Universidade Federal do Ceará

Departamento de Computação
Mestrado e Doutorado em Ciência da Computação

&

Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica
Settore Scientifico Disciplinare: INF/01

Ph.D. Thesis

Recommending places based on the wisdom-of-the-crowd

Igo Ramalho Brilhante

Supervisor	Supervisor	Supervisor
Prof. Jose de Macedo	Prof. Dino Pedreschi	Dr. Chiara Renso
Federal University of Ceará	Università degli Studi di Pisa	ISTI - CNR

Abstract

The collective opinion of a great number of users, popularly known as *wisdom of the crowd*, has been seen as powerful tool for solving problems. As suggested by Surowiecki in his books, large groups of people are now considered smarter than an elite few, regardless of how brilliant at solving problems or coming to wise decisions they are. This phenomenon together with the availability of a huge amount of data on the Web has propitiated the development of solutions which employ the wisdom-of-the-crowd to solve a variety of problems in different domains, such as recommender systems, social networks and combinatorial problems.

The vast majority of data on the Web has been generated in the last few years by billions of users around the globe using their mobile devices and web applications, mainly on social networks. This information carries astonishing details of daily activities ranging from urban mobility and tourism behavior, to emotions and interests. The largest social network nowadays is Facebook, which in December 2015 had incredible 1.31 billion mobile active users, 4.5 billion "likes" generated daily. In addition, every 60 seconds 510 comments are posted, $293,000$ statuses are updated, and 136,000 photos are uploaded. This flood of data has brought great opportunities to discover individual and collective preferences, and use this information to offer services to meet people's needs, such as recommending relevant and interesting items (e.g. news, places, movies). Furthermore, it is now possible to exploit the experiences of groups of people as a collective behavior so as to augment the experience of other. This latter illustrates the important scenario where the discovery of collective behavioral patterns, the wisdom-of-the-crowd, may enrich the experience of individual users. In this light, this thesis has the objective of taking advantage of the wisdom of the crowd in order to better understand human mobility behavior so as to achieve the final purpose of supporting users (e.g. people) by providing intelligent and effective recommendations. We accomplish this objective by following three main lines of investigation as discussed below.

In the first line of investigation we conduct a study of human mobility using the wisdom-of-the-crowd, culminating in the development of an analytical framework that offers a methodology to understand how the points of interest (PoIs) in a city are related to each other on the basis of the displacement of people. We experimented our methodology by using the PoI network topology to identify new classes of points of interest based on visiting patterns, spatial displacement from one PoI to another as well as popularity of the PoIs. Important relationships between PoIs are mined by discovering communities (groups) of PoIs that are closely related to each other based on user movements, where different analytical metrics are proposed to better understand such a perspective.

The second line of investigation exploits the wisdom-of-the-crowd collected through user-generated content to recommend itineraries in tourist cities. To this end, we propose an unsupervised framework, called TripBuilder, that leverages large collections of Flickr photos, as the wisdom-of-the-crowd, and points of interest from Wikipedia in order to support tourists in planning their visits to the cities. We extensively experimented our framework using real data, thus demonstrating the effectiveness and efficiency of the proposal. Based on the theoretical framework, we designed and developed a platform encompassing the main features required to create personalized sightseeing tours. This platform has received significant interest within the research community, since it is recognized as crucial to understand the needs of tourists when they are planning a visit to a new city. Consequently this led to outstanding scientific results.

In the third line of investigation, we exploit the wisdom-of-the-crowd to leverage recommendations of groups of people (e.g. friends) who can enjoy an item (e.g. restaurant) together. We propose GroupFinder to address the novel user-item group formation problem aimed at recommending the best group of friends for a $< user, item >$ pair. The proposal combines user-item relevance information with the user's social network (ego network), while trying to balance the satisfaction of all the members of the group for the item with the intra-group relationships. Algorithmic solutions are proposed and experimented in the location-based recommendation domain by using four publicly available Location-Based Social Network (LBSN) datasets, showing that our solution is effective and outperforms strong baselines.

Acknowledgements

First of all, I thank God first for this achievement. Thanks to my girlfriend, Clara Rebouças, for sharing the hard times, lovely and unforgettable moments, and for always motivating me to improve and to take a step forward.

I thank my family: my father, Newton, who always guided me with wisdom, showing me the ways of life; my mother, Ana Paula, for the immeasurable affection; my brother, Hugo, and sister, Izabelle. I thank my grandmothers, my aunts and uncles, godmother and godfather, and cousins for all the words of encouragement.

I thank my friends and university colleagues, Regis Pires, Samara Martins, Ivanildo Barbosa, Ticiane Linhares, Livia Almada, Paulo Rego, Alex, Samara, Josue, Pedro, Romulo, Rodrigo and Janaina. Thanks to friends, colleagues and teachers of CNR: from KDDLab Roberto Trasarti, Salvatore Rinzivillo, Paolo Cintia, Lorenzo Gabrielli, Riccardo Guidotti, Giulio Rossetti, Letizia Milli, Francesca Pratesi, Vittorio Romano, Chiara Falchi, Brunella Falchi, Mirco Nanni, Anna Monreale, Barbara Furletti, Caterina D'Angelo, prof. Dino Pedreschi and Fosca Giannotti; and from HPC Lab, Massimo Coppola, Patrizio Dazzi, Claudio Lucchese, Salvatore Orlando, Fabrizio Silvestri and Cristina Muntean. Thanks to my great friends David Araujo, Luca Pappalardo, Vinicius Monteiro and Amilcar Soares Jr.

I thank the teachers of the Federal University of Ceara for the teachings during this long period at the University. Thanks to Prof. Pierpaolo Degano for all efforts to achieve the collaboration between Federal University of Ceara and University of Pisa.

A very special thanks to my friend, partner and supervisor Prof. Jose Antonio F. de Macedo for the numerous and valuable discussions, contributing to this work, as well as for my professional growth. Thanks to my supervisor Chiara Renso who welcomed me during my stay in Pisa, guiding and giving me all the necessary support not only for work, but also for unforgettable moments. A very special thank to Raffaele Perego and Franco Maria Nardini, for teaching me priceless lessons and for directly collaborating in the achievements of this thesis.

Contents

List of Figures

List of Tables

Publications and Awards

Publications

Journals

1. Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, Chiara Renso: On planning sightseeing tours with TripBuilder. *Information Processing & Management* (IPM) 51(2): 1-15 (2015)

Conferences

1. Igo Ramalho Brilhante, Michele Berlingerio, Roberto Trasarti, Chiara Renso, José Antônio Fernandes de Macêdo, Marco Antonio Casanova: ComeTogether: Discovering Communities of Places. *Mobility Data Management* (MDM) 2012: 268-273

2. Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, Chiara Renso: Where shall we go today?: planning touristic tours with TripBuilder. *Conference on Information and Knowledge Management* (CIKM) 2013: 757-762

3. Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, Chiara Renso: TripBuilder: A Tool for Recommending Sightseeing Tours. *European Conference on Information Retrieval* (ECIR) 2014: 771-774

4. Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, Chiara Renso: Group Finder: an Item-driven Group Formation Framework. *Mobility Data Management* (MDM) (2016)

Workshops

1. Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, Chiara Renso: Scaling up the Mining of Semantically-enriched Trajectories: TripBuilder at the World Level. *Italian Information Retrieval Workshop* (IIR) 2015

1. Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, Chiara Renso: User-Item Group Formation with GroupFinder. *Italian Information Retrieval Workshop* (IIR) 2016

Special Tracks

1. Igo Ramalho Brilhante, José Antônio Fernandes de Macêdo, Franco Maria Nardini, Raffaele Perego, Chiara Renso: Planning sightseeing tours using crowdsensed trajectories. *SIGSPATIAL Special* 7(1): 59-66 (2015)

Awards

Best demo award

1. TripBuilder: A Tool for Recommending Sightseeing Tours Igo Ramalho Brilhante, Jose Antonio Macedo, Franco Maria Nardini, Raffaele Perego, Chiara Renso. *European Conference on Information Retrieval (ECIR) 2014 participants elected the winner of the ECIR 2014 best demo award.*

Chapter 1

Introduction

1.1 Context and Challenges

The collective opinion of a great number of users, popularly known as *wisdom of the crowd*, has been seen as powerful tool for solving problems. As suggested by Surowiecki in his books [134], large groups of people are now considered smarter than an elite few, regardless of how brilliant at solving problems or coming to wise decisions they are. This phenomenon together with the availability of a huge amount of data on the Web has propitiated the development of solutions which employ the wisdom-of-the-crowd to solve a variety of problems in different domains, such as recommender systems [128], social networks [100] and combinatorial problems [152, 151].

The vast majority of data on the Web has been generated in the last few years by billions of users around the globe using their mobile devices and web applications, mainly on social networks. This information carries striking details of daily activities ranging from urban mobility and tourism behavior, to emotions and interests. The largest social network nowadays is Facebook, which in December 2015 had incredible 1.31 billion mobile active users, 4.5 billion “likes” generated daily. In addition, every 60 seconds 510 comments are posted, $293,000$ statuses are updated, and 136,000 photos are uploaded[1]. This flood of data has brought great opportunities to discover individual and collective preferences, and use this information to offer services to meet people’s needs, such as recommending relevant and interesting items (e.g. news, places, movies). Furthermore, it is now possible to exploit the experiences of groups of people as a collective behavior so as to augment the experience of other. This latter illustrates the important scenario where the discovery of collective behavioral patterns, the wisdom-of-the-crowd, may enrich the experience of individual users. We strongly believe that the collective opinion and experience is better than individual ones: *Vox populi, vox Deli.*

These opportunities, therefore, bring several challenges on exploiting the wisdom-of-the-crowd to augment and develop new services to support users. First of all, the collection process and the mining of crucial information concerning the users’ behavior can be remarkably noisy. Secondly, there is a need to develop methodologies able to encompass all the elements, such as user preferences

[1]Source: `https://zephoria.com/top-15-valuable-facebook-statistics/`

and interests, spatial-temporal data, points of interest features, and so on.

Motivated by the challenges and the identified directions in the presented context, this thesis focuses on the problem of recommending personalized sightseeing tours for tourists that are visiting a new destination. This research topic has been receiving much attention in the last few years due to the enormous potential in the tourism environment for both industrial and scientific communities. According to *World Travel Tourism Council*[2], travel and tourism generated two trillion dollars directly in the global GDP and by 2025, international tourist arrivals are forecast to total $1,796,210,000$, generating expenditure of USD2,140.1bn, an increase of 4.2% pa[3]. These statistics therefore highlight the potential of tourism and the need to provide new solutions for the problems inherent in this area.

These facts combined with availability of a huge amount of data on the Web relating to millions of users has inspired us to exploit the benefits and peculiarities of the phenomenon of the wisdom-of-the-crowd in the context of studying previous tourist behavior in order to provide recommendations for personalized sightseeing tours for future tourists. Several key tasks are involved: (i) collecting data to represent the wisdom-of-the-crowd; (ii) integrating the wisdom-of-the-crowd with data sources about tourism available on the Web, like points of interest and their categories (e.g. museums, natural world); (iii) mining and discovery of essential information about the points of interest and tourist behavior; (iv) considering the nature of the tours, i.e. if they are based on individuals or groups of users; (v) dealing with spatio-temporal data that can represent the displacement of tourists in the city.

In front of these tasks, the proposal of this thesis is to investigate how to exploit the wisdom-of-the-crowd as the collective behavioral pattern to support individual users in "finding" the most suitable sightseeing tour based on their needs and preferences. In the next section, we describe the hypothesis and research questions investigated in this thesis.

1.2 Hypotheses and Research Questions

Complex challenges are posed when dealing with data about millions of people both for industry (e.g. tourism and e-commerce) and for the scientific community. Understanding in such a way as to identify implicit laws in order to build new business models or leverage existing ones is certainly a big issue to deal with. A second issue is the mining and discovery of patterns hidden in the data in order to explain some phenomena that govern our society. Several studies have been conducted in order to create theoretical frameworks to explain some phenomena. We can cite the paper by González *et al.* [72] reporting a study of $100,000$ anonymized mobile phone users which revealed that human trajectories show a high degree of temporal and spatial regularity and that individual travel patterns collapse into a single spatial probability distribution. Interestingly, this indicates that, despite the diversity of their travel history, humans follow simple reproducible patterns. These results show important considerations regarding the impact on phenomena driven by human mobility, such as epidemic prevention, emergency response and urban planning. It is

[2]http://www.wttc.org/

[3]Source: https://www.wttc.org/-/media/files/reports/economic\%20impact\%20research/regional\%202015/world2015.pdf

also worthwhile citing the astonishing studies about the spread of epidemic diseases on complex systems organized as *networks* [108, 109, 119, 94].

Considering these observations and the objective of this thesis, the main hypotheses that guided our work are presented below:

1. The mobility of users (e.g. people) can characterize points of interest, thus the movement of people between points of interest in their daily activities can lead to a different approach to group them together, based on the movements;
2. The wisdom-of-the-crowd based on both location-based social networking services and user-generated description of points of interest can leverage the development of a tourism knowledge base to favor the creation of personalized sightseeing tours for tourists thinking of visiting a new destination;
3. The wisdom-of-the-crowd produced from social network data combined with recommender system frameworks can contribute to identify meaningful groups of users who can enjoy a given item (e.g. city, music) together.

The first hypothesis motivated us to raise the following research question:

RQ1. *Can we study urban mobility on a global scale from the perspective of places, instead of users?*

This question led us to conduct a study of human mobility using datasets collected from enabled-GPS cars in three Italian cities. The resulting development of an analytical framework offers a methodology to understand how the points of interest of a city are related to each other from the displacement of people, on the basis of important and useful features of the points of interest.

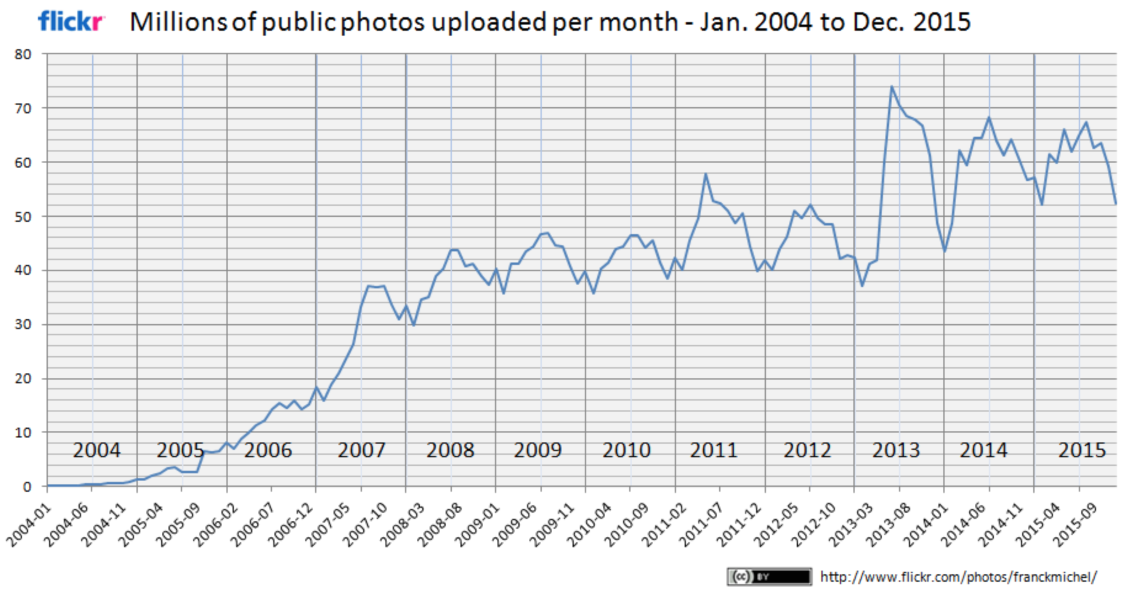


Figure 1.1: Empirical analysis of a dataset of public available on Flickr. Figure from [104].

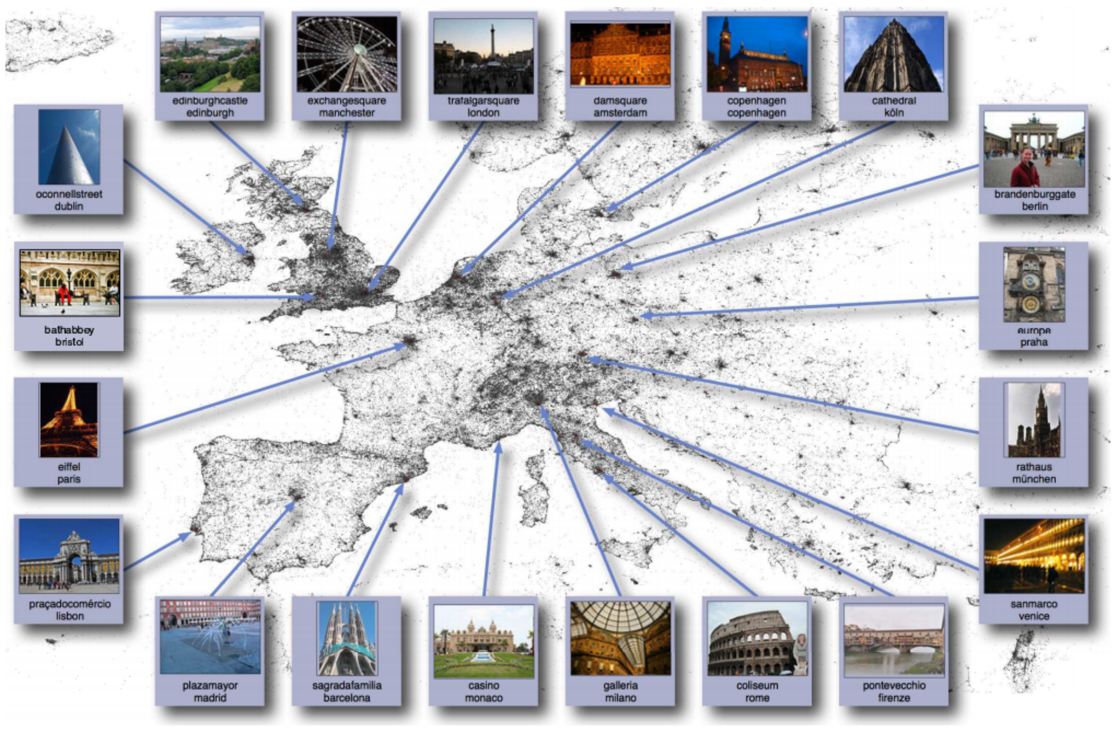


Figure 1.2: Representative photos found from 33, 393, 835 photos from 307, 448 Flickr users. Figure from [52].

Our first results gave us some insights stating that discovered patterns of people movement between points of interest in a city can contribute to generating new knowledge useful in several applications. As we have seen earlier, users tend to share data about their daily activities, and this happens especially when on holiday, or when visiting new places. The latter is the reason behind the second hypothesis, which guided us to investigate the possibility of exploiting geo-tagged photos from Flickr. This approach can leverage a knowledge based construction to support the creation of personalized sightseeing tours. An empirical study, illustrated in Figure 1.1 [4], shows remarkable behavior by Flickr users when uploading photos. The number of publicly uploaded photos is significant, since they represent users who share their photos for access by other users. An interesting finding relates to the peaks of each year. It is worthwhile noting that the peaks take place around the months of July and August, implying that users tend to upload more photos during their vacations. These results indicate that the large collection of Flickr photos could be useful to our task: taking advantage of the wisdom-of-the-crowd as a knowledge base to support users (e.g. tourists) visiting a new place. However, we still need some insights that could demonstrate, even empirically, that the uploaded photos could (in some way) represent interesting points of interest in the city, instead of just random objects found by the users.

Crandall *et al.* in [52] conducted a study of how to organize a large collection of geo-tagged photos collected from Flickr. The interesting result is that representative photos of the given area (e.g. clustering area) are usually related to the tourist attractions as shown in Figure 1.2. These

[4]Figure from `https://www.flickr.com/photos/franckmichel`

results highlight the powerful data available that can represent the typical tourist behavior in the cities which could provide essential information for new tourists. Inspired by these results, we raised the following research question:

RQ2. *Can we take advantage of the data provided by millions of users, also called wisdom-of-the-crowd, to support users (e.g. tourists) in planning their vacations to a new destination?*

As we have seen, the growth of available data about user activities may provide uncountable outcomes once we know how to appropriately deal with and manage it. On this basis, a second contribution we propose in this thesis is an unsupervised framework, called TripBuilder, which leverages large collections of Flickr photos and points of interest from Wikipedia to support tourists in those cities. More specifically, our framework creates a tourist knowledge base capturing the tourists' movement behaviors from Flickr photos combined with points of interest from Wikipedia to create personalized sightseeing tours in a given city considering the time allowance of the user and preferences. Based on this theoretical framework, we designed and developed a platform encompassing the main features required to create personalized sightseeing tours. This platform has been crucial to understanding tourists' needs when they are planning a visit to a new city. The designed system has been mentioned in the Brazilian news[5] and Communications of the ACM[6].

The third hypothesis motivated us to investigate how to exploit social ties between users in order to form meaningful groups of users to enjoy a given item, like a city, restaurant, etc.

It is worth noting that some application services may not be appropriate for individual users. In some situations, recommendations to groups of users are more relevant than individual ones. Consider for example, a day-trip, when people usually go with companions to share the travel experience. Other examples may be going to restaurant with a group of friends, or watching a certain film in the cinema. Social networks represent and highlight friendship networks and the high level of interaction between groups of users who are friends. Then, some works have been carried out in the literature to provide services for groups of users. In particular, the problem of recommending items (e.g. movies, books, etc) to a group of users [79, 121, 25] has been investigated. However, in some situations the group is not known a priori, in which case the user could be recommended an activity for a group of friends to enjoy (e.g. restaurant).

Considering the scenario of TripBuilder, some people usually go on their vacations with friends who are also interested in the destination. Therefore, we investigate the following research question:

RQ3. *How can we find out the best groups of users (e.g. friends) who can enjoy a given item together?*

In this thesis we investigate this problem by exploiting social networks jointly with recommender systems. In particular, we take advantage of the recommendations of items for users and how friends are linked to each other in order to find out the "best" groups of friends to enjoy a specific item. In this thesis, we present our framework GroupFinder that aims to deal with this issue.

[5] `http://goo.gl/Mmc09w`
[6] `http://goo.gl/yegM1z`

1.3 Thesis Contribution

The main contributions of this thesis are the following.

- We conducted a study on human mobility considering points of interest as the central objective from the perspective of complex networks. We presented a framework that encompasses useful features for points of interest that can be mined from GPS data. We experimented the methodology in three Italian cities highlighting the most important findings and comparing the results for each city. The results were published in [30].

- We proposed TripBuilder, an unsupervised framework for planning personalized sightseeing tours in cities. To this purpose we use: i) Flickr, to gather public photos (and their meta data), ii), Wikipedia to gather information regarding PoIs in the given city, iii) Google maps to estimate the time needed to move from one PoI to the next one in the sightseeing itinerary. The resulting knowledge base stores PoIs, their popularity, the time needed on average to visit them, the categories for which each PoI is relevant, and the patterns of movement of tourists that visited them in the past. In order to assess our system, we report on the building of a knowledge base covering three Italian cities which are important for tourism and guarantee variety and diversity in terms of size and the wealth of public user-generated content available: Rome, Florence, and Pisa. The resulting knowledge base, available for download to favor the reproducibility of results, is analyzed and its characteristics are here discussed. Finally, we report on several new experiments to evaluate the effectiveness and efficiency of all the components of our system and show that our solution outperforms competitive baselines. In particular, we assess TripBuilder's performance in providing budgeted sightseeing itineraries made up of actual PoI patterns tailored to the specific preferences of the tourist. The results were published in [34, 35, 31, 29].

- We design and develop a platform built upon TripBuilder to evaluate the proposed algorithms and the methodology to create the tourism knowledge base. In this platform, the user is able to create their personalized sightseeing tour considering the amount of time available and their preferences. The user can also balance personalization and popularity to modify the suggested tour. In addition, the platform has some social capabilities allowing the users to share the tours in such a way that the new user can re-use them as they need. The results of this chapter were published in [28].

- We formalize the user-item group formation problem aimed at recommending the best group of friends for a $< user, item >$ pair. We address this novel problem by combining user-item relevance information with the user social network (ego network), trying to balance the satisfaction of all the members of the group for the *item* with the intra-group relationships. We propose two different solutions that are accommodated into a framework called GroupFinder, which integrates the needed components and information sources. We instantiate the problem in the location-based recommendation domain and we experiment GroupFinder on four publicly available Location-Based Social Network (LBSN) datasets, showing that our solution is effective and outperforms strong baselines. The results were published in [36, 32].

1.4 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2 we present the state of the arts related to this thesis. In particular, we present works related to mobility data analysis and complex networks. Then we go through recommender systems and location-based recommender systems. Later, the works involving group recommendation are introduced. Finally, we present a recent research regarding the group formation problem.

In Chapter 3 we present a study on mobility data from the perspective of places instead of users. Here, we study how places are related to each other based on the movement of people between the places. In Chapter 4 we present how to build a knowledge base for tourism from Flickr photos and points of interest from Wikipedia to design an unsupervised framework to create personalized sightseeing tours – TripBuilder. Chapter 5 presents a user-friendly web application built upon TripBuilder allowing users to create their own personalized sightseeing tours. In Chapter 6 we present a novel framework called GroupFinder that finds the best group of friends of a given user and a suitable item on the basis of user's preferences (recommendations) and their social relationships. Finally, in Chapter 7 we draw the conclusions and describe future works based on this thesis.

Chapter 2

Related Works

This chapter presents the works related to this thesis. First, we present works concerning analysis of mobility data in Section 2.1. Then, we give basic foundations in Recommender Systems in Section 2.2 to support Section 2.3 which presents the important results in location-based recommender system. Finally, the works in group recommendation research fields are discussed in Section 2.4 and Section 2.5.

2.1 Mobility Data Analysis, Mining and Networks

2.1.1 Trajectories

Human mobility is a complex phenomena witnessed by a huge amount of interdisciplinary research in this topic, ranging from Physics to Sociology, Transportation Research and Computer Science [68, 67, 163]. In this sense, many efforts in the community have been done to develop new techniques to support better understanding of human mobility. The main object considered in this study is usually a *trajectory* of a *moving object* (e.g. person).

A trajectory is usually defined as the spatio-temporal evolution of a moving object (e.g. person). This evolution is typically represented as a sequence of positional observations represented by x and y coordinates of time-stamped sample points as collected by a tracking device, such as GPS tools or WI-FI sensors. However, many applications require more than coordinates, i.e., there is the need for semantic information inherent to the trajectories, which is usually done by means of annotations. A trajectory that has been enhanced with *annotations* is the definition of semantic trajectory as proposed by Parent *et al.* [118]. These annotations include the common case of "stop and moves" where segments of a trajectory representing the absence of movement are called "stops" while the parts representing the actual movement are called "moves" [132]. Depending on the application, stops and moves can be annotated in several ways, but it is common to associate stops to the visited points of interest [44]. In this thesis, we exploit semantic trajectory as the basis of the presented approach where the trajectories are seen as sequence of points of interest visited by the users.

Trajectories representing the movement evolution of individuals have witnessed an increasing interest in the last decade, especially due to the increasing availability of personal tracking device, ranging from GSM phone to the more sophisticated GPS-enables smart-phones, and the popularization of location-based services (LBS). The potentialities offered to several application domains by the analysis of huge amount of positioning data has opened new opportunities for developing analytical methods of this new form of data. Mobility data analysis has become a hot research topic since several methods on data mining and statistical techniques, tailored to trajectory data, have been proposed in the literature, like [68, 67, 163].

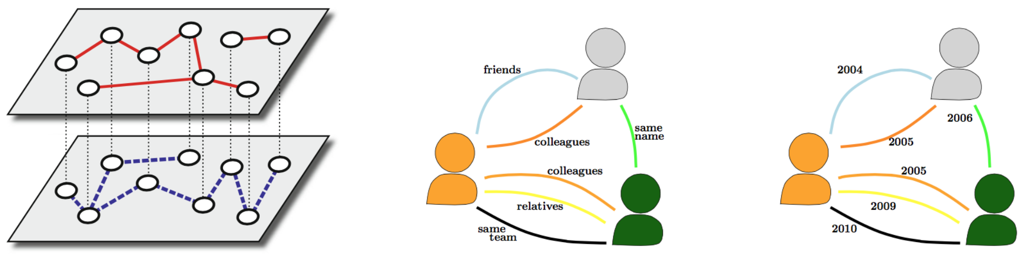
The task of analyzing large trajectory datasets can be carried out towards different directions. Basic statistics may be applied to trajectory data mainly to discover the distributions of people presence and origin-destination matrices [41]; other studies focus on trajectory data mining aiming at finding correlations in large datasets of positioning data [68]. Techniques to extract movement patterns include: (1) *clustering discovery* - finding groups of objects moving together – the authors in [105] propose a time-focused clustering of trajectories based on OPTICS algorithm [7]; (2) *sequential pattern discovery* - finding the most frequent sequences of places visited – the authors in [106] propose an algorithm to discovery *T-Pattern* from a trajectory dataset; (3) *flock detection* - extracting the convergence of people moving together for a certain amount of time [56, 68, 143]. From the analytics frameworks and data mining algorithms, software tools have been developed to encompass state-of-the-arts algorithms to deal with mobility data and to develop new algorithms with a core framework for testing and validating the results [135]. These previous techniques are mainly based on the geometric properties of trajectories thus trying to extract similarities or common behavior from the spatio-temporal dimension of the data. However, the semantic information is still missing and it is an important feature to be considering in this scope. This is the reason why a new research trend is growing to exploit these semantic rich information.

2.1.2 Networks

Besides the methods previously presented, some interesting works have envisioned how the objects interact with each other at a global scale. This perspective is usually associated to the paradigm of *complex networks*. The study of networks, or Network Science, is broadly interdisciplinary and important developments have occurred in many fields, including mathematics, physics, computer and information sciences, biology, and the social sciences [111] and have been receiving increasing attention by the scientific community, also due to the availability of massive network data from diverse domains, and the outbreak of novel analytical paradigms, which pose relations and links among entities, or people, at the center of investigation.

Networks are usually modeled as a graph $\mathcal{G} = (V, E)$, where the set of nodes V represent the involved entities and the set of edges E stand for any relationship between the entities. Depending on the domain and the objective, the graph may be undirected (social networks), directed (WWW networks), weighted or unweighted. Recently a new class of networks has been investigated to model many real-world complex systems where the same set of notes are connected via more than one type of connection, such as living organisms, human society and transportation system and critical infrastructure [22, 90] (see Figure 2.1).

Inspired by real-world scenarios such as social networks [4, 42], technology networks [1], the



(a) (b)

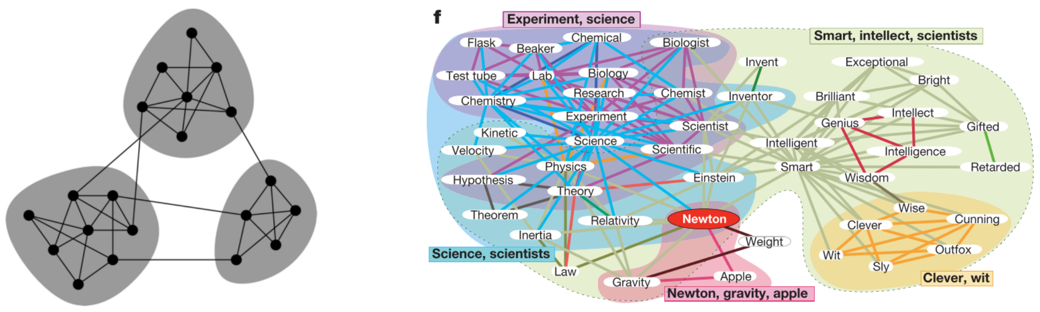
Figure 2.1: Two multiplex/multidimensional networks illustations: (a) network of nine nodes with two layers, the red (solid) and the blue (dashed) layer from [90]; (b) two social networks with different types of links between the users from [22].

World Wide Web [92, 57], biological networks [81, 82], and human movement [72],[145] the last few years have seen a wide, multidisciplinary, and extensive research devoted to the extraction of non-trivial knowledge from such networks. Predicting future links among the actors of a network [113, 37], detecting and studying the diffusion of information among them [71, 149], mining frequent patterns of users' behaviors ([17, 148, 48]), are only a few examples of problems studied in Complex Network Analysis. In this way, interesting researches are focused on the interplay between complex network and mobility data. A typical example is the study of spreading of cell phone viruses thru GSM phone calls [145, 13]. Besides of statistical analytics of the network as a whole, how densely connected nodes form groups is another important field in complex networks. These groups are called *communities*.

Community discovery in complex networks is a topic that is gaining more and more interest in the literature [50, 107, 129, 9, 97, 159]. Several approaches has been proposed from divisive graph partition algorithms, to random walk based approaches, from label propagation based methods, to clique percolation techniques. However, the literature is still missing a unique definition of the concept of community, and the diverse techniques lead all to different results, sometimes hard to compare to each other. Although a few measures of the quality of the results have been proposed so far (among which, the modularity), their definitions are still questionable (the modularity, for example, has a well known problem of resolution, and approaches that try to maximize it tend to create very large communities). A classification of community discovery methods is proposed in [50]. The authors classify the methods based on the different definitions of communities in the literature. Communities may involve several features like *overlapping*, *weighted* and/or *directed links*, and *dynamics*. Figure 2.4 illustrates two networks with the corresponding communities found using different methods.

Consequently, some works have taken advantage of community discovery techniques to understand mobility data. In [59], the authors propose a methodology to cluster trajectories by building a network of trajectories, where the links represent the similarity between two trajectories considering some constraints. Then, they apply a community algorithm based on modularity optimization in order to discover groups of trajectories that behaved similarly and that moved along the same portions of the road network.

The authors in [38] propose an approach to extract place-focused communities from social graphs by annotating the edges with check-in information from location-based social network like Foursquare (thru users' check-ins) to show the possibility to extract groups of friends who meet face-to-face for benefiting on-line social services. The authors propose a collection of co-location measures to evaluate the effectiveness of the approach. The main results suggest that the approach can find place-focused groups where users are often co-located. Yet, in [39] an interesting study over social and place-focused communities is presented. The authors investigate the evolution of tie structure within communities, concluding that the time period over which location data are aggregated has a substantial impact on stability of place-focused communities (communities of users that visit the same places). In particular, the authors investigate communities found in co-located networks of users that checked in at the same places and the on-line social network. In conclusion, local communities may be more useful than social communities for providing friend or points of interest recommendation when geographic information is considered.



(a) (b)

Figure 2.2: Example of communities found in two networks: (a) three communities found using modularity by Newman *et al.* (Figure from [110]); (b) communities found using the link community algorithm by Ahn *et al.* (Figure from [3]).

2.2 Recommender Systems

The growth of the Internet brought a series of new applications with a rich gamma of interactions mechanisms with these applications, which became the essential "windows" for the users to look for targeted contents, items, objects of their interest. For instance, news websites that provide many information about the society, sports, economics and so on; or e-commerces that sell different products to a broad range of target users. However, the number of available items in these systems has rapidly increased in such a way that users could not easily find what they need, making them give up from the search and, in the case of e-commerce, give up of the purchase. This problematic scenario leveraged the necessity for better strategies to help the users in finding the most relevant items to them from a huge collection of available items. This scenario led to the development of one of the most important topics in the last decade: *Recommender Systems* (RS).

Recommender systems are tools to support individuals in finding items from a large number of alternatives that a system, e.g e-commerce, may offer to them. The first idea of recommendation is likely the suggestion of the most popular items for the users under the assumption that most users are fine with the popular ones. As popular items were not enough anymore, new and sophisticated recommender systems have been developed to generate personalized and novel suggestions to the users, where these suggestions are highly relevant to the user preferences. Therefore, representing the the typical behavior and preferences of the user about the items in the systems became a key task to design effective recommender systems.

The common term to represent the entities in recommender systems is *item*. The items are defined according to the domain of the recommender system, for instance books, musics, movies, events, venues, foods and cities, to name a few. Indeed, recommender system is a multidisciplinary topic that can be incorporate into different applications and problems. Recommendation of movies (Netflix), music and artists (Spotify and LastFM), venues (Foursquare), news (Yahoo News) are examples of different applications and domains implementing recommender systems as mechanism to help users in finding the most relevant items to them.

Formally, recommender system has basis in an utility function that evaluates the importance or relevance of an item for a user. Let us denote $\mathcal{I} = \{i_1, \dots, i_m\}$ the set of items, and $\mathcal{U} = \{u_1, \dots, u_n\}$ the set of users, the utility function is usually represented by a function

$$R : U \times I \rightarrow \mathbb{R}.$$

The design of the recommender system determines the output of $R(\cdot, \cdot)$ which may also output integer values (e.g. $[1, 5]$), instead of real ones. This objective function is then used to evaluate the items for each user in order to generate a subset or a sorted list of items to the user. In the literature we find the recommendation problem treated as a prediction problem whose goal is to predict a score of the item i for the user u.

A taxonomy of recommender systems is provided in [40] that has become a classical way of distinguishing between recommender systems [121]. This taxonomy includes: *Content-based* where item content is used to match up against the user profile; *Collaborative Filtering* where ratings patterns are discovered to generate recommendations; *Demographic* whose recommendations are generated taking into consideration the demographic profile of the user, such as age, language and country; *Knowledge-based*, where the goal is to measure the utility of the recommendation for user by estimating how much the user needs match the recommendations; *Community-based* which recommends items based on the preferences of users friends; and *Hybrid* recommender systems that combine other RS techniques together using the advantages of one to fix the limitation of the other one.

In this thesis we discuss two of those classes of recommender systems that are most used: *content-based* recommender systems, where item content (e.g. movie year, actors) is taken into consideration to represent the items and to create user profile that are matched up against the item attributes in order to measure the relevance of the item for the user as a result of the recommendation process; and *collaborative filtering* recommender systems, where rating patterns based on the historical data of the users are analyzed over the items to generate suggestions through similarities between users or items, or they are used to learn a predictive model that is able to effectively predict or estimate the relevance of an item for the user.

2.2.1 Content-based Recommender Systems

A content-based recommender system (CBRS) has basis on the items' content to recommend similar items to those ones that the user has already liked before. The similarity of items is calculated based on the features associated with the compared items. CBRS approaches analyze the features of the items previously rated by a user to build a profile of user interests based on the features of the items rated by that user. The recommendation processing, then, consists in matching up the attributes of the user profile against the features of the item to be recommended. The result is a relevance score/ judgment that represents the user's level of interest in that item. The more accurate the profile is, the more effective the recommendations will be. Therefore, an important step in the content-based recommender systems is the technique used for item representation.

The items are represented by a set of features, also called *attributes* or *properties* [98]. In movie applications, for example, the year of the movie, actors, directors, description can be used as features for the items. A simple way to represent the items is then to use keywords-based profiles. This approach is especially suitable when each item is described by the same set of attributes and the possible values for each feature is known. In the case of textual description, keywords-based profiles are not effective as item representation, since simple string matching operation can not deal with *polysemy*, the presence of multiple meanings for one word, and *synonymy*, where multiples words have the same meaning [98].

A simple and very used model for representing the items is the Vector Space Model (VSM) broadly used in Information Retrieval (IR). In particular, VSM is used to spatially represent text documents, where each document can be seen as a vector in a m-dimensional space. Then, each dimension in the document vector corresponds to a term from the overall vocabulary of the document collection, which is weighted to indicate the degree of relevance between the document and the term. In content-based recommender system, this model can be used in such a way the items and users correspond to the documents, while the items' features are the terms of the overall vocabulary.

Let $T = \{t_1, \ldots, t_m\}$ be the set of terms in our vocabulary and $D = \{d_1, \ldots, d_n\}$ be the set of documents. Therefore, each document d_i is represented by its m-dimensional vector space $\vec{d_i} = \{w_{1i}, \ldots, w_{mi}\}$ such that w_{kj} is the relevance of the t_k for document d_j.

To evaluate how relevant a term t is to a document k, w_{tk}, we first need to point out important observations that help us to design the adequate weight function. As discussed in [98, 123]: (i) frequent terms are not necessary more relevant than rare terms; single occurrences of a term in a document are not more important than multiple occurrences; and documents with many terms are not more suitable than documents with less terms. TF-IDF (Term Frequency-Inverse Document Frequency) was developed based on these observations regarding text being the most commonly used term weighting framework. The intuition behind TF-IDF is that terms that occur frequently in one document and are not frequently found in many other documents are more likely to be relevant to the document, while frequent items that occur in several documents are not representative for a specific document. To compute TF-IDF, we need first to compute the term frequency of a term

t_k in a document d_i given by

$$\text{TF}(t_k, d_i) = \frac{f_{k,i}}{\max_z f_{z,i}},$$

where $f_{k,i}$ is the number of occurrence of term t_k in the document d_i, and $\max_z f_{z,i}$ stands for the maximum occurrence of any term z in any document i. With term frequency computed, we can calculate TF-IDF as

$$w_{k,i} = \text{TF-IDF}(t_k, d_i) = \text{TF}(t_k, d_i) \cdot log\frac{N}{n_k},$$

where n is the number of documents in the collection and n_k is the number of documents that have the term t_k. Analyzing the inverse term frequency component $log\frac{n}{n_k}$ we can see that the final score for TF-IDF is higher when n_k is lower, and lower for large n_k. This means that the term frequency of a term t_k in the document d_i is penalized if t_k occurs in many other documents.

Once we have computed the document vector $\vec{d_i}$, we can rely on a similarity function to find similar documents d_i with respect to a given vector (e.g. user profile vector) in the same m-dimensional space. Cosine similarity is broadly used to compute the similarity between two vector of an inner product space to measure the cosine of the angle between them given by:

$$cosine(\vec{d_i}, \vec{d_j}) = \frac{\vec{d_i} \cdot \vec{d_j}}{||\vec{d_i}|| \cdot ||\vec{d_j}||}.$$

Therefore, for a user profile vector $\vec{u}$ in the same m-dimensional space, we can compute the cosine similarity to find out documents that are relevant or similar to the user profile. The vector space model jointly with a vector-based similarity function are simple and very efficient ways for recommending items, mostly due to its simplicity and flexibility to be applied in different domains, such as music, movies, books, venues, etc.

Lops *et al.* in [98] highlight that keyword-based representations for the items and user profiles can give accurate performance, when the sufficient number of evidence of the user interests is available. However, this approach is not suitable for all applications. As previously discussed, keyword-based methods have some problems regarding *polysemy* and *synonymy*, what can lead to inaccurate results by the recommender system. To deal with this problem, an ontology-based representation might be used to integrate the recommender system with external knowledge bases to provide more semantic in the user profiles.

The Space Vector Model can then be used as a framework for the content-based recommender system. In the case of CBRS, the documents are the items and users, while the terms are the features associated with the items. In this way, we represent items and users as feature vectors in such a way similar items to the user profile can be found as recommendations to the user.

The content-based recommender systems have the advantage of: (i) *User Independence*, since the RS exploits the ratings provided by the user to build her own profile, and it does not need to compute the other users ratings as done in collaborative approaches; (ii) explaining how the recommender system works can be provided by explicitly listing the item features that caused the recommendation of that item - *Transparency*; (iii) overcoming the new item problem (item not rated by any user), once the item features can be match up against the user profile even when no user has rated that item.

The content-based recommender systems, on other hand, also have several drawbacks: (i) Domain knowledge is often needed, what might be problematic if the content of items (features) are not enough to discriminate items the user likes from items the user does not like. Therefore, automatic discovery and manual assignment of features to items could not be sufficient to define distinguishing aspects of items to capture and model the user interest; (ii) these RSs have the drawback of over specialization, where only items similar to those items previously rated by the user will be recommended, thus it does not favor for serendipity recommendations (unexpected recommendations); (iii) new users do not have enough ratings or feedbacks to create their profile, thus the system will not be able to provide reliable recommendations.

Not all of the item contents are available, which forces us to design different recommendation techniques from the content-based ones. In particular, when the ratings of users are present in the system, these can be used to discover patterns to support the recommendations of the items. These patterns may indicate users having similar preferences or behaviors and thus items of one could be used as a recommendation to the other; or they may be used to learn model that are capable of assisting the recommendations. In the next section we discuss the recommendation systems based collaborative filtering which uses the patterns on the recommendations of the items.

2.2.2 Collaborative Filtering Recommender Systems

This class of recommender systems relies on past user behavior to analyze relationships between users according to their interests in the items to find out new user-item associations corresponding to the recommendations to the users. The term *collaborative filtering* was devised by the developers of Tapestry, the first collaborative filtering recommender system [85]. This approach is an alternative to content-based methods when item contents are not available, and the user past behaviors (e.g. ratings, interactions with the system) can be taken into account to generate the recommendations. User past behavior is can be seen as *feedbacks* of the users that represent important information to boost the recommender systems: *explicit feedbacks* are associated with the explicit interest of the user about certain items usually given in the form of ratings (stars, like, dislike); and *implicit feedbacks* are indirect representation of the interests of the user by observing past behaviors such as the interaction of the user with the RS, browsing history and purchase, for instance.

The collaborative filtering recommender systems are mainly divided into two areas: *neighborhood-based* and *model*-based collaborative filtering techniques. In the neighborhood-based CF algorithms, similar users to the active user are identified to find out relevant items from these similar users. In the model-based CF algorithms, on the other hand, the ratings are used to learn a predictive model to produce recommendations for new items. In the following we discuss with more details each of these approaches.

Neighborhood-based Collaborative Filtering

Neighborhood-based methods, known also as k nearest neighbors (kNN), are inspired by the common principle of *word-of-mouth* [54]: "*one relies on the opinion of like-minded people or other* *trusted sources to evaluate the value of an item (movie, book, articles, album, etc.) according to his own preferences*". Let us consider the example illustrated in Figure 2.3. Adam has enjoyed three movies that were also enjoyed by other users. These in turn experienced other films that might be interesting to Adam. Therefore, the system might identify these users to somehow recommend the items.

Based on this observation, neighborhood-based techniques exploit the user-item ratings stored in the system to device groups of users, called *neighbors*, for the active user that have similar preferences in order to predict ratings for the new items. The neighborhood-based methods are approached in two ways: *user-based* or *item-based* recommendations. In user-based systems, the interest of a user u for a given item i is evaluated by the other users ratings for this item, the neighbors, that have similar rating patterns [54]. Item-based approaches [91], on the other hand, evaluate the relevance of an item i to a user u based on the ratings of u for items similar to i. In this case, two items are similar if users of the system tend to similarly rate them.

Let $r_{u,i}$ be the rating of user u for item i. The general framework for neighborhood-based methods consists of finding similar users by accomplishing a similarity computation between users (user-based) or items (item-based). The objective of this process is to find out the neighbors of each user in order to recommend items from a group of users with similar taste (user-based), and items that are similarly rated by the same set of users for a given item (item-based). Then, the ratings of the group for the item are somehow aggregated as a prediction score of the item. When the task is to generate a top-N recommendation, it is needed to select the k most similar users or items based on the similarity computation, and then aggregate the neighbors to get the top-N most relevant or scored items as the recommendation or prediction computation [133]. Each step is discussed as follows.

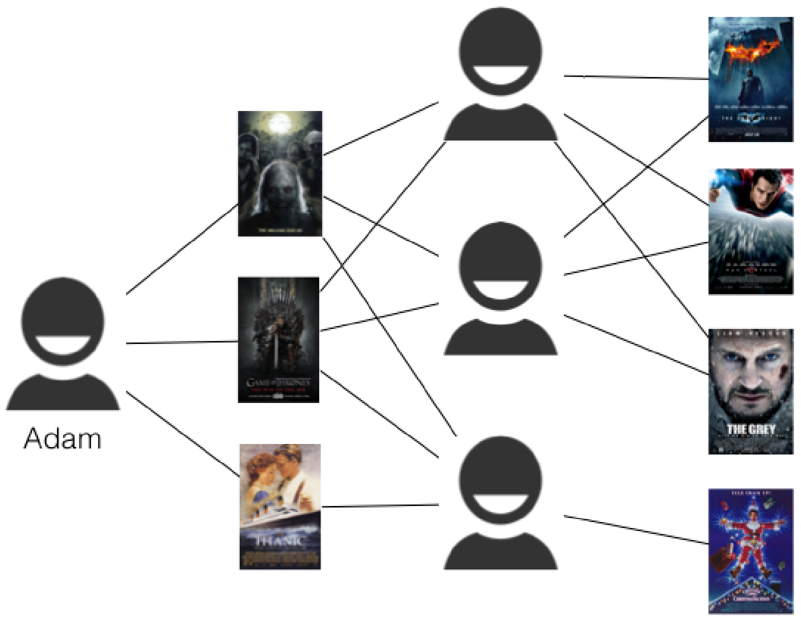


Figure 2.3: Example of neighborhood-based collaborative filtering for each group of users similar to the active user are identified to find out possible interesting items as recommendation.

Similarity Computation. In the case of user-based, given two users u and v, we need to compute the similarity between these two users according to their rating patterns. A common way to accomplish it is to compute the *Pearson correlation* between the two users, but other correlation-based similarities can also be used (e.g. Spearman rank correlation), as well as vector cosine-based similarity.

Let $I_{u,v} \subseteq \mathcal{I}$ is the item set summarizing the items that both the users u and v have rated, and $\bar{r}_u, \bar{r}_u$ are the average rating for the co-rated items in I of the user u and v, respectively. Considering the Pearson correlation as our measure, the similarity between two users u and v is given by

$$w_{u,v} = \frac{\sum_{i \in I_{u,v}} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{u,v}} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{u,v}} (r_{v,i} - \bar{r}_v)^2}}, \tag{2.1}$$

For item-based, we need to compute the similarity between two items i and j that were rated by the same users. Let $U_{i,j} \subset \mathcal{U}$ be the set of users who rated both items i and j, $\bar{r}_i$ and $\bar{r}_j$ the average rating of items i and j given by the users in $U_{i,j}$. The Pearson correlation is between the items i and j is then computed by

$$w_{i,j} = \frac{\sum_{u \in U_{i,j}} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{i,j}} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U_{i,j}} (r_{u,j} - \bar{r}_j)^2}}, \tag{2.2}$$

Prediction and Recommendation Computation. With the similarities between users or items computed to find out the neighbors, an aggregation of their ratings is used to predict the relevance of the items for the user. For the user-based methods, we can rely on the *weighted sum of the others' ratings* [133, 120]. Then, the recommendation of item i to user u is given by

$$R(u,i) = \bar{r}_u + \frac{\sum_{v \in U_{i,j}} (r_{v,i} - \bar{r}_v) \cdot w_{u,v}}{\sum_{v \in U_{i,j}} |w_{u,v}|} \tag{2.3}$$

For item-based, we might use a simple weighted average

$$R(u,i) = \frac{\sum_{j \in I_u} r_{u,j} \cdot w_{i,j}}{\sum_{j \in I_u} |w_{i,j}|}, \tag{2.4}$$

where I_u is the set containing all items rated by u [124].

Top-N Recommendation. Once we are able to compute $R(u,i)$ for a user u and item i, we can devise a top-N recommendation which consists of recommending the N most relevant items to the user, achieved by user-based or item-based algorithms. Regarding the user-based approach, the first step is to find similar k users, the neighbors, to the active user using Pearson correlation or any other similarity function. Then the relevance of the item i for the user u $R(u,i)$ is calculated for each item experienced by the neighbors. Finally, the top-N items not experienced by the user are recommended. For item-based recommendation, The algorithm first computes the k most similar items for each item according to a similarity function, for instance, equation 2.2. Then the items not yet experienced by the user are retrieved and compared to what the user has rated (seen) using for example the equation 2.4. In the end, the top-N items more relevant in terms of $R(u, i)$ are recommended to the user.

The choice between user and item-based algorithms is taken from five important criteria that should be considered [54]: (i) accuracy; (ii) efficiency; (iii) stability; (iv) justifiability; (v) and serendipity.

Regarding accuracy, it is important to understand the ratio between the number of users and items in the system, where a low number of neighbors, but with high similarity, is preferable to a large number of neighbors whose similarities are not trustworthy, leading to poor quality recommendations. As discussed in [54], item-based can produce more accurate recommendations when the number of users is much greater than the number of items; while user-based neighborhood methods are preferable in the cases the number of items is larger than the number of users in the system.

As we have seen, neighborhood-based CF algorithms require a similarity computation over the users or items. Then, the efficiency of the methods also depends on the ratio between users and items. In other words, when the system has more users than items, the item-based algorithm is more efficient as less memory and computation are required for processing similarities between the items.

This fact leads us to consider also the stability of the number of users and items in system. Naturally, if new items are frequently added into the system, for instance, the similarity computation between them will need to be frequently achieved, making the choice of a user-based algorithm more preferable. When the list of available items is not frequently changed, however, item-based algorithms might be more suitable.

Justifiability is an important characteristic for recommender systems when there is requirement for the explanation of the recommendations to the users. This may contribute to engage the user and enforce the confidence with the system. Item-based algorithms have advantage of justifiability, that is, the recommendation of an item can be easily explained as the neighbors of this item and the recommendation scores can be presented to the user as the justification for that recommendation. In the case of user-based, the justifiability can be difficult since the user may not know the neighbors, unless social networks are incorporated into the method.

Another very important aspect to be considered is related to serendipitous or novelty recommendations. Recommendations generated by item-based methods are highly related to the items the user has rated. As a result, the RS will recommend very similar items to those the user already rated, which does not favor for different items that might be relevant to the user – serendipitous or novel recommendations. In user-based algorithms, on the other hand, the user might be recommended by items provided by highly similar neighbors that might contribute to recommend types of items (e.g. movie genre) not seen or expected by the user, but still being relevant.

Neighborhood-based recommender systems are broadly used in different settings: with social networks, a variety of domains, etc. Spite of presenting some limitation, such as the *cold start* problem [133, 126, 2, 165], these approaches are still relevant and worthwhile due their simplicity and effectiveness for some applications [73].

Model-based Collaborative Filtering Recommender Systems

On a different line, model-based collaborative filtering algorithms work out by exploiting training data of ratings to learn a predictive model. Techniques from machine learning and data mining, for instance, are applied to recognize complex patterns to produce effective recommendations to the users. Among the techniques used in the model-based algorithms, we can cite Bayesian Belief Net CF algorithms, Clustering, Regression-based, Support Vector Machines, Latent Semantic Analysis and Latent Dirichlet Allocation [27, 75, 23, 73, 74, 16, 83, 85]. Grcar *et al.* presents experimental results in [73] of confronting the neighborhood-based algorithm (kNN) with Support Vector Machine (SVM) in the collaborative filtering framework. Interesting results show us some conditions where one collaborative filtering technique is more appropriate than the other. Regarding kNN and SVM, the authors show that kNN has better performance in the recommendations when the quality of data is higher. On the other hand, SVM is especially more adequate for sparse datasets. These insights are extremely useful at the moment of designing and developing a recommender system that is able to capture the users' rating patterns and, consequently, to produce effective recommendations.

Some of the most successful achievements of model-based systems are based on *matrix factorization* [85], where, in its basic form, matrix factorization characterizes both items and users by vectors of factors inferred from item rating patterns. The flexibility of adding additional information for modeling real-life situations combined with good scalability and recommendation (predictive) accuracy has popularized matrix factorization methods. Some of the additional information include *implicit* and *explicitly* feedbacks, *biases* of particular users and items, input sources about users and items (e.g. content), temporal dynamics and confidence levels to distinguish observed ratings from different input source, e.g. explicit feedbacks might be more relevant than implicit feedbacks. Koren *et al.*, who took part of the winner team of the Netflix Prize, present different modeling for matrix factorization by aggregating those additional information [85]. An interesting result is the importance of temporal dynamics to achieve accurate recommendations as the users may change their rating patterns over time. These results also motivated the development of other time-aware recommender systems presented in [88, 89, 84, 64].

2.3 Recommender Systems for Location-based Services

Recommender systems are broadly present in several domains, like movies, music, books and places. The diversity of domains requires the specialization of RSs to deal with specific characteristics of the users and items for the target domain. Consequently, the recommendation of locations (e.g. restaurants, museums) also needs deeper investigation to achieve high-quality recommendations by understanding the users behavior when associated with locations or venues. In this section, we discuss a comprehensive list of state-of-the-art for location-based recommender systems.

The advances in location-acquisition technologies, smartphones devices and the Web 2.0 technologies have enabled the creation and popularization of location-based services (LBS), such as the location-based social networking services (LBSNs) like Foursquare[1], Twinkle and Geolife [162, 163].

[1]`https://foursquare.com/`

These services allow users to share their locations and location-related-content with other users, such as geo-tagged photos in Flickr, check-ins in Foursquare, etc. The location dimension links the gap between the physical world and the digital on-line social networking services, bringing new opportunities and challenges in traditional recommender systems. Among the opportunities and challenges Bao *et al.* highlight: (i) the *complex objects and relations* represented by the location as the item in the RS, generating new relations between users, between locations, and between users and locations. Consequently, new recommendation scenarios (e.g. location and itinerary recommendations) require new methodologies for generating high-quality recommendations; and (ii) the *rich knowledge* regarding the location as one of the most important components to define the user's context, where location history can provide means of mining extensive knowledge about the user's behavior and preferences to be accurately assessed by recommender systems [12].

We have witnessed a variety of location-based social networking services with the basis on location sharing, but with some particularities. For example, in Flickr users share their photos that may be geo-tagged indicating locations where the user has been; the sharing of check-ins in Foursquare where the user inform the venue she is at that moment; and the sharing of GPS-based data where users do not only share a position, but a sequence of coordinates representing their trajectories as presented in Section 2.1. In front of these heterogeneous services, Bao *et al.* proposed an interesting classification of the location-based social networking services:.

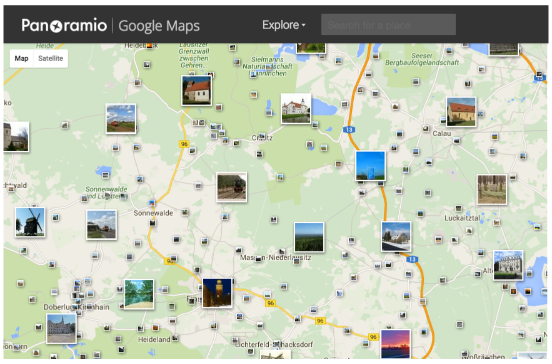
- **Geo-tagged-media-based**. It corresponds to those services in which users can add a location label to media content, such as text, photos and videos generated in the physical world by their devices. These tags are added to the content when it is created or added explicitly by the user. Example of these location-based social networking services are Flickr, Panoramio (Figure 2.4a), Twitter and Instagram.

- **Point-location-based**. Foursquare and Swarm (Figure 2.4c) are example of application with this service. The service allows the users to share their current locations, such as restaurants, museums or cities. With this kind of service, a venue (point of interest) is the main element determining the connections betweens users, while user-generated content such as comments (tips) and badges are associated with points of interest.

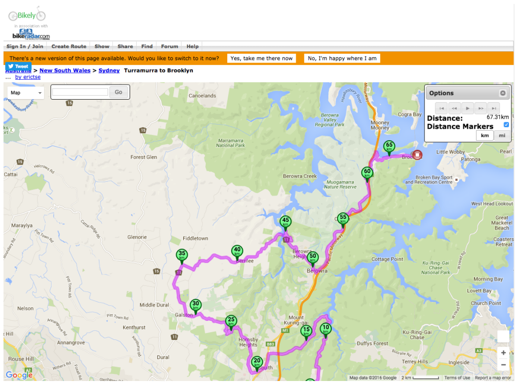
- **Trajectory-based**. In the trajectory-based social networking services like Bikely[2] (Figure 2.4b) and Runkeeper[3] users record both points of interest and routes connecting the points of interest. Other users can reference these experience by browsing the trajectory on a digital map or in the real world with a GPS-enable phone. The generated data are formed by sequences of geo-points (latitude and longitude) that might be enriched with semantic information, like points of interest and weather.

These classes show us how heterogeneous a recommender system might be designed for a location-based social networking. In front of this, Bao *et al.* in [12] proposed three taxonomies to categorize the recommender systems in LBSN according to: (i) the data source used by the

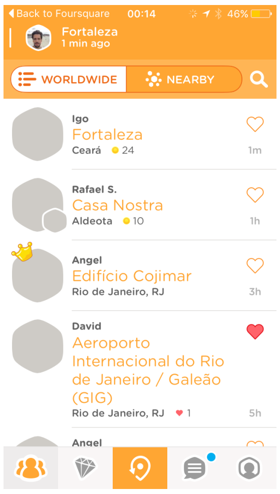
[2]`www.bikely.com/`
[3]`https://runkeeper.com/`



(a) Geo-tagged-media-based LBSN on Panoramio



(b) Trajectory-based LBSN on Bikely



Fortaleza
1 min ago

Igo
Fortaleza
Ceará 24
1m

Rafael S.
Casa Nostra
Aldeota 10
1h

Angel
Edifício Cojimar
Rio de Janeiro, RJ
3h

David
Aeroporto Internacional do Rio de Janeiro / Galeão (GIG)
Rio de Janeiro, RJ 1
5h

Angel

(c) Point-location-based LBSN on Swarm

Figure 2.4: Example of the three classes of location-based social networking: (a) shared geo-tagged photos by users in Panoramio; (b) trajectory shared by a user on Bikely; (c) venues sharing through users' check-ins on Swarm.

recommender system; (ii) the applied methodology for the recommendation; and (iii) the objective of the recommendation, such as location, users, activities or social media.

The categorization of recommender system according to the data source used is based on: the user profile, when data about the user (age, gender, preferences) are explicitly specified; the user geo-located content such as location, check-ins and geo-tagged social media; and the trajectories of the user representing her spatio-temporal evolution.

The authors also categorized the recommender system based on the methodology applied. They identified the content-based and collaborative filtering approaches which have basis on the traditional recommender system powered by the challenges of LBSN. In addition, the link analysis-based is another methodology used in the recommender system, including techniques like PageRank [117] and HITS (hypertext induced topic search) [87].

The last categorization is the recommendation objective. The recommender systems are classified according to their objectives:

1. **Location recommendation**. This category corresponds to the recommender systems fo-

cusing on locations as the major objective. Therefore, the objective is the recommendation of locations that can be twofolds: (i) stand-alone location recommendation, when users are recommended with individual locations based on the user's preference and possibly her location history; (ii) sequential location recommendation which recommends sequence of locations to be visited (e.g. tourist attractions) based on the user's preferences and some constraint inherent to the domain and application, such as time and cost. In this thesis we focus on this category of location-based recommender systems;

2. **User recommendation**. In some LBSN the objective is the recommendation of users, instead of locations. The recommender systems suggest friends based on user's location history, user with particular importance in the system such as local expertise and opinion leadership, communities of users in which the user might join based on her interests and activities.

3. **Activity recommendation**. As the name suggests, the recommendation objective is the activities the user can achieve in the locations. Therefore, the users are suggested by activities they may be interested at locations of their interest as well.

4. **Social media recommendation**. The objective is the recommendation of social media content like photos videos taking to consideration the user's location as well as the location meta-data of the social media.

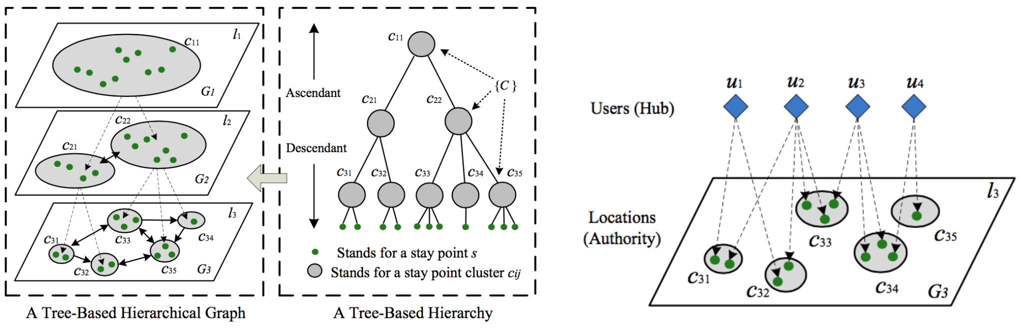
This thesis is mainly related to the location recommendation category. In particular, we introduce in Chapter 4 and 5 the proposed framework and system, respectively, for planning sightseeing tours in a city. Therefore, we present in the following sections some important achievements on this line for both stand-alone and sequential recommendations.

2.3.1 Stand-alone location recommendation

The most traditional format of location recommendation is likely the stand-alone. Here the recommendation outcome a list with the top N most relevant locations to the user considering some constraints or not. User's current location or past movements can be taken into account to devise the recommendation list.

In [76] the authors propose a system to make personalized recommendations of restaurants in a city. Their approach enhances the collaborative filtering solution by aggregating location information for generating recommendations.

A GPS-data-driven location-based social networking service is proposed in [162] where people can share their life experiences and connect to each other in the social network with their location histories. Then, in [164] the authors present a tree-base hierarchical graph (TBHG) to model multiple individuals' location histories jointly with a HITS (Hypertext Induced Topic Search)-based inference model to infer the interest of a location. The approach starts with GPS data of the users being mapped to sequences of stops, called *stay points*, determining spatial areas in which the user spent a minimum amount of time and within a maximum distance. To build the TBHG (Figure 2.5), it is needed to (i) cluster the stay points using a hierarchical clustering algorithm to build a tree-based hierarchy; and (ii) and build a graph at each level of the hierarchy connecting two consecutive clusters of the same level with a directed link. The built TBHG is then used to create the HITS-based inference model to estimate the users' experiences and location interests in a given region. In this model, the visit of a user to a cluster is represented as a directed link from the user to that cluster (Figure 2.5). A user is considered a hub if she has visited many clusters (locations), while a location is an authority if it has been accessed by many users. The generated scores for each user and location are used to find out the top-N locations in a given region. Note that the recommendation here is not personalized, i.e., the users are recommended with the same locations for a given N.



(a) (b)

Figure 2.5: (a) Tree-based Hierchical Graph on the left, and the Tree-based Hierarchy on the right; and (b) HITS-based inference model. Figures from [164].

Considering neighborhood-based CF schema, Xiao *et al.* propose a user-based similarity function considering the users' location history in [147]. They propose an algorithm called *maximal travel match*)(MTM), that considers semantic location history, i.e., like shopping malls, restaurant; mined from raw GPS data. Then, MTM algorithms evaluates the similarity between two users, what can be incorporated into a user-based collaborative filtering technique to make recommendations.

A step further location-based recommender system from GPS data is present in [161, 160]. In [161] the authors firstly model users' location (cluster of points) and activities histories. Then, location features and activity-activity correlations are mined from geographical databases and the Web, respectively, to use these data as input for a collective matrix factorization method. The location features are supported by a PoI database that gives the PoIs in an enclosing rectangle. Each location i is associated with a l-dimensional vector $\vec{q_i} = [q_{i1}, \ldots, q_{il}]$, for l different categories of PoIs, and q_{ij} the number of PoIs with the category j in the location i. Since some categories are more popular (e.g. restaurant) than others (e.g. cinema), the author apply TF-IDF (Section 2.2.1) over the feature vector as a normalization strategy. The activity-activity correlation is computed by query a search engine (e.g. Bing) given two activities, such as "Food and Drink" and "Shopping". The hit count returned by the search engine is then used as the correlation score. Finally, non-personalized locations and activities recommendations are made to the users using a matrix factorization algorithm (model-based collaborative filtering). This work is extended in [160], where the authors propose two other algorithms to incorporate users' features into the previous one: (i) a collective tensor and matrix factorization model is used; (ii) a ranking-based collective tensor and matrix factorization model is proposed. In these extensions, they proposed algorithms to make personalized recommendations, outperforming the previous one.

Lucchese *et al.* take advantages from Flickr geo-tagged photos and Wikipedia PoIs to recommend PoIs to tourist [101]. Their approach consists of creating an itinerary graph $G = (V, E, w)$, for the set of PoIs V, the set of edges E such that $e = (i, j)$ means that photos i and j are in the same album of at least one Flickr user or they share at least one Wikipedia category; and $w(i, j)$ is the number of Flickr user or Wikipedia categories. Later, an itinerary transition matrix is computed from G to estimate the conditional probability $P(j|i)$. Then, given a set U of PoIs representing already visited or interesting PoIs for a user, the random walk with restart computes the steady-state probability from each PoI in U. The results are aggregate using Hadamard product. They propose a random walk-based algorithm to recommend tourist points of interest. The algorithm takes advantage of knowledge mined from photo albums and Wikipedia to generate personalized recommendations of touristic points of interest according to the places previously visited by the user. This approach can be used to exploit a list of top-k points of interest, as well as when the user is currently visiting the city. They compared the approach with competitor to empirically evaluate its efficiency. Finally, the score for each item in $V \setminus U$ is found out to recommend the top-N most scored PoIs to the user.

A random walk-based approach for recommending venues from check-ins data is also proposed in [112]. In this work, the authors investigate issues related to behavioral, social and spatial information available in social networks to better generate recommendations. For this aim, they study the behavior of users in a large scale from two location-based social networking services: Foursquare and Gowalla. The proposed model is represented as a graph where the nodes are users and venues, where the steady-state probability is computed for each user by the random walk with restart framework and the transition matrix, considering the number of check-ins or not as weights for the edges in the graph. They compare their approach against relevant baselines, including neighborhood and model-based collaborative filtering algorithms, in 11 different cities. Another interesting results is that between 60% and 80% of users' visits are in venues that were not visited in the previous 30 days.

An interesting collaborative filtering approach is present in [12]. The authors propose a location-based and preference-aware recommender systems by considering user personal preferences and social opinions. The approach is divided into *off-line* and *on-line* modeling. In the *off-line* part, the individuals' personal preferences are modeled with a weighted category hierarchy (WCH) using the number of check-ins of the user at the venue normalized by the TF-IDF for each node in WCH; and the expertise of each user in a city with respect to different category of locations is inferred by estimating the expertise of the users for a given category using HITS computed over the relations between the users and the venues of that category. In the *on-line* phase, in turn, local expert users are selected in a spatial range that matches the user's preferences using a preference-aware candidate selection algorithm. A similarity function is proposed to compute the similarity between two users considering the WCH and its levels, instead of cosine-based similarity. These functions are then used in the user-based collaborative filtering schema. In the end, the top-N ranked locations are returned as the recommendation for the user.

In some application, not only the location is important, but also events that might take place somewhere. In front of this, Yin *et al.* [154, 153] propose a *location-content-aware recommender system* (LCARS) that recommends not only a set of locations, but also a set of events, such as concerts and exhibitions, in location-based social networking services. In this recommender systems, (spatial) items are locations or venues. Both local preference of the locations and item content information (for venues and events) are considered important for modeling user preferences and handling the data sparsity problem. LCARS is structured into two components: off-line modeling and on-line recommendation. The off-line modeling part, called LCA-LDA, is a location-content-aware probabilistic generative model, thus model-based collaborative filtering, that quantifies and incorporates both local preference and item content information into the spatial item recommendation process. It is designed to learn the interest of each individual user and the local preference of each individual city by capturing item co-occurrence patterns and exploiting item contents. The on-line recommendation part takes a querying user along with a querying city as input, and automatically combines the learned interest of the querying user and the local preference of the querying city to produce the top-N recommendations. The authors extend the Threshold-based Algorithm (TA) [61] to compute the top-N recommendations based on K sorted lists of latent topics discovered in LCA-LDA.

In [144], the authors propose a location-based recommender system for LBSNs based on users visited places, the location of each venue (e.g. restaurant), the social relationship among the users and the similarity between the users. The LBSN is seen as a graph containing two types of nodes (users and locations) and two types of edges to represent the friendship between users and the visits of users to the locations. They propose two algorithms based on the personalized Page Rank [80] by using the bookmark-coloring algorithm (BCA) [18]: *Friendship-based Bookmark-coloring Algorithm*, which takes into account only the friendship edges; and *Location-friendship Bookmark-coloring Algorithm* which reconciles social interaction and similarity in a common recommendation algorithm. The results highlight the importance of friendship and locations to get highly qualified recommendations.

Ye *et al.* also explore user preference, social influence and geographical influence for location (points of interest - POI) recommendations [150, 157]. They emphasize geographical influence due to the *spatial clustering phenomenon* exhibited in user check-in activities of LBSNs. The intuition is that users prefer to visit nearby locations rather than distant ones, and users may be interested in locations surrounded a location that users prefer. Therefore, they argue that the geographical influence among locations plays an important role in user check-in behaviors. To this aim, they propose a unified PoI recommendation framework, which fuses user preference to a POI with social influence and geographical influence. The proposed framework combines the ranked list of three collaborative filtering algorithms using a linear fusion framework. The CF approaches are: user-based CF using cosine similarity between two users and the venues they checked in; friend-based CF, which consider the user's friends, instead of neighbors, and the directional social influence of one user to another; and (iii) model-based CF using a naive Bayesian approach that accounts the probability of two checked in PoIs by the user are within a given distance in order to estimate a score for a new venue based on its distance to the user's check-in venues.

In [157], a survey on location/PoI recommendation is presented where the authors classify points of interest recommendation algorithms into four categories: pure check-in data based POI recommendation approaches, geographical influence enhanced POI recommendation approaches, social influence enhanced POI recommendation approaches and temporal influence enhanced PoI recommendation approaches.

2.3.2 Sequential location recommendation

The presented works so far have been focusing on recommending a ranked list of PoI/ places to the users. However, for the scenario where a user, e.g. tourist, would like to visit a sequence of places instead of picking one from the list, new methodologies are needed. In fact, many other works have been concentrated in this particular scenario that represents a variety of important applications in the context of tourism and urban activities.

Trips, *itineraries*, *travels*, *paths* and *sightseeing tours* or simply *tours* are some of those terms used to refer the sequence of places that is recommended to the users. In these works the main goal is to create personalized or non-personalized trips according to a set of constraints that are crucial in the problem definition. Due to the number and types of constraints, we can found different problem definitions depending on the objective and context of the proposal methodology. However, in general, the problem can be seen as a *trip planning problem.* In this section, we present the most relevant works whose aim is the recommendation or generation of itineraries for users.

Planning a travel itinerary or a trip is definitely a difficult and time-consuming task for tourists approaching their destination for the first time. Different sources of information such as travel guides, maps, on-line institutional sites and travel blogs are consulted in order to devise the right blend of Points of Interest (PoIs) that, a) best covers the subjectively interesting attractions, and, b) can be visited within the limited time planned for the travel. However, the user still needs to guess how much time is needed to visit each single attraction, and to devise a *smart* strategy to schedule them moving from one attraction to the next one. Furthermore, tourist guides and even blogs, reflect the point of view of their authors, and they may result to be not authoritative sources of information when the tourist preferences diverge from the most popular flow.

An early work on this topic is [70]. The authors use the Traveling Salesman Problem (TSP) as a starting problem to plan trips. A TSP with Activities and Lodging Selection (ALS) automatically selects PoIs and lodging. The Multiple Objective extension (MOTSP-ALS) minimizes transport and accommodation costs at the same time. The final step, i.e., Preference-based MOTSP-ALS, maximize the attractiveness of the lodging and the activities. The proposed model is very complex, and turns out to be very difficult to use also for the large amount of heterogeneous information required.

Yoon *et al.* propose a graph-based framework to recommend itineraries given a start and end point and the travel duration [155, 156]. First they generate a graph called $Location-InterestGraph$ from users' GPS trajectories by mining geographical regions where a user stay over a time threshold called *stay point.* Second, they find a list of feasible itineraries w.r.t to the given budget. Finally they sort the result by means of a location-interest metric that takes into account:

(i) *elapsed time ration* where itineraries that use as much available time as possible are considered to be better; (ii) *stay time ration* to favor the visiting time at the places instead of on the way traveling; (iii) *interest density ration* to highlight as many interesting locations as possible, i.e., popular locations and locations with cultural importance; (iv) and *classical travel sequence ration* where itineraries that revisit classical travel sequences of previous users are considered to be better. They compare the approach against baselines (Ranking-by-Time and Ranking-by-Interest) showing that the proposed approach can recommend relevant itineraries fulfilling the users' available travel time. They evaluate in terms of elapsed time ratio, stay time ratio, interest density ration and classical travel sequence.

Huang *et al.* propose an intelligent system to provide personalized recommendations of tourist attractions in an unfamiliar city by exploiting a tourism ontology [78]. The ontology, Figure 2.6, allows the integration of heterogeneous on-line travel information including attractions, open time, location, activity and admission fees. Based on a Bayesian network technique and the analytic hierarchy process (AHP) method, the system recommends tourist attractions to a user by taking into account the travel behavior both of the user and of other users. Spatial web services technology is embedded in the system to provide GIS functions. In addition, the system provides an interactive geographic interface for displaying the recommendation results as well as obtaining users' feedback. The experiments show that the system can provide personalized recommendations on tourist attractions that satisfy the user.

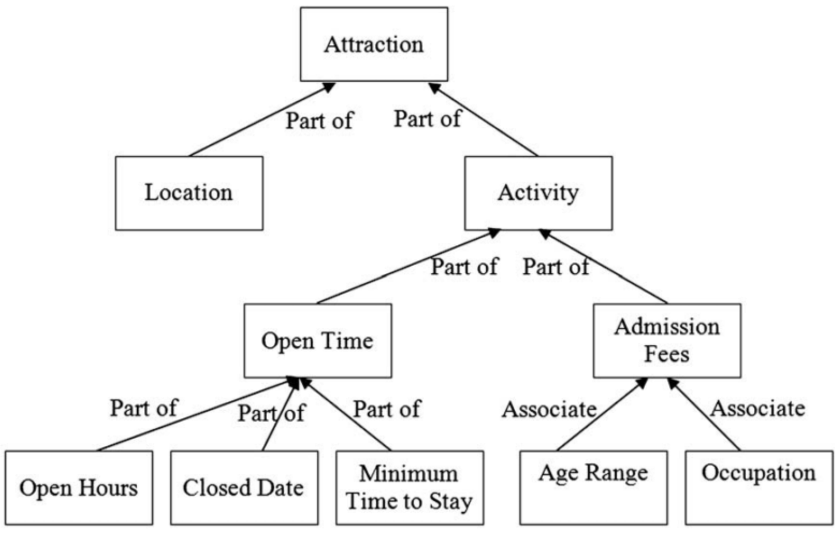


Figure 2.6: Concepts and their relationships in the ontology by Huang *et al.* Figure from [78].

Shang *et al.* propose and investigate a problem called *User Oriented Trajectory Search* (UOTS) for trip recommendation [127]. In contrast to conventional trajectory search by locations (where only the spatial domain is exploited), authors consider both spatial and textual domains in the new UOTS query. Given a trajectory data set, the query input contains a set of intended places given by the tourist and a set of textual attributes describing the tourist preference. If a trajectory is connecting/ close to the specified query locations, and the textual attributes of the trajectory are similar to the tourist preference, it will be recommended to the tourist for reference. The spatial distance is measured by the shortest path length between a given point and the closest point of the given trajectory normalized by a Sigmoid function. The textual distance, in turn, by means of Jaccard distance. The two distance functions are then linearly combined as the final result. This type of queries can bring benefits to tourists in many popular applications such as trip planning and recommendation. However, this approach does not take into account the user's time budget in order to optimize her trip. In fact, this approach can be used to retrieve a subset of trajectories from a very large set of trajectories to prune irrelevant trajectories w.r.t the user's interest.

An interesting approach to the trip recommendation problem is the one proposed by Vansteenwegen *et al.*, where authors define the Tourist Trip Design Problems (TTDP) [130, 142]. The orienteering problem, which originates in the operational research literature, is used as a starting point for modeling the TTDP. The problem involves a set of possible locations having a score and the objective is to maximize the total score of the visited locations, while keeping the total time (or distance) below the available time budget. The score of a location represents the interest of a tourist in that location. Scores are calculated using the vector space model and the TTDP is solved using a guided local search meta-heuristic. Authors compare their technique versus a competitor. Both algorithms are applied to a real data set from the city of Ghent. Results show that the approach turns out to be faster and produces solutions of better quality. Towards the generation of multiple tours, one for each day for instance, Vansteenwegen *et al.* presented the team orienteering problem (TOP) in [139] jointly with the Guided Local Search (GLS) algorithm combined with different local search heuristics to solve the TOP. Lately, they propose [138] a tourist expert system, called the "City Trip Planner"[4]. It is implemented as a web application that takes into account the interests and trip constraints of the user and matches these to a database of locations in order to predict personal interests.

An interesting variation of TOP is the team orienteering problem with time windows (TOPTW) [141, 96]. In this problem, the points of interest also hold a time window to represent their availability. In [141] an iterated local search (ILS) heuristic is proposed, while Lin *et al.* propose a simulated annealing heuristic, showing computational results competitive with other solution approaches (including ILS). A step further TOPTW is the Time Dependent Team Orienteering with Time Windows (TDTOPTW) formulation [140, 66], which is particularly suitable for using public transportation in planning the sightseeing tour.

A survey is presented in [137] for designing trips for a tourist approaching a new city based on existing models from the field of Operations Research. Using the Orienteering Problem and its extensions to model the tourist trip planning problem, the authors present different features and functionalities to deal with a number of practical tour planning problems. In addition, they also propose a set of interesting directions for this research area.

The orienteering problem is also employed in [53]. Here, De Choudhury *et al.* construct intra-city travel itineraries automatically by tapping a latent source reflecting geo-temporal traces left by millions of tourists. To do so, they firstly extract photo streams of individual users from Flickr. In the second step, they aggregate all user photo streams into a PoI graph. Itineraries are then

[4] `http://www.citytripplanner.com/en/home`

automatically constructed from the graph based on the popularity of the PoIs and subject to the user's time and destination constraints. The problem is modeled as an orienteering problem and they propose a variation of a recursive greedy algorithm to solve it. Their proposal explicitly needs a start location, an end location, the total number of locations to be visited in the trip and a set of locations to not be visited.

More recently, Gavalas *et al.* propose eCOMPASS system[5] , web and mobile application, to create multi-modal personalized sightseeing tours. The basis of the system is the Time Dependent Team Orienteering Problem with Time Windows (TDTOPTW). The system is organized into off-line and on-line phase, In the off-line or preprocessing phase, the set of points of interest are clustered based on the geographical criteria, using the global k-means algorithm by Likas *et al.* in [95]. The clustering process contributes to group close PoIs to be successively visited in the on-line phase, indicating that they can be visited by walking since PoIs in the same clusters are likely to be within a walking distance. In this phase it is also computed pairwise full (24h range) multi-modal time-dependent travel time profiles among all locations stored using the method in [55]. In the on-line phase, the user queries with preferences are used to create the tours. The algorithm, denoted by SlackRoutes, uses an iterated local search procedure inserting PoIs along the initial routes until no more insertion is feasible and then, new solutions are derived by perturbing the current solution. They also present a strategy to encompass lunch breaks in the tours.

Lu *et al.* [99], propose a novel data mining-based approach, namely Trip-Mine, to efficiently find the optimal trip which satisfies the user's travel time constraint based on the user's location. Authors also propose three optimization mechanisms based on Trip-Mine to further enhance the mining efficiency and memory storage requirement for optimal trip finding. They compare Trip-Mine with a BruteForce approach and a dynamic programming algorithm.

In [69], Gionis *et al.* propose efficient algorithmic solutions for recommending customized tours in urban settings, which considers (i) the different types of points of interest (categories), as well as the order in which the user wants to visit them, (ii) time budget or distance to be covered, (iii) the multiplicity bounds to allow users to specify the number of venues of a particular type that they want to visit, and (iv) the merit of visiting the included points of interest. These four constraints are considered to define the TOURREC problem. They propose two alternative instantiations to solve the referred TOURREC problem w.r.t a generic *satisfaction function* that measures the expected satisfaction of the user with respect to a candidate tour (sequence of points of interest): *additive satisfaction function* which accounts for satisfaction of each point of interest in the tour; and *coverage satisfaction function* when each points of interest is (spatially) associated with a set of attraction or activities that might be relevant to the user. The authors propose algorithms for each satisfaction function and evaluate them by using real datasets of *check-ins* from Foursquare and comparing against a greedy approach to show that their proposal can find better solutions.

As we can note up to here is that much of the efforts are concentrated in the development of algorithmic solutions to deal with the trip planning problem that has been seen NP hard in its different facades. Yahi *et al.*, however, propose a hybrid tour planning system, denoted by *Aurigo*, that combines a recommendation algorithm with interactive visualization techniques to

[5]`http://ecompass.aegean.gr/`

create personalized itineraries. In this system, the user is assisted to create her personalized trip by interacting with the system, given an origin and a destination, and the user's preferences. The user may decide to add or remove points of interest that are on the way between her origin and destination. The authors conducted a user case study with 10 participants to assess the benefits of *Aurigo*.

2.4 Group Recommendation

As we have seen, recommender systems have traditionally recommended a variety of items to be enjoyed by individual users: watching a movie on Netflix, listening to a song on Spotify, having a dinner in an Italian restaurant, and among many other examples. However, some types of items are better enjoyed with companions due to their natural collective characteristic. For instance, recommending a movie in the cinema for a single user is still relevant, but people usually go to the cinema in company of their friends, for instance, which indicates that the recommendation of a movie to be watched in the cinema could be driven to a groups of users instead. This new perspective has motivated a proliferation of new recommender systems to support recommendations to groups of users [79]. In this section we report the group-oriented recommender systems and main results found in the literature.

Recommendation for single users is not an easy task and it requires much efforts by the scientific and industrial community to find good solutions. Recommending to groups is even more complicated than recommending to individuals [121]. Here, the problem can be seen how to combine the individual user preferences to represent the preference of the whole group in such a way that each group member is not degenerated.

A classification of the recommendation to groups in collaborative recommender systems is presented in [115, 25] as illustrated in Figure 2.7. In this figure, the individual members of a group are represented on the left, in gray; each graticule represents the matrix of ratings by the users (horizontal) on the items (vertical). The graph shows the four representative cases of tackling the solution to recommendation by groups (one case for each matrix on the left of the figure). The circles show key information: they indicate the CF process phase where the unification is performed: "n users $\rightarrow$ group". They present four stages on which we can act in order to unify the group's users' data with the objective of obtaining the data of the group of users: similarity metric, neighborhoods, prediction and the recommendations stage.

In the similarity metric stage, the neighborhood of the group is found by using a similarity function able to calculate the similarity between a user and a group. This approach acts directly on the set of ratings of the groups of users to provide a set of neighbors for the group of users as presented in [115, 25]. The proposed similarity metric evaluate the number of cases in which user u's ratings intersects that of any of the members of the group. The intuition is that users having high intersection with the group members will be capable of proposing new items that the majority of the users in the group will enjoy [115].

Regarding the neighborhood stage, the neighbors of the group's users are unified in one neighborhood for the whole group. This approach has been studied by Bobadilla *et al.* in [24], proposing

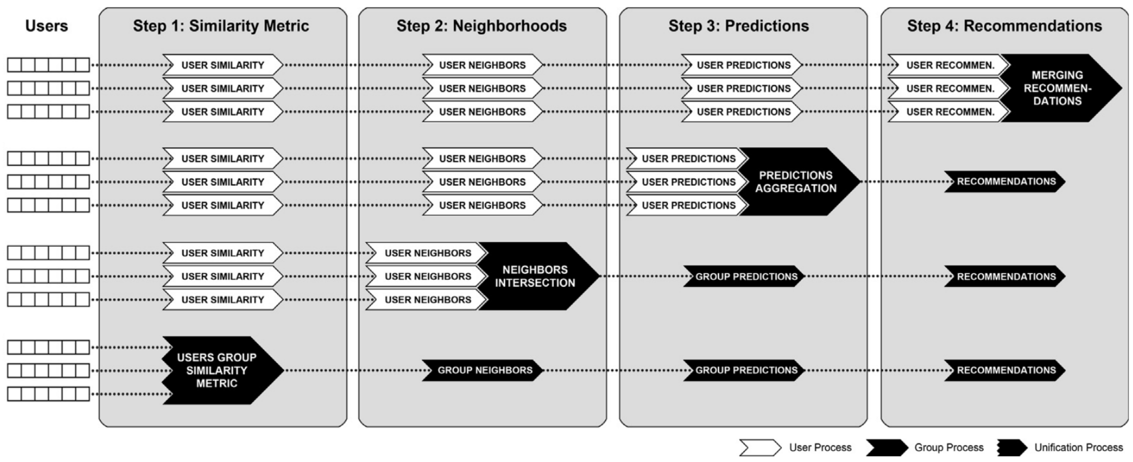


Figure 2.7: Classification of the recommendations to groups in Collaborative Filtering RSs. The figure represents the four representative cases for approaching the solution to group recommendations. Figure from [115].

the intersection of a large number (k) of neighbors of each user of the group.

In the prediction stage, the data unification is performed in CF process by aggregating the n individual predictions of n group members in one prediction of the whole group. We can highlight the works done by Berkovsky and Freyne [19], García et al. [65] and Christensen and Schiaffino [46].

In the last stage, the recommendations obtained for each individual user of the group are aggregated into one recommendation of the whole group. A rank aggregation approach on the individual lists of recommendations is presented by Baltrunas *et al.* [11].

This classification depicts the main problem of group recommendation which is to solve how to adapt to the group as a whole based on information about individual users' likes and dislikes [121]. The problem is solved by applying an aggregation strategy that combines individual ratings into a group rating. Aggregation strategy for group recommendation is also referred to as *group recommendation semantics* [5]. Many strategies exist to approach the problem in different ways. In [121] a list of eleven aggregation strategies inspired by Social Choice Theory is present and illustrated in Table 2.1. Two popular approach commonly used are Least Misery and Average [77, 114, 158, 15].

Hu *et al.* propose a novel group recommender system approach which accommodates both individual choices and group decisions in a joint model through a deep-architecture model built with collective deep belief networks (DBN) and dual-wing restricted Boltzmann machines [77]. Authors claimed that traditional methods that aggregate either uses' preferences or users' predictions are heavily sensitive to data and, consequently, they fail to learn group preferences when the data are slightly inconsistent with predefined aggregation assumptions. In particular, they propose a multi-layer model based on deep belief networks and Restricted Boltzmann Machines (RBM) to learn high-level features: *collective features* representing preferences of a group; *individual features* for individual-specific preferences; and *member features* to model the individual preference of a user when she makes choices as a group member. They design a dual-wing RBM (DW-RBM) on

Strategy	**How it works**	**Example**
Plurality Voting	Uses 'first past the post': repetitively, the item with the most votes is chosen.	A is chosen first, as it has the highest rating for the majority of the group, followed by E (which has the highest rating for the majority when excluding A).
Average	Averages individual ratings.	B's group rating is 6, namely $(4+9+5)/3$.
Multiplicative	Multiplies individual ratings.	B's group rating is 180, namely $4*9*5$.
Borda Count	Counts points from items' rankings in the individuals' preference lists, with bottom item getting 0 points, next one up getting one point, etc.	A's group rating is 17, namely 0 (last for Jane) + 9 (first for Mary) + 8 (shared top 3 for Peter).
Copeland Rule	Counts how often an item beats other items (using majority vote) minus how often it looses.	F's group rating is 5, as F beats 7 items (B,C,D,G,H,I,J) and looses from 2 (A,E).
Approval Voting	Counts the individuals with ratings for the item above an approval threshold (e.g. 6).	B's group rating is 1 and F's is 3.
Least Misery	Takes the minimum of individual ratings.	B's group rating is 4, namely the smallest of 4,9,5.
Most Pleasure	Takes the maximum of individual ratings.	B's group rating is 9, namely the largest of 4,9,5.
Average without Misery	Averages individual ratings, after excluding items with individual ratings below a certain threshold (say 4).	J's group rating is 7.3 (the average of 8,8,6), while A is excluded because Jane hates it.
Fairness	Items are ranked as if individuals are choosing them in turn.	Item E may be chosen first (highest for Peter), followed by F (highest for Jane) and A (highest for Mary).
Most respected person	Uses the rating of the most respected individual.	If Jane is the most respected person, then A's group rating is 1. If Mary is most respected, then it is 10.

Table 2.1: Overview of traditional aggregation strategies for group recommendation. Table from [121].

the top of the model (Figure 2.8), where one wing of the DW-RBM is connected to the group profile, and the other wing is connected to the collective features layer of the collective DBN. Their approach demonstrates to have good performance when compared with state-of-the-art models.

Bobadilla *et al.* present a collaborative filtering approach extended to groups of users and restricted groups of items that enables joint recommendations to groups of users and enables the recommendations to be restricted to items similar to a set of reference items [24]. For instance, a groups of four friends could request joint recommendations of films similar to "Avatar" or "Titanic",

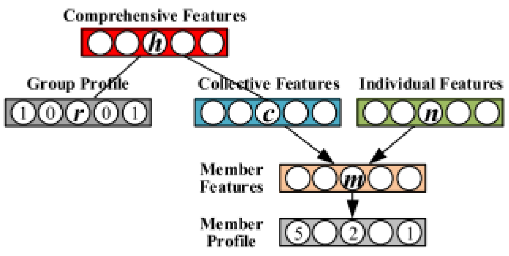


Figure 2.8: The dual-wing RBM proposed by [77] placed on the top of DBN, which jointly models the group choices and collective features to learn the comprehensive features of group preference.

which is motivated by the fact that users want recommendations of different types of items at different moments in their lives. They propose a collaborative filtering approach that considers user-user and item-item similarities, so that the k most similar users for each group member is used to compute the group neighbors from the intersection of its member neighbors. In addition, the s most similar items to the given restricted set of items are considered. They show that the traditional collaborative filtering approach does not resolve the investigated problem, while their proposal overcome the traditional CF in terms of number of recommendations, quality of the prediction and quality of recommendations.

Amer-Yahia *et al.* introduce the notion of *consensus function*, consisting of two components, *relevance* and *disagreement*, as a new aggregation strategy for group recommendation [6]. The group relevance is computed by means of *Average* and *Least Misery*, while group disagreement is calculated by two methods: *Average Pair-wise Disagreements* and *Disagreement Variance*. Then, these two methods are linearly combined to form the *consensus function*. Results conducted on Amazon Mechanical Turk with a comprehensive user study demonstrate that incorporating disagreements is critical to the effectiveness of group recommendation.

Later, Amer-Yahia *et al.* propose a new group recommendation model that takes into consideration the affinity between group members and how the affinity evolves over time – denoted GRECA [5]. They extend existing group recommendation semantics to include temporal affinity in recommendations and design an algorithm that produces temporal affinity-aware recommendations for ad-hoc groups. They propose two dynamic models to capture temporal affinities: a *discrete* model when time is discretized over a set of time periods; and a *continuous* model when time is represented as an exponential function that positively or negatively affects affinity over time. The user-item preference is computed taking into account the temporal affinity, absolute preferences (e.g. a predicted score by a RS), and relative preferences related to how close members of the user in the group like or dislike the item. Then, A time-aware group consensus (aggregation function) that combines group preferences (*Average* and *Least Misery*) with a group disagreement function as the group aggregation strategy. The results show substantial improvements in group recommendation quality when accounting for temporal affinities, improving the user satisfaction. In addition, the amount of satisfaction is variable and is dependent on the characteristics of the groups, like group size, group cohesiveness and affinity strength.

Apart from the aggregation strategies in Table 2.1, some works have applied rank aggregation methods for group recommendation [58, 11]. Dwork *et al.* developed a set of techniques for the rank aggregation problem [58]. Baltrunas and Ricci propose a method based on the ordinal ranking of items, where the result of a group recommendation process is an ordered list of items. They experimented different rank aggregation methods to evaluate the effectiveness of group recommendation. They claim that the aggregation method itself has not a big influence on the quality of the recommendation and that the performance depends on the group size and inner group similarity. Moreover, the quality of the recommendation can be increased, when the individual recommendations are not good, by aggregating the ranked list recommendations built for a group of users, which the target user belongs to.

These works have in common the goal of recommending items for a group of users considering some aspects inherent to the problem, such as the relevance of the item for the group members. In the next section, however, we report a new perspective in group recommendation defined as group formation problem, where the recommendations are taking into account to find out relevant groups of users. One of the contributions of this thesis shown in Chapter 6 is concerning the group formation problem.

2.5 Group Formation Problem

The group recommendation research track aims at recommending items, top-k item lists for instance, to a group of users in such a way the satisfaction of group members is maximized according to a group satisfaction function. A complementary work to group recommendation has been investigated by Basu Roi *et al.* in [15], where they consider the group formation problem in the context of recommender system, in other words, how to form groups such that the users in the formed groups are most satisfied with the suggested top-k items. In this section we present this work, [15], by detailing the investigated problem and the proposed algorithms. The results of this thesis in the field of group formation in the context of recommender system are introduced in Chapter 6.

Let us denote $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ as a set of items containing m items and $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ as a set of users with n users. Then, a group g represents a subset of users $g \subseteq \mathcal{U}$. As presented in Section 2.2, recommender systems aim at evaluating or predicting a score or relevance for a user u to an item i, where this score is usually denoted by $R(u, i)$. Thus, $R(\cdot, \cdot)$ expresses the possible satisfaction of u w.r.t the item i. In addition, let $R_g(g, i)$ be the function that measure the relevance of item i for the group g by using any group semantics (e.g. Least Misery).

Yet, let $\mathcal{I}_g^k$ be the recommended top-k item list for a group g, such that $\mathcal{I}_g^k \subseteq \mathcal{I}$ and $|\mathcal{I}_g^k| = k$. With those definitions in hands, we can rely on the group semantics methods presented in Section 2.4. Then, Basu Roi *et al.* consider in their work Least Misery and Aggregated Voting group semantics, which are very popular methods, to evaluate the relevance of an item i to a group of users g. Once the group satisfaction score of an item is computed, it is needed to compute the group satisfaction of the recommended item list $\mathcal{I}_g^k$ denoted by $R_g^s(\mathcal{I}_g^k)$. Three aggregation methods are proposed in [15]:

- **Max-aggregation**: Satisfaction of the group is calculated as the score $R_g(\cdot, \cdot)$ of the very top item in the list. Let i^1 be the item with the highest score in $\mathcal{I}_g^k$, then the group satisfaction:

$$R_g^s(\mathcal{I}_g^k) = R_g(g, i^1)$$

- **Min-aggregation**: Similar to Max-aggregation, this is measured as the score of the bottom item in $\mathcal{I}_g^k$, denoted by the item i^k, for the group. It is given by

$$R_g^s(\mathcal{I}_g^k) = R_g(g, i^1)$$

- **Sum-aggregation**: Satisfaction of the group for Sum-aggregation is measured as the sum os scores of all items in the recommended list of items $\mathcal{I}_g^k$, which is given by:

$$R_g^s(\mathcal{I}_g^k) = \sum_{i \in \mathcal{I}_g^k} R_g(g, i)$$

Based on these definition, Basu Roi *et al.* define in [15] the **Recommendation Aware Group Formation** (GF) problem as:

Recommendation Aware Group Formation Problem

Given items $i_1, i_2, \ldots, i_m$ and users $u_1, u_2, \ldots, u_n$, a group recommendation semantics Least Misery (LM) or Aggregate Voting (AV), two integer k and l, create a set of at most l non-overlapping groups, where each group g is associated with a top-k item set $\mathcal{I}_g^k$ in accordance with semantics LM or AV such that the aggregated group satisfaction of the created groups is maximized: $\sum_{j=1}^{l} R_j^s(\mathcal{I}_j^k)$.

The authors show that the problem is NP-hard by reducing from Exact Cover by 3-Sets (X3C), which is known to be NP-hard. They propose algorithmic solution to deal with the problem for both Least Misery (LM) and Aggregated Voting (AV) techniques. The approximation algorithms are designed according to the group semantics used (LM or AV) and the group satisfaction aggregation (MIN, MAX, SUM). We discuss these algorithms as follows.

The proposed algorithms follow a core framework that is composed by three basic steps: (i) the formation of a set of intermediate groups; (ii) the greedy selection of $l-1$ groups; and (iii) the formation of the l-th group. Each step is achieved according to the group semantics (LM and AV) and the group satisfaction aggregation function used (MIN, MAX, SUM) as described below.

Let us start from the algorithm GRD-LM-MIN that deals with the LM and Min-aggregation. At the step (i), each user has her top-k item sequence sorted by the preference score of the items in non-increasing order. Then, a set of intermediate groups g is created, where the groups are formed by a set of users who have the same top-k item sequence and the same preference score for the bottom item i^k in the sequence across all the users in g. The formation of the intermediate groups is achieved by using a hash map to hash each user u using her top-k item sequence and the preference score of the bottom item i^k. For instance, let $\langle i_1^u, \ldots, i_k^u \rangle$ be the top-k item sequence of u and $R(u, i^k)$ the preference score of the bottom item i^k. So, the pair $(\langle i_1^u, \ldots, i_k^u \rangle, R(u, i^k))$ is used as key and u as the value to hash each user. A heap is created from the hash map such that the set of users having the highest $R(u, i^k)$ can be efficiently retrieved. At the step (ii), $l-1$ groups are greedily formed by getting the $l-1$ intermediate groups. At each interaction, the highest LM score is retrieved from the heap and it used to find the corresponding group of users. In the end of this step, $l-1$ groups are obtained. Finally, at (iii), the l-th group is formed by all the remaining users from the hash map and the top-k LM score is assigned to this group. The objective function is then computed by the sum of the groups' satisfaction w.r.t their top-k item set $\sum_{j=1}^{l} R_l^s(\mathcal{I}_j^k)$. As discussed in [15], GRD-LM-MAX algorithm is similar to GRD-LM-MIN, but it hashes the user by the top item i^1 instead of the bottom one i^k.

For the algorithm GRD-LM-SUM, a similar framework is exploited, but here the step (i) is slightly modified. In GRD-LM-SUM, the intermediate groups are formed by hashing the users who have the same top-k item sequence and the same preference score for each item in the sequence. Then, in steps (ii) and (iii), the group satisfaction aggregation if computed over all k items, instead of the bottom item i^k.

What regards the Aggregated Voting group semantics, two other algorithms are proposed: GRD-AV-MIN and GRD-AV-SUM. Recall that in this case, the satisfaction score of a group is defined as the sum of the preferences scores of the individuals users in the group for the top-k item sequence. The algorithms are similar to those for Least Misery with some peculiarities present next. In GRD-LM-MIN, the top-k item sequence and the score of the bottom item are used to hash the users in the hash map and to create the heap. For GRD-AV-MIN and GRD-AV-SUM, the intermediate groups g are formed by hashing the users with only their top-k item sequence disregarding the preferences scores for the each item. Then, the group satisfaction score is aggregated according to the method: either the score of the bottom item i^k for GRD-AV-MIN; or the sum of scores for all items in the top-k item sequence in the case of GRD-AV-SUM.

The experiments were conducted on two very known datasets: *Yahoo! Music* and *Movie Lens.* Comparisons were done against different competitors. The results highlight the proposed algorithms and it emphasize the interestingness of group formation problem in the context of recommender system. Indeed, group formation in the perspective of RS may to complement the recommender systems as we present in Chapter 6 showing the obtained results from this thesis.

Chapter 3

ComeTogether: finding and characterizing communities of places in urban mobility

Much efforts have been done to model the mobility behavior of people powered by the popularization of devices equipped with positioning sensors, like GPS and GSM. The majority of the analysis works has been focusing on moving objects (e.g. people) to discovery patterns and knowledge that may explain the phenomena inherent to the mobility of people.

In this chapter, however, we present a different analysis perspective from the points of interesting by investigating research question **RQ1**: *Can we study urban mobility at the global scale from the perspective of places, instead of users?* To answer this question we need to combine urban places, like points of interest (PoI), with mobility information like trajectories of individuals moving within a city. In this chapter, we present a methodology based on complex network analysis encompassing the following steps: (i) we build a network of points of interests by connecting places using the individual trajectories passing through them; (ii) we then characterize the POIs based on the network features, finding different categories of places characterized by their position in the network; (iii) and finally we perform community detection, finding places grouped by dense patterns of mobility among them. A case study is presented on real trajectory datasets in the cities of Milan, Florence and Pisa, showing a view of the urban mobility which is complementary compared to the classical mobility mining. The experimental results show different mobility behaviors when comparing different cities, weekdays with weekends, or urban centers with external areas. We believe that these results interestingly add more insights on urban mobility and the kind of patterns that can be extracted by other available methodologies. This chapter is based on the published work [30].

3.1 Introduction

Human mobility is a complex phenomenon witnessed by a huge amount of interdisciplinary research in this topic, ranging from Physics to Sociology, Transportation Research and Computer Science [68, 67, 163, 20]. Despite the wide spectrum of research papers and results so far, many aspects and relationships of mobility data with the environment where movements take place are still to be fully understood. In this chapter we want to offer a new perspective where we look at the mobility data analysis focusing on a specific aspect: how can we characterize urban places (intended as Points of Interest) by the mobility of people visiting them? In turn, this gives a feedback on how the mobility is affected by the location or the existence of given places. In fact, there is a two-way relationship between how human mobility is affected by the location of places of interest (e.g.: people move to a newly opened bar), and how the places themselves are characterized and connected by the mobility of people (e.g.: bars frequently visited together).

With this in mind, Points of Interest can be characterized based on how people globally access them. However, we believe that a simple count of the number of visits of a given place, although certainly giving a measure of the attractiveness of that place, is not enough. It is in fact difficult to get a deep understanding on how that particular place is "lived by" people and how it "relates", from the point of view of mobility, to other places. We want then to enrich the available tools of analysis, by providing a framework that aims at characterizing urban places based on how people reach them and how people move among them. For example, are people crossing the whole city to reach them (as it usually happens for airports), or do they tend to come from nearby places (like a minor neighborhood store)?

We observe that, although mobility analysis has recently become a hot research topic in Computer Science and Transportation Research, the available approaches, to the best of our knowledge, fail to characterize, at global scale, the relationships among POIs based on how people access them. The specific aspect of understanding how objects interact at a global scale is usually associated to Complex Network Analysis. With this perspective in mind, the research questions we want to address in this chapter, based on the **RQ1** (Chapter 1), are the followings.

Question 1: Can we study urban mobility at a global scale from the perspective of places, instead of users?

Question 2: Are there any patterns of such mobility w.r.t places that we can detect?

Question 3: Can we characterize such patterns and find regularities or anomalies?

This chapter, that extends our previous work in [33] and [30], presents ComeTogether, a framework aimed at answering these questions by building and analyzing a complex network combining Points of Interests and traces of people movements. ComeTogether offers new analysis measures and techniques for classifying the POIs, that complement the available set of tools proposed by previous approaches. We experimented our methodology in a real case study where GPS trajectories are collected from private cars traveling in three Italian cities featured by complementary aspects, while the Points of Interest are downloaded from the Web. We discover different categories/ classes of POIs and we propose a method to group them in uniform classes that can be interpreted and understood. Then, after building a network of POIs by linking them using shared trips, we extract communities of POIs and introduce some measures, like the **compactness**, to compare them and give them an interpretation.

With respect to our previous work [30], the additional contribution presented in this chapter resides in i) a broader empirical evaluation, where we compare the methodology on three different cities and ii) a new evaluation strategy to extract meaningful local patterns that summarize the global urban mobility.

This remaining of this chapter is organized as follows: Section 3.2 poses the basic terminology and definitions for a proper understanding of the framework. Section 3.3 presents the methodology used in this chapter. Section 3.5 reports on the experimental results using three real GPS datasets. Section 3.6 contains a final discussion about the methodology and results.

3.2 Basic Concepts

Our approach is based on the study of a network of Points of Interest based on the trajectories of people visiting them. POIs represent locations in the city where a person may perform some activity. Usually we call POIs the places that are of interest for some specific application, like GPS navigators, Social Networks (e.g. FourSquare, Facebook, etc), maps (like OpenStreetMap), or others.

Formally, to the purpose of our work, a Point of Interest can be defined as follows.

Definition 1 (Point Of Interest) *A Point of Interest (POI) is a geographical object that is interesting for a specific application, usually associated to a human activity. A POI is a tuple $POI = (s, r, l, c)$ where s is the representative spatial point, r is the spatial area representing the extent of the object, l is the label of the POI, and $c = \{c_1, \ldots, c_j\}$ is a set of categories assigned to the POI from a category set $C : \forall 1 \leq i \leq j,\ c_i \in C$.*

An example of POI is the Eiffel Tower: the representative spatial point s is the center of the tower while the region r is the spatial extend of the base of the tower, the set c of categories may include, for example, “tourist attraction” or “monument” or “tower”, depending on the application, and the label represent the name “Eiffel Tower”.

The second main component of our network is the set of user movements. A movement of a person can be represented as a set of user position observations collected from a tracking device and creating the mobility history of an individual, defined as follows.

Definition 2 (User Mobility History) *Given a set of user's observations $\mathcal{O}_u$, the user's history is defined as an ordered sequence of spatio-temporal points $H_u = \langle o_1 \ldots o_n \rangle$ where $o_i \in \mathcal{O}_u$, $o_i = (x_i, y_i, t_i)$, with x_i, y_i spatial coordinates, t_i is an absolute timestamp and $\forall (i, j)\ i \leq j \Rightarrow t_i \leq t_j$.*

Since the user history represents the whole user movements, we need to distinguish the single *trajectory* as the part of the user history representing the a specific activity, such as *going to work*, *shopping* etc. To distinguish between the different trajectories in a user's history, we need to detect when a user stops for a time long enough to consider this stop as performing an activity in a PoI

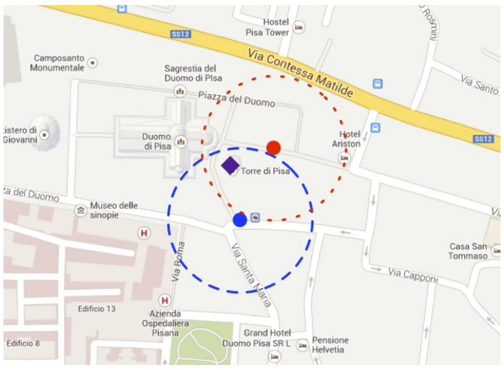


Figure 3.1: Example of assigning the POI *Leaning Tower* (purple diamond) to two candidate stops, depicted with a blue and a red dots.

and thus the end of that particular trip and the beginning of the next one. In the literature there are several approaches for stop detection - and therefore trajectory splitting - mainly following two main lines: clustering-based [26] and heuristic-based [146]. However, for computational efficiency reasons and for the sake of simplicity, here we take a different method as a trade-off between precision and efficiency. We search the points that change only in time. i.e. points that stay in the same spatial position for a certain amount of time quantified by the temporal threshold $MinStopTime$. Specularly, a spatial threshold $MaxStopArea$ is used to remove both the noise introduced by the imprecision of the device and the small movements that are of no interest for a particular analysis. These thresholds are used for detecting the *candidate stops* as defined below. The function $area()$ computes the size of the minimal convex region including a set of points and $\preceq$ is the operator of sequential inclusion without gaps.

Definition 3 (User's candidate stops) *We define the candidate stops $\mathcal{S}_u$ of user u as a sequence of tuples (a, t_s, d), where a is the sequence of spatio-temporal observations $\langle o_i, \ldots, o_j \rangle$, $area(a) \leq MaxStopArea$ with stop time starting at $t_s = o_i.t$ and time duration $d = o_j.t - o_i.t \geq MinStopTime$.*

For simplicity, we hereinafter indicate the start time of the k-th candidate stop of user u as $s^u_k.start$ and the end as $s^u_k.end$.

Associating the user stops to visited POIs is not trivial. In fact, depending on the tracking device, some parts of the track may be missing or inaccurate. Several techniques have been proposed in the literature for associating a POI to a stop in a GPS track, for example [122, 63].

In our approach, we associate a stop to a POI by fixing a spatial buffer around the candidate stop-start point and assigning the POIs within this spatial buffer to the stop. For instance, in Figure 3.1, we associated the PoI "Leaning Tower" to two different stops, by setting a spatial buffer of 50 meters.

However, several PoIs may be present in the same buffer area in very dense regions, so making not obvious the choice of the visited PoI [122, 63]. For this reason, we propose the new notion of Aggregated PoI as an aggregation of PoIs densely found in the same area. The idea is to associate a Aggregated PoI to a candidate stop when more than one PoI is present in the buffer area. Since the task of finding the best PoI to associate to a stop is outside the scope of this work, we choose here the simple solution of keeping all the possible visited PoIs, leaving for future investigation the integration into the methodology of a more sophisticated technique for stop-to-PoI association [63]. Formal definition of Aggregated PoI follows:

Definition 4 (Aggregated PoI) *An aggregated PoI, GPoI, is a set of PoIs* $\{POI_1, \dots, POI_n\}$ *aggregated by spatial neighborhood. The category set of a GPoI is the set of all the categories of the* POI_i*,* $i = 1, \dots n$*.*

The intuition behind the aggregated PoI is to group the spatially close PoIs into one single object as a node for the network. Each GPoI thus, in our definition, contains one or more PoIs. A spatially isolated PoI will be a GPoI with only one element. How to detect these PoIs aggregations? This depends on the specific application and the chosen implementation. A simple solution is to choose a radius threshold from the user history candidate stop and to collect all the POIs inside this radius. Another solution is to use a spatial clustering algorithm to discover the groups of POIs spatially close. As we see in Section 3.5, we choose the latter option in our implementation. However, the methodology is general and can be adapted to other aggregation methods. From now on, for the sake of readability, we still refer to PoI to indicate GPoI.

Given V the set of Aggregated PoIs, let $f : \mathcal{S} \to V$ be an assignment function from users' candidate stops to PoIs. Thus the function f applied to a candidate stop s returns the GPoI associated to it. Furthermore, let Δm be a threshold of maximum moving time between two any GPoIs. We next define the trajectory and the trip concepts as sequences of PoIs visited by the tracked user.

Definition 5 (Trajectory) *A trajectory* t_u *of a user* u *is represented as a sequence of POIs* $\langle (f(s_1^u), s_1.start, s_1.end), \dots (f(s_m^u), s_m.start, s_m.end)) \rangle$*.*

Definition 6 (Trip) *We define a trip of user* u *as a sub-sequence of the trajectory* t_u*,* $trip_u = \langle (f(s_j^u), s_j.start, s_j.end), \dots, (f(s_k^u), s_k.start, s_k.end) \rangle$ *where* $j < k$ *and such that* $s_i.start - s_{i-1}.start \leq \Delta m$*, for each* $i = j+1, \dots, k$*.*

In short, a trajectory t of a user u is the sequence of PoIs associated to the user's candidate stops in all the user history, while the user's trip is the sequence of POIs where the movement between two consecutive PoIs is no longer than the temporal threshold Δm. Trips represent the continuity of movement, cut by a given threshold Δm. This definition is mainly rooted on two observations:

- long stops at a frequently visited place suggest this place could be home or work and not at an activity performed in a PoI. Since we are interested in the activities related to the urban landmarks, we discard home and work locations cutting the trajectories during these stops.
- long moves between places terminate a trip. A long move usually hide a missing stop, therefore when a move becomes too long to be considered as a unique trip the trajectory is split.

We denote, therefore, with $Trip_u$ all the trips of user u. Points of Interest and trips are combined creating the POI network in Section 3.3. We refer to the following definition of network:

Definition 7 (Network) *A network is represented as a graph $G = (V, E, W)$ in which entities (the nodes in V) are linked by ties (the edges in E), representing any sort of connection, similarity or interaction. The strength of these connections are represented in W.*

Network analytics has been focusing on the characterization and measurement of local and global properties such as diameter, degree distribution, centrality, connectedness - up to more sophisticated discoveries based on graph mining, aimed at finding frequent subgraph patterns and analyzing the temporal evolution of a network. A branch of Complex Network Analysis has been focusing on the discovery of structures called *communities.*

Definition 8 (Community) *Communities are groups of nodes highly interactive, densely connected, or, more in general, highly similar, for a given definition of similarity between any two individuals.*

Some of the existing approaches for community detection focus on finding groups of nodes, while others put the links among entities at the center of the investigation (see Section 2.1). Since we are interested in analyzing movements between places visited by users and in grouping places according to their visits at the places by trips, we consider the edges as the main entities to be grouped. In addition, we also want to consider the possible overlap between different communities. Different places can, in fact, take part into more than one community, due to their role of spatial "bridges" between them.

Now we are able to introduce the ComeTogether methodology in the next section, which presents the main steps involved in the framework to accomplish the understanding urban mobility under the perspective of PoIs and, consequently, to answer the investigated research questions presented in Section 3.1.

3.3 The ComeTogether Methodology

Our proposed solution to the research questions illustrated in the introduction is a methodology called ComeTogether, combining different aspects of mobility and graph analysis. ComeTogether is composed of three main steps:

1. **Building the PoI network**. This first step *builds a mobility network* where each node is a PoI and each link represents the *relations* between two PoIs in terms of users' trips (definition 6);

2. **Points of interest network analysis**. At this step, we analyze the generated PoI network to characterize the PoIs based on the properties of the mobility graph stressing how users access the points of interest associated in the network. For example: are the PoIs visited by many users or few users? Do people tend to spend a long time or only short visits? Is there a trend to visit a given place from far places (thus producing more traffic) or are

people coming from the neighborhood (thus a POI having a local coverage)? These features, possibly combined, help in giving a characterization of POIs based on the mobility graph.

3. **Communities in points of interest networks**. This third step aims at providing means of analyzing groups of points of interest (communities) that are highly related to each other based on the discovered links in the points of interest network. The communities framework comes then to characterize the POIs globally as a group of places visited together by users: (i) How can we analyze these communities to better understand the mobility? (ii) What information is this analysis providing?

In the next sections we go through each step of the ComeTogether framework in more details.

3.3.1 Building the PoI network

The network we propose is composed of a set of nodes corresponding to the POIs where the moving users stopped to perform some activity. Having all the trips of all the m users denoted by $Trip = \bigcup_{1 \dots m}^{u} Trip_u$, we compute the PoIs network as:

Definition 9 (PoI network) *Given a set of GPoIs V and a set of users' trips $Trip$, we build the point of interest network $N = (V, E, W)$ where*
$E = \{e_{i,j} : \exists t \in Trip, \langle v_i, v_j \rangle \preceq t \vee \langle v_j, v_i \rangle \preceq t\}$ *and*
$W = \{w_{i,j} : w_{i,j} = |\{t_1, \dots, t_m\}|, \langle v_i, v_j \rangle \preceq t \vee \langle v_j, v_i \rangle \preceq t\}$.

In other words, the PoI network is an undirected weighted graph which summarizes all the trips of the users and each edge is weighted by the number of trips which share the movement between the same pair of PoIs.

Figure 3.2 illustrate the data transformation flow from the raw user history to the built PoI network modeling the user mobility behavior between points of interest. More in details, the process starts building the user history as a continuous sequence of points of interest sorted by time as shown in Figure 3.2(a). In Figure 3.2(b) the stops are identified considering $MaxStopArea$ = $50m^2$ and $MinStopTime$ = 30 minutes. Then the stops are spatially intersected with the set of PoIs V as shown in Figure 3.2(c): the red edge between the two stops has a duration which is greater than the $MaxMoveTime$ (e.g. 4 hours) therefore it is removed partitioning the user history into two trips. Finally, the two trips will contribute to the edges shown in Figure 3.2(d) where w and w' are the number of trips which share the same path respectively $POI_1 \rightarrow POI_2$ and $POI_3 \rightarrow POI_4$.

The network building process is summarized by the pseudo-code of Algorithm 1. Each user's observation is evaluated in the loop $4 - 17$. User's history is created on line 5, candidate stops are detected on line 6, while user's trajectory is constructed on line 7. Then, the procedure $createUserTrip$ is called on line 8 to create trips for the user according to the trajectories previously created and the given moving threshold Δm. Once the trips of a user are created, the loop of lines $9-16$ looks at each trip and find pairs $\langle v_i, v_j \rangle$ on lines $10-15$ in order to create new nodes (line 11) and edges (line 13) on the network. The edge weight is updated on line 14. Finally, the algorithm

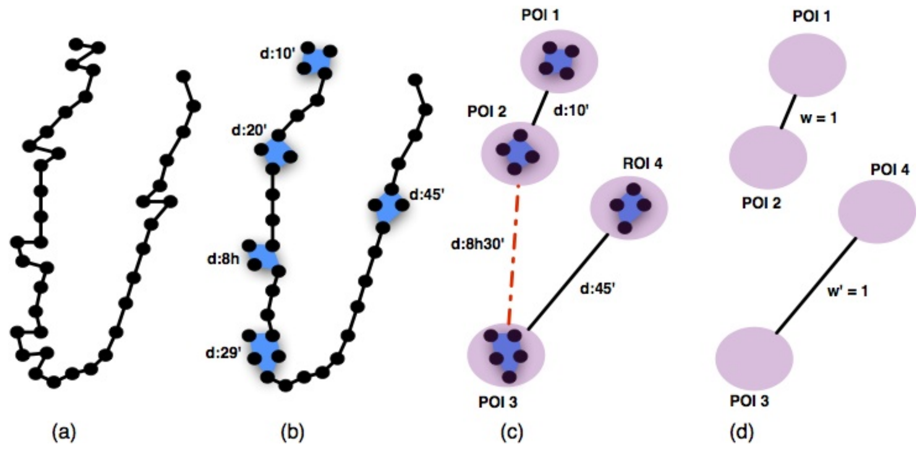


Figure 3.2: The building process of POI network from one user history: From the user history in (a), the candidates stops are computed in (b). The trips are found in (c), where a move of duration of $8h30'$ (thus exceeding a temporal threshold of 4 hours) splits the user history into two trips. Finally, the PoIs network is depicted in (d).

returns the PoI network $\mathbf{N} = (V, E, W)$, where V is the set of GPoIs, E is the set of edges and W is the set of weights of each edge in E.

3.3.2 PoI Network Analysis

Within the large set of available measures to quantify different phenomena in networks [109], we consider the *clustering coefficient* and the *average shortest path length*, as they are a good means for distinguishing real networks from random ones.

The number of triangles, representing the transitivity among three nodes in a network, is measured by the clustering coefficient of the nodes. This measure is used to investigate how clustered is the network, i.e., which is the probability of having an edge between two nodes A and C if there are the links A and B, and B and C. This property is usually evaluated in networks to identify some fundamental characteristics, where real networks, like biological and social networks, usually have larger cluster coefficient w.r.t to graphs randomly generated with the size number of nodes.

The distance between any two nodes in the network is also important for a first understanding of the structure. This is usually computed by the average shortest path between any two nodes in the network, contributing to understand if one node could be reached from another one in a few links on average. For this evaluation, low values are usually found in real networks.

We next present a characterization framework for the nodes in our networks, which is used in Section 3.5, together with the above measures, to study the basic properties of the PoI networks that we have built.

From network connectivity to mobility-related measures

Our aim here is to give a meaning of the nodes based on some properties representing their usage from the mobile users. Let us denote $\mathbf{A}$ a set of attributes assign to each node in the network. We

Algorithm 1: PoI Network Builder

```
Input: A set of positional observations $\mathcal{O}$,
a set of GPOIs $V$,
an assigning function $f$ from candidate stops to GPOIs,
a temporal threshold (minimum stop time) as $\Delta t$,
spatial threshold (maximum stop area) as $\Delta s$ for stop detection and
a temporal threshold (maximum moving time) as $\Delta m$ for creating users' trips
Output: POI network $\mathbf{N} = (V, E, W)$
1  $\mathbf{N}.V \leftarrow \emptyset$
2  $\mathbf{N}.E \leftarrow \emptyset$
3  $\mathbf{N}.W \leftarrow \emptyset$
4  for each $\mathcal{O}_u \in \mathcal{O}$ do
       // create users' history
5      $H_u \leftarrow$ userHistory($\mathcal{O}_u$)
       // identify candidate stops, def. 3
6      $S_u \leftarrow$ candidateStop($H_u, \Delta t, \Delta s$)
       // create user's trajectories based on def. 5
7      $T_u \leftarrow$ createUserTrajectory($f, S_u, V$)
       // create user's trips based on def. 6
8      $Trip_u \leftarrow$ createUserTrip($T_u, \Delta m$)
9      for each $t \in Trip_u$ do
10         for each $\langle v_i, v_j \rangle \preceq t$ do
               // create nodes $v_i$ and $v_j$
11             $\mathbf{N}.V \leftarrow V \cup \{v_i, v_j\}$
               // create edge $e_{ij}$
12             $e_{ij} \leftarrow \langle v_i, v_j \rangle$
13             $\mathbf{N}.E \leftarrow \mathbf{N}.E \cup e_{ij}$
               // update the weight $w_{ij}$ of the edge $e_{ij}$
14             update $w_{ij}$ in $\mathbf{N}.W$
15         end
16     end
17 end
18 return $\mathbf{N}$
```

define it as

$$\mathbf{A} = \{\mathbf{A}_{users}, \mathbf{A}_{stoptime}, \mathbf{A}_{movement}\}$$

Each attribute in **A** has a specific contribution to capture some relevant information about the nodes in the networks. They described below.

- *users* quantifies the number of users associated to the node (PoI), that is, the number of users that visited the PoI represented by that node. It gives some feedbacks regarding the popularity in terms of visits that such PoI has;
- *stoptime* is a relevant measure to quantify the duration of the visit. This is computed by the average stop time over all trips;
- *movement* represents the spatial dimension to capture how far are the node's neighbors. This attribute interestingly captures the notion of how far the users are willing to move to reach another PoI in the city. We can see it as the spatial proximity between two node in the network.

With these selection of attributes, we can intuitively introduce an interpretation for the nodes depending on the different values of each attribute. We qualitatively categorize the values of these attributes to *low* and *high*. Combining these values, we propose a set of classes composed by *Personal Spot*, *Popular Local*, *Popular Global*, *Hot Spot Local*, *Hot Spot Global* and *Undefined*. These classes are discussed below and they summarized in Figure 3.3.

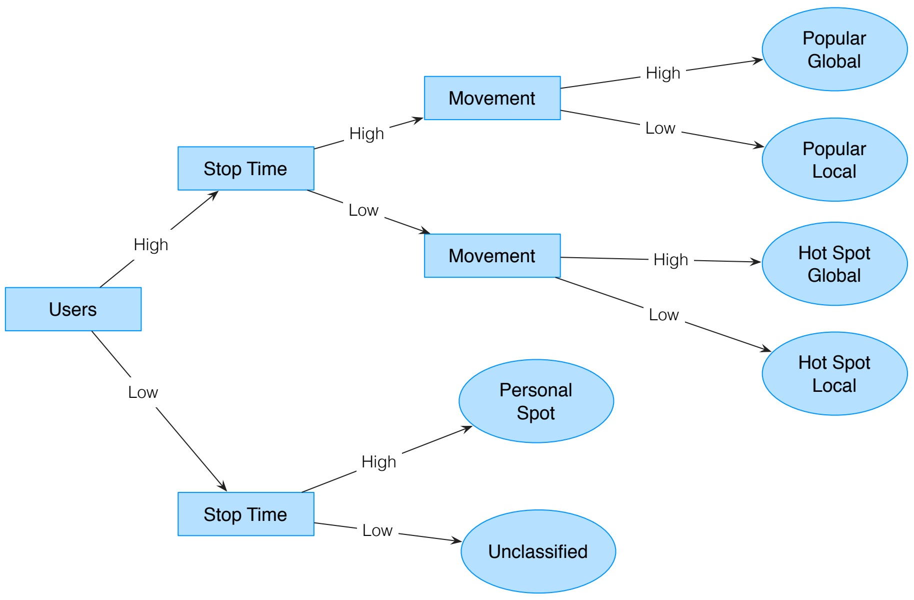


Figure 3.3: Summarization of node classes based on *users*, *stoptime* and *movement* attributes.

Personal Spot. Nodes with low number of users, but high stop time. This class stands for PoIs that a few users have visited, but they have spent long time. Therefore, they could be deemed as personal spots that are visited by a few users, and for a long time and thus it is probably of personal interest of the user. Examples of PoIs in this class are the gym or some clubs;

Popular Local. Nodes with high number of users and high stop duration, but low movement value. This represents popular places since many users are visiting the place for a long time, but in a local perspective, like places that are popular in their corresponding neighborhood. An example for this class could be a popular supermarket that is mainly visited by people from the neighborhood;

Popular Global. Nodes with high values of all the attributes. In essence, it corresponds to places that are popular, people tend to spend long time and tend to displace from distance places. An example could be an important Shopping Mall that attracts people from different parts of the city, and where they spend long time;

Hot Spot Local. Nodes with high number users, low stop time, and low spatial values. This class encompasses places where people move to spend short time moving for short distances. Possible examples could be a pharmacy or a bar, where many people that live close stop for a few minutes to buy some medicines or drinking a coffee. We could represent this class with facility places;

Hot Spot Global. Nodes with high number of users, low stop time and high movement. This class represents places that receive many people coming from many areas of the city to spend short time. We could interpret the airport as a member of this class, where people go there to bring or pick up friends or relatives and they tend to come from different parts of the city;

Undefined. All the other combinations are considered undefined, since they are not statistically meaningful w.r.t the attributes in **A**.

We can easily notice that local and global properties are mainly related to the values of the attribute movement, while hot spot and popular are related to the stop duration. This classification provides some interesting meanings to the PoIs in the city in addition to their standard categories. Indeed, while categories are static label assigned by some domain expert, these labels are given by the networks, based on where the places are located in the graph.

These attributes can still be used by location-based services taking into account the PoIs' characteristics. For instance, urban agents might identify PoIs that tend to cause traffic congestions (e.g. popular global PoIs) or PoIs for which people are willing to move far distances (e.g. global PoIs) and related them to possible traffic problems. Even location-based recommender systems could exploit the characteristics of the PoIs and users to produce meaningful recommendations.

There may be several ways to define a good threshold for splitting the distribution of the above attributes into low and high values. The details of the method chosen in this chapter are presented in Section 3.5. In general, there are at least two possibilities: exploiting domain knowledge from experts in a *top-down* fashion, or using a *bottom-up* approach where these values are directly inferred from data. In our experiments, we chose the bottom-up approach. However, our methodology is general and does not depend on the chosen strategy.

We can relate the above definitions to research **Question 1**, namely, *can we study urban mobility at a global scale from the perspective of PoIs instead of users?* We believe that the characterization of PoIs based on people's mobility is a possible way to answer this question.

Once we have identified such characteristics of the PoIs, it is still important to understand how they relate to each other at a global scale, i.e., how the movements among them create structures of groups known as communities. This leads to the research **Question 2**, i.e. *are there any patterns of such behaviors?* Finding communities is a way to find patterns in the POI network, as presented hereafter.

3.3.3 Communities of points of interests

Having the PoIs network N, we can identify communities of PoIs that are grouped based on the movements (trips) between them. This can be achieved, for example, by using the state-of-the-art algorithm presented in [3] (see Section 2.1). We obtain a set of communities $C = \{C_1, \ldots, C_n\}$ where each community is a subgraph of N, $C_i = \{(V_i, E_i) : V_i \subseteq V \text{ and } E_i \subseteq E\}, \forall 1 \leq i \leq n$. It is worth noticing that other choices of the community detection algorithm are possible, and that the resulting communities may differ, capturing different aspects of the connections among the nodes. However, our methodology does not depend on this choice, and we refer to the literature of community discovery for other algorithms that can be used based on the different kinds of connections to be extracted. In addition, the link community algorithm proposed by Ahn *et al.* in [3], is directly related to the ComeTogether framework, i.e., the framework exploits users' trips (movements) in the links of the network and link community uses as the central target the connections between the nodes, instead of the nodes, to find out the communities.

To characterize the communities extracted from the PoI network, we define some features inherent to the communities. We have considered two main scopes: *structural* and *mobility.* In particular, Let $C_i \in C$ be a community, Ei the edges of community i and V_i the nodes, we define the following features:

Compactness

It aims at measuring how trips tend to move inside the community. This function presents the levels of "fidelity" of each trip w.r.t the communities. Intuitively, the aim is to understand, from each community, if trips associated with its edges also move to edges belonging to other communities, given the notion of modular movements. Let $\mathcal{P}(Trip)$ and $\mathcal{P}(E)$ be power sets of $Trip$ and E respectively. Yet, let $\tau : C \rightarrow \mathcal{P}(Trip)$ be the function that retrieves a subset of trips that traversed a community C_i and $\sigma : \mathcal{P}(Trip) \rightarrow \mathcal{P}(E)$ is the function that retrieves a subset of edges given a subset of trips (edges traversed by the trips). Therefore, *Compactness* of community C_i can be measured as

$$Compactness(C_i) = \frac{|E_i|}{|\sigma(\tau(C_i))|}, \tag{3.1}$$

where $|E_i|$ is the number of edges of community C_i. It is computed by dividing the number of edges of the community by the number of distinct edges created by the community's trips. Values close to 1 means that the trips moved mainly over the edges of the community, while values close to 0 means the trips crossed many other edges from other communities.

Feature Similarity

Given the community C_i, let $\vec{F_i} \in \mathbb{R}^5$ be its feature vector, where each component is the number of nodes belonging to each node class (Personal Spot, Popular Local, Popular Global, Hot Spot

Local and Hot Spot Global) normalized so that the sum is 1: $\sum_j^5 \vec{F}_{ij} = 1$. Analogously, let us denote $\vec{F_N} \in \mathbb{R}^5$ the feature vector of the PoI network N. For instance, $\vec{F_i}$(Popular Global) $= 0.2$ and $\vec{F_N}$(Popular Global) $= 0.1$ stand for the probability of getting the node class Popular Global in the community C_i and in the whole PoI network N, respectively. In order to identify the communities that are very similar or dissimilar by means of the classes, we define the feature similarity $FeatureSim(\vec{F_N}, \vec{F_i})$ given by the cosine similarity between the the community feature vector and the PoI network feature vector:

$$FeatureSim(\vec{F_N}, \vec{F_i}) = \frac{\vec{F_N} \cdot \vec{F_i}}{||\vec{F_N}||||\vec{F_i}||}. \tag{3.2}$$

Note that values close to 1 mean that the community is similar to the whole PoI network in terms of node classes, while values close to 0 mean the community is dissimilar to the whole network, but it may still carry some particular characteristics.

The compactness and the feature similarity are the methods chosen to answer the research **Question 3**, namely *can we characterize these patterns?*. In section 3.5, we show the how these measures are used to characterize the communities found in an experimental evaluation conducted in three Italian cities.

3.4 Random Mobility Models

To evaluate the PoI networks built from the GPS data, we define four random models with the goal of comparing the real networks generated from real GPS data with randomly generated networks. This comparison aims at highlighting the similarities and particularities of the PoI networks. The random models are present below.

Fully Random Trip Model (FRT). Given t the number of trips, this model generates t completely randomized trips. This represents a naive approach taking into consideration none statistics from the real data.

Dist-based Random Trip Model (DRT). This model takes into account four features of a given set of trips to generate a new set of trips: (i) the number of users; (ii) number of trips per user; (iii) number of PoIs per trip; and (iv) the spatial extent of the trips. Although the trajectories are “artificial”, the features of both generated and real datasets are comparable according to the used features.

k-Random Trip Model (k-RT). This model takes into account a set of trip and randomizes the last k% points of each trip by getting PoIs of the same categories of the original PoI.

Random Graph (RG). Here the very known Erdős-Rényi Random Graph model is used to generate a random graph containing the same number of nodes and edges as the generated PoI networks. We recall that no trajectory dataset is used, but only the number of nodes and edges provided by the PoI networks.

3.5 Case Study on Different Cities

We experimented the ComeTogether methodology in a real case study involving three different cities in Italy: Milan, Florence and Pisa. We collected GPS tracks of users moving in these three cities, along with a number of PoIs downloaded from the web covering the same cities. In the following we compare the results of the analysis in these three urban areas, highlighting the main similarities and differences, giving possible explanations of the main interesting findings.

We follow the flow presented in the previous section, presenting, after the data and tools, the results of for each step composing the ComeTogether framework: building the PoI networks for each city; (ii) PoI network analysis; and the community discovery process over the built PoI networks.

3.5.1 Data and Tools

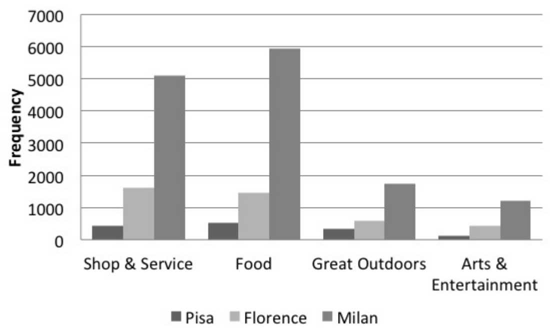
To build the PoI networks we essentially needed two datasets: a set of traces of users moving in a given area and a set of PoIs located in the tracked area. We focused the analysis in three well known cities in Italy - Milan, Florence and Pisa - characterized by some similarities and differences: Pisa and Florence are historical cities and oriented to tourism, whereas Milan is more business-oriented. In addition, Pisa is a small city, while Milan and Florence are larger than Pisa; Milan is located in the north of Italy while Florence and Pisa are in the center. Finally, Pisa is close to the sea while Florence and Milan are interior cities.

We downloaded the PoIs of these three cities from Foursquare[1] resulting in $1,403$ points of interest in Pisa, $4,074$ in Florence and $13,948$ for Milan. At the time of this writing, Pisa counts over $88,000$ inhabitants (around $200,000$ with the metropolitan area); Florence is the most populated city in Tuscany with approximately $370,000$ inhabitants, expanding to over 1.5 million on the metropolitan area; Milan has a population of about 1.35 million and about 8 million on the metropolitan area. Figure 3.4 shows the category distribution of the points of interest in the three focused cities: the four categories are *Shop & Service*, *Food*, *Great Outdoors* and *Arts & Entertainment*. Figure 3.4(a) shows the absolute number of PoIs for each category, while Figure 3.4(b) the values are normalized by the number of PoIs. As we can see, when we normalize by number of PoIs, the categories appear to be represented in similar relative percentages for the three cities. Moreover, we can see how the points of interest spread around each city in Figure 3.5. Milan is clearly the biggest city and more oriented to business leading to the highest number of PoIs.

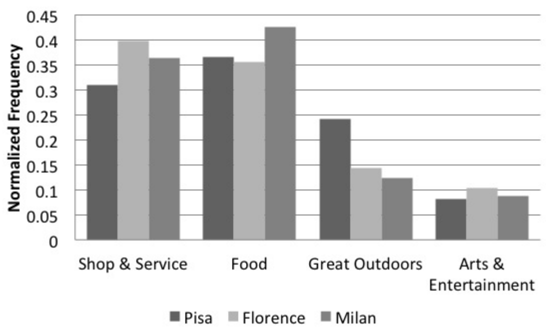
Our analysis is focused on four PoIs categories, since we believe that these categories cover the majority of the activities performed by people in an urban environment. We have discarded categories related to residence and work places. This choice has been taken since we would like to capture user movements in the city when related to the services that the city provides. This thus excludes places related to home and work.

The user traces were collected by the Italian company Octotelematics[2]. This company installs GPS devices on cars of citizens benefiting from an insurance discount. We have the traces of $42,775$ cars for a period of 5 weeks from May to June 2011 covering all the Tuscany region (thus including

[1] `http://www.foursquare.com`
[2] `http://www.octotelematics.it`

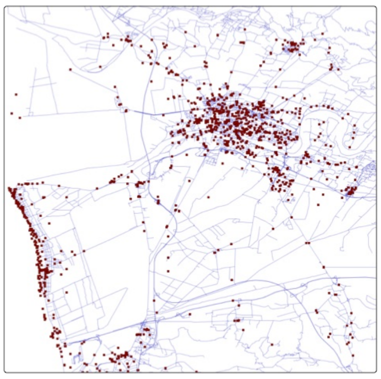


(a)

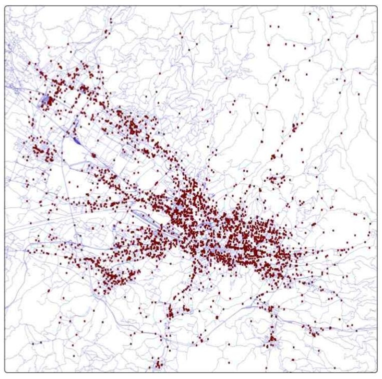


(b)

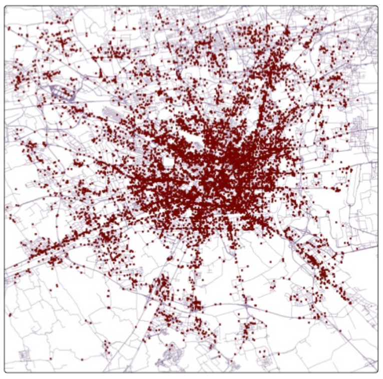
Figure 3.4: Category distribution for each city: Pisa, Florence and Milan. Four categories are considered: Shop & Service, Food, Great Outdoors and Arts & Entertainment. (a) absolute values and (b) for normalized values.



(a)



(b)



(c)

Figure 3.5: Points of interest for each city: (a) Pisa, (b) Florence and (c) Milan.

Florence and Pisa). We also have an additional car GPS dataset of 17, 087 users covering Milan for one week on April 2007.

3.5.2 Building the PoI network

Aggregated PoIs

As presented in the Section 3.3, the first preprocessing step for the PoI dataset is grouping together PoIs that are very close to each other, thus finding the Aggregated PoIs: these will be represented by the nodes of the PoI network. We choose a clustering approach to automatically find dense groups of PoIs. Alternative ways to build the Aggregated PoIs could be to use a fixed radius or to fix a spatial grid and aggregate the PoIs belonging to the same cell. Since the analyzed areas are characterized by very dense PoI regions alternated to more sparse regions, we decided to apply the density based clustering approach to aggregate the POIs.

We have used DBScan [60] as a density-based clustering algorithm. We used 200 meters and 1 member as parameters for the distance and neighborhood values. These values have been empirically selected running the algorithms with different parameters and finding a trade off between having too many single-member GPoIs and too large ones. The neighborhood parameter was set to one as we did not want to discard small clusters with only one element when those are the only one in the area (sometimes a very important place is not close to any other like an airport). The 200 meters value has been shown to be a good parameter for distance in our scenario, since larger values create very large cluster encompassing almost all the PoIs in the city.

Creating Trajectories and Trips

First of all, we computed the users' candidate stops. We set up 150 meters for $MaxStopArea$ and 20 minutes for $MinStopTime$ for the three cities. The assignment function f associating a stop to a GPoI is defined to assign for each user's candidate stop the closest aggregated PoI within the spatial distance of 100 meters. After defining f, we computed the trajectories - as time-stamped sequences of GPoIs- for the three cities.

The trajectories were also split into weekdays (WD) and weekends (WE). The observation behind this is that the people behaviors in terms of mobility and activities during the weekdays and weekend tend to significantly change. Thus, we want to capture the possible differences in the mobility of users in these two periods. Furthermore, we split each trajectory dataset into *trips*. We recall that the idea behind trips is to capture the continuous movement of the users between PoIs thus discarding long stops. We set 5 hours for Pisa and 6 hours for Florence and Milan as the threshold Δm to split trajectories into trips.

These values were empirically evaluated from the data and represent the intuition that a long stop (e.g. being at work or at home) cuts the continuity of movement between activities. With the above settings, the largest connected components (LCC) of the resulting networks contain at least 99% of the nodes.

3.5.3 PoI Network Analysis

We now present some basic analysis of the generated PoI networks for each city.

Table 3.1 summaries the generated PoIs networks for each city with its respective number of nodes and edges. As we can observe, the number of nodes is different from the number of PoIs since the nodes represent Aggregated PoIs. It is worth highlighting the difference between Milan and Florence. Looking at Figure 3.4 we clearly note that there are much more POIs in Milan than in Florence. However, Milan is spatially denser w.r.t PoIs (Figure 3.5), which reflects the fact that the number of nodes in Milan is smaller than in Florence when the PoIs are aggregated into aggregated PoIs. However, Florence is denser as network possibly due to the time span available of 5 weeks for Florence.

Table 3.1 also shows the properties of the networks, which include: average clustering coefficient (Avg. CC.), average degree of the nodes (k) and average shortest path length (l). Comparing to other results in the literature [109], we observe that the properties are consistent to the properties in real networks, such as biological networks and social networks. In particular, we see the popular small-world phenomenon in the PoI networks.

	Weekdays				
	Nodes	Edges	Avg. CC	k	l
Pisa	633	9310	0.2661	29.4154	2.4072
Florence	1837	65201	0.2316	70.9863	2.2566
Milan	1690	33876	0.2129	40.0899	2.4607
	Weekends				
	Nodes	Edges	Avg. CC	k	l
Pisa	576	3555	0.1922	12.3437	2.8831
Florence	1734	21198	0.1735	24.4498	2.7672
Milan	1575	10291	0.2271	13.0679	3.2008

Table 3.1: PoI Network properties for Pisa, Florence and Milan. Number of nodes, number of edges, average clustering coefficient (Avg. CC), average degree of the nodes (k), and average shortest path (l).

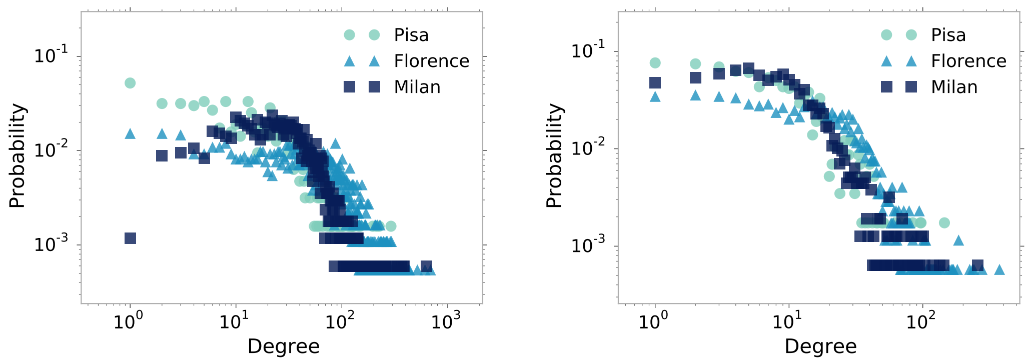


Figure 3.6: Degree distribution of the networks for weekdays (a) and weekends (b).

Pisa tends to concentrate the movements on weekdays mainly inside the city, while people tend to move to the coast on weekends (we remind that the dataset considered covers a summer period).

Looking at the clustering coefficient (Avg. CC) in Table 3.1, we report an interesting result. While in weekdays the clustering coefficient seems to be negatively correlated with the size of the cities, this correlation disappears in the weekends. This may suggest that, while the cities are comparable in terms of services and locations related to business, they offer very different attractions for the weekend, resulting in different patterns of mobility. We can also see in Table 3.1 that the average shortest path length l is smaller in the weekdays networks, due to a higher density of the networks.

Figure 3.6 shows the degree distribution of the networks. We observe that a few nodes (PoIs) are highly connected, while many nodes are sparsely connected. There are a very few places in the cities that link to many others such as very popular places like famous shopping malls or business areas. Most places in the cities tend to connect to a few other places. In our context, highly connected places are good candidates to attract movements and, consequently, induce more traffic within the urban area. We also see from Figure 3.6 that the degree distribution in our network does not depend on the size of the networks.

In Figure 3.7 we report the PoI network in Pisa, the smallest city considered. It can be clearly seen that visual analytics may not be of much help in this case. The rest of the methodology is intended to overcome this limitation.

Confronting PoI Networks and Random Models

Figure 3.8 shows the comparisons between the degree distribution of the randomly generated networks and the PoI networks. We recall that $0\% - RT$ corresponds to the original PoI network. There is a clear distinction between FRT and RG to the other networks. Note that in FRT and RG networks the node degrees tend to follow the average degree of the network.

Node classes

Here we address the research **Question 1** presented in Section 3.1: *Can we study urban mobility at a global scale from the perspective of places, instead of users?* As discussed earlier, we use a *bottom-up* approach to derive the values for the classes directly from the data. For this, we computed the median of each attribute (*users*,), and we classified as *low* those values lower than or equal to the median, and *high* otherwise. While this approach is limiting (only two classes, no usage of background information), it has the clear advantages of being simple, and easily replicable from city to city, without loss of semantics.

Figure 3.9 summarizes the percentage of the population of each class for each network. As we can see, about 70% of the nodes were classified into some class and only 30% were unclassified (except for Florence on weekends). We note a relevant number of *personal spots*, that reflect the fact that some PoIs are visited very often by a low number of people (like a gym or a neighborhood shop). On the other hand, we see a few PoIs attracting many people, measured by the percentage of *hot spot* and *popular* classes in the three cities.

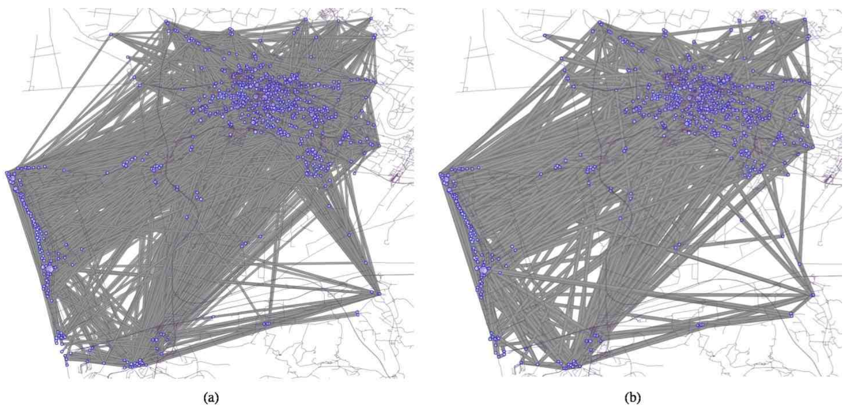
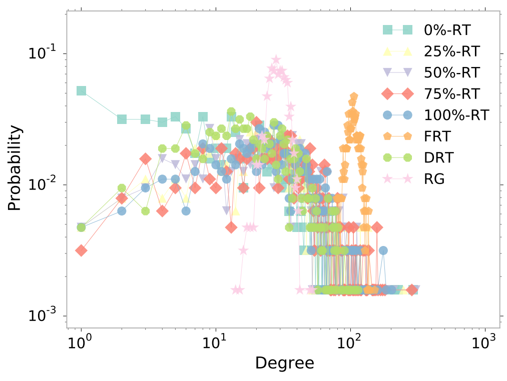
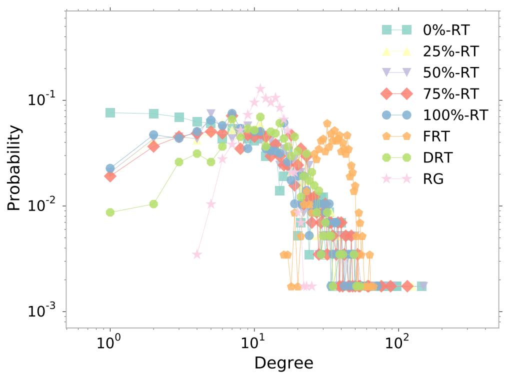


Figure 3.7: POI Network of Pisa in weekdays(a) and weekends(b)



(a) Pisa in weekdays



(b) Pisa in weekends

Figure 3.8: Comparison between PoIs Network and randomly generated networks.

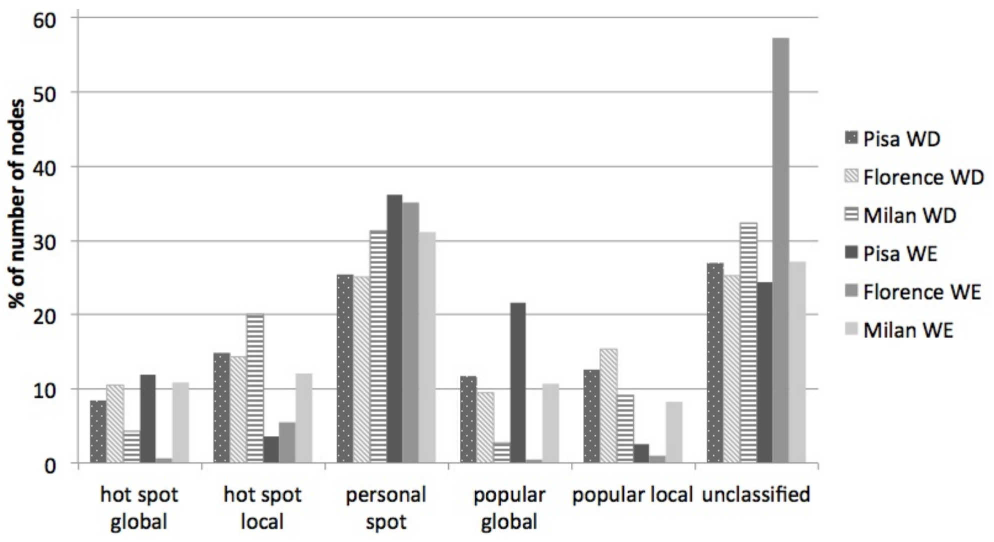


Figure 3.9: Node classes in the three networks for weekdays and weekends. Axis y corresponds to the % of number of number of each class.

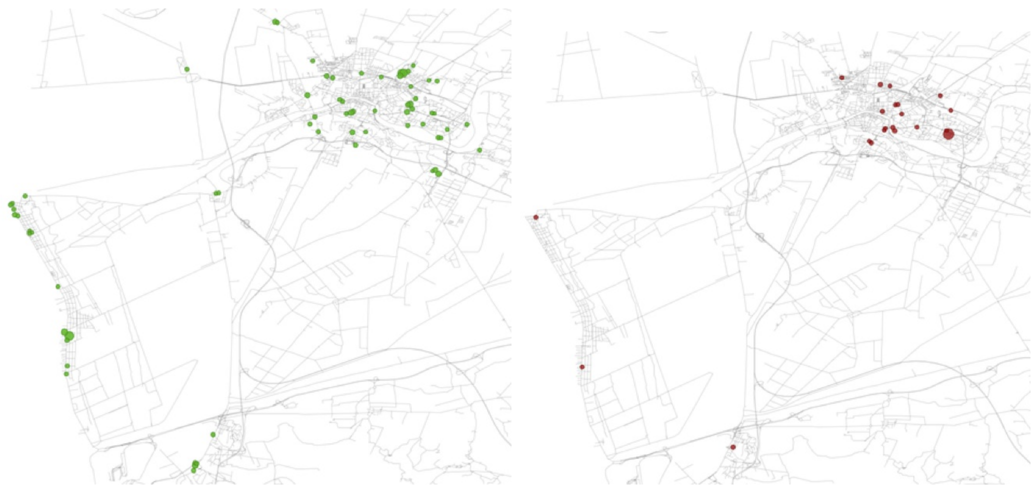


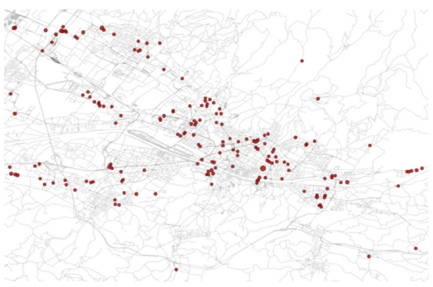
Figure 3.10: Hot spot global (a) and local (b) nodes in Pisa on weekends. We can see more global PoIs on the beaches and the concentration of local ones on the city center.

Looking at the results for Pisa, we can see that there are more global classes (*hot spot* and *popular*) on weekends than on weekdays. This reflects the possible behavior of coastal cities, where people tend to go to the beaches on weekends. In the case of Pisa, people usually move far distances to reach beach places represented by the global classes. Also, on the weekend there is an increasing number of *popular global* nodes, which means that we could find more places visited by many people, spending much time and moving longer distances to reach these places. Again, this behavior is mainly due to the popularity of the beaches that are a slightly more distant, around 10 kilometers and are mostly visited on weekends.

To get more insights, we looked at some PoIs with high number of users. We found, for example, *Centro Commerciale Carrefour*, which is a supermarket where people stop to buy groceries and home goods, as a *hot spot local*. People clearly move to this supermarket only for quick activities. A different case is a node including several points of interest like *Osteria la Griglia del maestro* and *Tirrenia Caffe Doc*, i.e., a restaurant and a bar, which are classified as *hot spot global*, since people tend to reach them from far places.

An additional step is to investigate where a specific node class is present in the city as shown in Figure 3.10a and Figure 3.10b. Interestingly, we see more hot spot global PoIs in Pisa than hot spot local PoIs- Figure 3.10a. However, hot spot local PoIs are mainly concentrated on the city center during weekends as shown in Figure 3.10b.

Recalling Figure 3.9 for Florence, we see that the percentage of both global and local hot spot are lower in weekends than in weekdays, while the percentage of personal spot increases. This shows that very few PoIs were visited by a high number of users, while most of the PoIs were visited by a low number of users, which contributes to the increase of personal spot as well as unclassified nodes. In fact, on weekends, there is a high percentage of unclassified nodes in Florence. Looking at Figure 3.11 we can see how the popular global (Figure 3.11a) and local (Figure 3.11b), nodes are spread around the city on weekdays, for instance. The figure shows that the popular global


(a)

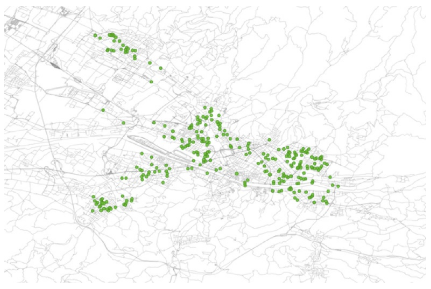
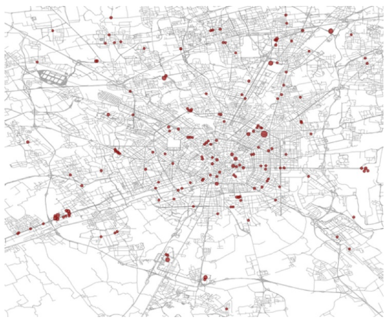

(b)

Figure 3.11: Popular global (a) and local (b) nodes in Florence on weekdays.


(a)

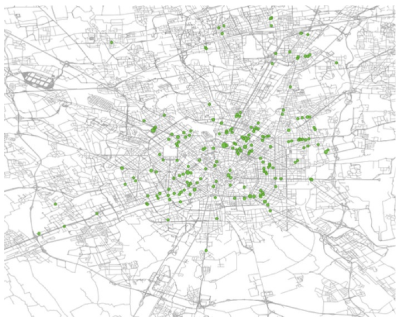

(b)

Figure 3.12: Hot spot global (a) and local (b) nodes in Milan on weekends.

nodes tend to be more spread in the city, while the popular local nodes are densely located in some specific areas of the city.

In Milan it is worth highlighting the growing percentage of hot spot global and popular global nodes from weekdays to weekends. This is an interesting result: in Milan, people tend to cover longer distances on weekends to arrive at more distant places than on weekdays no matter how long they are going to be there. Consequently, there is a decrease in the percentage of hot spot local and popular local nodes. Then, we observe that people tend to cover shorter distances from place to place on weekdays, while they appear to cover longer distances on weekends. In fact, in large urban areas, people move to close places mainly due to work daily routine and the long traffic lines in the streets. This is particularly different during weekends, likely on regular weekends without any special event and season, when people tend to go out less and, consequently, it is more likely to find shorter or none traffic lines. Figure 3.12 shows how the hot spot global (3.12a) and local (3.12b) nodes are distributed in Milan on weekends. Note that the airport gets clearly classified as hot spot global, as people cover longer distances to get there, then move back after short time.

3.5.4 Community discovery in PoI networks

After a first analysis on the PoI networks looking at the location and the classification of PoIs in the city, the second step is to look at the communities found in the network. This step mainly addresses the research **Question 2**: are there any patterns in these mobility networks that we

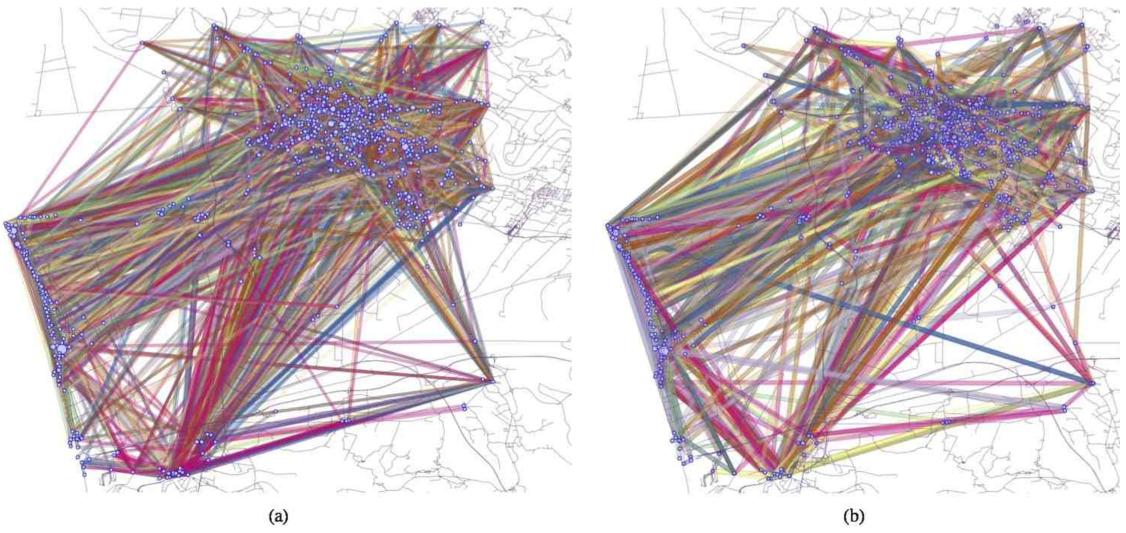


Figure 3.13: Communities in the PoI networks of Pisa for weekdays (a) and weekends (b).

can detect? The detection of the communities was performed by using the algorithm[3] proposed in [3], since it focuses on *link communities*, which is a good way to capture the mobility aspects of our networks, and also allows overlapping among communities. In addition, the weights of the links (number of trips) are considered in the community discovery process. Note that there are other interesting methods for discovering communities in networks, such as [51, 21, 20], and we plan to experiment the methodology with different community detection algorithms in the future, to better understand the mobility phenomena in our networks.

Figure 3.13 represents the same networks of previous Figure 3.7 with the difference that here the edge colors are mapped to the specific communities they belong to. Again, we can see that the visual approach is limited here and further analysis is needed.

Table 3.2 presents the number of communities found in each network. Figures 3.14 and 3.15 show the community size distribution and compactness cumulative distribution, respectively. One result that we see is that the distribution of community size tends to follows a power law, where a few communities have a great number of nodes, while many communities have a small number of nodes.

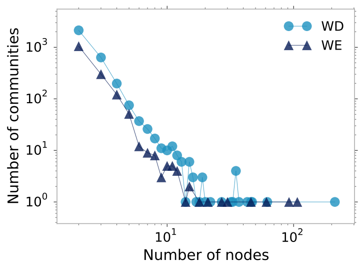
We can observe in Figure 3.14 that the community size distributions are very similar for all the networks. All of them present a skewed distribution, where a few communities contain many nodes, while many communities are formed by a few nodes. This result shows that the global mobility behavior tend to form only a few very connected groups (communities), while most communities are represented by particular movements in the city. The slight difference between networks on weekdays and weekends is due to the size of the networks in each period: networks on weekdays are larger.

Regarding Figure 3.15, we analyze the cumulative distribution of compactness of the communities. In Milan (3.15c) the communities present higher compactness compared to communities in Pisa (3.15a) and Florence (3.15b). In addition, there is a little difference between weekdays and

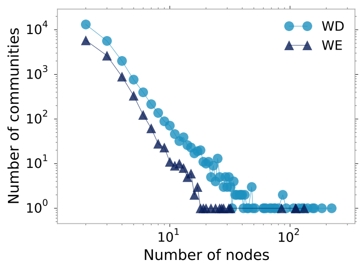
[3]The authors provide their implementation at `http://barabasilab.neu.edu/projects/linkcommunities/`

	WD	**WE**
Pisa	3197	1580
Florence	22834	9977
Milan	12313	4014

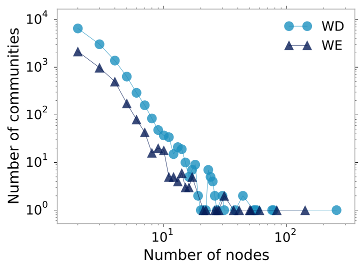
Table 3.2: Number of communities found in Pisa, Florence and Milan on weekdays (WD) and weekends (WE).



(a)



(b)



(c)

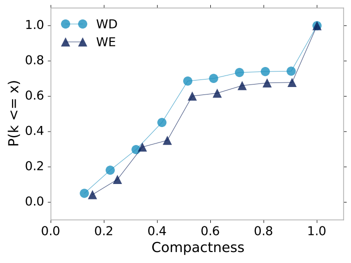
Figure 3.14: Community size distribution for each city: (a) Pisa, (b) Florence, (c) Milan; on log scale.

weekends in Milan. Since Milan is much denser, people do not have to travel around the city to find what they need and, besides, the movements tend to be longer than in Pisa and Florence.

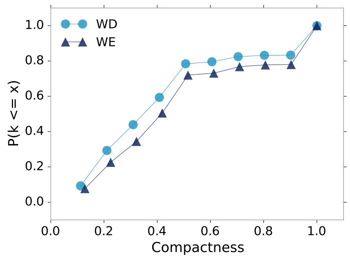
In contrast, Pisa presents the highest difference in compactness between weekdays and weekends, which means that people tend to travel around the city to get some amusements. Since Pisa is the smallest city, it is easier to reach any part of the city from any other and, consequently, this contributes for lower compactness.

3.5.5 Largest Communities

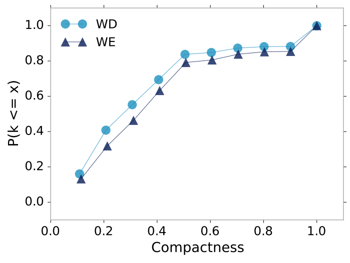
In front of the difficulty of analyzing such high number of communities, we present here some analysis about the largest communities discovered in our framework. These communities are more



(a)

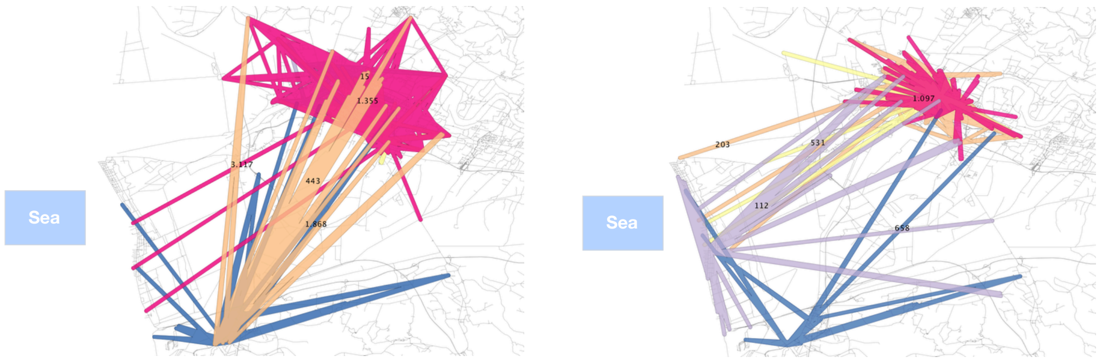


(b)



(c)

Figure 3.15: Compactness for each city: (a) Pisa, (b) Florence, (c) Milan.



(a) (b)

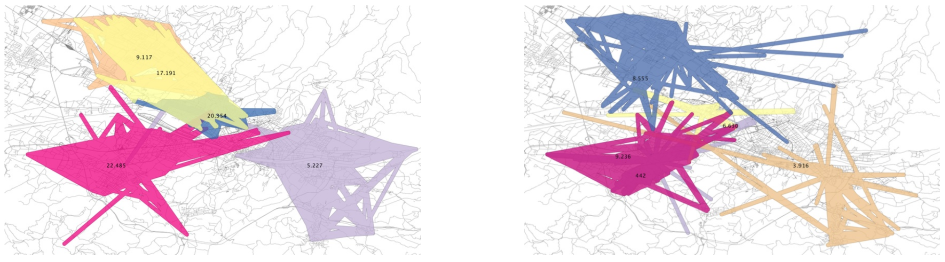
Figure 3.16: The five largest communities in Pisa on weekdays (a) and weekends (b).

representative in the sense that they encompass the majority of the movements in the city and, therefore, they are useful to give a deeper understanding. The five largest communities for each city on weekdays and weekends are presented in the following.

Figure 3.16 illustrates the selected largest communities in Pisa. On weekdays, Figure 3.16a, the movements are mainly between the center of Pisa and a nearby village. In addition, in the center of Pisa, there are some communities that clearly overlap: they contain some common POIs that are on their "borders". On weekends (Figure 3.16b), on the other hand, one of the largest communities comes out throughout the coast, featuring the typical mobility pattern in the area: the citizens tend to spend weekends on the beaches. We see that the mobility in the center of Pisa encompasses some of the largest communities and, consequently, significant concentration of movements among PoIs.

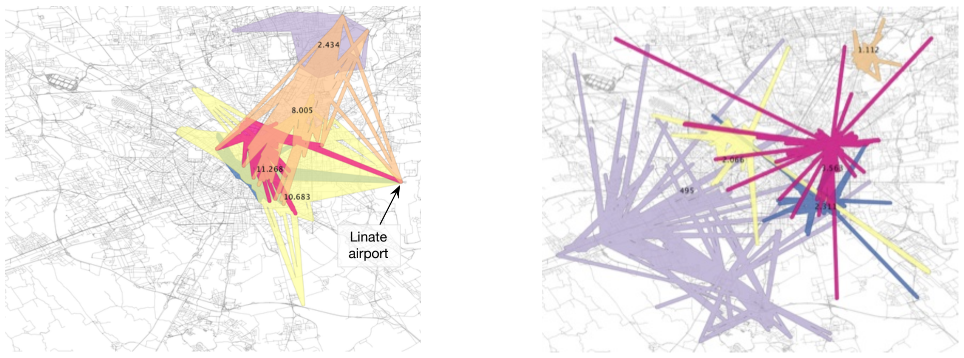
The five largest communities found in Florence are presented in Figure 3.17. On weekdays (Figure 3.17a) we can see an interesting splitting of the city. Interestingly, the communities tend to divide the city based on the connectivity (movements) among the POIs with overlapping among some communities. For instance, the community B (yellow) overlaps both community A (orange), in the northwest part of the town, and community C (blue), in the city center. Community B (yellow) highlights the interplay between two parts of Florence represented by community A (orange) and C (blue). The communities on weekends are in Figure 3.17b. By comparing communities on weekdays and weekends we can note the apparent presence of some communities on both periods, as well as some particular communities inherent to each time period. This observation raises the importance of developing techniques for dynamic community discovery algorithms in order to understand the evolution of the communities. However, this issue is out of the scope here, and we leave it for future work.

Figure 3.18 depicts the five largest communities in Milan on weekdays (Figure 3.18a) and weekends (Figure 3.18b). On weekdays, the movements among POIs mainly take place in the city center and on the north. The Linate airport (the one at the East part of the city highlighted in Figure 3.18a) interestingly takes part of some the largest communities on weekdays. Community B depicted in orange color stresses the interplay between the city center and north of Milan, represented by different communities.



(a) (b)

Figure 3.17: The five largest communities in Florence on weekdays (a) and weekends (b).



(a) (b)

Figure 3.18: The five largest communities in Milan on weekdays (a) and weekends (b).

On weekends (Figure 3.18b), in turn, we note the change from weekdays: the community on southeast comes out as one of the largest communities, gathering one of the major concentrations of movements in the city. Here, the Linate airport is not part of any of the five largest communities on the weekends, which tells us that on weekdays the airport plays an important role in the largest communities, and therefore, in the areas of the communities for which it takes part. This information is extremely useful for traffic agencies in order to capture probable places that cause traffic (the airport is one of them indeed). The airport still takes part of large communities on weekends (but not in the top 5).

The largest communities, therefore, encompass the global mobility behavior in the cities. They also show how the city is broken into areas that are connected each other by movements among POIs. This information is extremely useful for mobility manager because they can understand how traffic is forming in different areas of the city, and how the movement between different parts may contribute to produce traffic. For instance, large communities themselves are likely to be more dense of traffic since they represent large communities of POIs reached by people during the same trips. We can intuitively understand that this phenomenon can be worsened when other large communities overlaps.

3.5.6 Comparing Communities against the Network

We address here the research **Question 3** about finding regularities and anomalies in the patterns found. In order to do this, we make use of the $FeatureSim$ measure presented in Section 3.3 in the following way: we consider the global, city-wide mobility, as a feature vector filled with the percentages of POIs in each class (hot spot local, etc.). We then compute the same vectors for each single community extracted, and we compute its $FeatureSim$ with the city-wide feature vector. Then, we analyze the most similar and most dissimilar 5 communities. The intuition is that they should represent, respectively, patterns that globally reflect the mobility of the cities, and patterns that represent exceptions, or most typical characterization of the non-average mobility in a city.

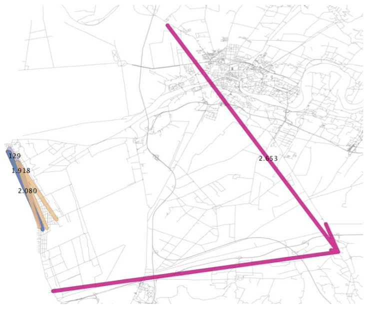
Tables 3.3, 3.4 and 3.5 present the least and the most similar communities to their respective networks, while Figures 3.19 (Pisa), 3.20 (Florence) and 3.21 (Milan) illustrate the communities on the map.

As previously discussed for Pisa, the common behavior on weekdays is mainly on the city center, while the typical mobility on weekends tend to reach the beaches. Looking at Figure 3.22, we confirm that the least similar communities are present on the beach (Figure 3.19a), while the most similar communities are present in the city center (Figure 3.19b) on weekdays. In contrast, the most similar communities on weekends are mainly located on the beach (Figure 3.19d). The reasoning is analogous for Florence and Milan in order to understand what seems to be global and particular pattern in each city. Although we have given some interpretation, more understanding would come with aid of urban agents of each city.

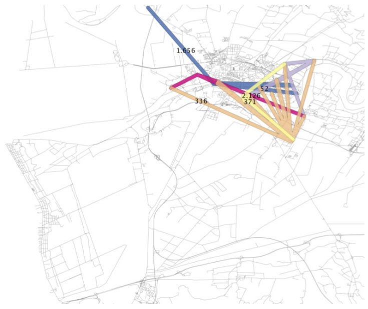
A further confirmation of the fact that the ComeTogether approach is providing a new perspective in analyzing human mobility connected to the visited places is given by Figure 3.22. It shows that no approaches based on frequency (e.g. frequent pattern mining) may lead to the same results as above, as there is no correlation between similarity and frequency (number of user and number of trips), i.e. there might be important communities that would not be found by looking at frequent patterns. The patterns may be used and interpreted by a mobility manager with background knowledge for understanding as well as making decision over mobility aspects in the city.

3.6 Discussion

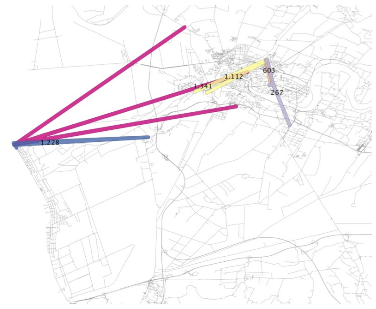
In this chapter, we have seen an exploratory study on the relation between people mobility and points of interest of an urban area at the global scale. We have based our work on complex network analysis combining mobility of users into a graph structure called PoI network, from which we have defined interesting features discussed throughout this chapter. The PoI networks of three cities have been analyzed to address the three research questions presented in Section 3.1. We have shown through our study how we may answer them: (1) *Can we study urban mobility at a global scale from the perspective of places, instead of users?* Our answer is positive, and we have defined different classes of places based on network properties to this aim; (2) *Are there any patterns of such mobility w.r.t places?* We have shown how to extract communities from the POI Network graph to find mobility patterns between the places; (3) *Can we characterize such patterns*



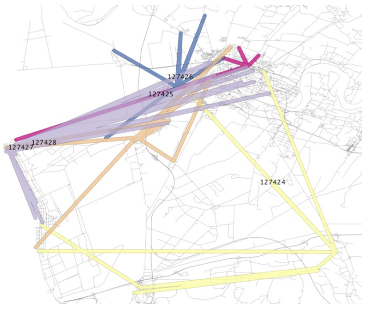
(a) Weekdays the least



(b) Weekdays the most similar

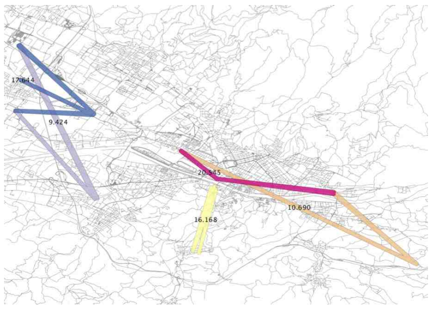


(c) Weekends the least similar

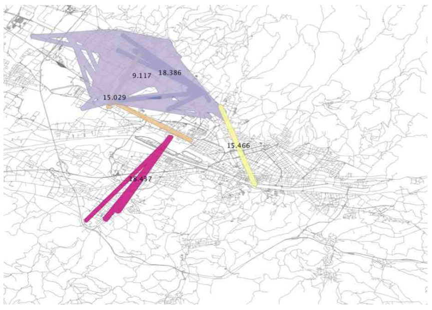


(d) Weekends the most similar

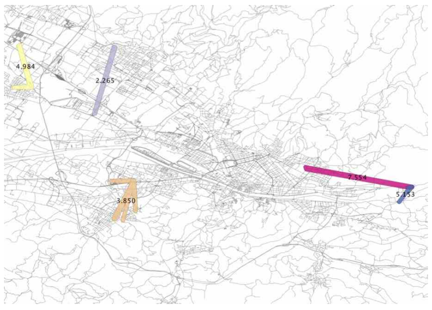
Figure 3.19: The least and most similar communities to the network in Pisa.



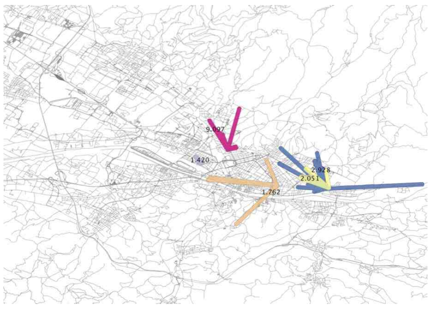
(a) Weekdaus the least similar



(b) Weekdays the most similar

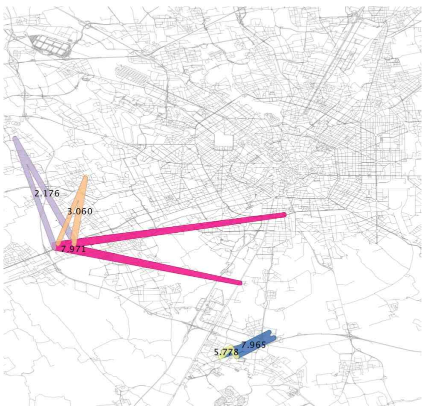


(c) Weekends the least similar

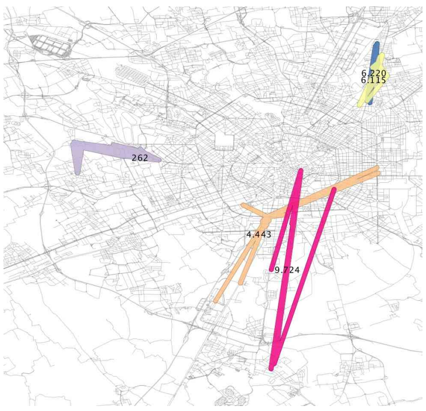


(d) Weekends the most similar

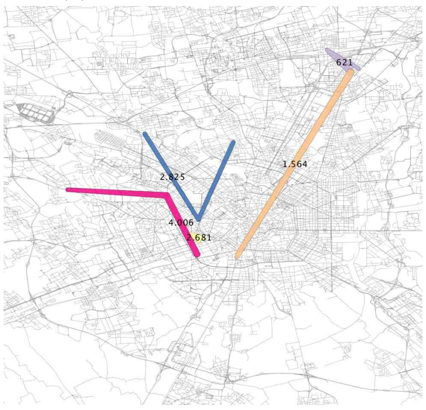
Figure 3.20: The least and most similar communities to the network in Florence.



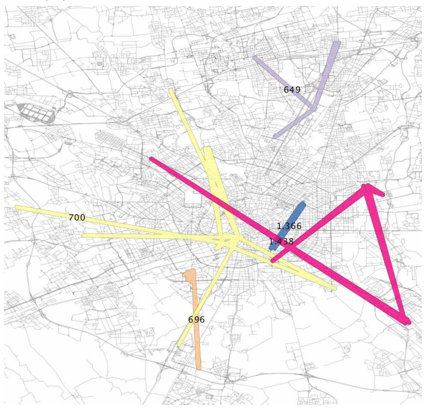
(a) Weekdays the least similar



(b) Weekdays the most similar



(c) Weekends the least similar



(d) Weekends the most similar

Figure 3.21: The least and most similar communities to the network in Milan.

Table 3.3: Pisa

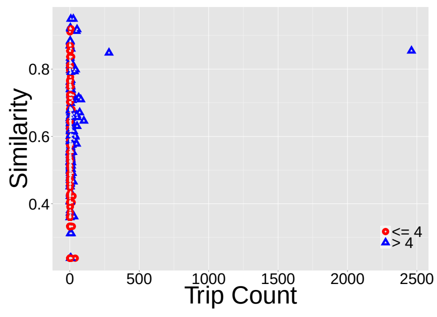
	Least similar			*Most similar*		
	Community	**#Nodes**	**Similarity**	**Community**	**#Nodes**	**Similarity**
Weekdays	1918	5	0.3120	52	6	0.9491
	129	5	0.2384	371	6	0.9491
	1353	5	0.2384	2126	5	0.9231
	2080	5	0.2384	1656	7	0.9197
	2653	5	0.2384	336	14	0.9181
Weekends	1112	5	0.3799	820	12	0.9792
	1228	5	0.3799	72	21	0.9617
	1341	5	0.3799	1406	6	0.9566
	267	5	0.2690	1256	7	0.9552
	603	5	0.2690	180	12	0.9431

Table 3.4: Florence

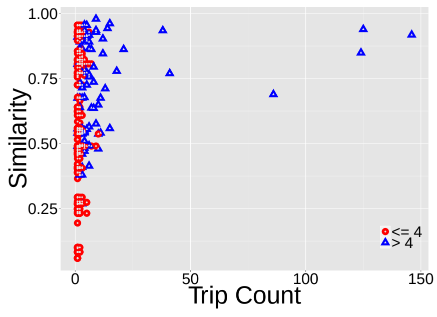
	Least similar			*Most similar*		
	Community	**#Nodes**	**Similarity**	**Community**	**#Nodes**	**Similarity**
Weekdays	9424	5	0.2673	9117	140	0.9924
	10690	5	0.2673	15029	5	0.9578
	16168	5	0.2673	15466	5	0.9578
	17644	6	0.2673	18386	13	0.9569
	20545	5	0.2673	18437	8	0.9469
Weekends	2265	5	0.0260	2928	15	0.9993
	3850	9	0.0260	9097	8	0.9984
	5153	5	0.0241	1420	5	0.9952
	7554	5	0.0241	1762	5	0.9952
	4984	5	0.0146	2051	8	0.9952

Table 3.5: Milan

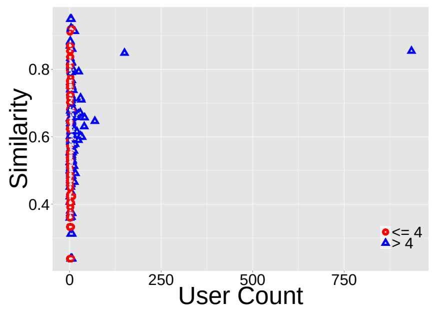
	Least similar			*Most similar*		
	Community	**#Nodes**	**Similarity**	**Community**	**#Nodes**	**Similarity**
Weekdays	2176	6	0.1117	9724	6	0.9907
	3060	5	0.1117	6220	9	0.9877
	5778	5	0.1117	262	15	0.9839
	7965	6	0.1117	6115	17	0.9793
	7971	5	0.1117	4443	13	0.9737
Weekends	4006	5	0.2826	700	15	0.9767
	621	5	0.2820	1438	10	0.9745
	1564	5	0.2200	649	5	0.9632
	2681	7	0.2200	696	8	0.9632
	2825	5	0.2200	1366	5	0.9632



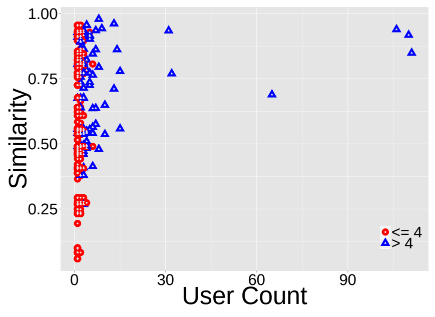
(a) Weekdays



(b) Weekends



(c) Weekdays



(d) Weekends

Figure 3.22: Correlation between number of trips (trip count) and *FeatureSim* in Pisa.

and find regularities or anomalies? We have defined two measures, namely *Compactness* and the *FeatureSim*, and shown how to use them in order to observe possible global and local patterns in each city. We have experimented our approach using real GPS datasets collected in three Italian cities with distinct characteristics.

Although the proposed PoI network is represented by an undirected graph, our methodology can be extended to directed PoI network when it is desired to capture the sequence of the visited PoIs. Note that, it is important to consider a community discovery algorithm able to work with directed edges in the network. The selected algorithm has been proposed by Ahn et al. can be extended for directed networks as presented in [3].

The results presented in this chapter have been extremely important to envision the investigation of two other problems presented in Chapters 4, 5 and 6, namely the sightseeing tours recommendation, user friendly web application and the group formation problem, that are addressed in this thesis with a number of contributions.

Chapter 4

Planning sightseeing tours based on the wisdom-of-the-crowd

Social networking services jointly with the advance of smart-phone devices have been essential to the collecting of an unforeseen amount of data generated by millions of users during their daily activities. The collective behavior found out in the data, known as wisdom-of-the-crow, can be a striking and powerful mechanism to devise new opportunities. As such, this chapter focuses on the research question **RQ2**: *Can we take advantage of data provided by the wisdom-of-the-crowd to support users (e.g. tourists) in planning their vacations to a new destination?*

This chapter presents TripBuilder, an unsupervised framework for planning personalized sightseeing tours in cities. To this aim, we exploit categorized Points of Interests (PoIs) from Wikipedia and albums of geo-referenced photos from the photo sharing social network of Flickr considered as traces revealing the behaviors of tourists during their sightseeing tours. We extract from photo albums spatio-temporal information about the itineraries made by tourists, and we match these itineraries to the Points of Interest (PoIs) of the city. The task of recommending a personalized sightseeing tour is modeled as an instance of the Generalized Maximum Coverage (GMC) problem, where a measure of personal interest for the user given her preferences and visiting time-budget is maximized. The set of actual trajectories resulting from the GMC solution is scheduled on the tourist's agenda by exploiting a particular instance of the Traveling Salesman Problem (TSP). Experimental results on three different cities show that our approach is effective, efficient and outperforms competitive baselines. This chapter is based on the published works [34, 35, 29].

4.1 Introduction

Tourists approaching their destination for the first time have to deal with the problem of planning a sightseeing itinerary that covers the most subjectively interesting attractions, and fits the time available for their visit. Precious information can be nowadays gathered from many digital sources, e.g., travel guides, maps, institutional sites, travel blogs. Nevertheless, the users still need to choose the preferred PoIs, guess how much time is needed to visit them and to move from one attraction to the next one. In this chapter we discuss TripBuilder, an unsupervised system helping tourists to build their own personalized sightseeing tour. Given the target destination, the time available for the visit, and the user's profile, our system recommends a time-budgeted tour that maximizes user's interests and takes into account both the time needed to enjoy the attractions and to move from one PoI to the next one. Moreover, the knowledge base feeding TripBuilder recommendation model is entirely and automatically extracted from publicly available Web services, namely, Wikipedia, Flickr and Google maps.

We observe that an increasing number of tourists share their travel photos on social networks. Unofficial estimates state that Flickr, one of the most popular photo-sharing platforms, collected about 518 million of public photos in 2012[1]. Each photo comes with very useful information such as: tags, comments and likes from Flickr social network, number of views, information about the user, timestamp, GPS coordinates of the place where the photo was taken. This allows us to roughly reconstruct the movements of users and their interests by analyzing the time-ordered sequence of their photos. However, the process of recognizing relevant PoIs given such set of photos is not trivial due to the noise present in the data. User tags are in many cases missing, wrong, or irrelevant for our purposes (e.g., *me and Ann, travel to Europe, Easter 2012*). Moreover, information available may be sparse and characterized by a skewed distribution.

Fortunately, in Wikipedia[2], we can find that most entities of interest for tourism are described in a dedicated page from which we can extract: the (multilingual) name of the PoI, its precise geographic coordinates, the categories to which the PoI belongs according to a weak but robust ontology (i.e., the PoI is a church, a square, a museum, a historical building, a bridge, etc). By spatially joining and re-conciliating tourists' photo albums and related information from Flickr with relevant PoIs data extracted from Wikipedia pages, we can derive a knowledge base that represents the behavior of people visiting a given city[3]. In this knowledge base the popularity of a PoI is estimated from the number of visitors that shot photos there, while from the timestamps of the first and last photos taken in a PoI we estimate the average time spent for the visit. Finally, we exploit the Wikipedia categories of the PoIs visited by a given tourist to build her user profile. For example, when a user takes many pictures of churches and museums, we can infer a preference for cultural/historical attractions. Analogously, we can aggregate this information at the level of itinerary to build a profile for each frequent visiting pattern.

We address the problem of planning the visit to the city as a two-step process. First, given the profile of the user and the amount of time available for the visit, we formalize and address the TripCover problem: choosing the set of itineraries across the PoIs that best fits user interest and respects the given time constraint. Then, the selected itineraries are joined in a sightseeing itinerary by means of a heuristic algorithm addressing the Trajectory Scheduling Problem (TrajSP), a particular instance of Traveling Salesman Problem (TSP).

The chapter is structured as follows: Section 4.2 introduces the TripCover problem and the

[1] `http://www.flickr.com/photos/franckmichel/6855169886/`
[2] `http://www.wikipedia.org`
[3] Hereinafter, we will consider cities as the destination targets of our users, although our technique is general and scale-independent.

approximation algorithm used to solve it. Moreover, the TRAJSP problem is defined and addressed in Section 4.3. Section 4.4 details the unsupervised method that builds the knowledge base, while Section 4.6 presents the experiments we perform to assess the effectiveness and the efficiency of our solution. Finally, Section 4.7 we conduct a final discussion of the chapter.

4.2 The TripCover Problem

Let $\mathcal{P} = \{p_1, \ldots, p_N\}$ be the set of PoIs in our city. Each PoI p is univocally identified by its geographic coordinates, a name, a radius specifying its spatial extent, and a *relevance vector*, $\vec{v_p} \in [0,1]^{|C|}$, measuring the normalized relevance of p w.r.t a set of categories C.

Symmetrically, let u be a user from the set $\mathcal{U}$, and $\vec{v_u} \in [0,1]^{|C|}$ the *preference vector* stating the normalized interest of u for the categories in C. The preference vector can be explicitly given by the user, or implicitly learned. Without loss of generality, we assume to know in advance the categories C, the relevance vectors $\vec{v_p}$, and the preference vectors $\vec{v_u}$ for all PoIs and users.

Definition 10 (User-PoI Interest) *Given a PoI p, its relevance vector $\vec{v_p}$, a user u, and the associated preference vector $\vec{v_u}$, we define the* User-PoI Interest *function $\Gamma(p,u) : \mathcal{P} \times \mathcal{U} \rightarrow [0,1]$ as:*

$$\Gamma(p,u) = \alpha \cdot sim(\vec{v_p}, \vec{v_u}) + (1-\alpha) \cdot pop(p)$$

where $sim(\vec{v_p}, \vec{v_u}) = \frac{\vec{v_p} \cdot \vec{v_u}}{||\vec{v_p}|| \, ||\vec{v_u}||}$ is the cosine similarity between the user preference and the PoI relevance vectors, $pop(p)$ is a function, ranging from 0 to 1, measuring the popularity of p, and $\alpha \in [0,1]$ is a parameter controlling how much user preference and popularity of PoIs have to be taken into account.

Definition 11 (PoI History) *Given a user u and the PoIs $\mathcal{P}$, the PoI history H_u of u is the temporally ordered sequence of m points of interest visited by u. Each PoI p of H_u is annotated with the two timestamps indicating the start time and the end time of the visit:*

$$H_u =< (p_1, [t_{11}, t_{21}]), \ldots (p_m, [t_{1m}, t_{2m}]) >$$

We can notice that having the start time and the end time we have an implicit representation of the time the user u has spent for her visit of p.

Definition 12 (Trajectory) *Given a PoI History H_u and a time threshold δ, we define a trajectory T_u any subsequence of H_u*

$$< (p_k, [t_{1k}, t_{2k}]), \ldots, (p_{k+i}, [t_{1(k+i)}, t_{2(k+i)}]) >$$

such that:

$$\begin{aligned} & i \geq 1 \\ & t_{1k} - t_{2(k-1)} > \delta, && if\ k > 1 \\ & t_{1(k+i+1)} - t_{2(k+i)} > \delta, && if\ (k+i) < m \\ & t_{1(k+j)} - t_{2(k+j-1)} \leq \delta, && \forall j\ s.t.\ 1 \geq j \leq i. \end{aligned}$$

The intuition is that trajectories are sequences of PoIs visited consecutively at the same "visit". They are obtained by cutting the user PoI history where the time interval between the visit to two subsequent PoIs is greater than a given threshold δ.

Example 1 *Let John Smith be a tourist who visited the city of Rome for two-days. John Smith's PoI history consists in the temporally ordered sequence of PoIs visited in the two days. As an example:*

$H_{JohnSmith}$ = *<(Colusseum, Tue[09.00,10.30]), (Roman forum, Tue[11.00,12.00]), (Spagna square, Tue[14.30,17.30], (St. Peter's Church, Wed[10.00-11.00]), (Vatican Museum, Wed[11.10,15.00]), (Trevi Fountain, Wed[16.30,17.00]), (Navona Square, Wed[17.20,18.00]), (Via Veneto, Wed[18.35,20.00])>.*

By using a threshold of 5 *hours as trajectory splitting criterium, from* $H_{JohnSmith}$ *we obtain the following two trajectories:*

$T^1_{JohnSmith}$ = *<(Colusseum, Tue[09.00,10.30]), (Roman forum, Tue[11.00,12.00]), (Spagna square, Tue[14.30,17.30]>*

$T^2_{JohnSmith}$ = *<(St. Peter's Church, Wed[10.00-11.00]), (Vatican Museum, Wed[11.10,15.00]), (Trevi Fountain, Wed[16.30,17.00]), (Navona Square, Wed[17.20,18.00]), (Via Veneto, Wed[18.35,20.00]>*

The time interval between the visits to Spagna square and St. Peters Church is in fact the only interval larger than the given threshold. □

By applying the same temporal splitting criterium to all the PoI histories of users $\mathcal{U}$ we obtain the set $\mathcal{S} = \{S_1, \ldots, S_M\}$ of relevant trajectories. Note that $\mathcal{S}$ results from a set-union operation disregarding timestamps. Finally, let $\rho(p) : \mathcal{P} \to \mathbb{R}$ be an estimate of the time needed to visit p, $\tau(p_i, p_j) : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ an estimate of the time a user needs to move from p_i to p_j, and $\vec{z} = (z_1, \ldots, z_M)$ be the total traveling time associated with each of the M trajectories in $\mathcal{S}$, obtained by exploiting $\tau(\cdot, \cdot)$. We are now ready to formulate the TripCover problem, i.e., the problem of generating an optimal personalized itinerary given tourist's preferences and her budget in term of available time to spend in the city.

TripCover Problem

TripCover(B): Given a tourist u, a set of PoIs $\mathcal{P}$, a time budget B, a set of trajectories $\mathcal{S}$, an User-PoI Interest function Γ, a cost function $\rho(p)$ and a vector $\vec{z}$. Find a subset of trajectories $\mathcal{S}^*$ of $\mathcal{S}$ that

$$\textit{maximize} \quad \sum_{i=1}^{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{P}|} \Gamma(p_j, u)\, y_{ij} \tag{4.1}$$

$$\textit{such that} \quad \sum_{i=1}^{|\mathcal{S}|} \sum_{j=1}^{|\mathcal{P}|} \rho(p_j)\, y_{ij} + \sum_{i=1}^{|\mathcal{S}|} z_i\, x_i \leq B \tag{4.2}$$

$$\sum_{i=1}^{|\mathcal{S}|} y_{ij} \leq 1, \quad \forall j \in \{1, \ldots, |\mathcal{P}|\} \tag{4.3}$$

$$\sum_{i=1}^{|\mathcal{S}|} x_i \geq \sum_{i=1}^{|\mathcal{S}|} y_{ij}, \quad \forall j \in \{1, \ldots, |\mathcal{P}|\} \tag{4.4}$$

where

$$y_{ij} = \begin{cases} 1 & \text{if PoI } j \text{ in trajectory } i \text{ is selected;} \\ 0 & \text{otherwise.} \end{cases}$$

$$x_i = \begin{cases} 1 & \text{if trajectory } i \text{ is selected;} \\ 0 & \text{otherwise.} \end{cases}$$

Without loss of generality, we assume $\forall S' \in \mathcal{S}$, $\sum_{p \in S'} \Gamma(p, u) > 0$. In fact, if this would not hold for a given user u and some trajectories, these trajectories could be filtered out. The TripCover(B) problem as formulated in (4.1) is an instance of the *Generalized Maximum Coverage* (GMC) problem that is proven to be NP-hard [47]. The constraint (4.2) and (4.3) ensure the time budget is satisfied, and each selected PoI is associated with only one trajectory, respectively. Moreover, (4.4) guarantees a selected trajectory if a PoI is selected. In particular, given a tourist u, TripCover(B) can be captured by the GMC formulation in the following way: i) the bins in GMC represent the collection $\mathcal{S}$ of trajectories; ii) the profit function $\Gamma(p, u)$ and the cost function $\rho(p)$ are bins-independent. They only depend on p and u. The TripCover(B) problem is thus NP-hard. An efficient greedy approximation algorithm for solving the GMC problem that achieves an approximation ratio of $e/(e-1) + \epsilon$, $\forall \epsilon > 0$ is proposed in [47]. We thus adapted this algorithm, whose source code has been kindly provided us by the authors, in order to take into account TripCover(B) specific constraints.

4.3 The TrajSP Problem

Once the solution of a given TripCover instance is computed, the trajectories in $\mathcal{S}^*$ need to be scheduled on the user agenda. To this purpose, we model trajectory scheduling as a Traveling Salesman Problem (TSP) aimed at finding the shortest path crossing all these trajectories. In the classic TSP, the goal is to find the shortest path connecting a given set of geographical points. Here, the task is different as it is defined over a set $\mathcal{S}^*$ of disjoint trajectories, i.e., trajectories not sharing any PoI. We consider these trajectories as bi-directional paths representing tourists' behaviors that must be preserved in the final solution. Therefore we have to connect trajectories in a single sightseeing tour by only considering their terminal PoIs (endPoIs) – the first and the last PoI of each trajectory.

In the following, we formally define our trajectory scheduling problem (TrajSP), we propose a local-search based algorithm to efficiently address it and, finally, we describe the simple approach used to schedule the final sightseeing tour on the user agenda.

4.3.1 Trajectory Scheduling Problem

Let $\mathcal{S}^* \subseteq \mathcal{S}$ be a set of disjoint trajectories, $\mathcal{P}^*$ the set of endPoIs, and $E = (e_{ij})$ the endPoIs matrix where $e_{ij} = 1$ if i and j are endPoIs of the same trajectory of $\mathcal{S}^*$, 0 otherwise. Moreover, let C be the symmetric cost matrix where c_{ij} is the time needed to move from endPoI i to endPoI j. TrajSP is defined as follows:

Trajectory Scheduling Problem

TrajSP: Given the set of endPoIs $\mathcal{P}^*$, the endPoIs matrix E, and the cost matrix C, find the tour $\hat{\mathcal{P}}$ that:

$$\textit{minimize} \quad \sum_{i=1}^{|\mathcal{P}^*|} \sum_{j=i+1}^{|\mathcal{P}^*|} c_{ij}\,\eta_{ij} \tag{4.5}$$

$$\textit{such that} \quad \sum_{i=1}^{k-1} \eta_{ik} + \sum_{j=k+1}^{|P^*|} \eta_{kj} = 1, \quad \forall k \in \{1,\ldots,|P^*|\} \tag{4.6}$$

$$\sum_{i,j \in S} \eta_{ij} + e_{ij} \leq |S| - 1, \quad (S \subset P^*, |S| > 2), \quad i < j \tag{4.7}$$

$$\eta_{ij} \leq 1 - e_{ij}, \quad \forall i,j \quad i \neq j \tag{4.8}$$

where

$$\eta_{ij} = \begin{cases} 1 & \text{if endPoI } i \text{ is connected to endPoI } j, \\ 0 & \text{otherwise.} \end{cases}$$

In the above formulation, only the costs between different trajectories' endPoIs have to be considered for minimizing cost (4.5), while (4.6) and (4.7) impose constraints on the *degree* (numeber of connections) of each endPoI and on *sub-tour elimination*, respectively [102]. Note that the degree constraint in (4.6) is set to 1 since each endPoI already has one fixed connection to the next PoI of the associated trajectory. The last constraint in (4.8) ensures that two endPoIs of the same trajectory are never connected together in the solution.

Although the number of possible solutions for TrajSP is lower than whose of the corresponding TSP formulation[4], finding the exact solution to TrajSP is still infeasible even for instances involving a small number of trajectories (we have about 82 billions of possible solutions with only 12 trajectories). Hence, we address TrajSP by proposing a *Local Search* heuristics that starts from a (given or random) tour $\hat{\mathcal{P}}$ connecting all trajectories in $\mathcal{S}^*$, and then applying local changes to $\hat{\mathcal{P}}$ by means of 2-OPT or 3-OPT strategies [102].

We now introduce an interesting property of the TrajSP problem that allows us to formalize how 2-OPT or 3-OPT strategies have to be applied. Given endPoI i, let $e(i) = j$ be the other endPoI of the same trajectory. Obviously $e(j) = i$ holds as well. Since we start from a tour connecting all the trajectories, each endPoI i is connected to both $e(i)$ and to an endPoI d of another trajectory in $\mathcal{S}^*$. Let denote with $n(i)$ the connected endPoI d. Note that only the link between endPoIs i and $n(i)$ can be modified, since the path between i and $e(i)$ is fixed by definition.

We now introduce a basic local-change operation $op(i,k)$ over two endPoIs i,k, such that $k \neq e(i)$. The local-change operation works by adding link (i,k) and removing links $(n(i),i)$ and $(n(k),k)$. It is clear that the application of $op(\cdot,\cdot)$ leads to a non-admissible solution for TrajSP, since we remove two links of the tour while we add only one. We thus need to perform some additional changes to reconnect the two endPoIs that remain disconnected. The result below formalizes how such changes aimed at restoring the feasibility of the current solution have to be done. In particular, given two endPoIs i, k, it states that we can apply 2-OPT or 3-OPT strategies chosen according to the value of $l(i,k)$, a function returning the minimum number of endPoIs connections between i and k.

Lemma 1 *Given a tour $\hat{\mathcal{P}}$ and a local-change operation $op(i,k)$ performed over two endPoIs i,k such that $l(i,k) > 1$, if $l(i,k)$ is even, then a 2-OPT strategy needs to be applied to produce a feasible tour, a 3-OPT strategy otherwise.*

Proof 1 *Given two endPoIs i,k, $l(i,k)$ can be either even or odd. First, suppose $l(i,k)$ is even. It means that the minimum number of connections between the endPoIs i and k is even. As a consequence, starting from the link $(n(i),i)$ (red dashed line) we arrive at $(e(k),k)$ (black dashed line), or from $(i,e(i))$ (black dashed line) we arrive at $(k,n(k))$ (red dashed line), as shown in Figure 4.1 (a) below.*

Operation $op(i,k)$ removes $(n(i),i)$ and $(n(k),k)$. Moreover, it adds (i,k) (see Figure 4.1 (b)). Note that, at this point the tour is disconnected. To have a feasible tour, we need to add the new link $(n(i),n(k))$ that joins the two disconnected endPoIs. Because this process adds two links $(i,k),(n(i),n(k))$ and removes two other links $(n(i),i),(n(k),k)$, it corresponds to applying a 2-OPT strategy.

We now study the case $l(i,k)$ is odd. Starting from $(i,n(i))$, leads us to $(k,n(k))$ (red dashed lines). Moreover, from $(i,e(i))$ we arrive at $(k,e(k))$ (black dashed lines). This configuration is

[4]We have exactly $(k-1)! * 2^{(k-1)}$ solutions for k trajectories in the case of TrajSP, and $(2k-1)!/2$ for the corresponding TSP formulation considering the two endPoIs of each trajectory.

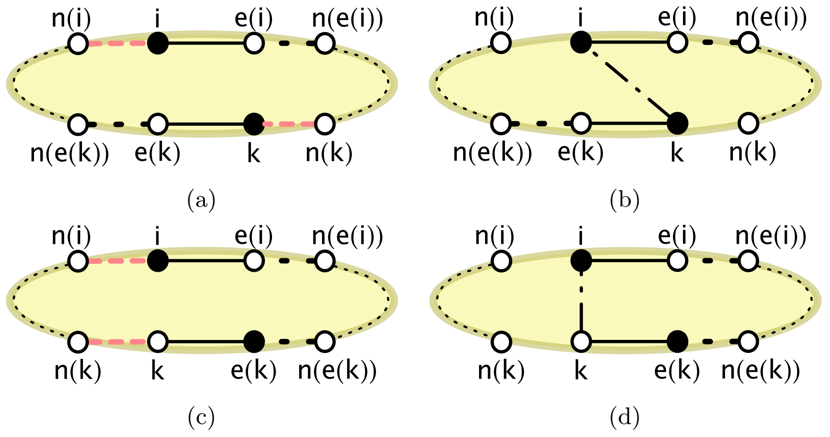


Figure 4.1: Examples of how $op(i,k)$ modifies the tour according to $l(i,k)$, i.e., the minimum number of endPoIs connections between i and k.

shown in Figure 4.1 (c). Applying $op(i,k)$ *to this particular case generates a different configuration (see Figure 4.1 (d)), i.e., one sub-tour (i.e., a closed path) from* i *to* k*, and one path from* $n(i)$ *and* $n(k)$*. To merge the sub-tour and the path, we need to remove one link from the sub-tour and connect it to the path, e.g.,* $n(i)$ *and* $n(k)$*. So, we can remove* $(e(i), n(e(i)))$*. We then need to add* $(e(i), n(i))$ *and* $(n(e(i)), n(k))$ *to obtain a tour crossing all the endPoIs. Because we add three new links and remove three other ones, we applied a 3-OPT strategy.* □

Local search algorithm. From the above discussion, it is easy to devise a local search algorithm that iteratively optimizes a given tour by applying 2-OPT or 3-OPT strategies and stops after a fixed number of iterations or when it converges to a locally optimal solution.

Scheduling the tour on the user agenda

Given the sightseeing tour $\hat{\mathcal{P}}$ computed for a given user and a time budget by our local search TrajSP algorithm, we schedule it on the user agenda by splitting the tour into the desired number m of slots (e.g., days). The solution involves identifying a starting endPoI in the tour and assigning the successive PoIs in $\hat{\mathcal{P}}$ to the current slot until the slot is filled and the next slot is considered. The choice of the starting endPoI can be done in two different ways: either by removing the most "expensive" connection (e.g., in terms of traveling time) between two endPoIs of the tour or by taking into account the closest endPoI to a given spatial position (e.g. user's hotel).

4.4 Building the Knowledge Base

Figure 4.2 depicts an overview of the TripBuilder architecture. The component related to *Data Collection* retrieves relevant data from Flickr, Wikipedia, and Google Maps. The second component called *Data Processing* extracts the knowledge used to devise relevant PoIs and model users' visiting behaviors from data provided by the *Data Collection* component. Given a budget B, the third component *Covering* deals with the exploitation of the models and the knowledge base to compute the solution to the TripCover(B) problem. The result is a set of trajectories in the chosen city on the basis of user interests and time budget (see Figure 4.3) that are finally scheduled on the user agenda by the fourth component *Scheduling*. Figure 4.3 illustrates an example showing geo-tagged photos matched with Wikipedia PoIs in order to generate a set of trajectories representing the past visits of tourists.

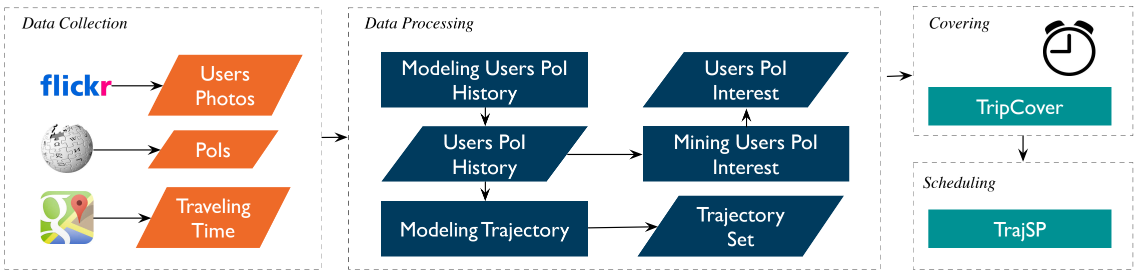


Figure 4.2: Overview of the unsupervised process used to build the TripBuilder knowledge base.

In order to assess TripBuilder we generate - in a complete unsupervised process - a knowledge base covering three Italian cities which are important from a sightseeing point of view and guarantee variety and diversity in terms of size and richness of public user-generated content available for download: Rome, Florence, and Pisa. The generation of the knowledge base for each of the cities is a multi-steps process that we are going to detail in the following.

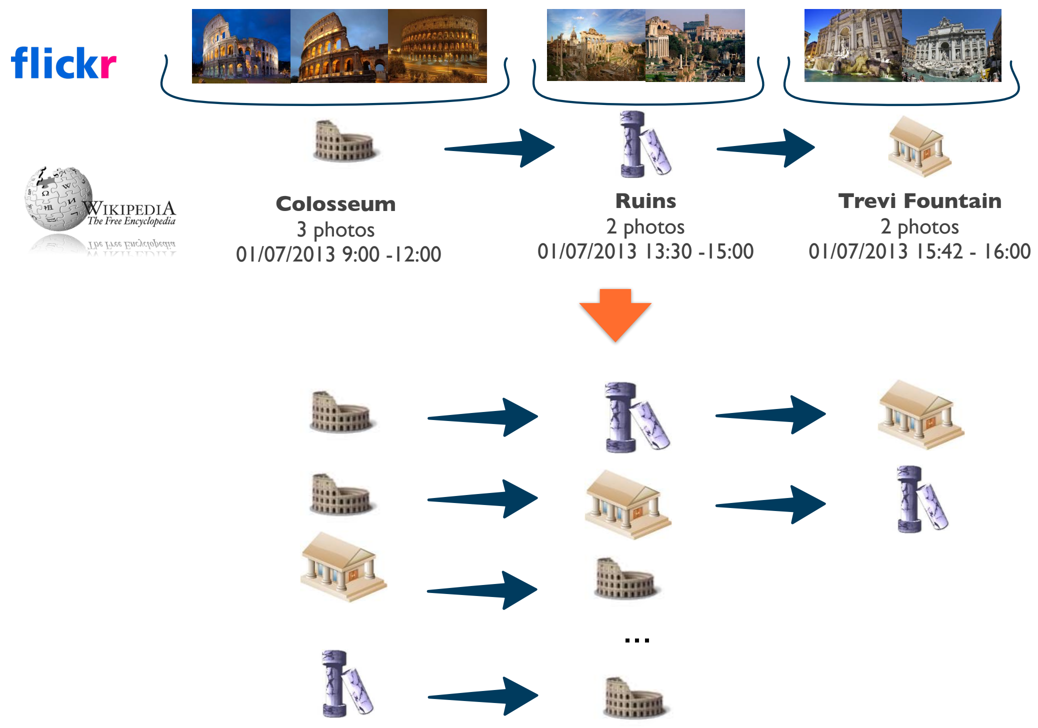


Figure 4.3: Data processing of TripBuilder: from Wikipedia PoIs and Flickr photos towards a knowledge base of tourist trajectories.

4.4.1 Points of interest discovery

The first step is to identify the set of PoIs in the target geographical region. Given the bounding box BB_{city} containing the city of interest, we download all the geo-referenced Wikipedia pages falling within this region. We assume each geo-referenced Wikipedia named entity, whose geographical coordinates falls into BB_{city}, to be a fine-grained Point of Interest. For each PoI, we retrieve its descriptive label, its geographic coordinates as reported in the Wikipedia page, and the set of categories the PoI belongs to. Categories are reported at the bottom of the Wikipedia page, and are used to link articles under a common topic. They form a hierarchy, although sub-categories may be a member of more than one category. By considering the set C of categories associated with all the PoIs, we generate the normalized relevance vector of each PoI. We then perform a density-based clustering to group in a single PoI sightseeing entities which are very close one to each other[5]. Clustering very close PoIs is important since a tourist in a given place can enjoy all the attractions in the surroundings even if she does not take photos to all of them. Moreover, it aims at reducing the sparsity that might affect trajectory data. To cluster the PoIs we use DBScan [60]. To build our dataset, we set 1 as the minimum number of points and 200 meters as ϵ. Finally, we obtain the relevance vector for the clustered PoIs by considering the occurrences of each category in the members of the clusters and by normalizing the resulting vector. At the end of this first step we have the set $\mathcal{P} = \{p_1, \ldots, p_N\}$ of PoIs and the relevance vector $\vec{v_p} \in [0,1]^{|C|}$ for each of these PoIs in a fully automatic way by exploiting Wikipedia as an external source of knowledge.

4.4.2 Users and PoI histories

As second step we need a method for collecting users $\mathcal{U}$ and the long-term itineraries crossing the discovered PoIs. We query Flickr to retrieve the metadata (user id, timestamp, tags, geographic coordinates, etc.) of the photos taken in the given area BB_{city}. The assumption we are making is that photo albums made by Flickr users implicitly represent sightseeing itineraries within the city. To strengthen the accuracy of our method, we retrieve only the photos having the highest geo-referenced accuracy given by Flickr[6]. This process thus collects a large set of geo-tagged photo albums taken by different users within BB_{city}. We preliminary discard photo albums containing only one photo. Then, we spatially match the remaining photos against the set of PoIs previously collected. We associate a photo to a PoI when *it has been taken within a circular buffer of a given radius having the PoI as its center.* To build our dataset, we empirically set it to 100 meters radius. Note that in order to deal with clustered PoIs, we consider the distance of the photo from all constituent members: in case the photo falls within the circular region of at least one of the members, it is assigned to the clustered PoI. Moreover, since several photos by the same user are usually taken close to the same PoI, we consider the timestamps associated with the first and the last of these photos as the starting and ending time of the user visit to the PoI. The PoI visiting time $\rho(p)$ is then estimated by computing for each PoI the average of these times. Moreover, the popularity of each PoI is computed as the number of distinct users that take at least one photo in

[5]Consider for example the beautiful marble statues in the *Loggia dei Lanzi* in Florence which are only a few meters far one from each other but have a distinct dedicated page in Wikipedia.

[6]`http://www.flickr.com/services/api/flickr.photos.search.html`

its circular region. The above process allows us to generate the set of users $\mathcal{U}$, their PoI history, and estimate for the popularity and visiting time of each PoI. Finally, the preference vector for each user is built by summing up and normalizing the relevance vectors of all the PoIs occurring in her PoI history.

4.4.3 Trajectories creation

In order to build the set $\mathcal{S}$ of trajectories we split users' PoI histories as detailed in Definition 11 (Poi History definition). To choose the splitting threshold δ, we derive the users' wisdom-of-crowds behavior by analyzing the inter-arrival time of each pair of consecutive photos taken in different PoIs. Therefore, for each city we compute the distribution of probability of the inter-arrival time $P(x \leq \delta)$ of pairs of consecutive photos. Then, we devise the time threshold δ such that $P(x \leq \delta) = 0.9$. Figure 4.4 shows the distributions of probability of inter-arrival times, i.e., $P(x \leq \delta)$, for all the pairs of consecutive photos in each dataset. Results show that while for Rome and Florence the resulting δ is about 5 and 6 hours respectively, for the smallest city of Pisa it decreases to about 3 hours.

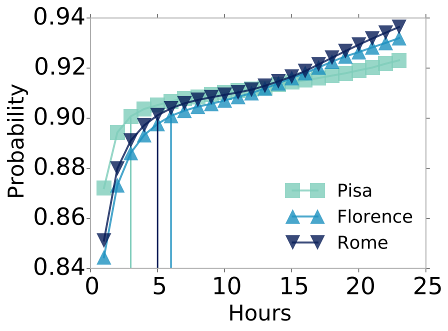


Figure 4.4: Plot shows the probability distribution of the inter-arrival time for pairs of consecutive photos taken in Rome, Florence, and Pisa. The vertical lines highlight the time thresholds corresponding to $P(x \leq \delta) = 0.9$.

4.4.4 Traveling time estimation

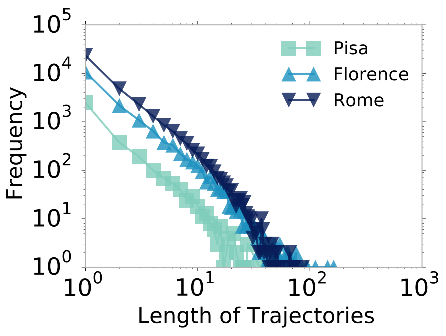
An important aspect of TripBuilder is that we recommend sightseeing tours fitting the available time budget and not just the set of PoIs to be visited. The sightseeing tour building step should therefore consider not only the PoI visiting time $\rho(p)$ but also the time $\tau(\cdot, \cdot)$ needed to move between consecutive PoIs in the itinerary. Since measuring intra-PoI moving time from the photo albums resulted to be inaccurate for not popular PoIs, we resort to an external service. Given a pair (p_i, p_j) of PoIs in a trajectory, we estimate $\tau(p_i, p_j)$ by querying Google Maps for the walking time between the PoIs. Naturally, this is an approximation since several variations may happen: the user having a car, using public transportation, taking a taxi. However, our method is parametric to these aspects, and the system can be easily adapted to consider the different choices. Moreover, most PoIs in our sightseeing cities are actually at walking distances.

4.5 Datasets statistics

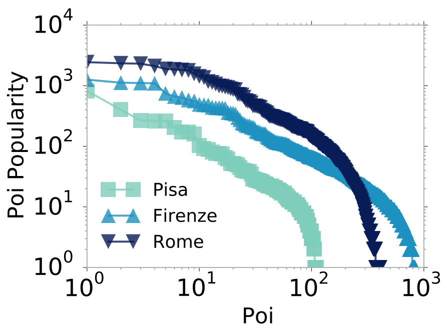
Table 4.1 shows the main characteristics of the three datasets. The second column reports the number of PoIs for each of the three cities. Note that these numbers refer to the result of the clustering phase, while the number of entities extracted from Wikipedia is 124, 1, 022, and 671 for Pisa, Florence and Rome, respectively. Furthermore, columns "Users" and "Photos" report the number of distinct users and photos retrieved from Flickr. Finally, column "Trajectories" reports the number of trajectories crossing at least two PoIs, while column "Traj. per PoI (avg.)" reports the average number of trajectories crossing each PoI.

City	**PoIs**	**Users**	**Photos**	**Trajectories**	**Traj. per PoI (avg.)**
Pisa	112	1, 825	18, 170	3, 430	7.20
Florence	891	7, 049	102, 888	16, 522	5.39
Rome	490	13, 772	234, 616	35, 522	20.51

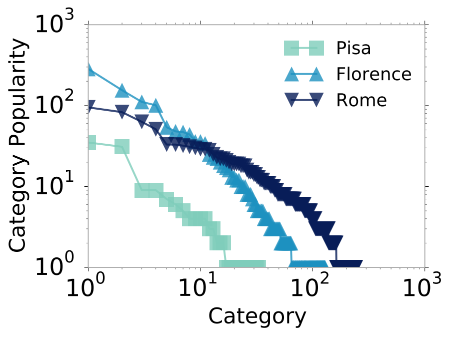
Table 4.1: Statistics regarding the three cities in our dataset.



(a)



(b)



(c)

Figure 4.5: Plots (a), (b), (c) show instead the distribution of trajectories length (as number of PoIs crossed), the popularity of PoIs and the popularity of categories in the three datasets.

Figures 4.5a and 4.5b, and 4.5c show three plots regarding the characteristics of the three datasets that have been made available for download to favor the reproducibility of experiments[7]. A general consideration common to all the three figures regards the skewness of the distributions that are plotted in log-log scale. In particular, Figure 4.5a reports the distribution of trajectory length expressed in term of the number of crossed PoIs. We can see that the slope of the three curves is very similar, only absolute values vary as expected. Note that the most frequent trajectory length in the plot is just 1 (see Figure 4.5a). These are obviously noisy trajectories, corresponding to the cases in which we can match only one single PoI to a photo album. These spurious trajectories are maintained in the datasets only for their contribution to the measure of PoI popularity, but do not belong to the set $\mathcal{S}$ of trajectories used by TripBuilder.

Figure 4.5b shows the distribution of PoI popularity in the three cities involved. Even in this case we can note strong similarities in the distributions, although the curves of Rome and Florence cross in the tail. This happens for a peculiarity in these two datasets: while we have more users and photos for Rome than for Florence, the opposite holds for the number of PoIs reported in the X axis.

The last plot reported in Figure 4.5c shows the popularity of categories associated with the PoIs. Even in this case we have a power-law distribution with a few very popular categories and most categories associated with relatively few PoIs. Wikipedia categories form in fact a (weak) hierarchy, and the most general topics are associated with many pages, while sub-categories are relevant only for precisely identified subsets of homogeneous PoIs.

Finally, Table 4.2 reports the top-3 most popular PoIs and categories in Pisa, Florence and Rome. We also report three examples per city of trajectories extracted from the dataset. Note that a popular PoI in Florence and Pisa is the baptistery which is entitled, in both cities, to Saint John the Baptist.

4.6 Experiments

We now assess the effectiveness of TripBuilder in: i) selecting a set of trajectories of interest for a given user (TripCover), and ii) scheduling that set on the user agenda (TrajSP). This is done by comparing its performance with those obtained by competitive baseline by means of evaluation metrics that consider the actual behavior of test users as mined from Flickr. Moreover, we present an evaluation of the efficiency of the TripBuilder framework together with a detailed evaluation of both TripCover and TrajSP solutions[8].

We conduct our experiments on the three datasets of Pisa, Florence, and Rome by varying the time budget and the parameter α affecting the contribution of PoIs/user-similarity and PoI-popularity to user profit. Moreover, we perform two different set of experiments that differ for the methodology used to choose the test users:

- *Random selection.* Here the set of users used to assess TripBuilder performance is randomly chosen. In particular, we consider for all the three cases a set of 100 test users randomly selected among the visitors having a Poi history longer than 10, 15, and 20 PoIs for Pisa,

[7]Interested readers can download the datasets from the URL: `https://github.com/igobrilhante/TripBuilder`

[8]All the experiments have been conducted on an Ubuntu Linux box with two Intel® Xeon® E5520 CPUs and 32 GB of RAM.

	Pisa	**Florence**	**Rome**
PoIs	Leaning Tower	Loggia dei Lanzi	Colosseum
	Miracoli Square	Ponte vecchio	Fontana di Trevi
	Battistero di San Giovanni	Battistero di San Giovanni	Pantheon
Categories	Churches	Palaces	Churches
	Palaces	Churches	Campitelli Quarter
	Museums	Architecture	Titular Churches
Trajectories	Battistero di San Giovanni, Miracoli Square, Leaning Tower.	Battistero di San Giovanni, Porta della Mandorla, Campanile di Giotto.	Cappella Sistina, Basilica di San Pietro in Vaticano, Obelisco Vaticano.
	Miracoli Square, Museo dell'Opera del Duomo, Leaning Tower.	Palazzo Vecchio, Galleria degli Uffizi, Piazza della Signoria.	Colosseum, Arco di Costantino, Foro Romano.
	Ebraic Cemetery, Battistero di San Giovanni, Leaning Tower	Ponte Vecchio, Forte Belvedere, Piazza della Signoria.	Colosseum, Foro Romano, Pantheon.

Table 4.2: Top-3 most popular PoIs and Categories in Pisa, Florence, and Rome. We also report three examples of trajectories per city extracted from the dataset.

Florence and Rome, respectively. The threshold on the length of the PoI history is set in order to be able to vary in a significant range the time budgets. This is because it is not feasible to evaluate a personalized 4-days itinerary in Rome with test users that actually visited only a few popular PoIs during a single day of visit. By using the above cutoff values, the users among which the 100 test users were chosen are 153, 679, and 930 in Pisa, Florence, and Rome, respectively.

- *Profile-based selection.* Here we select the test users among users who actually visited at least two of the three cities. In particular, given a user visiting two cities A and B, we used the preference vector obtained from the PoIs visited in city A to generate the personalized sightseeing tour in city B and vice-versa. In this way we avoid any possible bias to the specific categories used in the Wikipedia pages of a given city.

4.6.1 Effectiveness – TripCover

We compare the effectiveness of TripBuilder in selecting a budgeted set of trajectories of interest for a given user against the following baselines:

- **Trajectory Popularity (Tpop).** This baseline builds the tour by taking into account the normalized popularity of the trajectories in $\mathcal{S}$ computed as the sum of the popularity of the constituent PoIs divided by the length of the trajectory. It greedily adds to the visiting plan the most popular trajectories until the time budget is reached.

- **Trajectory Personalized Profit (Tppro)**. Given the preference vector of a tourist, this baseline sorts the trajectories in $\mathcal{S}$ by decreasing normalized user/PoI similarity. Such trajectory score is computed as the sum of user/PoI similarities of all the PoIs in the trajectory divided by the trajectory length. The baseline builds the personalized itinerary by adding once at a time the trajectories having the highest profit for the specific tourist until the total time budget is reached.

Experiments are conducted by providing to TripBuilder and the baseline algorithms the preference vector of each one of the test users in each city, along with a time budget varying in the range 1, 2, and 4 days (1/2, 1 day in the case of the small city of Pisa)[9]. We evaluate the performance of the three methods by means of the metrics defined below. Moreover, we also employ *recall* [10], a well-known IR metrics that in our settings measure the ability of the methods to predict PoIs and categories that match actual PoI histories of the users in the test set.

Personal Profit Score. Given a user u and a set of trajectories $\mathcal{S}^*$, S_u^{pro} is computed as the sum of the profits of the PoIs in $\mathcal{S}^*$ divided by the sum of the profits of all the PoIs. The user profit for a PoI (i.e., $sim(\vec{v_p}, \vec{v_u})$) is the cosine similarity between user preferences and PoI relevance vectors (see Definition 10):

$$S_u^{pro}(\mathcal{S}^*) = \frac{\sum\limits_{p \in \mathcal{S}^*} sim(\vec{v_p}, \vec{v_u})}{\sum\limits_{p \in \mathcal{P}} sim(\vec{v_p}, \vec{v_u})}.$$

Visiting Time Score. Given a set of trajectories $\mathcal{S}^*$, this score is computed as the sum of the visiting times for the PoIs in $\mathcal{S}^*$ normalized by the time budget B. Given a time budget, it assumes that high scored tours result to be interesting since they favor the time to enjoy attractions with respect to the intra-PoIs moving time:

$$S^{vt}(\mathcal{S}^*) = \sum_{p \in \mathcal{S}^*} \rho(p) \ / \ B.$$

Popularity Score. Given a set of trajectories $\mathcal{S}^*$, this score is computed by summing the popularity of the PoIs in $\mathcal{S}^*$. Note that the popularity $pop(p)$ of a given PoI p is normalized over the sum of the popularity of all the PoIs. As a consequence, $\sum\limits_{p \in \mathcal{P}} pop(p) = 1$:

$$S^{pop}(\mathcal{S}^*) = \sum_{p \in \mathcal{S}^*} pop(p).$$

Random Selection

Tables 4.3, 4.4, and 4.5 report the effectiveness measured for the recommended sightseeing tours on the basis of the metrics defined above in Pisa, Florence, and Rome, respectively. In particular, each table details the average per-user performance and its standard deviation for each previously-

[9]We assume the normal daily activity of a tourist in a city to be of twelve hours. Our solution is, however, completely agnostic w.r.t. the daily agenda and works with tourist-provided agenda defining different time slots as well.

defined metrics, and highlights in bold fonts the best per-budget figures. The first observation from the results is that, despite of their simplicity, both popularity (Tpop) and profit-based (Tppro) greedy strategies perform well, thus forming competitive baselines. In terms of Personal Profit Score (S_u^{pro}), our solution improves the baselines up to 86% in Pisa with an absolute improvement ΔS_u^{pro} of 0.30 (Tpop vs. TripBuilder, $\alpha = 0.75$), 178% in Florence with an improvement ΔS_u^{pro} of 0.203 (Tpop vs. TripBuilder, $\alpha = 1$), and 213% in Rome with an improvement ΔS_u^{pro} of 0.382 (Tpop vs. TripBuilder, $\alpha = 0.75$). In terms of S_u^{pro}, TripBuilder outperforms the baselines showing a behavior which is sensitive to the parameter α.

Results in Pisa (see Table 4.3) compared to Tppro demonstrate that our approach is better for 1/2-day budget with 6% of improvement, while Tppro is better for 1-day budget, but it improves only 1%. It is worth highlighting two situations: (1) when $\alpha = 0$, TripBuilder works by considering only the popularity (as Tpop) showing a performance in terms of S_u^{pro} that is similar to the Tpop baseline; (2) when $\alpha = 1$, TripBuilder considers only users' interest, and its performance becomes comparable with Tppro. We conclude that α plays an important role in TripBuilder to balance the contribution of users' profit/interest and PoI's popularity. For instance, the highest S_u^{pro} score in Pisa for the time budget of 1 day can be found when $\alpha = 0.75$ (i.e., 0.79). Moreover, TripBuilder builds tours that increase the Visiting Time Score (S^{vt}) up to 36 minutes in Pisa ($\Delta S^{vt} = 0.05$), about 279 minutes in Florence, and approximately 9.2 hours in Rome. Therefore, our algorithm is able to suggest itineraries that better match users' preferences w.r.t the baselines. In addition, TripBuilder works by favoring higher visiting time while contributing to lower intra-PoI movement time in its solutions.

	Days	Recall (PoIs)	Recall (Cat.)	S_u^{pro}	S^{vt}	S^{pop}
Tpop	1/2	0.792 (±0.17)	0.951 (±0.08)	0.348 (±0.05)	0.639 (–)	0.832 (–)
	1	0.894 (±0.10)	0.989 (±0.04)	0.655 (±0.06)	0.643 (–)	**0.914** (–)
Tppro	1/2	0.697 (±0.19)	0.819 (±0.18)	0.609 (±0.11)	0.648 (±0.04)	0.468 (±0.14)
	1	0.909 (±0.10)	0.966 (±0.07)	**0.804** (±0.06)	0.643 (±0.02)	0.780 (±0.10)
TripBuilder ($\alpha = 0.0$)	1/2	0.796 (±0.16)	**0.983** (±0.04)	0.400 (±0.05)	0.731 (–)	**0.836** (–)
	1	0.861 (±0.11)	0.989 (±0.04)	0.584 (±0.05)	0.640 (–)	0.887 (–)
TripBuilder ($\alpha = 0.25$)	1/2	**0.824** (±0.14)	0.946 (±0.09)	0.619 (±0.08)	**0.744** (±0.03)	0.717 (±0.09)
	1	0.914 (±0.09)	0.990 (±0.03)	0.751 (±0.05)	0.692 (±0.02)	0.899 (±0.02)
TripBuilder ($\alpha = 0.5$)	1/2	0.795 (±0.16)	0.917 (±0.12)	0.641 (±0.09)	0.736 (±0.03)	0.636 (±0.12)
	1	**0.927** (±0.08)	**0.993** (±0.02)	0.779 (±0.06)	0.692 (±0.02)	0.888 (±0.04)
TripBuilder ($\alpha = 0.75$)	1/2	0.772 (±0.15)	0.903 (±0.12)	**0.646** (±0.09)	0.731 (±0.03)	0.598 (±0.14)
	1	0.926 (±0.08)	0.991 (±0.03)	0.790 (±0.06)	0.689 (±0.02)	0.879 (±0.05)
TripBuilder ($\alpha = 1.0$)	1/2	0.769 (±0.16)	0.895 (±0.13)	0.644 (±0.09)	0.731 (±0.03)	0.579 (±0.14)
	1	0.918 (±0.09)	0.991 (±0.03)	0.788 (±0.06)	**0.693** (±0.02)	0.861 (±0.05)

Table 4.3: **Random Selection**: average effectiveness of TripBuilder and the baselines in Pisa.

In terms of Visiting Time Score (S^{vt}), the higher it is, the more relevant the itinerary can be considered for the user, since it is likely that she prefer to spend time visiting the PoIs than moving between them. As TripCover takes this factor into account (as a cost), it tends to exploit trajectories containing PoIs closer to each other to maximize the user profit. Consequently, we claim that TripBuilder is able to build tours that globally maximize S^{vt}. We can see from the

	Days	Recall (PoIs)	Recall (Cat.)	S_u^{pro}	S^{vt}	S^{pop}
Tpop	1	0.565 (±0.17)	0.905 (±0.08)	0.114 (±0.04)	0.601 (–)	0.591 (–)
	2	0.708 (±0.17)	0.952 (±0.06)	0.214 (±0.04)	0.607 (–)	0.732 (–)
	4	0.822 (±0.13)	0.982 (±0.04)	0.359 (±0.04)	0.609 (–)	0.836 (–)
Tppro	1	0.359 (±0.15)	0.720 (±0.17)	0.203 (±0.06)	0.648 (±0.05)	0.249 (±0.07)
	2	0.515 (±0.15)	0.864 (±0.10)	0.338 (±0.07)	0.637 (±0.04)	0.382 (±0.07)
	4	0.719 (±0.13)	0.961 (±0.05)	0.529 (±0.06)	0.631 (±0.03)	0.602 (±0.05)
TripBuilder ($\alpha = 0.0$)	1	**0.672** (±0.17)	**0.957** (±0.06)	0.196 (±0.04)	**0.792** (–)	**0.689** (–)
	2	**0.785** (±0.16)	**0.972** (±0.05)	0.325 (±0.04)	**0.722** (–)	**0.803** (–)
	4	0.860 (±0.10)	**0.986** (±0.03)	0.453 (±0.03)	**0.706** (–)	**0.869** (–)
TripBuilder ($\alpha = 0.25$)	1	0.572 (±0.15)	0.912 (±0.08)	0.316 (±0.05)	0.723 (±0.03)	0.457 (±0.06)
	2	0.749 (±0.13)	0.964 (±0.05)	0.457 (±0.05)	0.691 (±0.02)	0.663 (±0.05)
	4	**0.874** (±0.09)	**0.986** (±0.03)	0.594 (±0.04)	0.694 (±0.02)	0.826 (±0.02)
TripBuilder ($\alpha = 0.5$)	1	0.554 (±0.15)	0.903 (±0.09)	**0.317** (±0.05)	0.719 (±0.03)	0.435 (±0.06)
	2	0.742 (±0.13)	0.962 (±0.05)	**0.458** (±0.05)	0.689 (±0.02)	0.646 (±0.05)
	4	0.869 (±0.10)	**0.986** (±0.03)	0.595 (±0.04)	0.692 (±0.02)	0.822 (±0.03)
TripBuilder ($\alpha = 0.75$)	1	0.548 (±0.15)	0.897 (±0.09)	**0.317** (±0.05)	0.717 (±0.03)	0.427 (±0.06)
	2	0.741 (±0.13)	0.961 (±0.05)	**0.458** (±0.05)	0.688 (±0.02)	0.641 (±0.05)
	4	0.868 (±0.10)	**0.986** (±0.03)	**0.596** (±0.04)	0.692 (±0.02)	0.820 (±0.03)
TripBuilder ($\alpha = 1.0$)	1	0.546 (±0.15)	0.897 (±0.09)	**0.317** (±0.05)	0.716 (±0.03)	0.424 (±0.06)
	2	0.736 (±0.13)	0.962 (±0.05)	**0.458** (±0.05)	0.686 (±0.02)	0.638 (±0.05)
	4	0.866 (±0.10)	**0.986** (±0.03)	**0.596** (±0.04)	0.689 (±0.02)	0.817 (±0.03)

Table 4.4: **Random Selection**: average effectiveness of TripBuilder and the baselines in Florence.

results in the tables that TripBuilder uses more appropriately the time budget. The difference in terms of S^{vt} increases when larger budgets are considered. Moreover, this phenomenon is even more highlighted when dealing with larger cities. Indeed, in the case of Pisa, the three algorithms (i.e., Tpop, Tppro, TripBuilder) have quite similar S^{vt}, with slight gains for TripBuilder (from 30 to 34 minutes, with $\alpha = 0.75$). In the case of larger cities, i.e., Florence and Rome, TripBuilder remarkably outperforms the baselines. The rationale behind this result can be that in larger cities the intra-PoI traveling time tends to impact more the S^{vt} metrics.

In terms of Popularity Score (S^{pop}), results achieved with $\alpha = 0$ confirm that TripBuilder outperforms Tpop in the most cases. Moreover, the values of S^{pop} for TripBuilder decrease when increasing values of α as expected.

In terms of PoIs and categories recall, all algorithms get at least 69% of the relevant PoIs and 81% of the categories for Pisa. Regarding categories recall in Pisa (see Table 4.3), TripBuilder and Tpop present similar results, while both of them outperform Tppro. Moreover, looking at PoIs recall, TripBuilder gets better results than the baselines (with $\alpha = 0.5$): 92.7% compared with 89.4% of Tpop and 90.9% of Tppro for the 1-day time budget. Comparing the results for Florence and Rome, we observe that TripBuilder outperforms in both recall figures the baselines depending on α setting. Moreover, when larger budgets are employed (2 and 4-days budget), it always outperforms the baselines independently from the value of α (see Table 4.4 and Table 4.5). This behavior is due to the capability of TripBuilder of building tours with a higher Visiting

	Days	Recall (PoIs)	Recall (Cat.)	S_u^{pro}	S^{vt}	S^{pop}
Tpop	1	0.578 (±0.15)	0.881 (±0.07)	0.179 (±0.03)	0.471 (–)	0.642 (–)
	2	0.795 (±0.13)	0.949 (±0.05)	0.403 (±0.03)	0.463 (–)	0.836 (–)
	4	0.934 (±0.07)	0.991 (±0.02)	0.675 (±0.03)	0.485 (–)	0.953 (–)
Tppro	1	0.428 (±0.12)	0.708 (±0.14)	0.381 (±0.06)	0.456 (±0.04)	0.281 (±0.07)
	2	0.642 (±0.13)	0.872 (±0.09)	0.596 (±0.06)	0.443 (±0.03)	0.505 (±0.08)
	4	0.894 (±0.09)	0.979 (±0.03)	0.824 (±0.04)	0.447 (±0.02)	0.813 (±0.04)
TripBuilder ($\alpha = 0.0$)	1	**0.749** (±0.13)	**0.937** (±0.05)	0.369 (±0.03)	**0.786** (–)	**0.800** (–)
	2	**0.906** (±0.09)	**0.980** (±0.03)	0.603 (±0.04)	**0.754** (–)	**0.927** (–)
	4	0.950 (±0.05)	0.991 (±0.01)	0.742 (±0.03)	0.635 (–)	**0.961** (–)
TripBuilder ($\alpha = 0.25$)	1	0.680 (±0.12)	0.878 (±0.07)	0.558 (±0.04)	0.735 (±0.02)	0.563 (±0.05)
	2	0.889 (±0.08)	0.978 (±0.02)	0.757 (±0.04)	0.723 (±0.01)	0.837 (±0.03)
	4	0.966 (±0.04)	**0.995** (±0.01)	0.860 (±0.03)	**0.640** (±0.02)	0.955 (±0.01)
TripBuilder ($\alpha = 0.5$)	1	0.663 (±0.13)	0.854 (±0.09)	**0.561** (±0.04)	0.732 (±0.02)	0.528 (±0.06)
	2	0.885 (±0.08)	0.974 (±0.03)	0.760 (±0.04)	0.720 (±0.01)	0.809 (±0.04)
	4	**0.968** (±0.04)	**0.995** (±0.01)	0.865 (±0.03)	**0.640** (±0.02)	0.954 (±0.01)
TripBuilder ($\alpha = 0.75$)	1	0.654 (±0.13)	0.841 (±0.09)	**0.561** (±0.04)	0.731 (±0.02)	0.513 (±0.06)
	2	0.880 (±0.08)	0.973 (±0.03)	0.761 (±0.04)	0.718 (±0.02)	0.798 (±0.05)
	4	0.967 (±0.04)	**0.995** (±0.01)	**0.867** (±0.03)	0.639 (±0.02)	0.954 (±0.01)
TripBuilder ($\alpha = 1.0$)	1	0.651 (±0.13)	0.839 (±0.09)	**0.561** (±0.04)	0.731 (±0.02)	0.505 (±0.06)
	2	0.880 (±0.08)	0.970 (±0.03)	**0.762** (±0.04)	0.718 (±0.02)	0.792 (±0.05)
	4	0.967 (±0.04)	**0.995** (±0.01)	**0.867** (±0.03)	0.638 (±0.02)	0.953 (±0.01)

Table 4.5: **Random Selection**: average effectiveness of TripBuilder and the baselines in Rome.

Time Score within the time budgets. Consequently, more PoIs relevant for the specific user are likely to be visited. Moreover, we can see that the α parameter allows to fit the expectations of the user. For small values of α, results report higher recall values because trajectories crossing popular PoIs are preferred. When α increases, recall decreases because unexpected PoIs in the selected trajectories fitting the user interests are suggested. We believe these trajectories may constitute a source of serendipitous recommendations. Finally, the low standard deviations associated with the results in Tables 4.3, 4.4, and 4.5 prove that the performance of the three techniques are stable w.r.t. different users' profiles.

Profile-based Selection

In these experiments we select test users who visited two different cities A and B. We employ their preferences in city A to build and evaluate tours in city B and vice-versa. Among the $2,224$ users visiting both Florence and Rome and 814 visiting Pisa and Florence, we randomly chose 100 users having PoI histories longer than the thresholds discussed above. It is worth noting that for these experiments we need to "uniformize" the categories of the PoIs in the three cities, since Wikipedia provides different categories for each city. Therefore, we exploit the general categorization used within the TripBuilder Web application available online[10]: *Architecture*, *Arts*, *Churches*, *Entertainment*, *Monuments*, *Museums*, *Nature & Landmarks*.

[10]`http://tripbuilder.isti.cnr.it/`

The results of the experiments are reported in Tables 4.6, 4.7 and 4.8. As we can see, the results confirm the trends reported for the previous experiments showing that TripBuilder remarkably outperforms the baselines. Experiments in Pisa, Table 4.6, Tpop shows better results in terms of recall (Pois and categories), but TripBuilder still achieves good scores for recall, and it outperforms the baselines for S^{vt} and S_u^{pro}. Tables 4.7 and 4.8 show that TripBuilder outperforms the baselines in the most cases. Moreover, the similarity of these results with those obtained in the tests using random selection confirm that the performance of TripBuilder is not affected by the "over-specific" categorization used in Wikipedia pages. Finally, it is worth noting that the recall figures computed at the category level are remarkably higher that in the previous experiments. This is motivated by the lower number of categories used to conduct the experiments: covering the whole set of categories with a sightseeing itinerary is in this case much simpler. It is also worth noting that we do not report results of TripBuilder with $\alpha = 0$ here because we are conducting a profile-based experiment. TripBuilder with $\alpha = 0$ does not exploit personalization. For this reason, we do not consider it in this analysis.

	Days	Recall (PoIs)	Recall (Cat.)	S_u^{pro}	S^{vt}	S^{pop}
Tpop	1/2	**0.861** (±0.11)	**0.997** (±0.02)	0.352 (±0.04)	0.639 (–)	**0.832** (–)
	1	**0.915** (±0.08)	**1.000** (–)	0.661 (±0.06)	0.643 (–)	**0.914** (–)
Tppro	1/2	0.480 (±0.15)	0.781 (±0.18)	0.327 (±0.12)	0.608 (±0.03)	0.498 (±0.04)
	1	0.745 (±0.14)	0.951 (±0.09)	0.608 (±0.10)	0.629 (±0.02)	0.766 (±0.03)
TripBuilder ($\alpha = 0.25$)	1/2	0.744 (±0.15)	0.971 (±0.07)	**0.435** (±0.07)	**0.757** (±0.02)	0.736 (±0.06)
	1	0.863 (±0.10)	0.971 (±0.07)	0.653 (±0.05)	**0.709** (±0.02)	0.871 (±0.02)
TripBuilder ($\alpha = 0.5$)	1/2	0.669 (±0.15)	0.928 (±0.11)	0.427 (±0.08)	0.752 (±0.03)	0.681 (±0.07)
	1	0.871 (±0.11)	0.974 (±0.07)	0.661 (±0.04)	0.707 (±0.02)	0.875 (±0.02)
TripBuilder ($\alpha = 0.75$)	1/2	0.634 (±0.14)	0.917 (±0.11)	0.419 (±0.08)	0.749 (±0.03)	0.659 (±0.07)
	1	0.871 (±0.11)	0.974 (±0.07)	0.662 (±0.04)	0.705 (±0.02)	0.873 (±0.02)
TripBuilder ($\alpha = 1.0$)	1/2	0.626 (±0.15)	0.924 (±0.11)	0.420 (±0.08)	0.750 (±0.03)	0.650 (±0.07)
	1	0.869 (±0.11)	0.974 (±0.07)	**0.664** (±0.04)	0.701 (±0.02)	0.868 (±0.03)

Table 4.6: **Profile-based selection**: average effectiveness of TripBuilder and the baselines in Pisa obtained by exploiting the profiles from Florence visits.

4.6.2 Effectiveness – TrajSP

We now evaluate the effectiveness of TripBuilder in scheduling the candidate set of trajectories on the user agenda. We evaluate effectiveness by considering the *average path cost*, the ratio between the cost of the TrajSP solution – i.e., the length of the connections between trajectories endPoIs – over the total budget available (see Figure 4.6). The lower the ratio is, the better TripBuilder employs the time by minimizing the connections when solving the TrajSP instance. Our *Local Search* algorithm is compared with two baselines: i) *Random*, which builds the sightseeing tours by randomly connecting the endPoIs of the trajectories; and ii) *Nearest Trajectory*, which is an adaptation of the well-known "nearest neighbor" TSP greedy heuristic [102] that greedily constructs the solution by always selecting the trajectory with the closest endPoI. The results reported are referred to the whole set of 100 users in the test set. However, since we are evaluating heuristic

	Days	Recall (PoIs)	Recall (Cat.)	S_u^{pro}	S^{vt}	S^{pop}
Tpop	1	**0.583** (±0.15)	0.960 (±0.08)	0.120 (±0.03)	0.601 (–)	**0.591** (–)
	2	0.729 (±0.13)	**0.993** (±0.05)	0.221 (±0.04)	0.607 (–)	0.732 (–)
	4	0.835 (±0.09)	**1.000** (–)	0.365 (±0.04)	0.609 (–)	0.836 (–)
Tppro	1	0.271 (±0.09)	0.778 (±0.12)	0.111 (±0.03)	0.588 (±0.05)	0.289 (±0.03)
	2	0.411 (±0.10)	0.800 (±0.11)	0.190 (±0.04)	0.590 (±0.02)	0.442 (±0.03)
	4	0.632 (±0.11)	0.844 (±0.10)	0.341 (±0.06)	0.586 (±0.02)	0.651 (±0.02)
TripBuilder ($\alpha = 0.25$)	1	0.519 (±0.11)	**0.992** (±0.05)	0.248 (±0.03)	0.741 (±0.01)	0.523 (±0.02)
	2	0.745 (±0.12)	**0.993** (±0.05)	0.388 (±0.04)	0.721 (±0.01)	**0.739** (±0.02)
	4	0.849 (±0.09)	**0.993** (±0.05)	**0.524** (±0.04)	**0.692** (±0.01)	**0.844** (±0.01)
TripBuilder ($\alpha = 0.5$)	1	0.514 (±0.12)	**0.992** (±0.05)	0.248 (±0.03)	0.740 (±0.01)	0.519 (±0.02)
	2	0.739 (±0.12)	**0.993** (±0.05)	0.388 (±0.04)	0.721 (±0.01)	0.735 (±0.02)
	4	0.849 (±0.09)	**0.993** (±0.05)	**0.524** (±0.04)	**0.692** (±0.01)	**0.844** (±0.01)
TripBuilder ($\alpha = 0.75$)	1	0.512 (±0.11)	**0.992** (±0.05)	0.248 (±0.03)	0.740 (±0.01)	0.517 (±0.02)
	2	0.735 (±0.11)	**0.993** (±0.05)	0.388 (±0.04)	0.721 (±0.01)	0.734 (±0.02)
	4	0.849 (±0.09)	**0.993** (±0.05)	**0.524** (±0.04)	**0.692** (±0.01)	**0.844** (±0.01)
TripBuilder ($\alpha = 1.0$)	1	0.509 (±0.11)	**0.992** (±0.05)	**0.249** (±0.03)	**0.743** (±0.01)	0.513 (±0.02)
	2	**0.740** (±0.11)	**0.993** (±0.05)	**0.389** (±0.04)	**0.724** (±0.01)	**0.739** (±0.02)
	4	**0.852** (±0.09)	**0.993** (±0.05)	**0.524** (±0.04)	**0.692** (±0.01)	0.843 (±0.01)

Table 4.7: Profile-based selection: average effectiveness of TripBuilder and the baselines in Florence obtained by exploiting the profiles from Rome visits.

approaches that might be very sensitive to the starting conditions, we ran each experiment 5 times and averaged the results achieved. The stop condition for our TrajSP local search algorithm used in all tests is reaching $1,000$ iterations or 100 iterations without improvement in the solution cost.

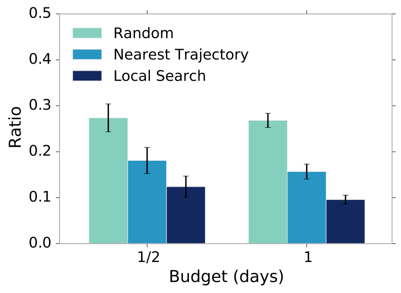
Results show that the Random baseline immediately diverges from the other competitors getting the worst paths. Nearest Trajectory performs better than Random and it is always able to provide an average connection cost lower than 0.3. On the other hand, our Local Search algorithm always outperforms both Random and Nearest Trajectories, with a notable improvement in all the three cities considered. In particular, it provides solutions having an average connection cost lower than 0.2. In other words, TripBuilder is able to address TrajSP even in big cities like Rome by adding at most 20% of the total time budget for inter-trajectory connections.

Efficiency

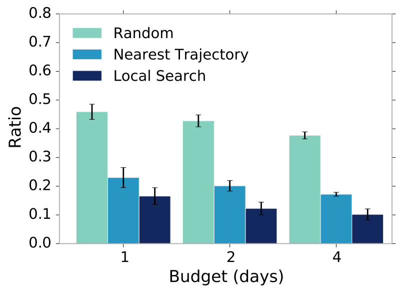
We now report results of experiments conducted to evaluate the efficiency of TripBuilder as a function of the time budget available for the three cities. In particular, we consider time budgets of 1, 2, and 4 days (1/2, 1 days in the case of the small city of Pisa), and, for each instance of the experiment (time budget, city), we run the algorithm for all the 100 distinct users in the test set and all the values of α used in the experiments reported in Table 4.3, 4.4 and 4.5. Figure 4.7 shows the average runtime of TripBuilder along with standard deviation. Moreover, the plot highlights the contributions of TripCover (bottom bars) and of our TrajSP local search algorithm (upper bars) to the overall runtime. It is worth noting that TripBuilder always completes the tour building process in a few seconds. Moreover, the lower the number of trajectories in the dataset,

	Days	Recall (PoIs)	Recall (Cat.)	S_u^{pro}	S^{vt}	S^{pop}
Tpop	1	**0.650** (±0.14)	0.838 (±0.08)	0.187 (±0.02)	0.471 (–)	**0.642** (–)
	2	**0.859** (±0.10)	0.975 (±0.06)	0.407 (±0.02)	0.463 (–)	0.836 (–)
	4	**0.955** (±0.06)	**1.000** (–)	0.676 (±0.02)	0.485 (–)	**0.953** (–)
Tppro	1	0.349 (±0.09)	0.767 (±0.12)	0.160 (±0.05)	0.416 (±0.03)	0.356 (±0.05)
	2	0.526 (±0.11)	0.791 (±0.11)	0.286 (±0.08)	0.427 (±0.02)	0.531 (±0.06)
	4	0.768 (±0.10)	0.853 (±0.09)	0.511 (±0.10)	0.414 (±0.02)	0.772 (±0.05)
TripBuilder ($\alpha = 0.25$)	1	0.627 (±0.09)	**0.883** (±0.10)	0.389 (±0.05)	**0.787** (±0.01)	0.624 (±0.03)
	2	0.824 (±0.08)	**1.000** (–)	**0.630** (±0.04)	**0.739** (±0.01)	**0.842** (±0.02)
	4	0.921 (±0.05)	**1.000** (–)	0.790 (±0.03)	**0.642** (±0.02)	0.937 (±0.01)
TripBuilder ($\alpha = 0.5$)	1	0.614 (±0.10)	0.878 (±0.10)	0.389 (±0.06)	0.785 (±0.01)	0.607 (±0.03)
	2	0.822 (±0.08)	**1.000** (–)	0.628 (±0.05)	0.737 (±0.01)	0.838 (±0.02)
	4	0.918 (±0.05)	**1.000** (–)	0.790 (±0.03)	0.641 (±0.01)	0.935 (±0.01)
TripBuilder ($\alpha = 0.75$)	1	0.612 (±0.10)	0.876 (±0.10)	0.388 (±0.06)	0.785 (±0.01)	0.604 (±0.03)
	2	0.820 (±0.08)	**1.000** (–)	0.628 (±0.05)	0.737 (±0.01)	0.837 (±0.02)
	4	0.916 (±0.05)	**1.000** (–)	0.789 (±0.03)	0.640 (±0.01)	0.935 (±0.01)
TripBuilder ($\alpha = 1.0$)	1	0.609 (±0.10)	0.872 (±0.10)	**0.391** (±0.06)	0.785 (±0.01)	0.602 (±0.03)
	2	0.817 (±0.08)	**1.000** (–)	0.629 (±0.05)	0.737 (±0.01)	0.835 (±0.02)
	4	0.917 (±0.05)	**1.000** (–)	**0.791** (±0.03)	0.641 (±0.02)	0.935 (±0.01)

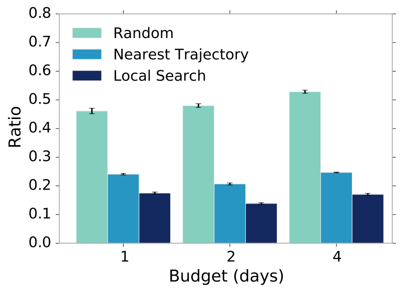
Table 4.8: **Profile-based selection**: average effectiveness of TripBuilder and the baselines in Rome obtained by exploiting the profiles from Florence visits.



(a) Pisa



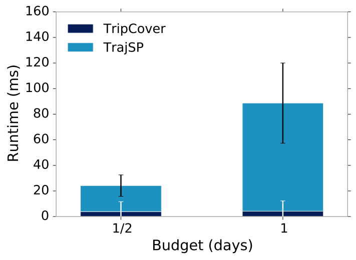
(b) Florence



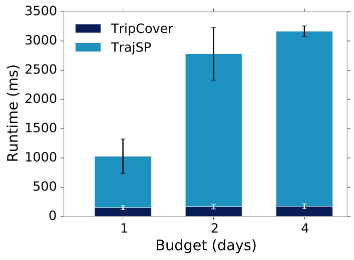
(c) Rome

Figure 4.6: Average path costs of the techniques employed to solve TrajSP as a function of the budget.

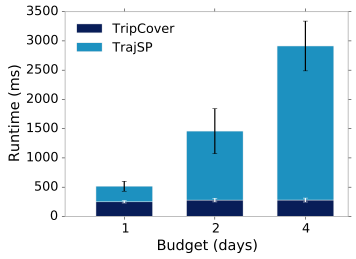
the more similar the runtime is for all budgets. On the other hand, long time budgets (e.g. 4 days) and large datasets impact the total runtime, as expected. Results also reveals low standard deviation of the average runtime thus confirming that the whole process is feasible for on-line applications.



(a) Pisa



(b) Florence



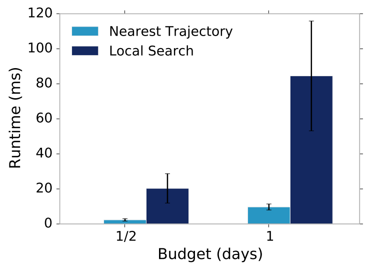
(c) Rome

Figure 4.7: Average total runtime as a function of the time budget for each city. Bottom bars refer to the average TripCover runtime while upper bars refer to the average TrajSP runtime.

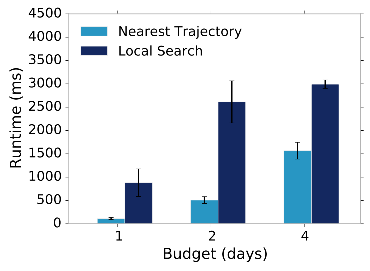
Figure 4.8 details the average runtime of the techniques employed to solve TrajSP. As before, we compute the efficiency by averaging the runtime obtained in 5 runs for each one of the 100 users of the test set. The Random technique obviously has the best runtime performance even if it is not effective in providing solutions as good as the other approaches do. For this reason, we do not report its results in Figure 4.8. Results confirm that both the approaches for TrajSP can be exploited in practice. In general, Nearest Trajectory is always faster than Local Search. However, since the execution times are in any case limited, and Local Search remarkably outperforms Nearest Trajectory in effectiveness it has to be preferred.

4.7 Discussion

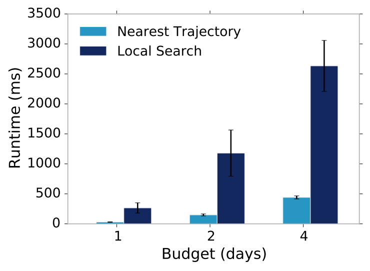
In this chapter, we comprehensively discussed TripBuilder, our unsupervised framework for recommending personalized sightseeing tours. TripBuilder addresses the problem of planning the visit to a city of touristic interest as a two-steps process. First, given the profile of the user and the amount of time available for the visit, the set of itineraries across the PoIs that best fits the user interests and respect the time constraint is chosen. This is done by solving the TripCover problem by means of an approximation algorithm. Then, the selected trajectories are joined in a sightseeing itinerary by means of a local search algorithm addressing TrajSP, a particular instance of the Traveling Salesman Problem. TripBuilder generates the budgeted sightseeing tours by composing popular trajectories performed by real tourists as mined from Flickr Photo albums. Moreover, since both PoIs characteristics and user preferences are mapped into the same



(a) Pisa



(b) Florence



(c) Rome

Figure 4.8: Average runtime of Nearest Trajectory and Local Search by varying the time budget.

categorization automatically extracted from Wikipedia, it is able to personalize the recommended itinerary and even consider the popularity of each PoIs as estimated from the number of photos available for it. Regarding the definition of the user-item interest function, we proposed the use of a PoI popularity function, but this function can be generalized to capture both popular and non-popular points of interest.

We experimented TripBuilder with data collected for three cities different for their size and the amount of user-generated content available for download. The process exploited to mine such content from Flickr, Wikipedia and Google Maps and to build the TripBuilder knowledge base has been detailed, and an analysis of the data collected has been provided. We evaluated our framework by considering the performance of the algorithms proposed to address both the TripCover and the TrajSP problems. The proposed solutions resulted to outperforms the baselines in terms of all the metrics adopted for assessment. Our solution suggests itineraries that better match user preferences. Moreover, such itineraries present higher visiting time and, consequently, lower intra-PoI movement time than the baselines. Furthermore, we assessed the performance of our TSP-based local search heuristic to schedule a set of trajectories into the user agenda. Finally, the tests conducted to demonstrate the efficiency of TripBuilder show that it computes a four-day personalized sightseeing tours of Rome in about 3 seconds thus confirming that our approach can be fruitfully deployed in on-line applications.

The design of TripBuilder has allowed us to develop platform to encompass the required features to create personalized sightseeing tours in the city. The next chapter presents the platform and a web application by detailing its architecture and the main functionalities.

Chapter 5

TripBuilder platform to create personalized sightseeing tours

In the previous chapter we presented the TripBuilder framework jointly with algorithmic solutions to solve the corresponding problems. The results in the previous chapter motivated us to design and develop platform to concretize all the theoretical and algorithmic results.

This chapter, therefore, presents the TripBuilder platform jointly with a user friendly web application[1] encompassing the required artifacts for planning personalized and time-budgeted sightseeing tours in a given city. We present the proposed system by describing the main components used to design an extensible and scalable architecture and the functionalities provided by the interface. We also report the designed and implemented architecture using open-sourced Big Data technologies to scale up the system in the worldwide level. This chapter is a result of [28, 31].

5.1 Introduction

As we have seen in the chapters 2 and 4, planning sightseeing tours is a difficult and time-consuming task, not only for tourists approaching their destination for the first time, but also for tourists who already visit the destination, as they are willing to explore it with a different perspective. Many efforts have been done to develop methodologies and framework to support tourists in achieving these tasks as reported in Chapter 2. TripBuilder system is proposal in this context, since we believe that the development of tools or applications comes to be essential to concretize the theoretical and algorithmic results for planning sightseeing tours.

The system must allow the users (tourists) to create sightseeing tours with personalized points of interest based on the explicit preferences given by the user of the system. In addition, it is also important to provide an user friendly interface that is not only easy to use, but it is also able to provide the needed information about the generated tours, such as time of tour, information about each point of interest, travel information between two consecutive points of interest, etc.

[1]TripBuilder can be accessed in `http://tripbuilder.isti.cnr.it/`

This chapter presents TripBuilder tool as a web application [28][2], a user friendly and interactive system for planning a time-budgeted sightseeing tour of a city on the basis of the points of interest and the patterns of movements of tourists mined from user-contributed data. The knowledge needed to build the recommendation model is entirely extracted in an unsupervised way from two popular collaborative platforms: Wikipedia and Flickr. TripBuilder interacts with the user by means of a friendly Web interface that allows her to easily specify personal interests and time budget. The sightseeing tour proposed can be then explored and modified.

This chapter is organized as follows. We first describe the platform and its main components: data collection, data processing and the TripBuilder engine; in Section 5.2. Section 5.3 presents the user interface of the web application and the most important functionalities of the system. Then, we discuss a scalable and distributed architecture to scale up the mining of semantically-enriched trajectories for TripBuilder at the world level. Finally, a final discussion is presented in Section 5.5.

5.2 TripBuilder Platform

Our designed platform is organized into four modules to generate time-budgeted sightseeing tour (see Figure 5.1): i) *Data Collection*, ii) *Data Processing*, iii) *Data Storage*, and, iv) *TripBuilder Engine*. In the followings we discuss each module with its responsibilities.

5.2.1 Data Collection

The collect of data is a crucial and key task to obtain the collective behavior represented by the wisdom-of-the-crowd for creating personalized and time-budgeted sightseeing tours. This module then collects the required data to build the knowledge base representing the behavior of tourists that have visited the city. In particular, it exploits geo-tagged Flickr photos and Wikipedia points of interest.

It is composed by two different modules that retrieve the relevant information from Flickr and Wikipedia. The first one queries Flickr to retrieve the meta-data (user id, time-stamp, tags, geographic coordinates, etc.) of the sequences of photos taken in the given geographic area, e.g. the city bounding box. An important assumption here is that photo albums implicitly represent sightseeing itineraries within a city. To strengthen the accuracy of our method, this module retrieves only the photos having the highest geo-referencing precision. This process thus collects a large set of geo-tagged photo albums taken by different users in the given geographic area. Table 5.1 illustrates a sample of geo-tagged collected photos.

The second module collects PoIs from Wikipedia. In particular, we assume each geo-referenced Wikipedia named entity, whose geographical coordinates falls into a given area, to be a fine-grained Point of Interest (see Table 5.2). For each PoI, we retrieve its descriptive label, its geographic coordinates as reported in the Wikipedia page, and the set of categories the PoI belongs to that are reported at the bottom of the Wikipedia page. Then, Flickr photos and Wikipedia PoIs are

[2]This work was awarded on European Conference on Information Retrieval 2014 held in Amsterdam, Netherlands, as the best demo paper elected by the attendees.

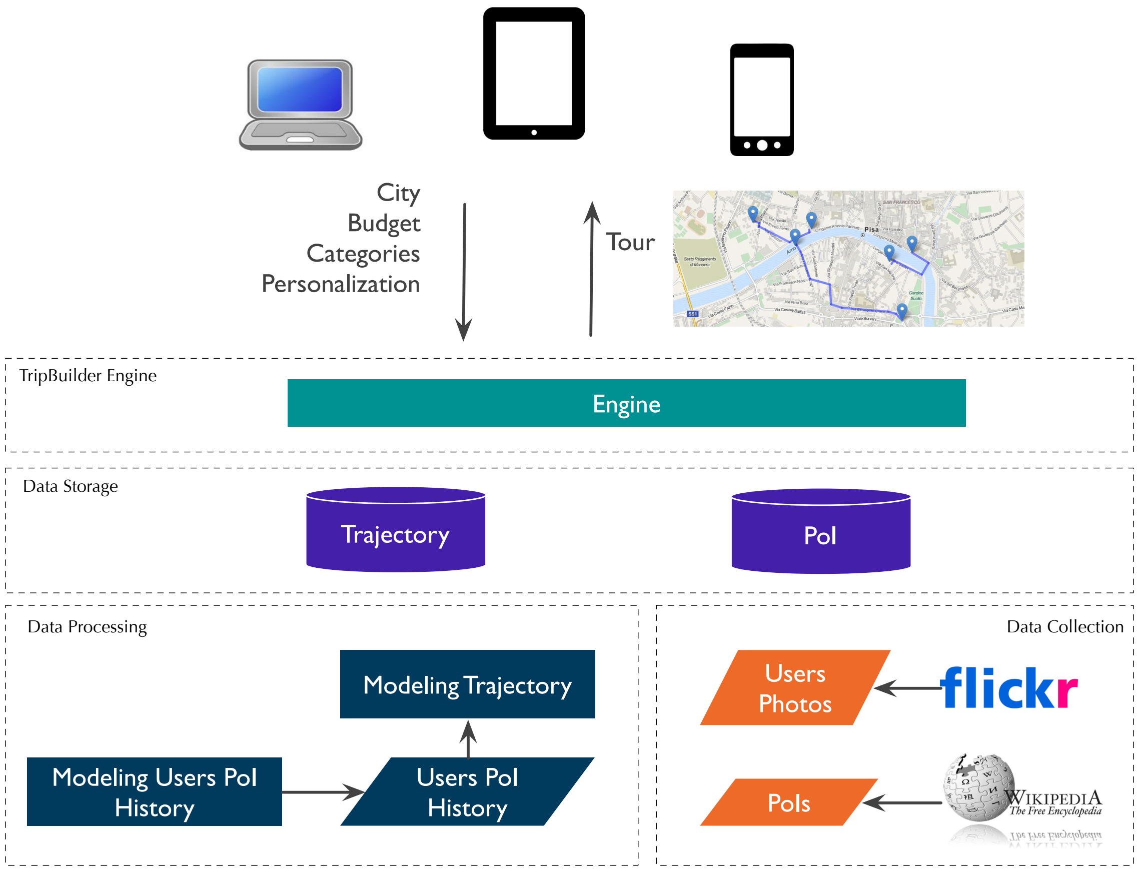


Figure 5.1: Architecture of TRIPBUILDER. We outline the four modules of the system, i.e. *Data Collection*, *Data Processing*, *Data Storage* and *TripBuilder Engine*.

matched by spatial proximity according to their coordinates. After that, the matched PoIs and photos are used by the data processing module to generate trajectories representing the visits of the tourists.

5.2.2 Data Processing

Once photos and points of interest are collected, it is necessary to perform data clean and transformation. These are important tasks to automatically generate high-quality data to be used by the system.

user id	photo id	date taken	date upload	latitude	longitude
0	1	2009-07-25 05:26:32	1248534250	43.723335	10.394546
0	2	2009-07-25 05:31:51	1248534638	43.723335	10.394546
		...	...		

Table 5.1: Example of the geo-tagged photos collected from Flickr.

PoI id	name	latitude	longitude	categories
0	Porta del Leone	43.72374167	10.397325	Porte di Pisa
1	Chiesa di San Michele degli Scalzi	43.70584167	10.41904167	Chiese di Pisa
		...	...	

Table 5.2: Example of the points of interest collected from Wikipedia.

This module consists of different components each one manipulating part of the data previously collected to devise personalized tours and clean data. In particular, the modules here transform sequences of photos from Flickr into trajectories crossing Wikipedia PoIs to be used in the TripBuilder engine module. Moreover, popularity and characteristics of PoIs are computed by considering the number of distinct photos taken, the date taken and date upload of the photos, and the Wikipedia categories. The data obtained are then stored by means of the "Data Storage" module by using a structured database schema. This is an important point in favor of TripBuilder flexibility: different sources of information for trajectories and PoIs can be easily integrated into the system by changing/updating only the two lowest layers.

5.2.3 Data Storage

This component is responsible for storing, querying and indexing trajectory and PoI data. It is composed by a database management system that efficiently provides information to the "TripBuilder Engine" component. This component contains a well defined schema to enable flexibility in integrating other data sources. Geo-spatial indexes are used for searching spatial objects, such as PoIs and tourist traces, within a given region (e.g. polygon). The system also takes advantage of indexes over PoI categories and tourist traces, both represented as arrays, to efficiently retrieve relevant PoIs to the user preferences.

A MongoDB instance is used to store the processed data. We chose MongoDB[3] due to its flexibility to change the data schema, its efficient index implementation over arrays of objects, geo-spatial indexing and its capability of returning streams of data as they are inserted (Capped collections). In particular, our trajectories contain arrays of points of interest with their data (e.g. categories, visiting time). Thus, MongoDB supports the indexing of the trajectories by means of their arrays of points of interests which are queried using the categories present in the given user preferences. In this sense, we do not need to retrieve all trajectories of the target city, but only those trajectories that contain at least one point of interest that is related to the user preference.

5.2.4 TripBuilder Engine

It is the core of the architecture. It receives a set of trajectories crossing a set of PoIs, a time budget, user preferences and a factor used to tune the level of personalization as input, and generates the personalized sightseeing tour. As presented in Chapter 4, the problem of building

[3]https://www.mongodb.org/

the tour is modeled as an instance of the well-known Generalized Maximum Coverage problem [47] and TrajSP as an instance of Travel Salesman problem [35].

In details, the trajectories retrieved by the Data Storage module for the user are matched up against the preferences of the user u to compute for each point of interest p, contained in the trajectories, the user-item interest $\Gamma(u, p)$ introduced in Chapter 4. The result from this process jointly with the time budget, user preferences and the level of personalization are then used by the TriBuilder engine to compute the personalized sightseeing tour for the user u.

In conclusion, the systems presented in this section allows us to develop useful application to support tourists to plan personalized sightseeing tours. The next section presents the web application that consumes the services provided by the TripBuilder platform through a well-defined Restful API.

5.3 The Web Application and Functionalities

The web application built upon the TripBuilder platform currently covers four cities: Pisa, Florence, Rome and Amsterdam. The data of the three first cities are the same used in Chapter 4 to experiment and evaluate TripBuilder. The city of Amsterdam was added to enrich the system and confirm the feasibility to add new cities using the proposed methodology in this thesis. In this section we go through a list of screen-shots of the web application to highlight the generated tours and the other functionalities of the web application.

The user initially need to select the desired city to create the tour. As we can see in Figure 5.2 shows the very first web page with the four cities supported by the system. There is also a component in the page that is integrated to Panoramio[4]. Spite of using only Flickr as the main data source for collecting geo-tagged photos, Panoramio is also a very interesting and rich data source with a huge number of photos uploaded by several users. Once the user selects a city, for instance Amsterdam, she is moved to the next page where she can customize the options to create the sightseeing tour (Figure 5.3). We can see in this figure a drop-down menu named *Preferences* where the user can select the number of days of the tour and he can customize her preferences regarding the categories covered in the system: *Architecture*, *Arts*, *Churches*, *Entertainment*, *Monuments*, *Museums* and *Nature & Landmarks*. For each category, the user can give a score ranging from 0 (low interest) to 5 (high interest) points. In addition, the user is also able to configure the level of personalization and popularity of the generated tour related to the user-item interest function defined in Chapter 4. With the settings done, the user can create the tour by clicking on "Create Personalized Agenda" as shown in Figure 5.3 on right side, e.g., two-days tour (agenda). Then, the user can explore each day and the points of interest to be visited, the visiting duration, photos (Figure 5.5) and the distance to move from one to next point of interest.

The generated tours can be easily saved into the user account (e.g. using Facebook) to be retrieved later either to be modified or to be used during the visit as shown in Figure 5.4. We recall that the philosophy or many motivation behind TripBuilder is the exploitation of historical data of past tourists in the city that can be used to create and enrich the tourism knowledge base

[4]http://www.panoramio.com/

of the system to support new tourists to discovery relevant sightseeing tours in their destinations. In front of this, the web application allows the user to share her create agenda on Facebook (see Figure 5.7) in such a way her friends can take advantage of it when they are visiting the same city.

Beyond the creating of sightseeing tours, the web application has also other useful and interesting capabilities to support the exploration when visiting a city. In particular, the user may wish to find out the most popular PoIs in the city as shown in Figure 5.6. In our empirical observation, we noted that Flick photos indeed can represent the typical tourism behavior of the cities, where the most popular places are usually represented by a great number of photos and users.

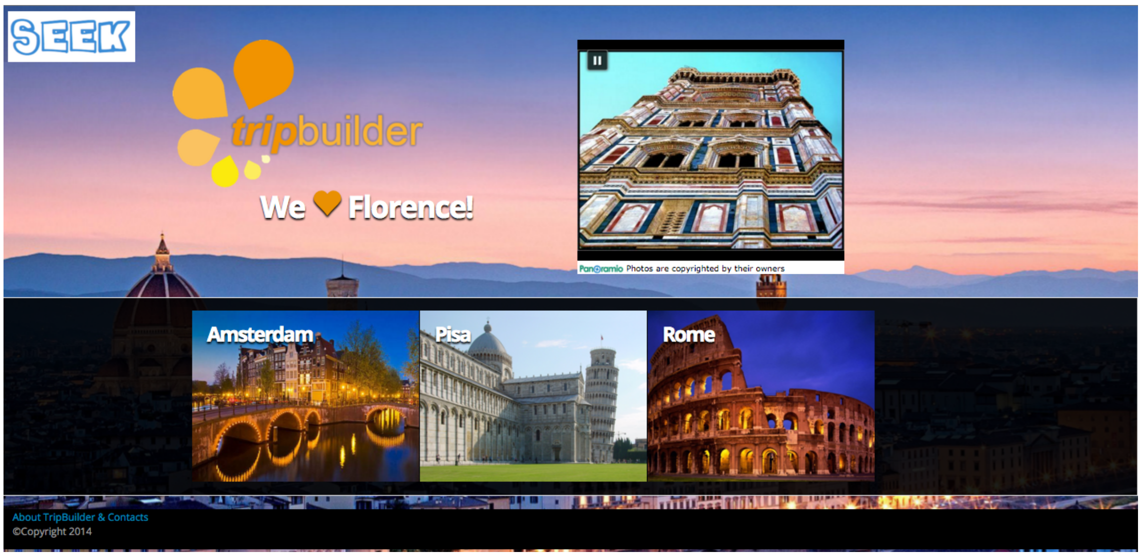
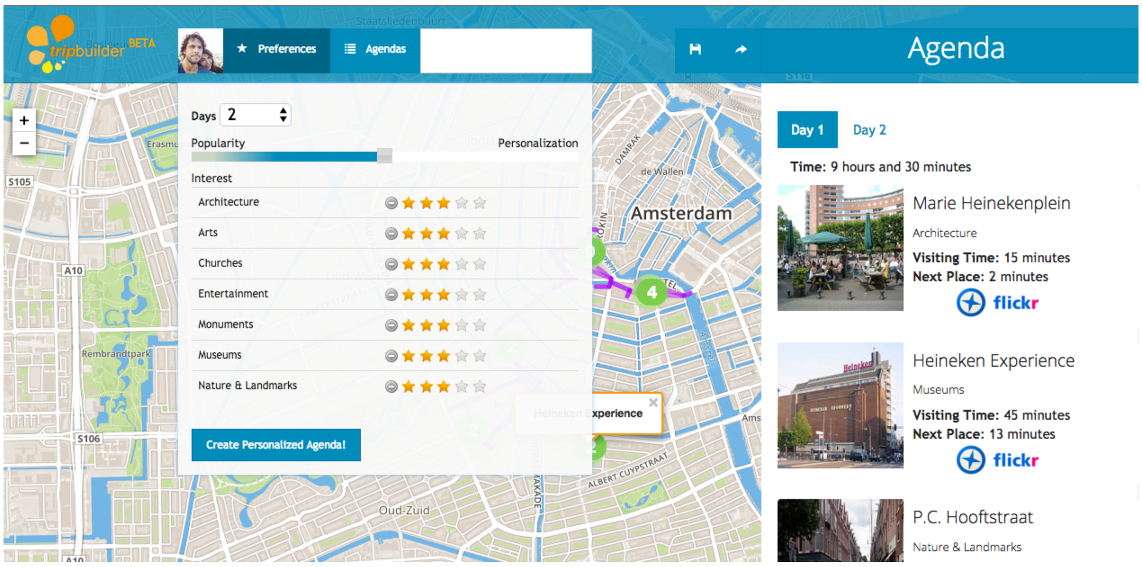

SEEK
tripbuilder
We ♥ Florence!
Panoramio Photos are copyrighted by their owners
Amsterdam
Pisa
Rome
About TripBuilder & Contacts
©Copyright 2014

Figure 5.2: A screen-shot of the Web interface that lets users interact with TripBuilder to select the targeted city in the system. This screen is the very first step on the web application.

5.4 Towards a distributed architecture

The growth of the number of cities covered by the system is imminent in order to have tourist application in the worldwide scale. Thus, we need to design a scalable and distributed architecture to collect and process geo-tagged photos, points of interest and trajectories to generate a complete knowledge base representing the tourists' behaviors in the cities. In the architecture presented in Section 5.2 (Figure 5.1), there are two modules responsible for the collection and processing tasks. These modules can therefore be improved to achieve higher scalability and to cover cities around the globe.

Thus, we present in this section a distributed and scalable architecture for data collection and processing [31]. In particular, we rely on open-sourced Big Data tools that allows us to perform streaming and batching processing in a cloud computing environment with several computational nodes. Figure 5.8 shows our distributed architecture divided into three layers: *Stream Layer*, *Batch Layer* and *Distributed Storage Layer*.



Days 2

Popularity — Personalization

Interest

Architecture	★★★☆☆
Arts	★★★☆☆
Churches	★★★☆☆
Entertainment	★★★☆☆
Monuments	★★★☆☆
Museums	★★★☆☆
Nature & Landmarks	★★★☆☆

Day 1 Day 2

Time: 9 hours and 30 minutes

Marie Heinekenplein
Architecture
Visiting Time: 15 minutes
Next Place: 2 minutes

Heineken Experience
Museums
Visiting Time: 45 minutes
Next Place: 13 minutes

P.C. Hooftstraat
Nature & Landmarks

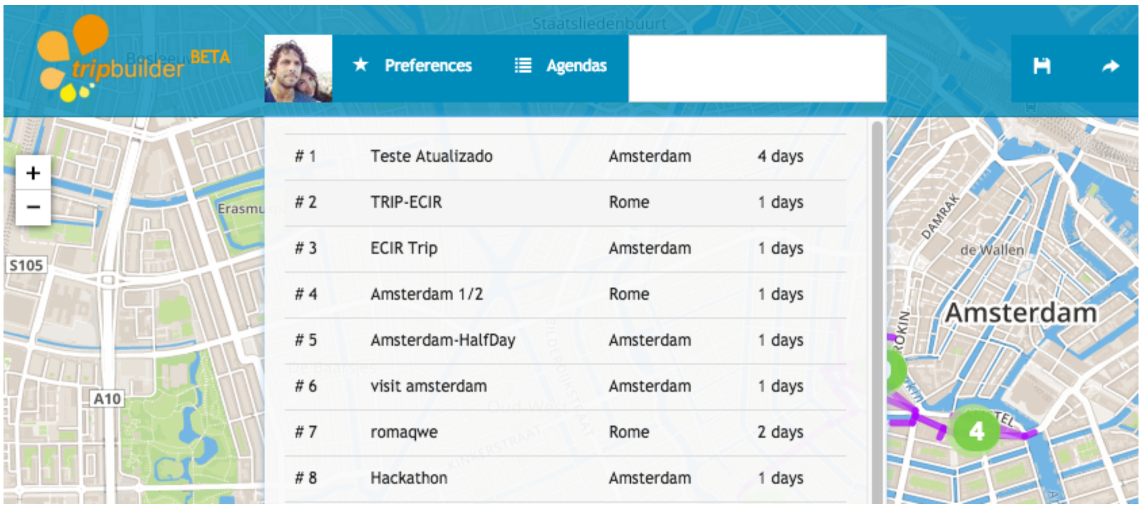
Figure 5.3: Screen-shot illustrating the component for setting the preferences, number of days of the tour, level of personalization and also the details of the created sightseeing tours for each day.

5.4.1 Stream Layer with Apache Storm

Similarly to the data collect in Figure 5.1, this layer is composed of two different modules that retrieve the relevant information from Flickr and Wikipedia by receiving city bounding boxes. However, here the city bounding box are treated as streams of data used to discovery photos and points of interest in a distributed fashion. In particular, each item of the stream is used by *Wikipedia PoI Discovery* to collect PoIs from Wikipedia, where, for each PoI, we retrieve its descriptive label, its geographic coordinates as reported in the Wikipedia page, and the set of categories the PoI belongs to, which are reported at the bottom of the Wikipedia page. This step generates another streams of data containing the discovery points of interest. Then, these new streams are used by *Photo Discovery* to query Flickr to retrieve the required meta-data (user id, time-stamp, tags, geographic coordinates, etc.) of photo albums. Then, photos from Flickr and PoIs from Wikipedia are matched by spatial proximity according to their coordinates. Figure 5.8 highlights the components on the Stream layer.

The stream layer is built by means of Apache Storm[5], a free and open source distributed real-time computation system. Apache Storm allows to reliably process unbounded streams of data. Storm organizes the computation in a graph, called *topology*, where data flows through nodes, called *bolts*. Our stream layer is thus able to crawl Flickr and Wikipedia in a real-time fashion by receiving from an input Kafka[6] queue a given bounding box representing the target geographic area. The results of the real-time computation are stored on a distributed data storage such as HDFS or HBase. Figure 5.9 highlights the topology responsible for processing streams on TripBuilder, where *spout* nodes read data streams like city bounding boxes, PoIs, passing them through *bolt* nodes (**Wiki** and **Photo**) to discovery PoIs and photos respectively, which are stored

[5]`https://storm.apache.org`
[6]`http://kafka.apache.org/`



# 1	Teste Atualizado	Amsterdam	4 days
# 2	TRIP-ECIR	Rome	1 days
# 3	ECIR Trip	Amsterdam	1 days
# 4	Amsterdam 1/2	Rome	1 days
# 5	Amsterdam-HalfDay	Amsterdam	1 days
# 6	visit amsterdam	Amsterdam	1 days
# 7	romaqwe	Rome	2 days
# 8	Hackathon	Amsterdam	1 days

Figure 5.4: List of saved sightseeing tours saved by the user. She can then open and edit any of those tours.

by HDFS bolt nodes. Note that, this topology is highly scalable where *spout* and *bolt* nodes can have as many instances as needed spread across several machines. In our first experiments, we have obtained around 2, 600 points of interest, 1, 302, 356 photos for 20 cities, which gives us an average of 65, 118 photos per city and 500 per point of interest. The collected data are stored in the Distributed Storage Layer for future batch processing.

5.4.2 Batch Layer with Apache Spark

This layer is made up of different components each one manipulating the data previously collected. It is in charge of cleaning and transforming the data by means of distributed computing frameworks like Apache Hadoop[7] and Spark[8] to speed up the data processing step. In particular, the modules here transform sequences of photos from Flickr to sequences of visited Wikipedia PoIs, i.e., trajectories, to be used in the TRIPBUILDER module. Moreover, this step is in charge of computing popularity and other important characteristics of PoIs by considering meta-data and information extracted both from Flickr and Wikipedia. We take advantage of the functional capabilities of Spark to distribute and parallelize the computation on our cloud cluster. Spark has shown to be a great tool for large-scale data processing. The cleaned and transformed data obtained are then stored on the "Distributed Data Storage" layer. This is an important point in favor of enabling the flexibility of TRIPBUILDER: different sources of information for trajectories and PoIs can be easily integrated into the system by modifying only the two lowest layers. Moreover, the approach taken allows to scale to large geographic areas as the two layers effectively exploits modern state-of-the-art technologies for distributed and parallel computation.

[7]`http://hadoop.apache.org`
[8]`http://spark.apache.org`

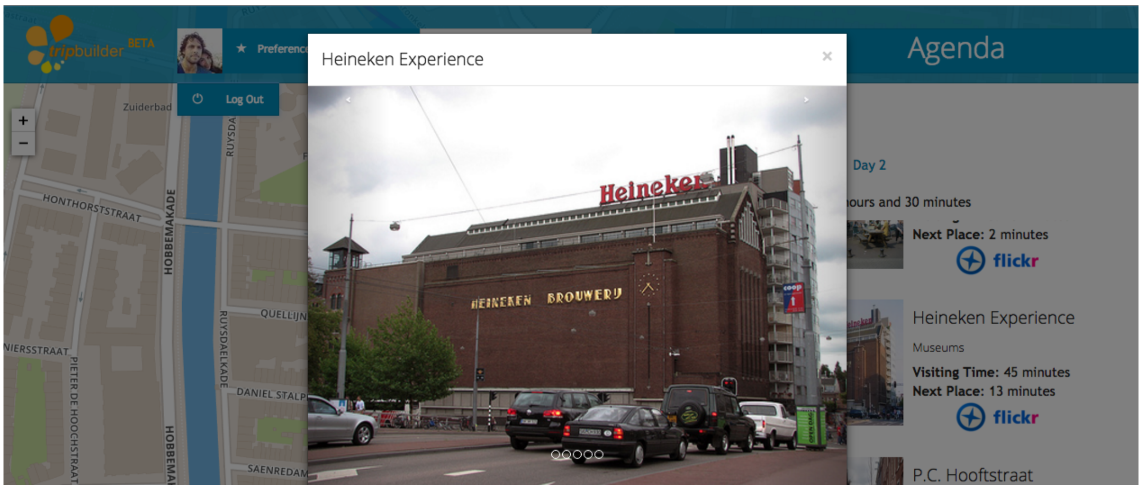


Figure 5.5: Among the information about the points of interest such as time needed to visit, the user is also able to find photos of the point of interest from Flickr and Panoramio. We see in this example a photo of an important museum in Amsterdam that is part of the generated tour.

5.4.3 Distributed Data Storage

This component is responsible for storing, querying and indexing trajectory and PoI data. It is composed by a database management system and a distributed file system that efficiently provides information to the "TripBuilder Engine" component and a distributed data storage to support *Stream* and *Batch* layers. The database component contains a well-defined schema to enable flexibility in integrating other data sources. Geo-spatial indexes are used for searching spatial objects, such as PoIs and tourist traces, within a given region (e.g. polygon). The system also takes advantage of indexes over PoI categories and tourist traces, both represented as arrays, to efficiently retrieve relevant PoIs to the user preferences. Moreover, the distributed file system is built by using the Apache Hadoop Distributed Filesystem (HDFS) jointly with HBase and MongoDB. We choose the HDFS technology as it is a mature solution for storing data in distributed environments. As an example, it provides effective and efficient mechanisms to deal with faults thus preventing us to avoid data loss in case of hardware problems.

5.5 Discussion

In this chapter we presented the TripBuilder tool as a web application to create personalized sightseeing tours. We detailed the main components of the systems which include data collection, processing, storage and the tour creation engine. We presented the user friendly interface of the web application and its main functionalities including the creation of the personalized sightseeing tours, and mechanisms to help the users in exploring the city and in sharing the tours with friends from their social network.

We finally presented an augmented version of the TripBuilder architecture, focusing on the modules responsible for collecting and processing data. We detailed how we designed a distributed and scalable architecture by using open-sourced Big Data tools that allows us to distribute stream-

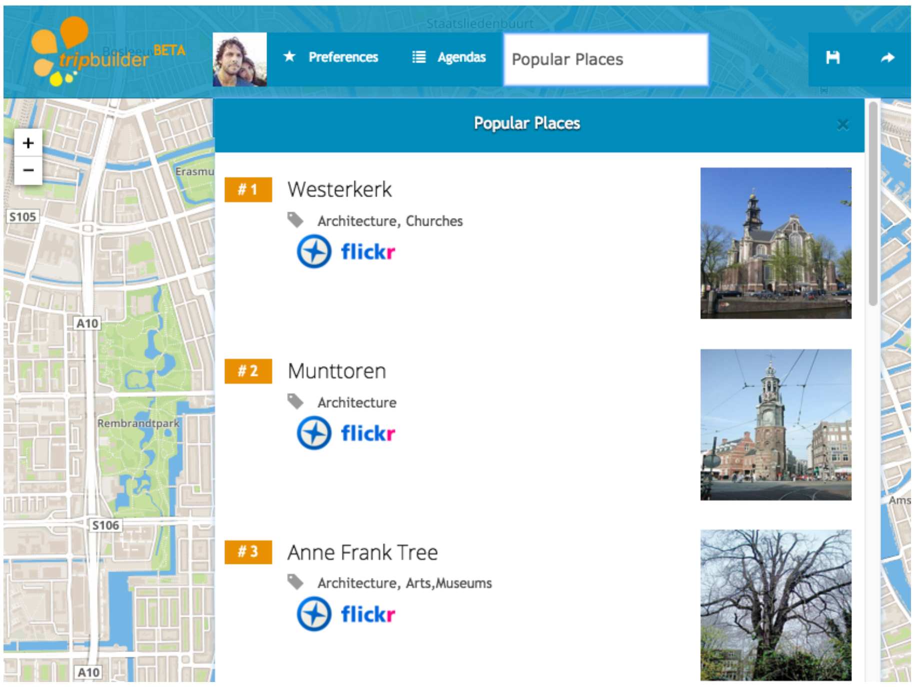


Figure 5.6: Popular places of the city are mined from the collected Flickr photos given important insights for the tourists.

ing and batch computation across several computation nodes in a cloud environment.

The results from Chapters 4 and 5 leverage the following thought: a person usually visit a city in the companions of other ones, like friends, family, etc. This means that in some applications, there is a highly collective appeal. Based on that, we present in the next chapter a framework whose objective is to provide groups by observing the users' preferences and they are related to each other in the social network.

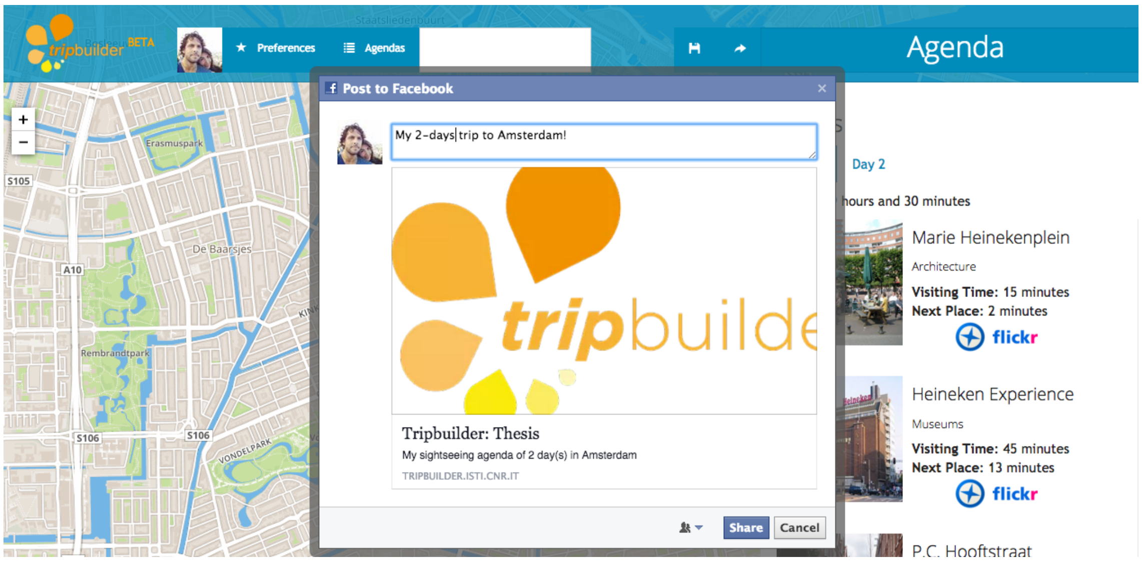


Figure 5.7: Users can save and retrieve their created tours in order to share them with other users (e.g. friends) that might take advantage of it to plan their visit in the city.

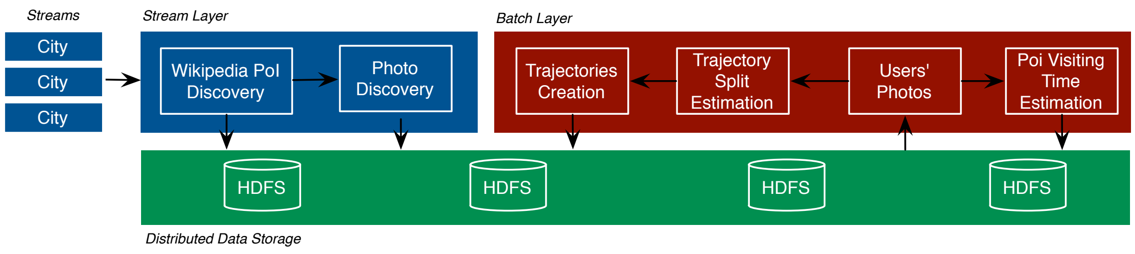


Figure 5.8: Layers of the distributed and scalable architecture of TripBuilder for collecting and processing data.

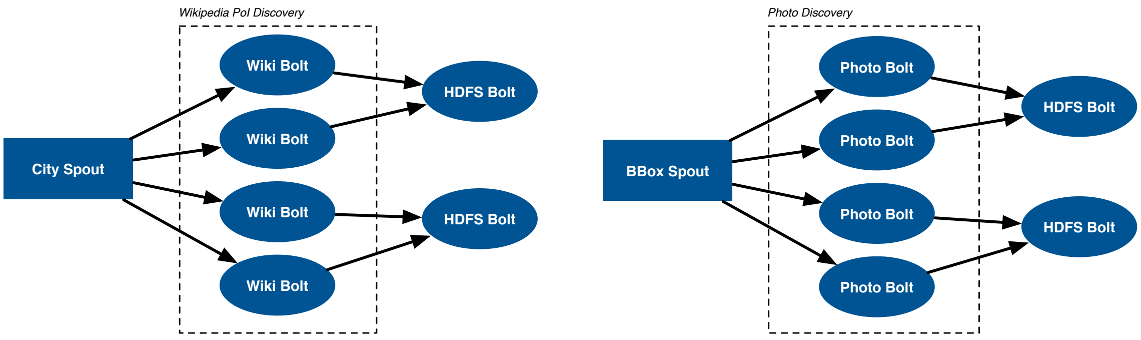


Figure 5.9: TripBuilder Storm topology.

Chapter 6

GroupFinder framework for group formation problem

The work carried out in the area of group recommendation has demonstrated the importance of this problem in real applications due to the need to find relevant and significant items for a group of users instead of individual ones as presented in Chapter 2. The advance of the methodologies has conducted us to a complementary view for group recommendation problem known as the *group formation* problem in the context of recommender systems presented in Chapter 2. In the group formation problem, the goal is to find or form a group of users considering the users' recommendations generated by a recommendation system. In this chapter we investigate the research question **RQ3**: *How can we find out the best groups of users (e.g. friends) who can together enjoy a given item?*.

This chapter presents a novel framework called *GroupFinder* that encompasses algorithmic solutions to address a novel group formation problem considering the recommendations of users, as well as the users' friendship represented by social networks. This chapter is based on the published works [36, 32].

6.1 Introduction

Nowadays, we are witnessing a pervasive use of recommendation systems to support choices in our daily activities, from the most traditional recommendations on books and music, like Amazon[1] and Netflix[2] discussed in Chapter 2, just to mention well-known examples, to the mobile recommendations of attractions to visit and tour itineraries to follow, like TripAdvisor[3], Gogobot[4] and TripBuilder[5] presented in Chapter 4 and 5. We observe that these last activities are usually better enjoyed with travel companions, thus shifting the problem from recommending a single item to a single user (as typical in the traditional cases) to a new paradigm of recommendation that

[1]`http://www.amazon.com`
[2]`http://www.netflix.com`
[3]`http://www.tripadvisor.com`
[4]`http://www.gogobot.com`
[5]`http://tripbuilder.isti.cnr.it`

takes into account a group of users. Traditional recommendation systems primarily focus on identifying relevant items to single individuals using well-known techniques like collaborative filtering [136, 103, 124], or matrix factorization [86]. When the recommendation targets groups of users it is referred to as "group recommendation" and the main goal is to identify items that may have a large consensus among a previously-known group of users [14]. The group recommendation problem is typically hard to solve since a group can be characterized by a diverse mixture of preferences, and finding a trade-off among these preferences may bring to unsatisfactory recommendations for some of the users. In this work we address a complementary perspective of the group recommendation problem. Given a user and a recommended item, we want to suggest the "best" group of friends with whom to enjoy the recommended item. In the TripBuilder system, for example, we want to find a group of friends for the given user with who to enjoy a recommended city, or even a generated sighseeing tour in a city.

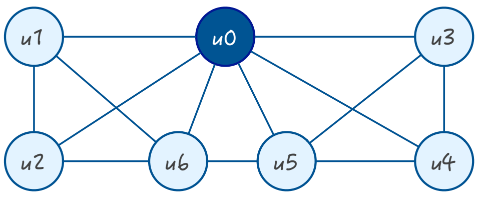
Basu Roi *et al.* has made a great contribution in [15] by investigating the group formation problem from a group recommendation perspective. In spite of that, we believe that an essential characteristic is missing for the group formation: the social networks. In last decade we could see the popularization of social network-based applications and the development of techniques to capture how users are related to each other through interactions (e.g. comments, chat, likes, tags) between them. Therefore, we believe that the social aspect of the users is an essential feature for group formation that may help to find out groups of users that better fulfill the users' expectations.

Consider for example a user who has been recommended to visit Paris: we want to be able to suggest the travel companions who can join her in visiting Paris. Such group should ideally have interest in visiting Paris and also be friend each other to facilitate the staying together. Thus we need to balance the strength of the group internal friendship with the group members interest in traveling to Paris. Considering this last scenario, we design a recommendation technique suggesting the "best" group of k friends for a pair $< user, item >$ taking into account both the social relations and the preferences of the user and the group. Since this approach focuses on the formation of the group based on an item and a user, we refer to it as *User-Item Group Formation* problem. In the remaining of the chapter we often refer to it as UI-GF or simply *group formation* for the sake of readability.

Let us consider the simple example with 7 users and 3 items depicted in Figure 6.1. In this example the items represent destinations that are suggested by a recommender system for tourism. We are interested in finding the best group of 3 users who can enjoy visiting *Florence* together with user u_0. Figure 6.1 (a) reports the relevance score s (ranging from 1 to 5, the higher the value, greater the relevance) of the cities for each user, while Figure 6.1 (b) shows the social network of user u_0 (i.e. her ego network), where links represent friend relationships. A trivial solution would be choosing the users with the highest relevance scores for *Florence*: users u_3, u_4, and u_2. However, when we look at social relationships the perspective changes: the network in Figure 6.1b shows that u_0's friend u_2 is not friend of u_3 and u_4. Indeed, a better group of u_0's friends to enjoy item i_2 should include u_3, u_4 and u_5, since these three users are all friends each other still having a good relevance score for *Florence*. This simple example illustrates the advantage of considering either user-item relevance and the strength of interpersonal relations in a solution addressing the

s	u_0	u_1	u_2	u_3	u_4	u_5	u_6
Pisa	2	3	1	2	2	1	3
Florence	2	1	4	5	5	2	2
Rome	2	4	3	1	1	3	1

(a)



(b)

Figure 6.1: Toy instance of our group formation problem. Table (a) reports the relevance scores of three items for seven users, while the graph in (b) shows the ego network of user u_0 having the same set of users.

group formation problem. To the best of our knowledge this work is the first one that considers both social relations and user-item relevance in group formation.

The main topics covered by this chapter can be summarized as follows:

- we formalize the user-item group formation problem aimed at recommending the best group of friends for a $< user, item >$ pair. We address this novel problem by combining user-item relevance information with the user social network, trying to balance the satisfaction of all the members of the group for the *item* with the intra-group relationships;
- we propose two different solutions that are accommodated into a framework called GroupFinder integrating the needed components and information sources;
- we instantiate the problem in the location-based recommendation domain and we experiment GroupFinder on four publicly available Location-Based Social Network (LBSN) datasets, showing that our solution is effective and outperforms strong baselines.

The rest of the chapter is organized as follows. In Section 6.2 we present the formation of the UI-GF problem. Section 6.3 discusses the algorithmic solutions and Section 6.4 describes the components of the GroupFinder framework. The results of the experiments conducted to assess GroupFinder are reported in Section 6.6, whereas in section 6.7 we present a discussion.

6.2 The User-Item Group Formation problem

Given a user u, her social network $\mathcal{S}$ and an item i suggested to u, UI-GF aims at discovering the group of k friends of u that maximizes a measure modeling the "satisfaction" of the group for the recommended item. Our measure of satisfaction considers both the interest in the recommended item for every member of the group and the intra-group social relations.

We denote with $\mathcal{S}_G = \{\mathcal{U}, E\}$ a social network where $\mathcal{U}$ is the set of users, and E the set of undirected edges modeling the friendship relation between pairs of users in $\mathcal{U}$. Edges $e_{ij} \in E$ are associated with a normalized weight $w(u_i, u_j)$ measuring the *strength* of the friendship between u_i and u_j.

Let $\mathcal{I}$ be a set of items of interest for the users in $\mathcal{U}$. Given a generic user $u \in \mathcal{U}$ and any of the items $i \in \mathcal{I}$ we denote with $R(u, i)$ the relevance for u of i. It is worth noticing that our approach is totally independent from the technique used to generate recommendations of items. Therefore, the relevance metric R can be instantiated with different measures depending on the application and the recommender system, with the only requirement regarding the possibility to compute $R(u, i)$ for every user $u \in \mathcal{U}$ and item $i \in \mathcal{I}$.

Since our approach is based on the combination of two orthogonal dimensions, namely user-item relevance and friendship relations, we define a measure of relevance of an item for pairs of friends. To this purpose, we adapt two well-known relevance aggregation methods from the state of the art [79, 5, 6, 116] (see Chapter 2 for more aggregation methods) to the pairwise case.

Definition 13 (Pairwise User-Item Relevance) *Given an item $i \in \mathcal{I}$ and $u, v \in \mathcal{U}$, we define $R_P(u, v, i)$ to be a generic function measuring the pairwise relevance of i for the two users u, v. We can easily derive two different pairwise user-item relevance measures $R_P(\cdot,\cdot,\cdot)$ from the group-level counterparts, namely the Aggregated Voting (the sum of the recommendation score of the item for each member) and the Least Misery (the minimum of the recommendation scores of the item for each member):*

- $R_{PAV}(u, v, i) = R(u, i) + R(v, i)$ *(*Pairwise Aggregated Voting *measure);*
- $R_{PLM}(u, v, i) = min_{z\in\{u,v\}} R(z, i)$ *(*Pairwise Least Misery *measure).*

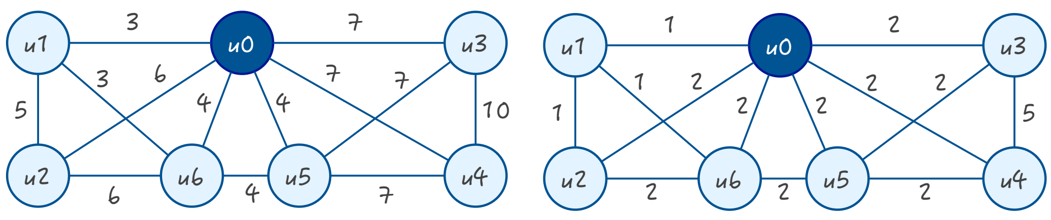
In our group formation approach we need to combine these two dimensions: **friendship** and **item relevance** for the group.

- **Friendship**. The best group for enjoying an item should be preferably formed by people that are all friends each other. Thus, the strength of the friendship relationship between all the members of the proposed group must be taken into account.
- **Item relevance for the group**. The recommended item should be interesting for all the members of the proposed group. The group relevance for a given item is easily captured by the group-level relevance models like Aggregated Voting and Least Misery. However, the pairwise versions of them allow us to weight differently the interest of a given item for a pair of users on the basis of their friendship.

We consider both these two aspects in the definitions of the *pairwise satisfaction* function measuring the “strength” of the relevance of a given item i for two users u and v, and of the User-Item Ego Network where edges are weighted according to such pairwise satisfaction and the normalized weight $w(u, v)$ measuring the strength of the friendship between u and v.

Definition 14 (Pairwise Satisfaction) *Given an item $i \in \mathcal{I}$ and $u, v \in \mathcal{U}$, the pairwise satisfaction of users u and v w.r.t. the item i is defined as $PS(u, v, i) = w(u, v) \cdot R_P(u, v, i)$.*

Definition 15 (User-Item Ego Network) *Given a user u, an item i, and an integer θ, the User-Item Ego Network of u w.r.t i is defined as an undirected weighted graph $\Gamma^{\theta}_{u,i} = (F, E)$ where $F \subseteq \mathcal{U}$ is the set of friends of u at a distance lower than or equal to θ in the original graph $\mathcal{S}_G$, and E is the set of edges weighted by the pairwise satisfaction $PS(\cdot,\cdot, i)$.*



(a) (b)

Figure 6.2: Application of the Aggregated Voting (a) and Least Misery (b) pairwise user-item relevance functions w.r.t. item *Florence* in the previous example.

Considering again the example reported in Figure 6.1, Figures 6.2a and 6.2b show the user-item ego networks obtained by weighting edges according to the Pairwise Aggregated Voting and Pairwise Least Misery measures, respectively, for item *Florence*. The values on the edges represent thus the pairwise satisfaction $PS(\cdot, \cdot, i_2)$.

We model the UI-GF problem of finding the "best" group of k friends of user u for item i as the problem of finding the densest k-subgraph over the user-item ego network. In this formulation the densest k-subgraph problem has the objective of finding the subgraph of exactly k users that maximizes the weighted pairwise satisfaction density. In this way, we select from F a group of k users characterized by strong friendship relations and high interest w.r.t the proposed item i:

User-Item Group Formation Problem

Given a user u, an item i, her user-item ego network $\Gamma^{\theta}_{u,i}$, and an integer k, the User-Item Group Formation problem asks to find the subgraph $G_{u,i} = (F_u, E_u)$ of $\Gamma^{\theta}_{u,i}$, $|F_u| = k$ that maximizes the weighted pairwise satisfaction density:

$$\max_{\forall G_{u,i} \subseteq \Gamma^{\theta}_{u,i}, |F_u| = k} \rho(G_{u,i}) = \frac{2 \cdot \sum_{\forall t,v \in F_u} PS(t, v, i)}{k \cdot (k-1)} \tag{6.1}$$

Solving the user-item group formation problem thus requires to compute the densest k-subgraph maximizing the pairwise satisfaction. The densest k-subgraph problem is NP-hard since it generalizes the clique problem as presented in [8]. In the following section we thus propose an approximation algorithm and an heuristic to address the UI-GF problem.

6.3 Addressing the UI-GF Problem

We address the formulation of the UI-GF problem given in Definition 6.1 by means of the greedy approximation algorithm (Greedy) proposed in [8], and a k-Nearest-Neighbor heuristic ($k-$NN). Both algorithms exploit a measure of pairwise satisfaction aggregated at the level of each user. Let v be a user in our User-Item Ego Network $\Gamma^{\theta}_{u,i} = (F, E)$. The *Aggregated User Satisfaction*, $\phi(v, i)$, is defined as the sum of the pairwise satisfaction computed over all its neighbors (e.g. friends), i.e.,

$$\phi(v, i) = \sum_{x \in F} PS(v, x, i)$$

6.3.1 Greedy algorithm

The Greedy algorithm is an approximation algorithm to solve the densest k-subgraph problem that has been introduced in [8]. The pseudo code of the algorithm is shown in Algorithm 2. It works by repeatedly removing from the input user-item ego network $\Gamma^{\theta}_{u,i}$ the node x with the minimum value of $\phi(x, i)$ (line 3), and by updating the values $\phi(v, i)$ of its neighbor nodes v accordingly. This process is repeated until exactly k nodes are left (condition on line 2).

Algorithm 2: Greedy algorithm from [8] adapted to the UI-GF problem.

```
   Input: User $u$, item $i$, $\Gamma^{\theta}_{u,i}$, integer $k$
   Output: $G_{u,i} = (F_u, E_u), |F_u| = k$
1  $G_{u,i} \leftarrow \Gamma^{\theta}_{u,i}$
2  while $|F_u| > k$ do
       // use a Fibonacci heap to find the node x
3      $x \leftarrow$ node with minimum $\phi(x, i)$ in $G_{u,i}$
4      update $\phi(v, i)$ of every neighbor $v$ of $x$
5      remove $x$ from $G_{u,i}$
6  end
7  return $G_{u,i}$
```

Complexity Analysis. The complexity of the algorithm depends on the values of $\phi()$ that weights the relations on the graph. As claimed in [131, 43], Greedy can be implemented in linear time $O(n + m)$, for m edges and n nodes, when the image of the function $\phi()$ is a subset of $\mathbb{N}_0$. In many real applications, however, $\phi()$ is not an integer value. In this work for example, $\phi() \in \mathbb{R}$ as our pairwise satisfaction is the product between a normalized weight and a pairwise user-item relevance. The algorithm in this case needs to use a different strategy to efficiently find the node with minimum $\phi()$ and update the $\phi()$ of its neighbors. Charikar *et al.* suggest the use of a Fibonacci heap to hold the nodes indexed by their $\phi()$ values to obtain a final complexity of $O(m + n \log n)$ [43]. Using a Fibonacci heap, we are able to extract the node with minimum $\phi()$ with complexity $O(\log n)$, and update $\phi()$ of a given node in $\Theta(1)$ [49, Chapter 19]. As the algorithm removes at most n nodes, and updates at most m neighbors (edges), Greedy with Fibonacci heap has a complexity of $O(m + n \log n)$, for m edges and n nodes in the User-Item Ego Network.

Approximation Analysis. Asahiro *et al.* studied the Greedy algorithm and proposed tight bounds on the worst case approximation ratio R, which is related to the value of k [8]. Two cases are studied by the authors and they are presented as follows.

1. The approximation ratio R of Greedy when $\mathbf{n/3 \leq k \leq n}$ is given by

$$(1/2 + n/2k)^2 - O(n^{-1/3}) \leq R \leq (1/2 + n/2k)^2 - O(1/n).$$

This result demonstrates, for example, that for $k = n/2$, the bounds are $9/4 \pm O(1/n)$, which improves on naive lower and upper bounds of 2 and 4, respectively [8].

2. The approximation ratio R of GREEDY when $\mathbf{k < n/3}$ is given by

$$2(n/k - 1) - O(1/k) \leq R \leq 2(n/k - 1) + O(n/k^2).$$

The choice of k is therefore application-dependent, which is determinant to establish the corresponding approximation ratio. In the context of this thesis whose goal is to find the "best" groups of friends with who to enjoy a particular item, the number of members k in the group will be in general less than $n/3$, where n is likely to be more than 100 and less than 1000 as we will see later in Figure 6.6. Thus, this observation conducted us the second case, when $\mathbf{k < n/3}$.

Running Example. Figure 6.3 illustrates the execution of GREEDY for the example in Figure 6.2a using Aggregated Voting as the pairwise user-item relevance function and $k = 3$. At the first iteration, the user u_1 has the minimum $\phi()$, i.e., $\phi(u_1, Florence) = 11$ (Figure 6.3a). User u_1 is removed and its neighbors are updated in Figure 6.3b. The next user to be removed is u_6, where $\phi(u_6, Florence) = 10$ in the current graph shown in Figure 6.3b and 6.3c. The iterations ends up when we find $k = 3$ friends of u_0, in this example, u_3, u_4 and u_5 (Figure 6.3d).

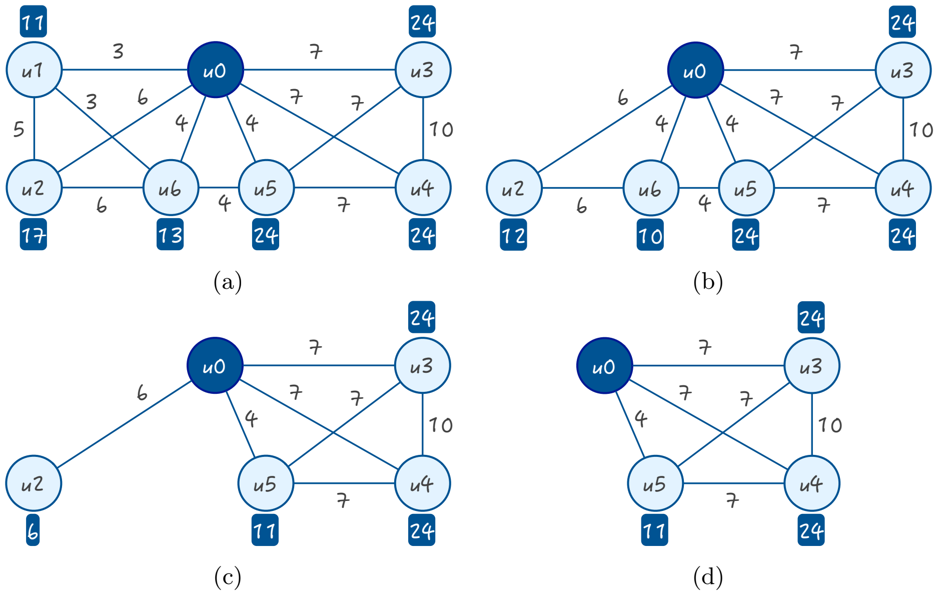


Figure 6.3: Running example of the GREEDY algorithm for the example in Figure 6.2a using Aggregated Voting, the item *Florence* and $k = 3$.

6.3.2 Nearest Neighbor Dense k-Subgraph (k-NN)

The k nearest neighbor is a well-known non-parametric technique successfully employed in several domains ranging from recommender systems to clustering. Here, we employ k-NN on the user-item ego network (Algorithm 3) to retrieve the k neighbors of u having the highest values of $\phi()$. First, the nodes $v \in F$ are sorted by $\phi()$ in the descending order (line 1) to create the list L. Then, the first k nodes having the highest values of $\phi()$ are selected to create the set U of nodes (line 2). Finally, the algorithm creates the subgraph of $\Gamma^{\theta}_{u,i}$ induced by U as the result.

Algorithm 3: Nearest Neighbor Dense k-Subgraph – k-NN

```
  Input: User $u$, item $i$, $\Gamma^{\theta}_{u,i} = (F, E)$, integer $k$
  Output: $G_{u,i} = (F_u, E_u), |F_u| = k$
  // create a list of nodes ordered by $\phi()$
1 $L \leftarrow$ sort $v \in F$ in descending order of $\phi(v, i)$
2 $U \leftarrow$ first $k$ nodes of $L$
3 $G_{u,i} \leftarrow$ subgraph of $\Gamma^{\theta}_{u,i}$ induced by $U$
4 return $G_{u,i}$
```

Complexity Analysis. The algorithm sorts all the nodes in $\Gamma^{\theta}_{u,i}$ in $O(n \log n)$. At most n nodes are selected to create the set U in $O(n)$. Finally, the subgraph induced by U is created in $O(m)$. Therefore, the final complexity of k-NN is bounded by $O(m + n \log n)$.

Approximation Analysis. For this algorithm, we do not provide any bounds for the approximation ratio. However, we empirically study its solutions by confronting them against solutions from other methods (including Greedy) in Section 6.6. In summary, k-NN has not achieved denser subgraphs than Greedy, but it has obtained better results in other important evaluation metrics, such as precision and recall as discussed later in Sections 6.5 and 6.6.

Running Example. Figure 6.4 illustrates the execution of k-NN for the example in Figure 6.2a using Aggregated Voting as the pairwise user-item relevance function and $k = 3$. The nodes $v \in F$ are sorted by $\phi(v, i)$ as shown in Figure 6.4b. The first k nodes are selected as the set U. Finally, the subgraph induced by U is created (Figure 6.4c).

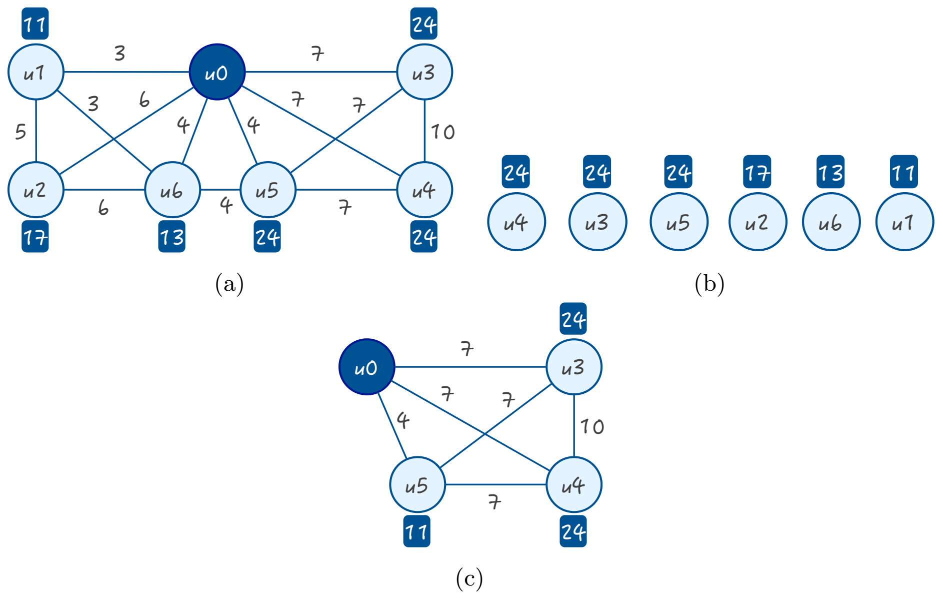


Figure 6.4: Running example of the k-NN algorithm for the example in Figure 6.2a using Aggregated Voting, the item *Florence* and $k = 3$.

6.4 GroupFinder Framework

The algorithms detailed in the previous Section are integrated in the GroupFinder framework that includes three different components (see Figure 6.5): *Recommender System*, *Social Network Manager* and *Group Finder Engine*.

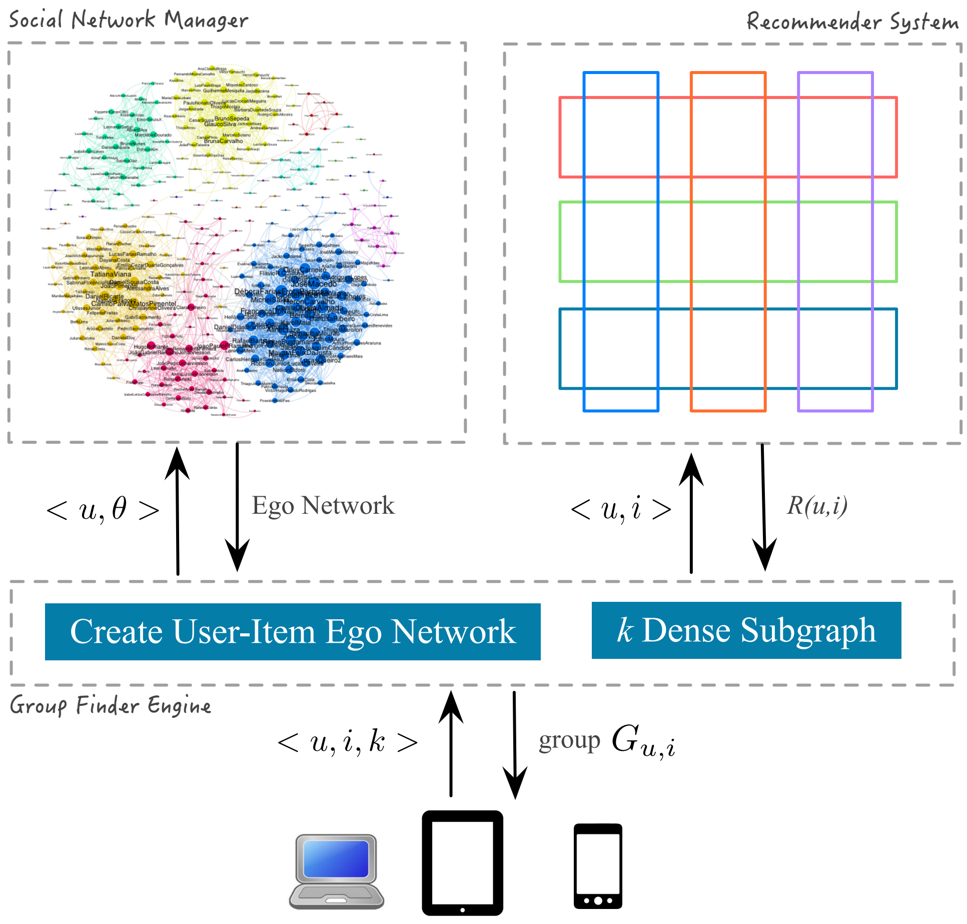


Figure 6.5: The components of the GroupFinder framework: *Recommender System*, *Social Network Manager* and *Group Finder Engine*. The input is the triple $< u, i, k >$ representing the user, the item and the size of the group and the output is the recommended group $G_{u,i}$.

Recommender System

This component is in charge of providing the relevance $R(u, i)$ of the item i for user u. It is a functional component exposing an interface for receiving pairs (u, i) and replying back the associated relevance score $R(u, i)$. It is worth recalling that our solution is independent from the recommendation strategy. Hence, the component may implement a specific recommender technique based on the target application domain, or acting as an intermediate towards an external service. The flexibility of this component may contribute to incorporate GroupFinder in environments with an existing recommender system to leverage new services in the line of user-item group formation.

Social Network Manager

This component manages the information about the social network connecting the users. In particular, given a user u and an integer θ, it retrieves the ego network of focal node u. The ego network consists of node u and its neighborhood composed by all nodes of the social network to whom u has a connection at some path length lower or equal to θ. For example for $\theta = 1$, the ego network of u is formed by u and all the nodes to whom u is directly connected to (the direct friends) plus the ties, if any, among the friends. Besides computing the ego network, the component also evaluates the normalized weight $w(u_i, u_j)$ measuring the *strength* of the friendship between any pair of users in the ego network. Our UI-GF solution is independent from the method used to compute such friendship strength.

Group Finder Engine

This component implements the algorithmic solutions for approaching the UI-GF problem. Given a request composed by a triple (u, i, k), it coordinates the interaction with the other two components aimed at obtaining the user ego network and the relevance scores of item i for all the members of u ego network. Then, it builds the user-item ego network $\Gamma^{\theta}_{u,i}$ by exploiting the pairwise satisfaction function computed for each pairs of users. Finally, the densest k-subgraph is computed by means of the algorithms detailed in Section 6.3 and returned as result of the UI-GF instance.

It is worth noticing that the proposed modular design allows GroupFinder to be very flexible since it permits to easily encompass different (external) recommender systems, social networks and friendship metrics.

6.5 Experimental Settings

We propose a comprehensive assessment of GroupFinder against state-of-the-art baselines by employing four public LBSN datasets. We first introduce the datasets then we detail the baseline algorithms and the metrics used for evaluation. Finally, we discuss the results of the experiments conducted in Section 6.6.

6.5.1 Datasets

We employ four publicly available datasets collected from three popular LBSN services: Foursquare, Brightkite, and Gowalla. These datasets provide the users registered to the social networks and the venues where the users checked-in, typically entertainments like restaurants and cinemas or tourist attractions like museums and monuments.

Foursquare[6] is a popular LBSN where users check in at places to inform friends on where they are. Thanks to the authors of [93, 125], we downloaded a dataset containing users check-ins, places,

[6] `https://foursquare.com/`

users ratings of the places, and the social graph connecting users[7]. Starting from this dataset, which is called hereinafter Foursquare, we built a second dataset by selecting only the check-ins falling in the bounding box of New York city[8]. This second dataset is called in the following Foursquare (New York). We also used two other LBSN datasets collected from Brightkite and Gowalla[9] made available from the authors of [45]. These datasets, as the previous one, record user check-ins and the social network connecting users. However, they do not report the users' ratings as shown in Table 6.1.

Dataset	Users	Items	Links	Ratings	Check-ins
Foursquare	485,381	1,143,089	13,549,236	2,809,580	1,021,965
Foursquare (NY)	55,252	74,149	945,422	623,437	157,064
Brightkite	58,228	772,966	214,078	–	4,491,143
Gowalla	196,591	1,280,969	950,327	–	6,442,890

Table 6.1: Statistics regarding the four datasets used in the experiments: Foursquare, Foursquare (New York), Brightkite and Gowalla.

Table 6.1 shows some statistics about the datasets. Foursquare is the largest one in terms of number of users, with a very large social network made up of about thirteen millions edges. Gowalla has the largest number of check-ins. The degree distributions of the users in the social networks are shown in Figure 6.6. As expected, all the datasets present a power-law distribution in the node degrees: the majority of the users have a limited number of friends, while only a few users have thousands or more friends. This is an important consideration as the degree distribution affects the size of the user-item ego network $\Gamma^{\theta}_{u,i}$. In the following experiments, we set $\theta = 1$ to consider only "direct" friends of the user in her user-item ego network.

6.5.2 Computing the relevance scores

GroupFinder relies on a given recommendation technique to produce the relevance scores of items for users. For the experiments we use a content-based recommender system that exploit the meta-data associated with venues to measure user-item relevance scores. To this purpose, we downloaded the categories of venues using the Foursquare API[10] for all the datasets. Let us denote the set of categories as C. This allows us to compute for each venue $i \in \mathcal{I}$ its *relevance vector* $\vec{v}_i \in [0,1]^{|C|}$ measuring the normalized relevance of i w.r.t the set of categories C. Moreover, we computed for each user $u \in \mathcal{U}$ her *preference vector* $\vec{v}_u \in [0,1]^{|C|}$ stating the normalized interest of u for the same set of categories C. As in [98, 35, 34], the preference vector of each user is obtained from the data. To this end we exploited either the normalized ratings of the category (if available), or the normalized number of check-ins in venues belonging to each category.

[7] Available at `https://archive.org/details/201309_foursquare_dataset_umn`
[8] `https://www.flickr.com/places/info/2459115`
[9] Available at `https://snap.stanford.edu/data/`
[10] `https://developer.foursquare.com/`

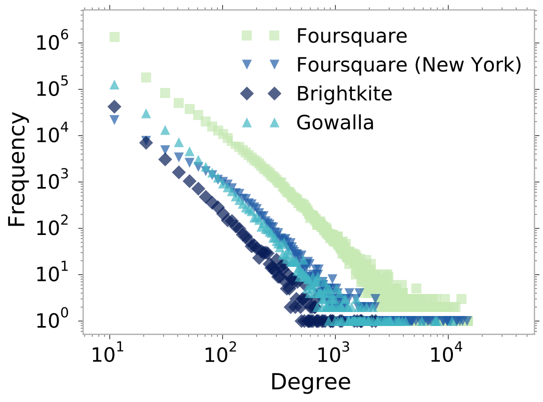


Figure 6.6: Degree distribution of the social networks of the four datasets used in the experiments: Foursquare, Foursquare (New York), Brightkite and Gowalla.

The relevance score $R(u, i)$ of an item i for a user u is computed as the cosine similarity between the user's preference vector $\vec{v}_u$ and item's relevance vector $\vec{v}_i$ [98, 35, 34]:

$$R(u,i) = \frac{\vec{v}_u \cdot \vec{v}_i}{||\vec{v}_u|| \times ||\vec{v}_i||}$$

This process allows us to compute the relevance $R(u, i)$ of a given item i for every user u. These relevance values are in fact needed to build the user-item ego network $\Gamma^{\theta}_{u,i}$ by means of our pairwise satisfaction function using either Aggregated Voting (PAV) or Least Misery (PLM) measures (see Definition 13). For the experiments we adopt a binary function $w(u, v)$ to model the relationships between pairs of users ($w(u, v) = 1$ iff u and v are friends, $w(u, v) = 0$ elsewhere). It is worth noticing that our formalization allows any strength function to be used. When available, the information about the interactions between pairs of users in the social network (e.g., the number of messages exchanged, the number of likes or comments, the number of common friends, etc) could be fruitfully used to model more accurately the strength of the relationship.

6.5.3 Ground-truth groups

To evaluate the quality of the groups proposed by GroupFinder we compare them against ground-truth groups, i.e., groups of friends that actually enjoyed a specific venue. We extracted these ground-truth groups from the four datasets. In particular, we looked in the datasets for sets of users who *checked in at the same place within a fixed temporal window.* In addition, we considered a user to be member of a group *only if she is friend of at least one of other group members.* In this way we obtained groups of users who enjoyed the place where they checked in, together with their friends.

As an example let us consider 6 users friends with each other who checked in at places i_1 and i_2 in two different days:

$$(u_0, i_1, 2015/10/09\ 13:30), (u_1, i_1, 2015/10/09\ 13:50)$$
$$(u_2, i_1, 2015/10/09\ 14:10), (u_3, i_2, 2015/11/02\ 08:23)$$
$$(u_4, i_2, 2015/11/02\ 09:01), (u_5, i_2, 2015/11/02\ 08:50)$$

By considering a temporal window of 1 hour, users u_0, u_1, u_2 form a ground-truth group since they checked in at place i_1 within the same temporal window. In the same way, users u_3, u_4, u_5 form another ground-truth group for item i_2.

In our experiments, we set a temporal window of 4 hours and consider only groups with at least 4 members. This led us to devise 1, 495 for Foursquare, 258 ground-truth groups for Foursquare (New York), 24, 996 for Brightkite and 27, 997 for Gowalla. Gowalla has the largest number of ground-truth groups since it also has the largest number of check-ins (see the check-ins column in Table 6.1).

Experiments are conducted for each ground-truth group, by arbitrarily selecting one of the users as the *focal node*. We then ask GroupFinder to suggest a group of friends for this specific user and venue. The remaining users of the ground-truth group are of course those we would like to find in the group suggested by GroupFinder.

6.5.4 Performance Metrics

We assess the quality of the group recommended by GroupFinder and the baselines solutions on the basis of different metrics. The first metric is exactly the **weighted pairwise satisfaction density** used in Definition 6.1. This metric is exactly the one that our algorithms (and the baselines) try to maximize. It thus allows to assess the effectiveness of the various algorithms in approximating the densest k-subgraph of the user-item ego network. The other performance metrics exploit instead the ground-truth groups above discussed.

Let $\hat{F}_{u,i}$ be a ground-truth group for user u and venue i extracted as previously discussed from a LBSN dataset. Moreover, let $F_{u,i}$ be the group generated by GroupFinder or the baselines solutions for the same user u and venue i. To evaluate the effectiveness of the various algorithms in suggesting groups possibly similar to the ones mined from actual data, we used two well-known information retrieval metrics: *precision* and *recall* [10].

Precision. This metric computes the fraction of members in $F_{u,i}$ that also appear in the ground-truth group $\hat{F}_{u,i}$:

$$precision(F_{u,i}) = \frac{|\hat{F}_{u,i} \cap Fu, i|}{|F_{u,i}|}$$

Recall. This metric computes the fraction of actual group members in $\hat{F}_{u,i}$ that are present in the suggested group $F_{u,i}$:

$$recall(F_{u,i}) = \frac{|\hat{F}_{u,i} \cap F_{u,i}|}{|\hat{F}_{u,i}|}$$

The rationale behind using these two metrics is that the higher the precision and recall are, the more similar to the actual choices of real LBSN users the suggested groups. In the experimental evaluation reported below, the figures of precision and recall reported are the average ones computed over all the ground-truth groups in the specific dataset.

6.5.5 Baselines

We compare the performance of GroupFinder with two baselines: i) *Densest k-Subgraph* and ii) *Top k-Nodes*.

Densest k-Subgraph ($DkSP$)

This baseline is a known algorithm from [62] that aims at selecting the densest k-subgraph from a graph G. It works by first identifying three candidate k subgraphs by applying the following three procedures:

Procedure 1. Select $k/2$ arbitrary edges from the graph, then return the set of nodes incident with these edges, adding arbitrary nodes to this set if its size is lower than k.

Procedure 2. Create two disjoint sets H and C. The set H includes the $k/2$ nodes with highest $\phi()$ in the input graph G. The set C is created by selecting $k/2$ nodes from $G \setminus H$ with the highest $\phi()$ w.r.t. the nodes in H. Return the subgraph induced by the set $H \cup C$.

Procedure 3. Let $W_2(u, v)$ be the function that returns the number of paths of length 2 between two nodes u and v. Let H be the set with $k/2$ nodes with the highest $\phi()$ in the input graph G. For every node v in H compute $W_2(v, w)$ for all $w \in G$, and create a set H^v with $k/2$ nodes with the highest $W_2(v, w)$. Then, create the set B^v with the $k/2$ neighbors x of v with the highest $\phi()$ w.r.t. the set H^v. Finally, return the subgraph G'_v induced by the set $H^v \cup B^v$, adding arbitrary nodes to this set if its size is smaller than k.

Each one of the previous procedures generates a candidate k subgraph. The $DkSP$ algorithm returns the densest k-subgraph among these three candidates.

Top k-Nodes (k-Top)

Top k-Nodes is a trivial heuristic to compute the densest k-subgraph without considering the edges. It forms the group by retrieving the k nodes of the user-item ego network with the highest value of $R(\cdot, i)$. Note that in this approach the relationships among the users are not considered. Consequently, it does not use the pairwise satisfaction function.

6.6 Experiments

6.6.1 Effectiveness

We now evaluate the proposed algorithms by using the performance metrics defined above to demonstrate the effectiveness of our proposals.

Weighted Pairwise Satisfaction Density

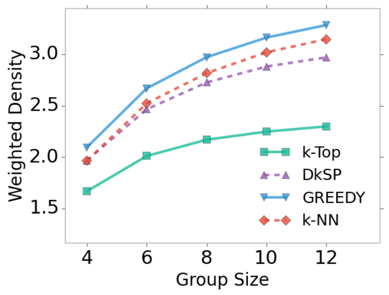
This metric highlights the friendship level of the group jointly with the interestingness of its members for the item i, i.e., the group satisfaction. Good solutions tend to have more connections within the group while keeping an high score of the venue for the involved users. The results reported in Figure 6.7 show the weighted density of the subgraphs found by varying the size k of the group. Here, densest k-subgraph-based approaches tend to overcome k-Top. k-Top considers only the user interest measured by $R(\cdot, \cdot)$, thus it may generate groups in which the members are not actually friends in the social network. This explains the lower weighted density obtained with k-Top. Interestingly, $DkSP$ performs remarkably better with PAV than with PLM pairwise user-item relevance.

Greedy and k-NN algorithms outperform $DkSP$ and k-Top in terms of weighted density with both PAV and PLM pairwise user-item relevance. Greedy performs better than k-NN thus highlighting that its greedy strategy results to be more effective in finding dense subgraphs. Greedy outperforms $DkSP$ from 6% to 17% for PAV, and from 26% to 46% for PLM. This means that it suggests groups characterized by a good balance between friendship and users' relevance, avoiding to include users who are not interested in the item or users that are not well-connected with the rest of the group members.

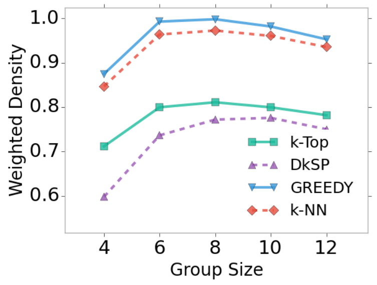
The above results highlight the effectiveness of our solutions to approximate the densest k-subgraphs and the importance of considering the relationships and the relevance of the item for the users. However, we have still to assess how meaningful the suggested groups are for the targeted users. To this end we now analyze the effectiveness of the proposed algorithms by considering **precision** and **recall** figures that allow us to understand if the groups proposed by GroupFinder are really relevant w.r.t. the ground-truth groups mined from the data.

Precision and Recall

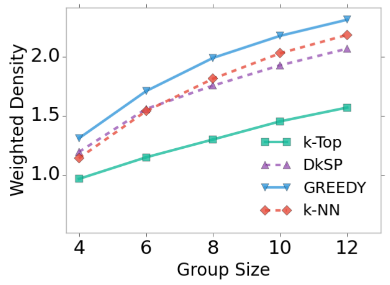
Although k-NN does not produce solutions as good as Greedy in terms of approximating the densest k-subgraph, it has obtained important scores in terms of precision and recall. Figure 6.8 depicts the results for precision. As we can see, both Greedy and k-NN outperform k-Top and $DkSP$ in terms of precision with both PAV and PLM metrics. We can observe that on average Greedy achieves better results on Foursquare datasets, while k-NN demonstrates a better performance on the Brightkite and Gowalla datasets. It is worth highlighting that the improvement is higher for smaller values of k, while for larger groups the difference decreases. Moreover, Greedy and k-NN are able to suggest more precise groups when using the PLM user-item relevance. As shown in Table 6.2, the precision measured for k-NN using PLM results to be up to 14%, 3%, 5%, 6% higher than the one with PAV for Foursquare, Foursquare (New York), Brighkite and Gowalla datasets, respectively. This results highlight an interesting user behavior: real users tend to invite at a venue the friends that are expected to like it, while they rarely invite a friend to enjoy a certain venue if they know she does not like it. This behavior is better captured by the pairwise least misery relevance that considers the minimum among the user-item relevance scores for forming the group.



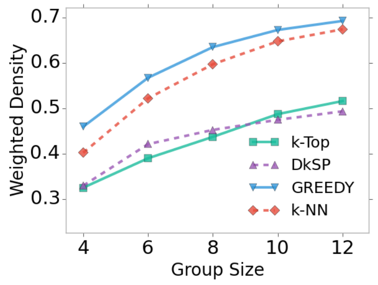
(a) Foursquare PAV



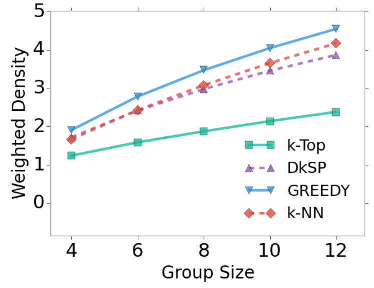
(b) Foursquare PLM



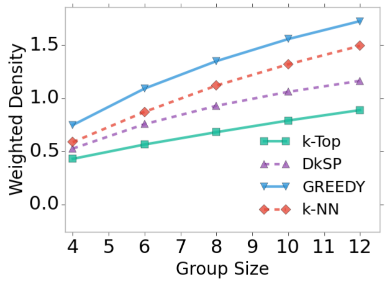
(c) Foursquare (New York) PAV



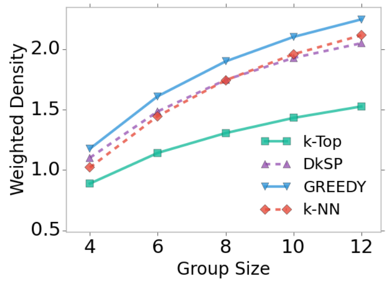
(d) Foursquare (New York) PLM



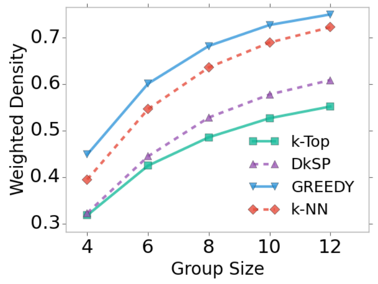
(e) Brightkite PAV



(f) Brightkite PLM

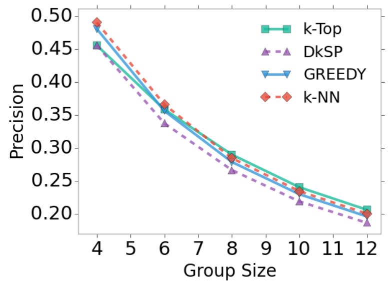


(g) Gowalla PAV

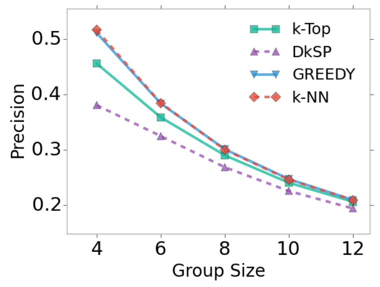


(h) Gowalla PLM

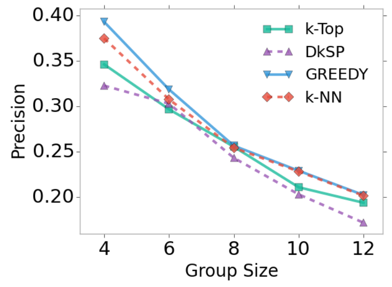
Figure 6.7: Weighted density of the groups computed with the various algorithms employing PAV and PLM on the four datasets: Foursquare, Foursquare (New York), Brightkite and Gowalla.



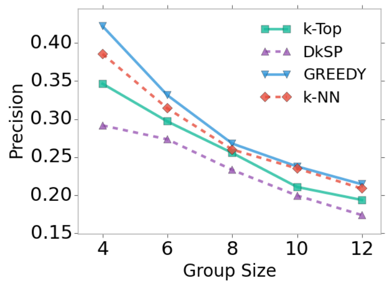
(a) Foursquare PAV



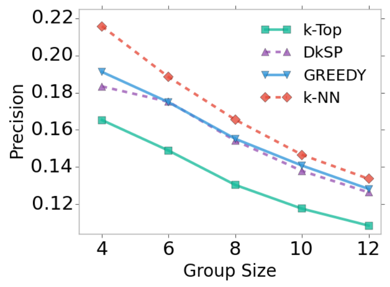
(b) Foursquare PLM



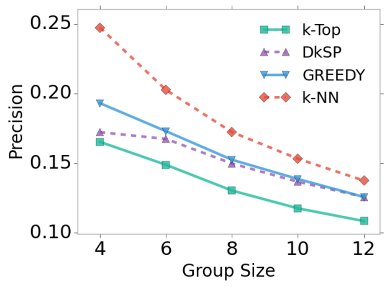
(c) Foursquare (New York) PAV



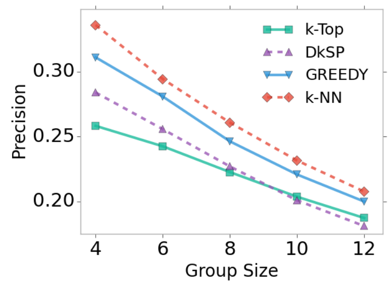
(d) Foursquare (New York) PLM



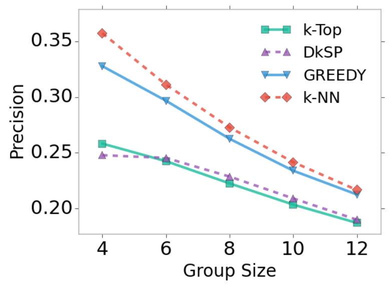
(e) Brightkite PAV



(f) Brightkite PLM

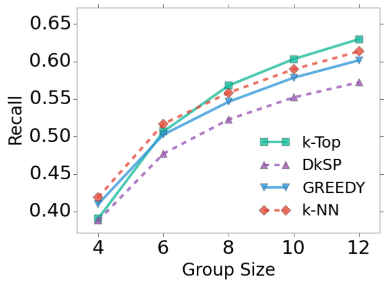


(g) Gowalla PAV

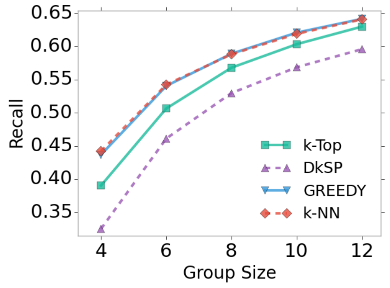


(h) Gowalla PLM

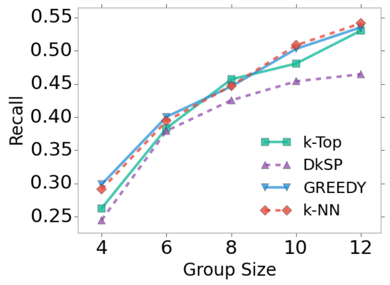
Figure 6.8: Precision computed on the basis of the groups suggested by the various algorithms employing PAV and PLM w.r.t. the ground-truth groups for the four datasets: Foursquare, Foursquare (New York), Brightkite and Gowalla.



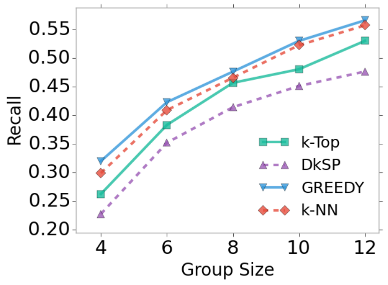
(a) Foursquare PAV



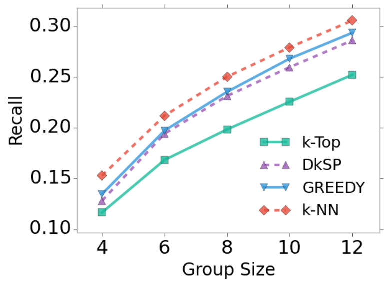
(b) Foursquare PLM



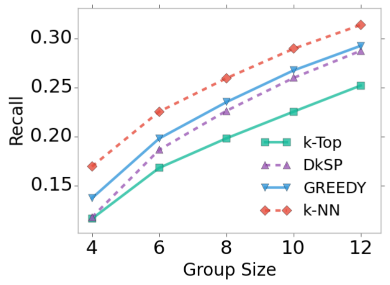
(c) Foursquare (New York) PAV



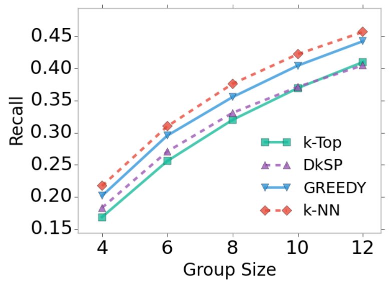
(d) Foursquare (New York) PLM



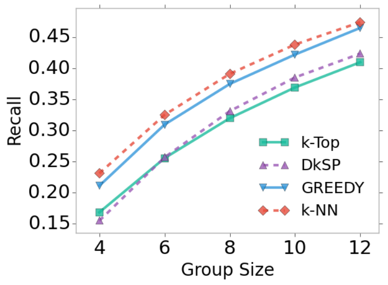
(e) Brightkite PAV



(f) Brightkite PLM



(g) Gowalla PAV



(h) Gowalla PLM

Figure 6.9: Recall computed on the basis of the groups suggested by the various algorithms employing PAV and PLM w.r.t. the ground-truth groups for the four datasets: Foursquare, Foursquare (New York), Brightkite and Gowalla.

		Foursquare		**Foursquare (NY)**		**Brighkite**		**Gowalla**	
	k	p	r	p	r	p	r	p	r
Greedy	4	6.2	6.6	7.2	7.1	0.9	2.0	5.3	4.8
	6	7.6	7.5	3.9	5.6	−1.0	0.4	5.6	4.8
	8	7.9	7.8	4.3	6.7	−1.7	−0.1	6.5	5.5
	10	7.5	7.3	3.7	5.4	−1.5	−0.2	5.9	4.4
	12	6.8	6.6	5.9	5.7	−1.8	−0.3	6.3	5.1
	24	4.9	5.1	5.1	5.6	−3.0	−1.6	6.0	4.6
k-NN	4	5.2	5.5	2.7	2.6	14.6	11.0	6.2	6.5
	6	4.9	4.9	2.0	3.8	7.5	6.4	5.6	4.9
	8	5.4	5.3	2.1	4.2	4.1	3.7	4.6	4.0
	10	5.0	4.9	3.0	2.8	4.4	3.7	4.3	3.8
	12	4.4	4.4	3.5	2.9	2.8	2.6	4.3	3.8
	24	2.6	2.7	3.5	4.5	0.2	0.6	3.6	2.9

Table 6.2: Improvements (%) of *precision* (p) and *recall* (r) by varying k for Greedy and k-NN when using PLM instead of PAV.

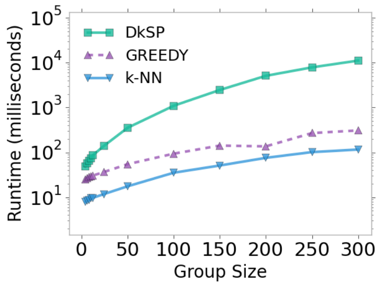
A similar behavior is confirmed when evaluating the performance of the algorithms by using the **recall** metric (Figure 6.9). The plots in the figure shows that Greedy and k-NN achieve higher recall when PLM is used. Interestingly, Figures 6.9 (a) and (b) shows that when PAV is used the k-Top baseline exhibits better recall than Greedy and k-NN on the Foursquare datasets. This does not hold for the tests using PLM where Greedy and k-NN algorithms outperforms k-Top (and *DkSP*) in all the tests conducted. The advantage of Greedy using PLM instead of PAV is up to 7% in the two Foursquare datasets. Moreover, it is up to 5% and 4% for k-NN in Foursquare and Foursquare (New York), respectively (see Table 6.2). These results thus confirm the previous findings from the analysis employing the precision metric. For the Brightkite and Gowalla datasets, Greedy and k-NN are always the best group formation approaches regardless the pairwise user-item relevance function used. It is however worth reminding that the best results are obtained when PLM instead of PAV is used, with improvements in recall for the k-NN method of up to 11% for the Brightkite dataset, and of up to 6% for the Gowalla one (Table 6.2). The relatively high values of precision and recall achieved by our solutions demonstrate that they are indeed able to suggest meaningful and relevant groups of friends with whom to enjoy a given venue.

6.6.2 Efficiency

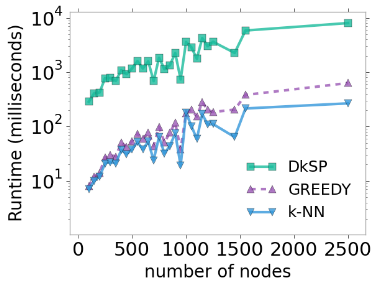
In this section we report the results of an experimental evaluation of the computational efficiency of the Greedy, k-NN, and *DkSP* algorithms. This evaluation has the purpose of showing their applicability to real-world applications. The analysis does not consider k-Top because this baseline does not exploit the user-item ego network to form groups.

We analyze the efficiency of the approaches by means of two different sets of experiments: (i) by varying the size k of the groups; (ii) by varying the number of nodes in the user-item ego network for a fixed value of k ($k = 100$). Figure 6.10a reports the results of the first analysis. The Figure plots the average time needed by the three algorithms to compute a solution as a function of the value of k. *DkSP* exhibits acceptable execution times with an average runtime lower than 5 seconds for values of k up to 200. *DkSP* uses three procedures to compute its solution and this process affects its efficiency. On the other hand, Greedy and k-NN shows very good executions times as they need less than one second to compute their solution with $k = 300$.

In the second analysis, we investigate how the size of the user-item ego network affects the average runtime of the algorithms. Intuitively, the larger the network is, more nodes and edges have to be explored by the algorithms when computing the solution. For conducting these tests we set $k = 100$ and we vary the number of nodes composing the user-item ego network. Results in terms of average runtime are reported in Figure 6.10b. When the ego network is composed of $1,000$ nodes, all algorithms shows an average runtime lower than 1 second. Moreover, when the focal users have a larger number of friends, Greedy and k-NN result still very efficient in computing the groups. Their average runtime is in fact below 1 second when dealing with ego networks of up to $2,500$ nodes. This is not true for *DkSP*. Its performance get worse mostly due to the time needed to compute three procedures over the same user-item ego network.



(a)



(b)

Figure 6.10: Execution time for the textscGreedy, k-NN, and *DkSP* algorithms as a function of the size k of the groups (a), and of the size of the User-Item Ego Networks for a fixed value of k ($k = 100$). For each tests the results plotted have been averaged over 5 runs.

6.7 Discussion

Finding the best group of companions with whom to visit an attraction or travel to a tourist destination is the motivation that inspired our work in proposing a novel recommendation task suggesting the best group of friends with whom to enjoy a specific item. In this chapter we introduced a formalization of our user-item group formation problem modeled as an instance of the k-densest subgraph problem over the user-item ego network. We presented two solutions embedded into GroupFinder, a modular framework proposed as a general solution to easily provide user-item group recommendations in different domains. Evaluations of our proposals were conducted in the mobile recommendations domain by using four different publicly available location-based social networking datasets. The results of extensive experiments showed that the proposed solutions outperform the baselines in effectively finding groups of friends who can jointly appreciate a suggested venue (e.g. restaurant, cinema, etc).

Chapter 7

Conclusions and Future Works

7.1 Conclusions

The popularization of social networking services and the advance of smartphones, in particular GPS-enabled devices, have favored the capture of a huge amount of data generated by millions of users during their daily activities. Access to this information on the Web is extremely important to find out collective patterns of the user behaviors and opinion in an implicit or explicitly way, characterizing the *wisdom-of-the-crowd*. Accordingly, many opportunities can occur to take advantage of the wisdom-of-the-crowd as a crucial dimension to better understand users' behavior, in particular from mobility data, in order to favor the creation of new services and application to enrich the society as a whole.

In this thesis, we presented noteworthy proposals; to study mobility data from the perspective of points of interest; to take the wisdom-of-the-crowd as a next level to support users (e.g. tourists) visiting a new city; to recommend groups of users according to their friendship relationships and their preferences towards a set of items from an item-driven framework in the context of recommender systems.

First, we presented a study about mobility data collected, from GPS-enabled vehicles, from the perspective of points of interest (PoI), in contrast to the works where the focus is on the users, in order to answer the research question **RQ1** described in Chapter 1: *Can we study urban mobility on a global scale from the perspective of places, instead of users?* We applied techniques from complex network fields to identify relationships between PoIs based on the displacement of the users *summarized* in the PoI networks. We proposed a classification for PoIs to identify important features regarding the popularity of the PoI, the amount of time the users spend at the PoI, and the distance traveled by the users from one PoI to another. Later, we applied a community discovery algorithm to the PoI networks in order to mine groups of PoIs that are closed related to each other. Instead of spatial proximity, user movements were observed. This study was conducted in three GPS datasets from three Italian cities demonstrating the utility of the proposal and the interesting findings from the data.

Based on the wisdom-of-the-crowd, we presented TripBuilder, an unsupervised framework that combines information from the social networking service Flickr and Wikipedia PoIs with aim of generating a rich knowledge base encompassing tourists' behaviors during their visits to a given city. This work relates to the research question **RQ2**: *Can we take advantage of the data provided by millions of users, also called wisdom-of-the-crowd, to support users (e.g. tourists) in planning their vacations to a new destination?* In particular, our framework exploits Flickr photos and Wikipedia PoIs to reconstruct the tourists' trajectories in the city in order to benefit other tourists visiting the same city based on the individual user's preferences in terms of PoI categories and time budget. Some properties of the PoIs and trajectories are mined from the data, the wisdom-of-the-crowd, such as the categories and the estimated time needed to visit each point of interest. The problems TripCover and TrajSP were formulated as a basis for creating sightseeing tours, where solutions to approach the problems have been introduced. We experimented our framework with datasets of three tourist Italian cities by conducting an extensive experimental evaluation to demonstrate the effective and efficiency of our proposal.

Later, we described the TripBuilder platform that encompasses the required capabilities to create personalized sightseeing tours in a city. We gave details of the system and its main components as well as the major functionalities regarding the tour creation, tour exploration (e.g. time to visit each PoI, photos), and the possibility to share the created tours using social networking services. In addition, we presented our architecture designed to scale up TripBuilder to a worldwide level by exploiting open-sourced Big Data tools for distributed storage, batch processing and stream processing of Flickr photos, Wikipedia PoIs and trajectories.

Lastly, we presented a complementary view for group recommendation research through an item-driven group formation approach considering the recommendations of items for the users and social networks according to the basis for the friendship between the users. This work answers the research question **RQ3**: *How can we find out the best groups of users (e.g. friends) who can enjoy a given item together?* We presented our framework GroupFinder which encompasses solutions to approach the User-Item Group Formation (UIGF) problem. The problem is formalized as the k most dense subgraph problem, which allows us to model both user interest in a given item and the social relationship between them in order to identify the best group of users to enjoy a given item. We experimented the proposal by exploiting datasets of check-ins from three location-based social networking services. The results show the relevance of the problem to boost recommender system with UIGF and the effectiveness and efficiency of the proposal solutions compared to strong baselines.

7.2 Future Works

We envision several important future works to support our results and contribute *for novelties in the research tracks.*

Recommendation of personalized sightseeing tours

In this thesis we discussed how semantically enriched trajectories derived from user-generated content from web services like Flickr and Wikipedia offer a solid background for planning personalized sightseeing tours. Therefore, we envision several important advances towards a complete framework that considers wisdom-of-the-crowd to fulfill the tourists' needs in planning their visits to a new city.

Linked data. The increasing availability of Linked Open Data (LOD) sources has brought new opportunities to integrate different data sources so as to semantically enrich trajectories and points of interests. LOD may fulfill the lack of information we typically experience managing trajectories from user-generated data and this additional information will bring to better recommendations combined with the capability of explaining the recommendation itself.

Smart Cities. Nowadays, we see the opportunity of integrating crowd-generated trajectories with smart cities environments, such as the physical sensors used for collecting data for several aspects of a city (pollution, traffic, etc). Similar to the LOD case, here we will have the opportunity to create a huge new potential resource to enrich recommendations to the tourism industry.

Real-time Services. It is crucial to keep the tours up-to-date with the most recent events in the city, like special discounts for museums, restaurants, events, etc. How to deal with it and how to collect this amount of user data to provide real-time support to tourists will be a significant challenge.

Group Recommendation. The fact that people usually do not travel alone highlights the importance of recommending tours for groups of people instead of individuals. The task here is to balance the recommendation to satisfy the distinct preferences inherent to each user in the group. This may be a hard task since other issues may come up: the ability to influence people, the distinct roles of members of the group such as the leader, etc.

Hierarchical Sightseeing Tours. So far we have considered the recommendation of sightseeing tours for a single city. If the tourist is willing to travel around different cities, they would need to use the system to generate the tour for each city separately. To overcome this limitation, the design of a hierarchical approach would bring several benefits to help users to travel around many cities.

Time-aware PoIs and Trajectories. Another important challenge is related to the temporal dimension of PoIs and trajectories. Some PoIs and trajectories might be influenced by the time of day: most people visit beaches during the day, in addition some sights (e.g. the Colosseum, the Eiffel tower) could appear different in sunlight as compared during the night. Consequently, better personalized tours for the users may be suggested when the temporal importance and relevance of the PoIs and trajectories are taken in consideration.

Tour-based Hotel Selection. During the scheduling of a trip, tourists choose the city, select a hotel and note down the PoIs to visit. Although the POI tour can be generated by TripBuilder, it still lacks the ability to help choose a hotel. The choice of hotel is usually influenced by traditional constraints like price, ratings, etc. However, knowing the location of the sights on the tour will make it easier to identify a hotel in a strategic position.

Personalized Visiting Time. TripBuilder uses the crowd-generated content to infer an approximate visiting time for each PoI and this information can be used by all tourists. However, people usually have preferences and the tourists might wish to spend more time at some preferred places compared to other less interesting attractions. A personalization of visiting time could be very appropriate for tourists who can adapt their tours by splitting their time allowance for specific attractions according to their preferences.

Novelty and Serendipitous Recommendations. The definition of the user-item interest, $\Gamma(\cdot,\cdot)$, relies on the relevance of the item for the user and also on the popularity of the PoI (Definition 10 Chapter 4). We believe that it is important to define a generic function to model the user-item interest in such a way that different functions might be used to capture different perspective. In the current formulation, the popularity of the PoI is considered, but we believe that considering the PoIs in the long-tail of the popularity distribution may bring enormous benefit for the users when using real applications (e.g. Web and mobile) in order to favor novelty and serendipitous recommendations. A possible solution could be to generalize the popularity function $pop(\cdot)$ in such a way as to be able to account for popularity and non-popularity PoIs.

Advances in User-Item Group Formation

Based on the results of GroupFinder and the user-item group formation problem, we can highlight some future works that will encourage group formation in the recommender system as an indispensable feature for recommender systems of items that can be enjoyed in group. The objective would be to engage users by offering not only item recommendations, but also group recommendations. We therefore intend to study different formulations of the UI-GF problem by modeling it as a *densest "at most" k-subgraph* problem. This is clearly a more complex formulation that also needs more experiments in real world applications. Other research directions we envisage include extending to lists of recommendations and the applicability to other domains.

Bibliography

[1] ADAMIC, L., LUKOSE, R., PUNIYANI, A., AND HUBERMAN, B. Search in power-law networks. *Physical Review E 64*, 46135 (2001).

[2] ADOMAVICIUS, G., AND TUZHILIN, A. Toward the Next Generation of Recommender Systems: a Survey of the State of the Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering 17*, 6 (2005), 734–749.

[3] AHN, Y.-Y., BAGROW, J. P., AND LEHMANN, S. Link communities reveal multiscale complexity in networks. *Nature 466*, 7307 (2010), 761–764.

[4] AIELLO, W., CHUNG, F., AND LU, L. A random graph model for massive graphs. In *STOC* (2000), ACM, pp. 171–180.

[5] AMER-YAHIA, S., OMIDVAR-TEHRANI, B., ROY, S. B., AND SHABIB, N. Group Recommendation with Temporal Affinities. In *EDBT 2015: Proceedings of the 18th International Conference on Extending Database Technology* (2015), pp. 421–432.

[6] AMER-YAHIA, S., ROY, S. B., CHAWLA, A., DAS, G., AND YU, C. Group Recommendation : Semantics and Efficiency. *Proceedings of the VLDB Endowment 2*, 1 (2009), 754–765.

[7] ANKERST, M., BREUNIG, M., AND KRIEGEL, H. OPTICS: ordering points to identify the clustering structure. *ACM SIGMOD Record* (1999).

[8] ASAHIRO, Y., IWAMA, K., TAMAKI, H., AND TOKUYAMA, T. Greedily Finding a Dense Subgraph. *Journal of Algorithms 34*, 2 (2000), 203–221.

[9] B., L., Z., Z., AND P., Y. A general framework for relation graph clustering. *Knowledge and Information Systems 24* (2010), 393–413.

[10] BAEZA-YATES, R., RIBEIRO-NETO, B., ET AL. *Modern information retrieval*, vol. 463. ACM press New York, 1999.

[11] BALTRUNAS, L., AND RICCI, F. Group Recommendations with Rank Aggregation and. In *Proceedings of the fourth ACM conference on Recommender systems. ACM* (2010), pp. 119–126.

[12] BAO, J., ZHENG, Y., AND MOKBEL, M. F. Location-based and preference-aware recommendation using sparse geo-social networking data. *Proceedings of the 20th International Conference on Advances in Geographic Information Systems - SIGSPATIAL '12* (2012), 199.

[13] Barabási, A.-L., and Albert, R. Emergence of scaling in random networks. *Science 286* (1999).

[14] Basu Roy, S., Amer-Yahia, S., Chawla, A., Das, G., and Yu, C. Space efficiency in group recommendation. *The VLDB Journal 19*, 6 (Dec. 2010), 877–900.

[15] Basu Roy, S., Lakshmanan, L. V., and Liu, R. From group recommendations to group formation. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 1603–1616.

[16] Bell, R., Koren, Y., and Volinsky, C. Modeling relationships at multiple scales to improve accuracy of large recommender systems. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'07 5*, 1 (2007), 95–104.

[17] Benevenuto, F., Rodrigues, T., Cha, M., and Almeida, V. Characterizing user behavior in online social networks. In *Internet Measurement Conference* (2009), pp. 49–62.

[18] Berkhin, P. Bookmark-Coloring Algorithm for Personalized PageRank Computing. *Internet Mathematics 3*, 1 (2006), 41–62.

[19] Berkovsky, S., and Freyne, J. Group-based recipe recommendations: analysis of data aggregation strategies. *Proceedings of the fourth ACM conference on ...* (2010), 111–118.

[20] Berlingerio, M., Calabrese, F., Lorenzo, G. D., Nair, R., Pinelli, F., and Sbodio, M. L. Allaboard: A system for exploring urban mobility and optimizing public transport using cellphone data. In *ECML/PKDD (3)* (2013), H. Blockeel, K. Kersting, S. Nijssen, and F. Zelezný, Eds., vol. 8190 of *Lecture Notes in Computer Science*, Springer, pp. 663–666.

[21] Berlingerio, M., Coscia, M., and Giannotti, F. Finding and characterizing communities in multidimensional networks. In *ASONAM* (2011), IEEE Computer Society, pp. 490–494.

[22] Berlingerio, M., Coscia, M., Giannotti, F., Monreale, A., and Pedreschi, D. Foundations of Multidimensional Network Analysis. *2011 International Conference on Advances in Social Networks Analysis and Mining* (jul 2011), 485–489.

[23] Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent Dirichlet Allocation. *Journal of Machine Learning Research 3* (2003), 993–1022.

[24] Bobadilla, J., Ortega, F., Hernando, A., and Bernal, J. Generalization of recommender systems: Collaborative filtering extended to groups of users and restricted to groups of items. *Expert Systems with Applications 39*, 1 (Jan. 2012), 172–186.

[25] Bobadilla, J., Ortega, F., Hernando, a., and Gutiérrez, a. Recommender systems survey. *Knowledge-Based Systems 46* (July 2013), 109–132.

[26] Bogorny, V., Kuijpers, B., de Macêdo, J. A. F., Moelans, B., and Vaisman, A. A. A model for enriching trajectories with semantic geographical information. In *ACM-GIS* (2007).

[27] Breese, J. S., Heckerman, D., and Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. *Proceedings of the 14th conference on Uncertainty in Artificial Intelligence 461*, 8 (1998), 43–52.

[28] Brilhante, I., Macedo, J., Nardini, F., Perego, R., and Renso, C. Tripbuilder: A tool for recommending sightseeing tours. In *Advances in Information Retrieval*, M. de Rijke, T. Kenter, A. de Vries, C. Zhai, F. de Jong, K. Radinsky, and K. Hofmann, Eds., vol. 8416 of *Lecture Notes in Computer Science*. Springer International Publishing, 2014, pp. 771–774.

[29] Brilhante, I., Macedo, J. A., Nardini, F. M., Perego, R., and Renso, C. Planning sightseeing tours using crowdsensed trajectories. *SIGSPATIAL Special 7*, 1 (May 2015), 59–66.

[30] Brilhante, I. R., Berlingerio, M., Trasarti, R., Renso, C., Macedo, J. A. F. d., and Casanova, M. A. **ComeTogether: Discovering Communities of Places in Mobility Data**. In *Proceedings of the 2012 IEEE 13th International Conference on Mobile Data Management (mdm 2012)* (Washington, DC, USA, 2012), MDM '12, IEEE Computer Society, pp. 268–273.

[31] Brilhante, I. R., de Macêdo, J. A. F., Nardini, F. M., Perego, R., and Renso, C. Scaling up the mining of semantically-enriched trajectories: Tripbuilder at the world level. In *IIR* (2015).

[32] Brilhante, I. R., de Macêdo, J. A. F., Nardini, F. M., Perego, R., and Renso, C. User-item group formation with groupfinder. In *IIR* (2016).

[33] Brilhante, I. R., de Macedo, J. A. F., Renso, C., and Casanova, M. A. Trajectory data analysis using complex networks. In *Proceedings of the 15th Symposium on International Database Engineering &#38; Applications* (New York, NY, USA, 2011), IDEAS '11, ACM, pp. 17–25.

[34] Brilhante, I. R., Macedo, J. A., Nardini, F. M., Perego, R., and Renso, C. Where shall we go today? planning touristic tours with tripbuilder. In *22nd International Conference on Information and Knowledge Management (CIKM)* (October 2013).

[35] Brilhante, I. R., Macedo, J. A., Nardini, F. M., Perego, R., and Renso, C. On planning sightseeing tours with TripBuilder. *Information Processing & Management 51*, 2 (Mar. 2015), 1–15.

[36] Brilhante, I. R., Macedo, J. A., Nardini, F. M., Perego, R., and Renso, C. GroupFinder: an item-driven group formation framework. In *Proceedings of the 2016 IEEE 17th International Conference on Mobile Data Management (MDM 2016)* (Washington, DC, USA, 2016), MDM '16, IEEE Computer Society.

[37] Bringmann, B., Berlingerio, M., Bonchi, F., and Gionis, A. Learning and Predicting the Evolution of Social Networks. In *IEEE Intelligent Systems* (2010), vol. 25, pp. 26–35.

[38] Brown, C., Nicosia, V., Scellato, S., Noulas, A., and Mascolo, C. The importance of being placefriends: discovering location-focused online communities. In *Proceedings of the 2012 ACM workshop on Workshop on online social networks* (New York, NY, USA, 2012), WOSN '12, ACM, pp. 31–36.

[39] Brown, C., Nicosia, V., Scellato, S., Noulas, A., and Mascolo, C. Social and place-focused communities in location-based online social networks. *European Physical Journal B 86*, 6 (2013), 1–11.

[40] Burke, R. Hybrid web recommender systems. *The adaptive web* (2007), 377–408.

[41] Calabrese, F., Colonna, M., Lovisolo, P., Parata, D., and Ratti, C. Real-time urban monitoring using cellular phones: a case-study in rome. *IEEE Transactions on Intelligent Transportation Systems* (2010).

[42] Castro, R. D., and Grossman, J. Famous trails to paul erdös. *Mathematical Intelligencer 21* (1999), 51–63.

[43] Charikar, M. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization* (London, UK, UK, 2000), APPROX '00, Springer-Verlag, pp. 84–95.

[44] Chiara Renso, Stefano Spaccapietra, E. Z. *Mobility Data: Modeling, Management, and Understanding.* Cambridge University Press, 2013.

[45] Cho, E., Myers, S. A., and Leskovec, J. Friendship and mobility: User movement in location-based social networks. In *Proc. SIGKDD'11*, ACM, pp. 1082–1090.

[46] Christensen, I. a., and Schiaffino, S. Entertainment recommender systems for group of users. *Expert Systems with Applications 38*, 11 (may 2011), 14127–14135.

[47] Cohen, R., and Katzir, L. The generalized maximum coverage problem. *Information Processing Letters 108*, 1 (2008), 15–22.

[48] Cook, D., Crandall, A., Singla, G., and Thomas, B. Detection of social interaction in smart spaces. *Cybernetics and Systems 41*, 2 (2010), 90–104.

[49] Cormen, T. H., Leiserson, C. E., and Rivest, R. L. *Introduction to Algorithms , Second Edition*, vol. 7. 2001.

[50] Coscia, M., Giannotti, F., and Pedreschi, D. A classification for community discovery methods in complex networks, 2011.

[51] Coscia, M., Rossetti, G., Giannotti, F., and Pedreschi, D. Demon: a local-first discovery method for overlapping communities. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2012), KDD '12, ACM, pp. 615–623.

[52] Crandall, D. J., Backstrom, L., Huttenlocher, D., and Kleinberg, J. Mapping the world's photos. In *Proceedings of the 18th International Conference on World Wide Web* (New York, NY, USA, 2009), WWW '09, ACM, pp. 761–770.

[53] De Choudhury, M., Feldman, M., Amer-Yahia, S., Golbandi, N., Lempel, R., and Yu, C. Automatic construction of travel itineraries using social breadcrumbs. In *Proc. HT* (2010), ACM, pp. 35–44.

[54] Desrosiers, C., and Karypis, G. *Recommender Systems Handbook.* Springer US, Boston, MA, 2011, ch. A Comprehensive Survey of Neighborhood-based Recommendation Methods, pp. 107–144.

[55] Dibbelt, J., Pajor, T., and Wagner, D. User-Constrained Multi-Modal Route Planning. *Networks 6*, 3 (2012), 10.

[56] Dodge, S., Weibel, R., and Lautenschütz, A.-K. Towards a taxonomy of movement patterns. *Information Visualization 7* (June 2008), 240–252.

[57] Donato, D. Graph structures and algorithms for query-log analysis. In *CiE* (2010), pp. 126–131.

[58] Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. Rank aggregation methods for the Web. *Proceedings of the 10th international conference on World Wide Web* (2001), 613–622.

[59] El Mahrsi, M. K., and Rossi, F. Clustering par optimisation de la modularité pour trajectoires d'objets mobiles. In *Actes des 8èmes journées francophones Mobilité et Ubiquit* (2012), pp. 12–22.

[60] Ester, M., peter Kriegel, H., S, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, pp. 226–231.

[61] Fagin, R., Lotem, A., and Naor, M. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences 66*, 4 (2003), 614–656.

[62] Feige, U., Kortsarz, G., and Peleg, D. The dense k-subgraph problem. *Algorithmica 29* (1999), 2001.

[63] Furletti, B., Cnr, K. I., Cintia, P., Cnr, K.-I., and Spinsanti, L. Inferring human activities from GPS tracks. *Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing*, August 2015 (2013), 5.

[64] Gao, H., Tang, J., Hu, X., and Liu, H. Exploring temporal effects for location recommendation on location-based social networks. *Proceedings of the 7th ACM conference on Recommender systems - RecSys '13* (2013), 93–100.

[65] Garcia, I., Sebastia, L., and Onaindia, E. On the design of individual and group recommender systems for tourism. *Expert Systems with Applications 38*, 6 (jun 2011), 7683–7692.

[66] Gavalas, D., Konstantopoulos, C., Mastakas, K., Pantziou, G., and Vathis, N. Heuristics for the time dependent team orienteering problem: Application to tourist route planning. *Computers and Operations Research 62* (2015), 36–50.

[67] Giannotti, F., Nanni, M., Pedreschi, D., Pinelli, F., Renso, C., Rinzivillo, S., and Trasarti, R. Unveiling the complexity of human mobility by querying and mining massive trajectory data. *Very Large Database 20*, 5 (2011).

[68] Giannotti, F., and Pedreschi, D., Eds. *Mobility, Data Mining and Privacy - Geographic Knowledge Discovery.* Springer, 2008.

[69] Gionis, A., Lappas, T., Pelechrinis, K., and Terzi, E. Customized tour recommendations in urban areas. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining* (New York, NY, USA, 2014), WSDM '14, ACM, pp. 313–322.

[70] Godart, J.-M. Combinatorial optimisation based decision support system for trip planning. In *Information and Communication Technologies in Tourism 1999.* Springer, 1999, pp. 318–327.

[71] Gomez-Rodriguez, M., Leskovec, J., and Krause, A. Inferring networks of diffusion and influence. In *KDD* (2010), pp. 1019–1028.

[72] González, M., Hidalgo, C., and Barabási, A.-L. Understanding individual human mobility patterns. *Nature*, 453 (2008), 479–482.

[73] Grcar, M., Fortuna, B., and Mladenić, D. kNN Versus SVM in the Collaborative Filtering Framework. *Learning* (2005), 5–9.

[74] Hofmann, T. Collaborative filtering via gaussian probabilistic latent semantic analysis. *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval - SIGIR '03*, v (2003), 259.

[75] Hofmann, T. Latent semantic models for collaborative filtering. *ACM Transactions on Information Systems 22*, 1 (2004), 89–115.

[76] Horozov, T., Narasimhan, N., and Vasudevan, V. Using location for personalized POI recommendations in mobile environments. *International Symposium on Applications and the Internet (SAINT'06)* (2006), 6 pp.–129.

[77] Hu, L., Cao, J., Xu, G., Cao, L., Gu, Z., and Cao, W. Deep Modeling of Group Preferences for Group-Based Recommendation. *AAAI Conference on Artificial Intelligence (AAAI 2014)* (2014), 1861–1867.

[78] Huang, Y., and Bian, L. A bayesian network and analytic hierarchy process based personalized recommendations for tourist attractions over the internet. *Expert Systems with Applications 36*, 1 (2009), 933–943.

[79] Jameson, A., and Smyth, B. Recommendation to groups. *The Adaptive Web* (2007), 596–627.

[80] Jeh, G., and Widom, J. Scaling personalized web search. *Proceedings of the twelfth international conference on World Wide Web WWW 03 ofthe12th*, [19] (2003), 271.

[81] Jeong, H., Mason, S. P., Barabasi, A. L., and Oltvai, Z. N. Lethality and centrality in protein networks. *Nature 411*, 6833 (May 2001), 41–42.

[82] Jeong, H., Tombor, B., Albert, R., Oltvai, Z. N., and Barabási, A. L. The large-scale organization of metabolic networks. *Nature 407*, 6804 (October 2000), 651–654.

[83] Koren, Y. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (2008), pp. 426–434.

[84] Koren, Y. Collaborative filtering with temporal dynamics. *Communications of the ACM 53*, 4 (Apr. 2010), 89.

[85] Koren, Y., Bell, R., and Volinsky, C. Matrix Factorization Techniques for Recommender Systems. *Computer 42*, 8 (2009), 42–49.

[86] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer 42*, 8 (Aug. 2009), 30–37.

[87] Langville, A. N., Meyer, C. D., and FernÁndez, P. Google's pagerank and beyond: The science of search engine rankings. *The Mathematical Intelligencer 30*, 1 (2008), 68–69.

[88] Lathia, N., Hailes, S., and Capra, L. Temporal Collaborative Filtering With Adaptive Neighbourhoods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (2009), pp. 796–797.

[89] Lathia, N., Hailes, S., Capra, L., and Amatriain, X. Temporal Diversity in Recommender Systems. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval* (2009), pp. 210–217.

[90] Lee, K. M., Min, B., and Goh, K. I. Towards real-world complexity: an introduction to multiplex networks. *European Physical Journal B 88*, 2 (2015).

[91] Lemire, D., and Maclachlan, A. Slope one predictors for online rating-based collaborative filtering. *CoRR abs/cs/0702144* (2007).

[92] Leskovec, J., Huttenlocher, D., and Kleinberg, J. Predicting positive and negative links in online social networks. In *WWW* (2010), ACM, pp. 641–650.

[93] Levandoski, J. J., Sarwat, M., Eldawy, A., and Mokbel, M. F. LARS: A location-aware recommender system. In *Proceedings - International Conference on Data Engineering* (2012), pp. 450–461.

[94] Leventhal, G. E., Hill, A. L., Nowak, M. A., and Bonhoeffer, S. Evolution and emergence of infectious diseases in theoretical and real-world networks. *Nature communications 6* (2015), 6101.

[95] Likas, A., Vlassis, N., and J. Verbeek, J. The global k-means clustering algorithm. *Pattern Recognition 36*, 2 (2003), 451–461.

[96] Lin, S.-W., and Yu, V. F. A simulated annealing heuristic for the team orienteering problem with time windows. *European Journal of Operational Research 217*, 1 (Feb. 2012), 94–107.

[97] Loe, C. W., and Jensen, H. J. Comparison of communities detection algorithms for multiplex. *Physica A: Statistical Mechanics and its Applications 431* (2015), 29–45.

[98] Lops, P., de Gemmis, M., and Semeraro, G. Content-based recommender systems: State of the art and trends. In *Recommender Systems Handbook*, F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, Eds. Springer US, 2011, pp. 73–105.

[99] Lu, E., Lin, C., and Tseng, V. Trip-mine: An efficient trip planning approach with travel time constraints. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on* (2011), vol. 1, IEEE, pp. 152–161.

[100] Lu, K., Zhou, W., and Wang, X. Social network of the competing crowd. In *Behavior, Economic and Social Computing (BESC), 2014 International Conference on* (Oct 2014), pp. 1–7.

[101] Lucchese, C., Perego, R., Silvestri, F., Vahabi, H., and Venturini, R. How random walks can help tourism. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2012), vol. 7224 LNCS, pp. 195–206.

[102] Matai, R., Mittal, M. L., and Singh, S. *Traveling salesman problem: An overview of applications, formulations, and solution approaches.* Prof. Donald Davendra(Ed.), InTech, 2010.

[103] McLaughlin, M. R., and Herlocker, J. L. A collaborative filtering algorithm and evaluation metric that accurately model the user experience. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2004), SIGIR '04, ACM, pp. 329–336.

[104] Michel, F. How many public photos are uploaded to flickr every day, month, year? `https://www.flickr.com/photos/franckmichel/6855169886/in/dateposted/`, 2014. Last access in 2016-03-01.

[105] Nanni, M., and Pedreschi, D. Time-focused clustering of trajectories of moving objects. *Journal of Intelligent Information Systems 27*, 3 (Nov. 2006), 267–289.

[106] Nanni, M., Pedreschi, D., Pinelli, F., Cnr, I., Moruzzi, V. G., Giannotti, F., Nanni, M., Pinelli, F., Pedreschi, D., Pinelli, F., Cnr, I., and Moruzzi, V. G. Trajectory pattern mining. *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '07* (2007), 330–339.

[107] Narasimhamurthy, A., Greene, D., Hurley, N., and Cunningham, P. Partitioning large networks without breaking communities. *Knowl Inf Syst 25* (2010), 345–369.

[108] Newman, M. E. J. Spread of epidemic disease on networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics 66*, 1 (2002).

[109] Newman, M. E. J. The Structure and Function of Complex Networks. *SIAM Review 45*, 2 (2003), 167.

[110] Newman, M. E. J. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences of the United States of America 103*, 23 (jun 2006), 8577–82.

[111] Newman, M. E. J., Ed. *Networks: An Introduction.* Oxford University Press, 2010.

[112] Noulas, A., Scellato, S., Lathia, N., and Mascolo, C. A random walk around the city: New venue recommendation in location-based social networks. In *Proceedings - 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust and 2012 ASE/IEEE International Conference on Social Computing, SocialCom/PASSAT 2012* (2012), pp. 144–153.

[113] Nowell, D., and Kleinberg, J. The link prediction problem for social networks. In *CIKM '03* (2003), ACM, pp. 556–559.

[114] O'connor, M., Cosley, D., Konstan, J., and Riedl, J. PolyLens: a recommender system for groups of users. *ECSCW 2001* (2001).

[115] Ortega, F., Bobadilla, J., Hernando, A., and Gutiérrez, A. Incorporating group recommendations to recommender systems: Alternatives and performance. *Information Processing and Management 49* (2013), 895–901.

[116] O'connor, M., Cosley, D., Konstan, J. A., and Riedl, J. Polylens: a recommender system for groups of users. In *ECSCW 2001* (2001), Springer, pp. 199–218.

[117] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank Citation Ranking. *World Wide Web Internet And Web Information Systems 54*, 1999-66 (1998), 1–17.

[118] Parent, C., Spaccapietra, S., Renso, C., Andrienko, G., Andrienko, N., Bogorny, V., Damiani, M. L., Gkoulalas-Divanis, A., Macedo, J., Pelekis, N., Theodoridis, Y., and Yan, Z. Semantic trajectories modeling and analysis. *ACM Computing Surveys 45*, 4 (2013), 42:1–42:32.

[119] Pastor-Satorras, R., Castellano, C., Van Mieghem, P., and Vespignani, A. Epidemic processes in complex networks. *Rev. Mod. Phys. 87* (Aug 2015), 925–979.

[120] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work* (New York, NY, USA, 1994), CSCW '94, ACM, pp. 175–186.

[121] Ricci, F., Rokach, L., and Shapira, B. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook*. 2011, pp. 1–35.

[122] Rocha, J. A. M. R., Times, V. C., Oliveira, G., Alvares, L. O., and Bogorny, V. Db-smot: A direction-based spatio-temporal clustering method. In *IEEE Conf. of Intelligent Systems* (2010).

[123] Salton, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[124] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web* (New York, NY, USA, 2001), WWW '01, ACM, pp. 285–295.

[125] Sarwat, M., Levandoski, J. J., Eldawy, A., and Mokbel, M. F. LARS*: An efficient and scalable location-aware recommender system. *IEEE Transactions on Knowledge and Data Engineering 26*, 6 (2014), 1384–1399.

[126] Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. Methods and metrics for cold-start recommendations. *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval SIGIR 02 46*, Sigir (2002), 253–260.

[127] Shang, S., Ding, R., Yuan, B., Xie, K., Zheng, K., and Kalnis, P. User oriented trajectory search for trip recommendation. In *Proc. EDBT* (2012), ACM, pp. 156–167.

[128] Shang, S., Hui, P., Kulkarni, S. R., and Cuff, P. W. Wisdom of the crowd: Incorporating social influence in recommendation models. In *Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on* (Dec 2011), pp. 835–840.

[129] Souam, F., Aïtelhadj, A., and Baba-Ali, R. Dual modularity optimization for detecting overlapping communities in bipartite networks. *Knowledge and Information Systems online first* (2013).

[130] Souffriau, W., Vansteenwegen, P., Vertommen, J., Berghe, G., and Van Oudheusden, D. A personalized tourist trip design algorithm for mobile tourist guides. *Applied Artificial Intelligence 22*, 10 (2008), 964–985.

[131] Sozio, M., and Gionis, a. The Community-search Problem and How to Plan a Successful Cocktail Party. *Proc. SIGKDD* (2010), 939–948.

[132] Spaccapietra, S., Parent, C., Damiani, M. L., de Macêdo, J. A. F., Porto, F., and Vangenot, C. A conceptual view on trajectories. *Data Knowl. Eng. 65*, 1 (2008), 126–146.

[133] Su, X., and Khoshgoftaar, T. M. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence 2009*, Section 3 (2009), 1–19.

[134] Surowiecki, J. *The Wisdom of Crowds: Why the Many are Smarter Than the Few and how Collective Wisdom Shapes Business, Economies, Societies, and Nations.* Doubleday, 2004.

[135] Trasarti, R., Rinzivillo, S., Pinelli, F., Nanni, M., Monreale, A., Renso, C., Pedreschi, D., and Giannotti, F. Exploring Real Mobility Data with M-Atlas. *Matrix* (2010), 624–627.

[136] Ungar, L. H., and Foster, D. P. Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems* (1998), vol. 1, pp. 114–129.

[137] Vansteenwegen, P., and Souffriau, W. Trip planning functionalities: state of the art and future. *Information Technology & Tourism 12*, 4 (2010), 305–315.

[138] Vansteenwegen, P., Souffriau, W., Berghe, G., and Oudheusden, D. The city trip planner: an expert system for tourists. *Expert Systems with Applications 38*, 6 (2011), 6540–6546.

[139] Vansteenwegen, P., Souffriau, W., Berghe, G. V., and Oudheusden, D. V. A guided local search metaheuristic for the team orienteering problem. *European Journal of Operational Research 196*, 1 (jul 2009), 118–127.

[140] Vansteenwegen, P., Souffriau, W., and Oudheusden, D. V. The orienteering problem: A survey. *European Journal of Operational Research 209*, 1 (2011), 1–10.

[141] Vansteenwegen, P., Souffriau, W., Vanden Berghe, G., and Van Oudheusden, D. Iterated local search for the team orienteering problem with time windows. *Computers & Operations Research 36*, 12 (dec 2009), 3281–3290.

[142] Vansteenwegen, P., and Van Oudheusden, D. The mobile tourist guide: an or opportunity. *OR Insight 20*, 3 (2007), 21–27.

[143] Wachowicz, M., Ong, R., Renso, C., and Nanni, M. Finding moving flock patterns among pedestrians through collective coherence. *IJGIS 25*, 11 (2011).

[144] Wang, H., Terrovitis, M., and Mamoulis, N. Location recommendation in location-based social networks using user check-in data. *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2013), 374–383.

[145] Wang, P., González, M., Hidalgo, C., and A.-L.Barabási. Understanding the spreading patterns of mobile phones viruses. *Science*, 324 (2009), 1071–1076.

[146] Xiao, J. Clustering spatial data for join operations using match-based partition. In *CIMCA/IAWTIC* (2005).

[147] Xiao, X., Zheng, Y., Luo, Q., and Xie, X. Finding similar users using category-based location history. *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems - GIS '10*, 49 (2010), 442.

[148] Yan, X., and Han, J. gSpan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2002. Proceedings. 2002 IEEE International Conference on* (2002), ICDM '02, IEEE, pp. 721–724.

[149] Yang, J., and Leskovec, J. Modeling information diffusion in implicit networks. In *ICDM* (2010), pp. 599–608.

[150] Ye, M., Yin, P., Lee, W.-C., and Lee, D.-L. Exploiting geographical influence for collaborative point-of-interest recommendation. *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information* (2011), 325–334.

[151] Yi, S. K. M., Steyvers, M., Lee, M. D., and Dry, M. Wisdom of Crowds in Minimum Spanning Tree Problems. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society* (2010).

[152] Yi, S. K. M., Steyvers, M., Lee, M. D., and Dry, M. J. The wisdom of the crowd in combinatorial problems. *Cognitive Science 36*, 3 (2012), 452–470.

[153] Yin, H., Cui, B., Sun, Y., Hu, Z., and Chen, L. LCARS: A Spatial Item Recommender System. *ACM Trans. Inf. Syst. 32*, 3 (2014), 11:1–11:37.

[154] Yin, H., Sun, Y., Cui, B., and Chen, L. LCARS : A Location-Content-Aware Recommender System. *Lecture Notes in Computer Science 3290* (2013), 492–508.

[155] Yoon, H., Zheng, Y., Xie, X., and Woo, W. Smart itinerary recommendation based on user-generated gps trajectories. *Ubiquitous Intelligence and Computing* (2010), 19–34.

[156] Yoon, H., Zheng, Y., Xie, X., and Woo, W. Social itinerary recommendation from user-generated digital trails. *Personal and Ubiquitous Computing 16*, 5 (2012), 469–484.

[157] Yu, Y., and Chen, X. A Survey of Point-of-Interest Recommendation in Location-Based Social Networks. *AAAI Workshops - Trajectory-Based Behavior Analytics* (2015), 53–60.

[158] Yu, Z., Zhou, X., Hao, Y., and Gu, J. TV Program Recommendation for Multiple Viewers Based on user Profile Merging. *User Modeling and User-Adapted Interaction 16*, 1 (June 2006), 63–82.

[159] Zhana Kuncheva, and Giovanni Montana. Community detection in multiplex networks using locally adaptive random walks. *ArXiv* (2015).

[160] Zheng, V. W., Zheng, Y., Xie, X., and Yang, Q. Towards mobile intelligence: Learning from GPS history data for collaborative recommendation. *Artificial Intelligence 184-185* (2012), 17–37.

[161] Zheng, V. W., Zheng, Y., Xie, X., and Yang†, Q. Collaborative location and activity recommendations with gps history data. *International World Wide Web Conference* (2010).

[162] Zheng, Y., Chen, Y., Xie, X., and Ma, W.-Y. GeoLife2.0: A Location-Based Social Networking Service. *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, 49 (2009), 357–358.

[163] Zheng, Y., Xie, X., and Ma, W.-Y. GeoLife: A Collaborative Social Networking Service among User, location and trajectory. *IEEE bullettin 33*, 2 (2010).

[164] Zheng, Y., Zhang, L., Xie, X., and Ma, W.-Y. Mining interesting locations and travel sequences from GPS trajectories. *Proceedings of the 18th international conference on World wide web - WWW '09*, 49 (2009), 791.

[165] Zhou, K., Yang, S.-h., and Zha, H. Functional Matrix Factorizations for Cold-Start Recommendation. *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval* (2011), 315–324.