random pickup point for a future ridesharing request, we use the fact that the sub-trips start and end at public transportation stops. Hence, we choose the pick-up and drop-off stations among these stops for the service requests. In addition to serving as a good pick-up and drop-off locations, this facilitates the request parameters by knowing the right timing that is derived from the transportation stop's schedule. Therefore, a ridesharing request is issued given the parameters (source, destination and departure time). After a service is found, we connect it to the overall trip. And the gap is filled with the newly found more efficient alternative mode.

It is worth noticing that the added service will improve the arrival time since the new generated sub-plan makes use of the time gained by the ridesharing service. For example, since we will use a service to fill the gap then it is possible that we may arrive in a shorter time. To deal with this case, the planner issues another trip request from the drop-off station to the destination given the new departure time. The new trip will in turn be an input to the core planner for further optimizations.

The proposed algorithm is shown in Algorithm 8. It starts by iterating over the edges of the reference trip, starting from the origin point to the destination. For each edge, the algorithm queries a ridesharing service giving the source and destination points as the departure and arrival stops of the current edge (line 2). Later, the cost of the original edge is compared with the cost of adding the new ridesharing service (line 3). The cost(service) function here is a function that calculates the trip cost of a specific service. The cost in the current implementation is the trip duration, however in the future work we plan on a more complex cost. The idea is to check if injecting the new service will improve the trip or not. If yes, it will be added and will be discarded otherwise (line 4). The addition of the new edge will cause a better arrival time since the cost is lower. Therefore, the algorithm calculates another trip staring from the arrival stop in order to make benefit of the gained time (line 5). Finally, the path originating from the next edge is replaced with the new trip and the algorithm continues iterating over the new edges. When the iteration is completed the algorithm terminates with the reference trip being optimized with the added ridesharing edges.

```
Input: reference trip: trip
Result: Optimized trip with gaps filled
1  /* Iterate over each edge in the trip                          */
2  for current, next in trip.getPaths() do
3      /* Query for a ridesharing service between the endpoints of
          the edge                                                  */
4      $service \leftarrow selectService(current, next)$;
5      /* Check if the ridesharing service is faster than the
          existing solution                                         */
6      if (cost(service) < cost(edge(current, next))) then
7          /* Replace the existing edge with the ridesharing service
              */
8          replaceEdge(current, next, service);
9          /* Issue a new query from the edge endpoint to the
              destination                                           */
10         $newTrip \leftarrow calculateTrip(next, rtrip.dest)$;
11         /* Replace the sub-trip from the edge endpoint to the
              destination by the new sub-trip                       */
12         replaceTrip(next, trip.tail, newTrip);
13     end
14 end
```

Algorithm 8: Inject Service Algorithm

5.3.4 Selecting the Driver/Service

Drivers are moving in real time and finding their future positions is somehow complex. We may use some learning or probabilistic approaches to calculate the future position of a car. However, this is unnecessary due to the fact that in ridesharing services the driver must confirm the request. This means that a candidate driver can simply reject a trip request if it is not along his way.

As emphasized above, a ridesharing service is highly dynamic, which requires adjusting the plan by requesting it in real-time. However, the request should anticipate the future positions of the potential drivers. To overcome this problem, we propose the use of a range query around the arrival stop (pick-up location) so as to select, as candidates for ridesharing, cars that may reach this stop in a specific time in the near future. We send these cars the request with the passenger's waiting time constraint, then we choose the one who accepts it. In case some candidates were chosen but moved away from the stop, the problem can be easily fixed by the driver simply discarding the request.

The range query must take into account two main constraints. The rider pick-up time window constraint and the driver waiting time constraint. The rider pick-up time window constraint is the acceptable interval of time the rider should be picked up at. This is translated into the radius of the range query that covers all the cars that can reach the pick-up position while satisfying the time window. The driver waiting time constraint is the time a driver can wait in the area for the rider to arrive. This is taken into account by first querying the drivers that can reach the rider's pick-up location before the pick-up time window. Then filter them based on the waiting time each driver can wait. The final set of filtered drivers receive the request to be accepted or rejected.

5.3.5 Triggering the Service Injection

An important question that arises here is where to inject these ridesharing services? There is no direct answer to this question because it depends on the user preferences. If a user wishes to minimize the travel cost, then it is better to limit the services injection to only the long distance gaps. Ridesharing services usually cost more than public transportation systems, thus limiting their use to only long gaps is helpful. If the user favors optimizing time, then more services are to be injected whenever possible. A balanced approach is to check the overall cost that is composed of both time and money after each service injection. Then select the one with the lowest cost. Another approach is to track the reference trip in real-time, and only inject a ridesharing service when some unexpected delay is detected. In this case, the trip will be flexible to events and adapt in real-time to offer an optimized service to the user.

5.3.6 Customizing RETRy

It is crucial to note that RETRy is highly customizable and does not depend only on external services and data on the web. Instead, RETRy may work in different configurations based on each need. For example, it may work on local data and local functions provided by the user. To do so, we introduce two functions in algorithms 6 and 8, calculateTrip and selectService. These functions are they key functions behind customizing RETRy. Users may extend those functions to write their own implementation of calculating the trips and selecting the ridesharing functions. Therefore, users are not limited to any service mentioned in the paper and the framework is generic to support different usages.

Adding custom services or functions must be taken with deep caution. The more optimized the algorithm, the better the performance of the framework. Users may rely on quick shortest path algorithms like CSA [28] and lightweight connections representations as proposed in Chapter 4.

On the other hand, it is important to note that using external services makes the framework lighter, thus able to be integrated on small devices with limited memory capabilities.

5.4 Evaluation

In this section we evaluate the different implementations of RETRy and its core algorithm. We first evaluate the use of a single reference trip as a base plan (as in Algorithm 6), then we study the effect of including the K-nearest trips (as in Algorithm 7). The evaluation criteria is based on comparing different trip plans with and without integrating ridesharing services.

The implementation of RETRy is carried out using Python 3 programming language. The execution environment is a Windows 10 machine with 8GB of RAM and a core i5 processor with 1.70 GHz of processing power.

5.4.1 Query and Service Selection

The trip queries were formed by randomly choosing different source and destination points. Each query set consists of 200 queries representing random trips in the Ile-de-France region, which is one of the most crowded areas in France. First, we chose a random source and destination for a query. Then, we issue a car driving trip and a multimodal public transportation trip. Later RETRy optimizes the trip by injecting the ridesharing service and finally comparing the results. We have run our queries over one week with a one hour time window each day. The time window choice is to be able to visualize how trips vary with respect to different query hours (morning, mid-day and night).

We chose Google Transit as our trip planning service since it integrates multiple public transportation modes and provides a practical use via an API[2]. Uber[3] was selected as our ridesharing service since the API provides methods to estimate a ridesharing request and get an estimate of both cost and duration. Notice that the algorithm is independent of the services used.

5.4.2 Single Reference Trip

At first we evaluate the approach by creating a reference trip at a time in each query as in Algorithm 6. In order to optimize the trip, we iterate over each sub-trip and try to inject the ridesharing service in its place. However, since we are injecting the ridesharing service on each edge, we may sometimes have consecutive trips that use a ridesharing mode e.g. trip A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D where the edges between (A and B) and (B and C) are ridesharing services. Therefore, we aggregate the whole sequence by replacing it with one trip from the from the first departure location to the last arrival location (from A to C). This may lead to an earlier arrival, lower cost and thus a more optimized trip.
The mentioned steps were executed on RETRy and the results are shown in figures 1 to figure 4. It is worth to mention that calculating the reference trip and injecting the ridesharing services takes around 8 seconds.

Duration with respect to Transportation Mode

Figure 5.4 shows the trip duration with respect to the hours when the queries were issued. For each hour, a group of different transportation modes are shown with their corresponding values. Analyzing the figure, we notice that the average trip duration (92 minutes) using public transportation is optimized using the injection of ridesharing services (70 minutes) with a gain of approximately 22 minutes in average. This result is better optimized when aggregating the trip (63 minutes) with an average total gain of 29 minutes.

[2]https://developers.google.com/maps/documentation/directions/
[3]https://www.uber.com

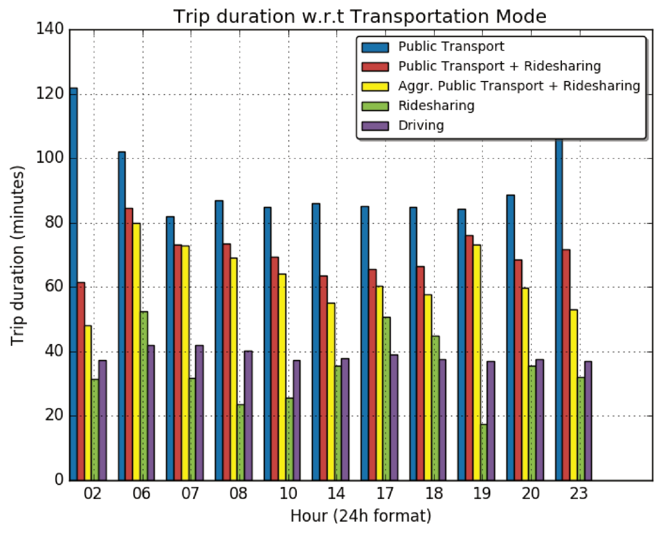


FIGURE 5.4: Average Trip Duration w.r.t Travel Mode

We notice that the driving and the ridesharing modes are the fastest according to the simulations (approximately 40 minutes), however, this solution is not feasible for all passengers. The first reason is that not all passengers can travel by car. The second is that using ridesharing services alone is expensive.

Cost with respect to Transportation Mode

In addition to what precedes, we must keep in mind that ridesharing services exist for small distances, therefore relying on them alone is not possible for long trips. Figure 5.5 shows that the injection of ridesharing trips costs around a minimum estimate of 15 euros which is improved slightly to 11 euros when compressed, while ridesharing alone is nearly double the value (22 euros) which is obviously not preferred.

Expected versus Real-time Trips

For practical reason, that is grouping the tests in the same program, the previous experiments considered trips that are planned hours before their execution. However, to ensure the reliability of the proposed solution for timely execution, we set up a simulation script which schedules and calls the ridesharing services at the exact time when they were planned. Then, we

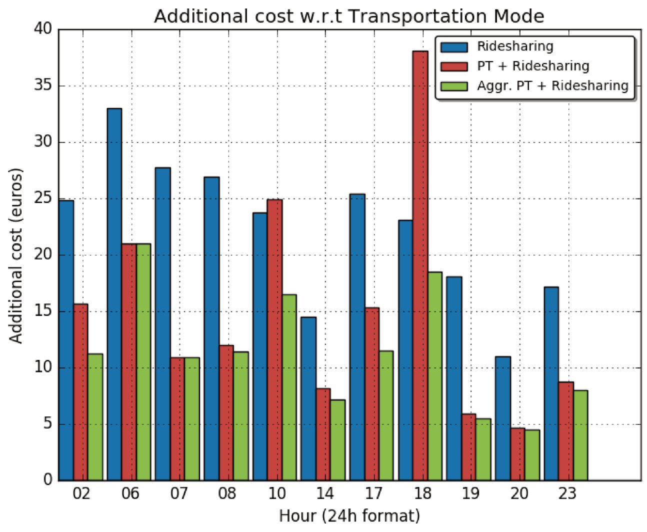


FIGURE 5.5: Trip cost w.r.t Travel Mode

compare the planned arrival time with the real one. The results are shown in Figure 5.6. The positive points show an early arrival with a max of 16 minutes ahead of time, while the negative points show a late arrival with a max of 7 minutes delay.

Thus, timely trip plan execution is almost as efficient as the offline trip plan, and sometimes more efficient than the offline version. This variation is normal since the real time conditions vary. This test shows however that this variation is rather small, and that our previous experiments are valid. The indirect outcome is to show the stability of the duration between the planned and the online (re)planned trips.

5.4.3 K-Nearest Stops

In this section we study the possibility of improving the results with the querying technique introduced in Algorithm 7. First, we query the nearest K stops (with K = 5), then we calculate trips to destination by taking these stops as starting points. Finally, we add the required path to pass from the starting point to each nearest stop. Despite that this method will clearly result in a worse trip than the reference trip, it is possible that it will result in a optimized trip when ridesharing services are injected.

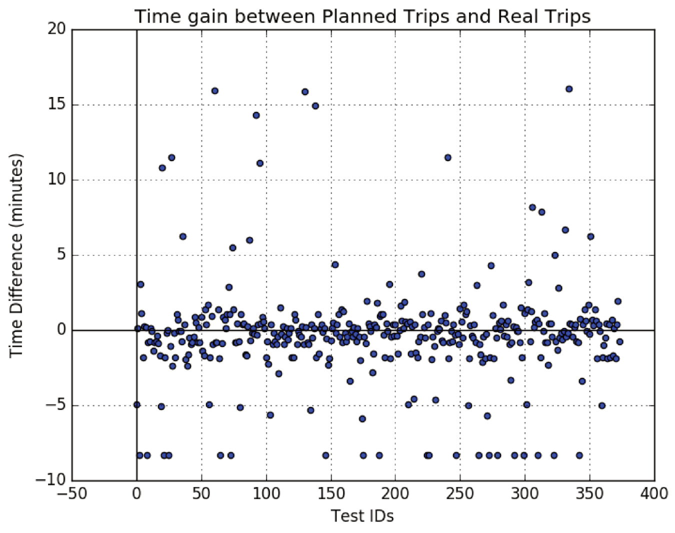


FIGURE 5.6: Expected Arrival Time vs Real Arrival Time

We evaluated this approach by issuing the same queries as in Section 5.4.1 according to the different proposals in algorithms 6 and 7. The results are shown in Figure 5.8. The figure represents the durations of the single and k-nearest trips. A clearer view is shown in Figure 5.7 that presents the time gain by using the K-nearest trips method. The time gain is calculate as follows: $(trip2.duration - trip1.duration) * 100/trip1.duration$, where trip2 is the K-nearest trip and trip1 is the single reference trip. The results show that using the K-nearest stops method (Algorithm 7) will either improve or give the same results as the previous algorithm. The average gain is approximately 8 percent which can be taken as a considerable one especially when adding up the improvement we achieved compared to only using public transportation planners.

However, this improvement comes with a calculation drawback which increases the execution time of the algorithm. Figure 5.9 shows the execution time in both approaches. On average, the first approach takes 10 seconds to query and optimize the trip, while the second approach takes up-to 60 seconds. This increase is typical since more trips are queried where each one undergoes an optimization phase in order to finally select the fastest one. Nevertheless, we observe that the increase ratio remains reasonable and the execution still doable in near real-time. For instance, for 5 neighbors at both origin and destination w(which results in 25 reference trips), the increase ratio of the execution time is only between 4 and 5.

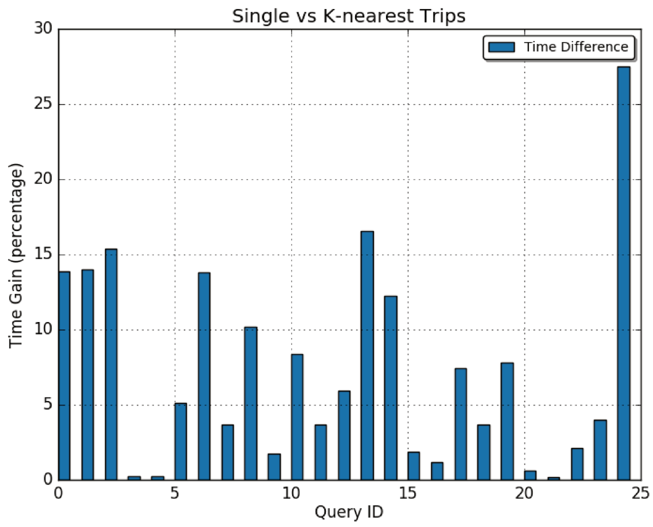


Figure 5.7: Single vs K-nearest Trips - Gain Percentage

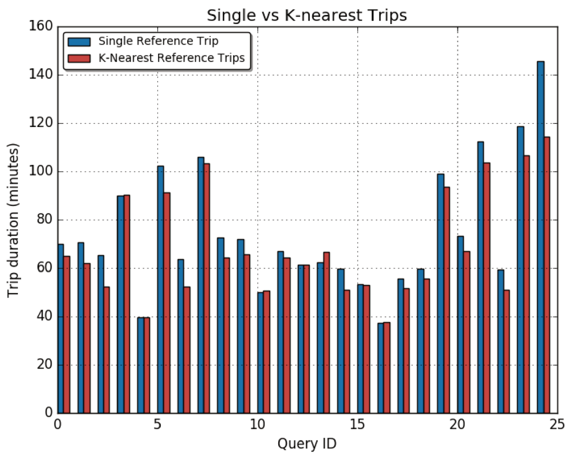


Figure 5.8: Single vs K-nearest Trips - Trip Duration

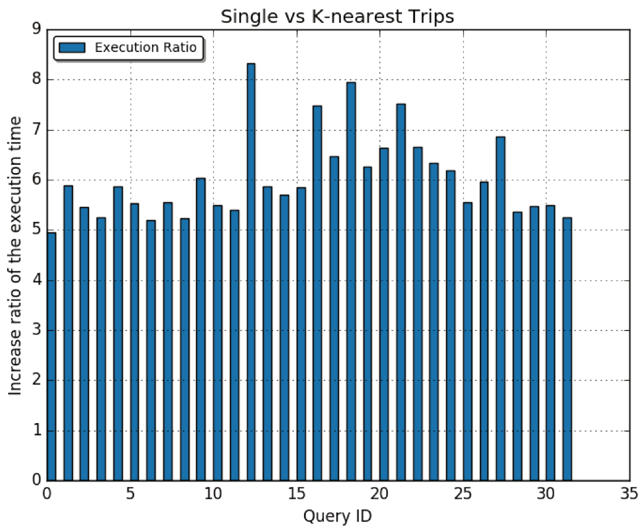


FIGURE 5.9: Single vs K-nearest Trips - Execution Time

As a final comparison, Figure 5.10 shows the difference of using a single reference trip and K-nearest trips approaches according to different values of K. The results shows that the duration and cost gain is nearly similar. However, the execution time significantly increases with the increase of K, which is predictable since the number of trip queries is increasing. Therefore, we conclude that K=3 is enough to achieve better trips with a reasonable amount of time.

5.4.4 Visualization

Our system is split into two main applications. The back-end python core and the front-end Django web application. It allows to interactively query trips and see the results with different choices of transportation modes. Users can see the time and cost difference in using driving, ridesharing, public transportation or our integration solution. A screenshot of the web application is shown in Figure 5.11, with the first page showing the query interface and the corresponding results in page two. The results are shown as both a table and a graph showing the different durations and costs across the different trip modes i.e. PT, PT + Ridesharing, Aggr. PT + Ridesharing, Ridesharing and Driving. The panel below shows the trip details for each mode where the user may see each trip step in the plan. The map on the right shows the corresponding plot of the trip and is changed when a different mode is selected.

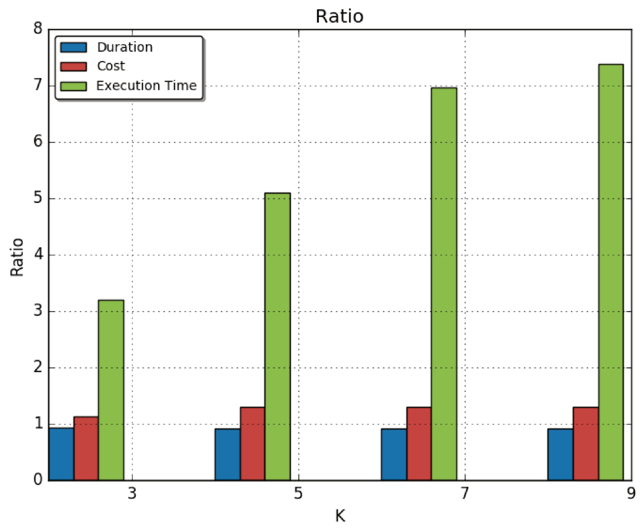


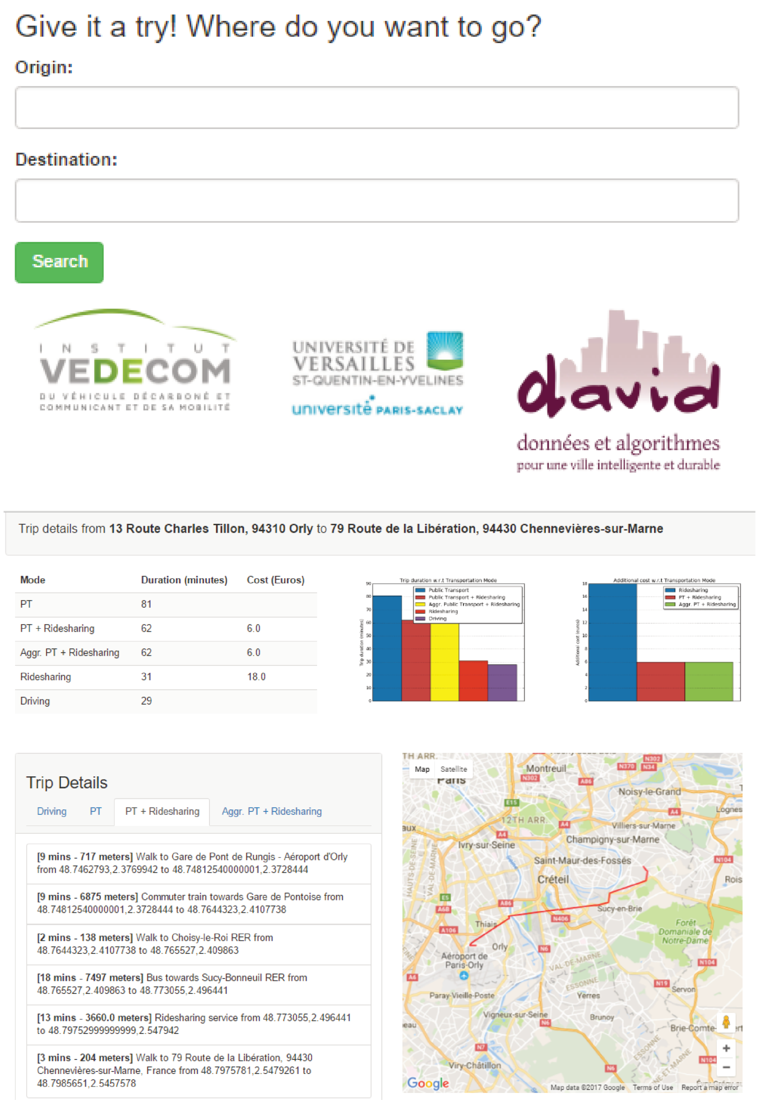
FIGURE 5.10: Gain ratio with respect to different values of K

5.5 Conclusion

Ridesharing services play an important role as part of the existing transportation services. Current solutions propose ridesharing as separate services without really integrating them within planned solutions. The problem facing this integration is the complex characteristics ridesharing services retain. They are highly dynamic and do not have the notion of fixed stop locations or schedules. This complicates the planning problem by massively increasing the search space resulting in a very high execution time.

Our approach solves the problem by presenting a light-weight framework that makes use of existing trip planning algorithms and optimizes them by injecting ridesharing services when possible. These characteristics make it possible to use the system on mobiles or embedded systems without being overwhelmed with heavy computations. Results show that using ridesharing services in multimodal trip planners will decrease the trip duration with an acceptable extra cost.

Future work target enabling multi-criteria querying (such as number of transfers, preferred modes and locations) which is challenging, especially across different services. In addition, we also target integrating a real-time travel monitor with the ability to automatically detect delays or unexpected events and adapt the trip based on them.



Give it a try! Where do you want to go?

Origin:

Destination:

Search

Trip details from **13 Route Charles Tillon, 94310 Orly** to **79 Route de la Libération, 94430 Chennevières-sur-Marne**

Mode	Duration (minutes)	Cost (Euros)
PT	81	
PT + Ridesharing	62	6.0
Aggr. PT + Ridesharing	62	6.0
Ridesharing	31	18.0
Driving	29	

Trip duration w.r.t Transportation Mode
Public Transport
Public Transport + Ridesharing
Aggr. Public Transport + Ridesharing
Ridesharing
Driving
Trip duration (minutes)
Additional cost w.r.t Transportation Mode
Ridesharing
PT + Ridesharing
Aggr. PT + Ridesharing
Additional cost (euros)
Map
Satellite

Trip Details

Driving | PT | PT + Ridesharing | Aggr. PT + Ridesharing

[9 mins - 717 meters] Walk to Gare de Pont de Rungis - Aéroport d'Orly from 48.7462793,2.3769942 to 48.74812540000001,2.3728444

[9 mins - 6875 meters] Commuter train towards Gare de Pontoise from 48.74812540000001,2.3728444 to 48.7644323,2.4107738

[2 mins - 138 meters] Walk to Choisy-le-Roi RER from 48.7644323,2.4107738 to 48.765527,2.409863

[18 mins - 7497 meters] Bus towards Sucy-Bonneuil RER from 48.765527,2.409863 to 48.773055,2.496441

[13 mins - 3660.0 meters] Ridesharing service from 48.773055,2.496441 to 48.79752999999999,2.547942

[3 mins - 204 meters] Walk to 79 Route de la Libération, 94430 Chennevières-sur-Marne, France from 48.7975781,2.5479261 to 48.7985651,2.5457578

FIGURE 5.11: The Graphical User interface of RETRy

Chapter 6

Conclusion

This thesis targeted the problems that rise from transportation data integration. At first we have seen how standardization solutions fail to support the ultimate goal of a unified model to represent transportation information. The solution turned quickly into another problem with many organizations rushing to propose new standards and pushing it to the market. The result was loads of standards that are scattered among its adopters resulting in a new diverse representation problem. We moved into the domain of automatic schema matching, which seemed a possible solution to handle schema heterogeneity of transportation datasets. However, our studies showed that existing approaches fail to detect the complex characteristics in transportation schemas. More specifically, the diverse and complex representations of geospatial characteristics which play a main role in representing transportation information and units. On this manner, we proposed an approach for an automatic detection of these properties by using geospatial web services as mediators that guide the matching task. The approach uses dataset instances and a statistical similarity matrix to find relations between the dataset and the web service response. This similarity is then used to conclude the mappings from one dataset to another via transitivity. The results showed good improvement compared with state of art approaches.
After the schema level integration, we focused on how to identify links between different transportation services and their entities. The links we search for represent the physical links that connect different services together. To this end, we explored using the open data techniques as the fuel to build the giant connections between all the given services. However, we noticed limitations in the existing interlinking tools that are suitable to equivalence detection other than finding richer types of links. Existing tools miss the support for complex types of links and the methods to detect them. On this behalf, we proposed a new model that enables a flexible and customized way of

discovering and generating links between transportation datasets. The new model and the corresponding framework were able to improve the quality of the links, thus enabling a richer network graph that connects all the added transportation services together.

Moving to the service level, we noticed an isolation of the new mobility services from existing journey planners. New mobility services are proposed as alternative solutions and not really as part of the multimodal planned journeys. This isolation was mainly because of the complex characteristics held by these services. Unlike the known public transportation services, these services have no notion of fixed timetables and fixed boarding stops, which complicates the integration a lot with many complex computations and search space. Our approach handled this problem with a set of algorithms that enables this integration by building trips above what we introduced as reference trips to solve the problems of unknown timetable and missing fixed pickup stops. The evaluation showed an interesting improvement on both cost and travel time dimensions with a very small acceptable additional cost due to the addition of ridesharing services.

Summing up, we handled three problems related to transportation data integration at three levels: schema level integration, instance level integration and finally service level integration. The experiments showed that the solutions may provide passengers with better trips that integrates all the nearby services to provide them with the optimal trip plan.

Our work still has high potential for future improvements. On the schema level, it is required to solve the problem of data sampling to select the top queries that will be matched with the web service response. This will improve the speed of the algorithm by decreasing the number of web service calls. In addition a smarter query formulator is needed to efficiently get more relevant information from the web service. Integrating geospatial querying solutions shown in [103] may help increasing the accuracy of the query formulator. It is worth stating that the use of a web service to bridge the gap between different dataset representations could apply to other domains as long as web services are provided for these datasets.

Regarding the instance level, a possible extension to the work is to adapt the approach to handle the dynamicity of the connections. This will make us able to maintain the status of existing connections and handle new services, such as dynamic ride-sharing, car sharing, etc. The problem here is how to track connections' evolution in real time. How can we make use of external events that may affect their use, etc.? Furthermore, some performance

optimization is to be considered for both the automatic matching and interlinking approaches.
Finally, on the service level, future work target enabling multi-criteria querying (such as number of transfers, preferred modes and locations) which is challenging, especially across different services. In addition, we also propose targeting the integration of a real-time travel monitor with the ability to automatically detect delays or unexpected events and adapt the trip based on them.

Bibliography

[1] Patrick Arnold and Erhard Rahm. “Enriching ontology mappings with semantic relations”. In: *Data and Knowledge Engineering* 93 (2014), pp. 1–18.

[2] Spiros Athanasiou et al. “GeoKnow: Making the web an exploratory place for geospatial knowledge”. In: *ERCIM News* 96 (2014), pp. 12–13.

[3] Sören Auer, Jens Lehmann, and Sebastian Hellmann. *Linkedgeodata: Adding a spatial dimension to the web of data*. Springer, 2009.

[4] David Aumueller et al. “Schema and ontology matching with COMA++”. In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM. 2005, pp. 906–908.

[5] Hannah Bast et al. “Route planning in transportation networks”. In: *Algorithm Engineering*. Springer, 2016, pp. 19–80.

[6] Montserrat Batet et al. “An information theoretic approach to improve semantic similarity assessments across multiple ontologies”. In: *Information Sciences* 283 (2014), pp. 197–210.

[7] Richard Bellman. “On a routing problem”. In: *Quarterly of applied mathematics* 16.1 (1958), pp. 87–90.

[8] Annabell Berger, Martin Grimmer, and Matthias Müller-Hannemann. “Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks”. In: *International Symposium on Experimental Algorithms*. Springer. 2010, pp. 35–46.

[9] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. “Generic schema matching, ten years later”. In: *Proceedings of the VLDB Endowment* 4.11 (2011), pp. 695–701.

[10] Christian Bizer. “Evolving the Web into a Global Data Space.” In: *BNCOD*. Vol. 7051. 2011, p. 1.

[11] Christian Bizer, Tom Heath, and Tim Berners-Lee. “Linked data-the story so far”. In: *Semantic Services, Interoperability and Web Applications: Emerging Concepts* (2009), pp. 205–227.

[12] Paolo Bouquet, Luciano Serafini, and Stefano Zanobini. "Semantic coordination: a new approach and an application". In: *The Semantic Web-ISWC 2003*. Springer, 2003, pp. 130–145.

[13] Daniela F Brauner et al. "An Instance-based Approach for Matching Export Schemas of Geographical Database Web Services." In: *GeoInfo*. 2007, pp. 109–120.

[14] Bin Cao et al. "Sharek: A scalable dynamic ride sharing system". In: *Mobile Data Management (MDM), 2015 16th IEEE International Conference on*. Vol. 1. IEEE. 2015, pp. 4–13.

[15] Alessandro Cecconi. "Integration of cartographic generalization and multiscale databases for enhanced web mapping". In: (2003).

[16] Michelle Cheatham and Pascal Hitzler. "String similarity metrics for ontology alignment". In: *International Semantic Web Conference*. Springer. 2013, pp. 294–309.

[17] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. "A comparison of string metrics for matching names and records". In: *Kdd workshop on data cleaning and object consolidation*. Vol. 3. 2003, pp. 73–78.

[18] William W Cohen. "Data integration using similarity joins and a word-based information representation language". In: *ACM Transactions on Information Systems (TOIS)* 18.3 (2000), pp. 288–321.

[19] Alberto Colorni and Giovanni Righini. "Modeling and optimizing dynamic dial-a-ride problems". In: *International transactions in operational research* 8.2 (2001), pp. 155–166.

[20] Sergio Consoli et al. "Producing linked data for smart cities: The case of Catania". In: *Big Data Research* 7 (2017), pp. 1–15.

[21] Kenneth L Cooke and Eric Halsey. "The shortest route through a network with time-dependent internodal transit times". In: *Journal of mathematical analysis and applications* 14.3 (1966), pp. 493–498.

[22] Olivier Corby, Rose Dieng, and Cédric Hébert. "A conceptual graph model for w3c resource description framework". In: *Conceptual Structures: Logical, Linguistic, and Computational Issues* (2000), pp. 468–482.

[23] Isabel F Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. "Agreement-Maker: efficient matching for large real-world schemas and ontologies". In: *Proceedings of the VLDB Endowment* 2.2 (2009), pp. 1586–1589.

[24] Káthia Marçal De Oliveira et al. "Transportation ontology definition and application for the content personalization of user interfaces". In: *Expert Systems with Applications* 40.8 (2013), pp. 3145–3159.

[25] Daniel Delling, Thomas Pajor, and Renato F Werneck. "Round-based public transit routing". In: *Transportation Science* 49.3 (2014), pp. 591–604.

[26] Thomas Devogele. "A new Merging process for data integration based on the discrete Fréchet distance". In: *Advances in spatial data handling*. Springer, 2002, pp. 167–181.

[27] Thomas Devogele, Christine Parent, and Stefano Spaccapietra. "On spatial database integration". In: *International Journal of Geographical Information Science* 12.4 (1998), pp. 335–352.

[28] Julian Dibbelt et al. "Intriguingly simple and fast transit routing". In: *Experimental Algorithms*. Springer, 2013, pp. 43–54.

[29] Edsger W Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[30] Hong-Hai Do and Erhard Rahm. "COMA: a system for flexible combination of schema matching approaches". In: *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment. 2002, pp. 610–621.

[31] AnHai Doan and Alon Y Halevy. "Semantic integration research in the database community: A brief survey". In: *AI magazine* 26.1 (2005), p. 83.

[32] Marc Ehrig. *Ontology Alignment: Bridging the Semantic Gap, volume 4 of Semantic Web And Beyond Computing for Human Experience*. 2007.

[33] Marc Ehrig and Steffen Staab. "QOM–quick ontology mapping". In: *The Semantic Web–ISWC 2004*. Springer, 2004, pp. 683–697.

[34] Marc Ehrig and York Sure. "Ontology mapping–an integrated approach". In: *The Semantic Web: Research and Applications*. Springer, 2004, pp. 76–91.

[35] Jérôme Euzenat. "An API for ontology alignment". In: *The Semantic Web–ISWC 2004*. Springer, 2004, pp. 698–712.

[36] Jérôme Euzenat, Pavel Shvaiko, et al. *Ontology matching*. Vol. 333. Springer, 2007.

[37] Jérôme Euzenat, Petko Valtchev, et al. "Similarity-based ontology alignment in OWL-lite". In: *ECAI*. Vol. 16. 2004, p. 333.

[38] Robert W Floyd. "Algorithm 97: shortest path". In: *Communications of the ACM* 5.6 (1962), p. 345.

[39] Masabumi Furuhata et al. "Ridesharing: The state-of-the-art and future directions". In: *Transportation Research Part B: Methodological* 57 (2013), pp. 28–46.

[40] Aldo Gangemi et al. "Sweetening wordnet with dolce". In: *AI magazine* 24.3 (2003), p. 13.

[41] *Geographic information - Geography Markup Language (GML) (ISO 19136:2007)*. Standard. 2009.

[42] *Geographic information - Spatial data infrastructures - Part 1: Reference model*. Standard. 2012.

[43] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. "S-Match: an algorithm and an implementation of semantic matching". In: *ESWS*. Vol. 3053. Springer. 2004, pp. 61–75.

[44] Michael B Gurstein. "Open data: Empowering the empowered or effective data use for everyone?" In: *First Monday* 16.2 (2011).

[45] Fayçal Hamdi et al. "Alignment-based partitioning of large-scale ontologies". In: *Advances in knowledge discovery and management*. Springer, 2010, pp. 251–269.

[46] Fayçal Hamdi et al. "Geomrdf: A geodata converter with a fine-grained structured representation of geometry in the web". In: *arXiv preprint arXiv:1503.04864* (2015).

[47] Fayçal Hamdi et al. "TaxoMap in the OAEI 2009 alignment contest". In: *The Fourth International Workshop on Ontology Matching*. 2009.

[48] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[49] Oktie Hassanzadeh et al. "A declarative framework for semantic link discovery over relational data". In: *Proceedings of the 18th international conference on World wide web*. ACM. 2009, pp. 1101–1102.

[50] Wei Hu, Yuzhong Qu, and Gong Cheng. "Matching large ontologies: A divide-and-conquer approach". In: *Data & Knowledge Engineering* 67.1 (2008), pp. 140–160.

[51] Yan Huang et al. "Large scale real-time ridesharing with service guarantee on road networks". In: *Proceedings of the VLDB Endowment* 7.14 (2014), pp. 2017–2028.

[52] *Intelligent transport systems - DATEX II data exchange specifications for traffic management and information - Part 1: Context and framework*. Standard. 2011.

[53] *Intelligent transport systems - Geographic Data Files (GDF) - GDF5.0 (ISO 14825:2011)*. Standard. 2011.

[54] *Intelligent transport systems - Public transport - Identification of Fixed Objects in Public Transport (IFOPT)*. Standard. 2012.

[55] *Intelligent transport systems - Traffic and travel information messages via traffic message coding - Part 1: Coding protocol for Radio Data System -*

Traffic Message Channel (RDS-TMC) using ALERT-C (ISO 14819-1:2013). Standard. 2013.

[56] Afraz Jaffri, Hugh Glaser, and Ian C Millard. "Managing URI synonymity to enable consistent reference on the Semantic Web". In: *In Proceedings of the Workshop on Identity, Reference, and the Web (IRSW) at ESWC2008*. Citeseer. 2008.

[57] Prateek Jain et al. "Ontology alignment for linked open data". In: *The Semantic Web–ISWC 2010*. Springer, 2010, pp. 402–417.

[58] Yves R Jean-Mary, E Patrick Shironoshita, and Mansur R Kabuka. "Ontology matching with semantic verification". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 7.3 (2009), pp. 235–251.

[59] Yannis Kalfoglou and Marco Schorlemmer. "Ontology mapping: the state of the art". In: *The knowledge engineering review* 18.01 (2003), pp. 1–31.

[60] Michael M Kostreva and Malgorzata M Wiecek. "Time dependency in multiple objective dynamic programming". In: *Journal of Mathematical Analysis and Applications* 173.1 (1993), pp. 289–307.

[61] Spyros Kotoulas et al. "Spud semantic processing of urban data". In: *Web Semantics: Science, Services and Agents on the World Wide Web* 24 (2014), pp. 11–17.

[62] James Larson, Shamkant B Navathe, Ramez Elmasri, et al. "A theory of attributed equivalence in databases with application to schema integration". In: *Software Engineering, IEEE Transactions on* 15.4 (1989), pp. 449–463.

[63] Robert Laurini and Derek Thompson. *Fundamentals of spatial information systems*. Vol. 37. Academic press, 1992.

[64] Jon Jay Le Grange et al. "The GeoKnow Generator: Managing Geospatial Data in the Linked Data Web". In: *Linking Geospatial Data* (2014).

[65] Luiz André P Paes Leme et al. "Instance-based OWL schema matching". In: *Enterprise Information Systems* (2009), pp. 14–26.

[66] Marie-Jeanne Lesot, Maria Rifqi, and Hamid Benhadda. "Similarity measures for binary and numerical data: a survey". In: *International Journal of Knowledge Engineering and Soft Data Paradigms* 1.1 (2008), pp. 63–84.

[67] Juanzi Li et al. "Rimom: A dynamic multistrategy ontology alignment framework". In: *Knowledge and Data Engineering, IEEE Transactions on* 21.8 (2009), pp. 1218–1232.

[68] Shuo Ma, Yu Zheng, and Ouri 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.

[69] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. "Generic schema matching with cupid". In: *VLDB*. Vol. 1. 2001, pp. 49–58.

[70] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. "Similarity flooding: A versatile graph matching algorithm and its application to schema matching". In: *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE. 2002, pp. 117–128.

[71] Eduardo Mena et al. "OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies". In: *Distributed and parallel Databases* 8.2 (2000), pp. 223–271.

[72] Luc Donnet Michel Etienne and Marc Florisson. *Dossier d'architecture Chouette*. http://www.chouette.mobi/wp-content/uploads/20170201-Chouette-V3.4-Architecture-V1.6.odt. 2017.

[73] George A Miller. "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11 (1995), pp. 39–41.

[74] Matthias Müller-Hannemann and Karsten Weihe. "Pareto shortest paths is often feasible in practice". In: *International Workshop on Algorithm Engineering*. Springer. 2001, pp. 185–197.

[75] Axel-Cyrille Ngonga Ngomo and Sören Auer. "Limes-a time-efficient approach for large-scale link discovery on the web of data". In: *integration* 15 (2011), p. 3.

[76] Ian Niles and Adam Pease. "Towards a standard upper ontology". In: *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*. ACM. 2001, pp. 2–9.

[77] Natalya F Noy and Mark A Musen. "Anchor-PROMPT: Using non-local context for semantic matching". In: *Proceedings of the workshop on ontologies and information sharing at the international joint conference on artificial intelligence (IJCAI)*. 2001, pp. 63–70.

[78] Ariel Orda and Raphael Rom. "Minimum weight paths in time-dependent networks". In: *Networks* 21.3 (1991), pp. 295–319.

[79] Ariel Orda and Raphael Rom. "Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length". In: *Journal of the ACM (JACM)* 37.3 (1990), pp. 607–625.

[80] M Tamer Özsu and Patrick Valduriez. *Principles of distributed database systems*. Springer Science & Business Media, 2011.

[81] Manfred Padberg and Giovanni Rinaldi. "A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems". In: *SIAM review* 33.1 (1991), pp. 60–100.

[82] Jeff Z Pan. “Resource description framework”. In: *Handbook on ontologies*. Springer, 2009, pp. 71–90.

[83] Julien Plu and François Scharffe. “Publishing and linking transport data on the web: extended version”. In: *Proceedings of the First International Workshop on Open Data*. ACM. 2012, pp. 62–69.

[84] *Public transport - Network and Timetable Exchange (NeTEx) - Part 1: Public transport network topology exchange format*. Standard. 2014.

[85] *Public transport - Reference data model - Part 1: Common concepts*. Standard. 2016.

[86] *Public transport - Service interface for real-time information relating to public transport operations - Part 1: Context and framework*. Standard. 2015.

[87] Bastian Quilitz and Ulf Leser. “Querying distributed RDF data sources with SPARQL”. In: *European Semantic Web Conference*. Springer. 2008, pp. 524–538.

[88] Erhard Rahm and Philip A Bernstein. “A survey of approaches to automatic schema matching”. In: *the VLDB Journal* 10.4 (2001), pp. 334–350.

[89] Erhard Rahm, Hong-Hai Do, and Sabine Maßmann. “Matching large XML schemas”. In: *ACM SIGMOD Record* 33.4 (2004), pp. 26–31.

[90] Yves Raimond, Christopher Sutton, and Mark B Sandler. “Automatic Interlinking of Music Datasets on the Semantic Web.” In: *LDOW* 369 (2008).

[91] Yves Raimond et al. “The Music Ontology.” In: *ISMIR*. Citeseer. 2007, pp. 417–422.

[92] Lionel Savary and Karine Zeitouni. “Automated linear geometric conflation for spatial data warehouse integration process”. In: *The 8th AGILE Conference on GIScience. Estirl, Portugal*. 2005.

[93] François Scharffe and Jérôme Euzenat. “MeLinDa: an interlinking framework for the web of data”. In: *arXiv preprint arXiv:1107.4502* (2011).

[94] François Scharffe, Yanbin Liu, and Chuguang Zhou. “Rdf-ai: an architecture for rdf datasets matching, fusion and interlink”. In: *Proc. IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US)*. 2009.

[95] François Scharffe et al. “Enabling linked data publication with the Datalift platform”. In: *Proc. AAAI workshop on semantic cities*. 2012, No–pagination.

[96] Frank Schulz, Dorothea Wagner, and Karsten Weihe. “Dijkstra’s algorithm on-line: an empirical case study from public railroad transport”. In: *Journal of Experimental Algorithmics (JEA)* 5 (2000), p. 12.

[97] Dennis Shasha, Jason TL Wang, and Rosalba Giugno. "Algorithmics and applications of tree and graph searching". In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2002, pp. 39–52.

[98] Shashi Shekhar and Hui Xiong. *Encyclopedia of GIS*. 1st. Springer Publishing Company, Incorporated, 2007. ISBN: 0387359753, 9780387359755.

[99] Bilong Shen, Yan Huang, and Ying Zhao. "Dynamic ridesharing". In: *SIGSPATIAL Special* 7.3 (2016), pp. 3–10.

[100] Amit P Sheth. "TO SEMANTICS". In: *Interoperating geographic information systems* 495 (1999), p. 5.

[101] Pavel Shvaiko and Jérôme Euzenat. "A survey of schema-based matching approaches". In: *Journal on Data Semantics IV*. Springer, 2005, pp. 146–171.

[102] Pavel Shvaiko and Jérôme Euzenat. "Ontology matching: state of the art and future challenges". In: *Knowledge and Data Engineering, IEEE Transactions on* 25.1 (2013), pp. 158–176.

[103] Marek Smid and Petr Kremen. "OnGIS: Semantic Query Broker for Heterogeneous Geospatial Data Sources". In: *Open Journal of Semantic Web (OJSW)* 3.1 (2016), pp. 32–50.

[104] Claus Stadler et al. "Linkedgeodata: A core for a web of spatial open data". In: *Semantic Web* 3.4 (2012), pp. 333–354.

[105] Giorgos Stoilos, Giorgos Stamou, and Stefanos Kollias. "A string metric for ontology alignment". In: *The Semantic Web–ISWC 2005*. Springer, 2005, pp. 624–637.

[106] *Traffic and Travel Information (TTI) - TTI via Transport Protocol Experts Group (TPEG) Extensible Markup Language (XML) - Part 1: Introduction, common data types and tpegML (ISO/TS 24530-1:2006)*. Standard. 2006.

[107] *UTMC Framework Technical Specification*. Standard. 2009.

[108] Julius Volz et al. "Silk-A Link Discovery Framework for the Web of Data." In: *LDOW* 538 (2009).

[109] Holger Wache et al. "Ontology-based integration of information-a survey of existing approaches". In: *IJCAI-01 workshop: ontologies and information sharing*. Vol. 2001. Citeseer. 2001, pp. 108–117.

[110] Sascha Witt. "Trip-based public transit routing". In: *Algorithms-ESA 2015*. Springer, 2015, pp. 1025–1036.

[111] Sascha Witt. "Trip-Based Public Transit Routing Using Condensed Search Trees". In: *arXiv preprint arXiv:1607.01299* (2016).

[112] Shuxin Yuan and Chuang Tao. "Development of conflation components". In: *Proceedings of Geoinformatics, Ann Arbor* (1999), pp. 1–13.

[113] Li Yujian and Liu Bo. "A normalized Levenshtein distance metric". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1091–1095.

[114] Stefano Zanobini. "Semantic coordination: the model and an application to schema matching". PhD thesis. PhD thesis, International Doctorate School in Information and Communication Technology, University of Trento, Trento (IT), 2006.