A System for Multi-Passenger Urban Ridesharing Recommendations with Ordered Multiple Stops

Eleonora D'Andrea [a], Beatrice Lazzerini [a], and Francesco Marcelloni [a]

[a] Dipartimento di Ingegneria dell'Informazione, University of Pisa, Largo Lucio Lazzarino 1, 56122 Pisa, Italy

{eleonora.dandrea@for.unipi.it, beatrice.lazzerini@unipi.it, francesco.marcelloni@unipi.it}

Corresponding author:

Dr. Eleonora D'Andrea
Tel.: +39 050 2217467
Fax: +39 050 2217600
Email: eleonora.dandrea@for.unipi.it
Address: Dipartimento di Ingegneria dell'Informazione, University of Pisa, Largo Lucio Lazzarino 1, 56122 Pisa, Italy

Abstract

Traffic and air pollution caused by the increasing number of cars have become important issues in nowadays cities. A possible solution is to employ recommender systems for efficient ridesharing among users. These systems, however, typically do not allow specifying ordered stops, thus preventing a large amount of possible users from exploiting ridesharing, e.g., parents leaving kids at school while going to work. Indeed, if a parent desired to share a ride, he/she would need to indicate the following constraint in the path: the stop at school should precede the stop at work. In this paper, we propose a ridesharing recommender, which allows each user to specify an ordered list of stops and suggests efficient ride matches. The ride-matching criterion is based on a dissimilarity between the driver's path and the *shared path*, computed as the shortest path on a Directed Acyclic Graph with ordering constraints between the stops defined in the single paths. The *dissimilarity value* is the detour requested to the driver to visit also the stops of the paths involved in the ride-share, respecting the visiting order of the stops within each path. Results are presented on a case study involving the city of Pisa.

Keywords: *ridesharing recommender, shortest path, sustainable transport, sequential ordering problem, directed acyclic graph, green vehicle routing problem.*

1. Introduction

Recently, the use of Intelligent Transportation Systems for the management of different mobility aspects of today's *smart cities* has spread to overcome the problems caused by the increasing number of private cars moving in the city. In fact, the huge number of private cars causes, on the one hand, greenhouse gas emissions and air pollution, and, on the other hand, traffic congestions, bottlenecks, incidents, and lack of parking spaces. Road transport contributes for about one-fifth of the total emissions of carbon dioxide (CO_2) in the European Union (EU), with private cars alone responsible for about 12% of CO_2 emissions [1]. About this, the EU climate target consists in reducing CO_2 emissions by 20% by 2020, compared to 1990 [1].

Even though city administrations direct efforts toward alternatives to private cars (e.g., by promoting the use of public transports, bikes, electric cars, etc.), these remain the users' favorite transportation mode, as they allow the maximum flexibility (e.g., pick-up and drop-off at desired time and place) and comfort (e.g., privacy, security). Indeed, private cars account for around 83.2% of EU's inland passengers transport, and Italy is the second EU country for motorization rate, over 600 cars per 1000 inhabitants [2]. On the contrary, public transports do not offer a door-to-door service [3], and a flexible scheduling timetable, while bikes are reluctantly used in the case of bad weather conditions. With an average car occupancy in Europe of about 1.5 passengers (according to the ridesharing service *BlaBlaCar* [4]), city administrations are also introducing High-Occupancy Vehicle (HOV) lanes/areas to encourage carpooling, and to reduce the number of single-occupancy vehicles.

This paper focuses on an approach for diminishing the number of vehicles moving in the city. Our proposal may help reduce energy consumption and emission, thus complying with the objective of the so-called Green Vehicle Routing Problems (GVRP), which aim to design effective routes to meet environmental and economic concerns [5]. A possible answer to the problem we deal with can be found in (peer-to-peer) ridesharing or carpooling services, exploiting a form of collaboration between users. Ridesharing is the practice according to which at least two users (a driver and a passenger) share a portion of a trip using the same vehicle [6]. Several online ridesharing services have spread in recent years as an economical and easy-to-use form of collaborative transportation system among users, aimed at reducing the main problems in today's cities, i.e., traffic and pollution. *BlaBlaCar* [4] is a ridesharing peer-to-peer platform for long-distance trips. *Uber* [7] is well-known for several services in addition to the classic Uber taxi service. *UberPop* is the Uber peer-to-peer ridesharing service. *UberPool* is an extension of the classic Uber ridesharing service with dedicated driver, in which a pool of up to four users share the classic Uber ride. *Flinc* [8] allows the sharing of both regular and occasional trips, by exploiting a social network for trust-based rides. *RideshareOnline* [9] mainly focuses on multi-modal ridesharing services for commuters. Such services provide direct economic benefits for users, e.g., money savings in fuel, tolls, parking fees, while societal and environmental benefits for the city are less traffic, less pollution, and less need for parking lots [10].

In the literature, different ridesharing systems have been proposed. They differ from each other in some features such as the matching criterion between rides or between users, and the kind of trip. Regarding the matching criterion, rides can be matched taking into account: i) origin and destination points of the trips, ii) pick-up/drop-off points of users, or designed meeting points along the trip, iii) keywords (cities, regions, Points Of Interests (POIs), etc.) associated with the trip, iv) users' needs, habits, and constraints, v) overlapping portions of the trips, and vi) a combination of these criteria. Furthermore, the ride-matching criterion can be aimed, e.g., at minimizing the overall travel distance, the overall travel time, the cost in money of the ride, or maximizing the

number of user requests served. Regarding the kind of trip, we can make a distinction based on: i) the frequency (regular, commute, or occasional); ii) the length (long-distance, e.g., in the non-urban context, or short-distance, e.g., in the urban context); iii) the presence of additional stops in the trip before the destination point (with or without ordering constraints). Other characterizing criteria can be: i) the number of users matched in a ride; ii) the temporal constraints related to the pick-up or drop-off times of passengers; iii) the presence of stay points; iv) the allowance of detours (i.e., the pick-up or drop-off points for the passengers do not have to be exactly on the route of the driver's trip); v) the allowance for multi-hop ridesharing (i.e., passengers' transfers between drivers). Furthermore, we can discriminate between *dynamic* and *static* ridesharing. In the former case, the ride-matching is done on a very short notice on incoming requests from users (users' trips), while in the latter case, it is done offline on a set of fixed trips known in advance [11]. The main challenge in ridesharing systems is to provide an effective recommendation criterion, i.e., an efficient matching between rides (or users), by fulfilling the (often) conflicting objectives of: meeting users' needs and privacy, respecting origin and destination points, or scheduling times, etc. [6]. In fact, when a user chooses a transportation mode from an origin point to a destination point, he/she considers several aspects of the ride: cost, travel time, flexibility, pick-up and drop-off points, privacy, security, etc. Some of the aspects mentioned above are difficult to be directly controlled in public transport or long-distances ridesharing services, where often the constraints for the pick-up/drop-off locations are not flexible for the user.

In this paper, we propose a ridesharing recommendation system to group similar users, with similar mobility needs. Recommendation systems aim to provide users with a list of recommended items (or people) to meet their preferences, based on the collection and processing of user activities or item data [12]. While the majority of existing ridesharing systems deal with planned and long-distance trips [13], we propose a ride-matching criterion for static ridesharing systems in the *urban* context, where short-distance multiple-stops trips are typically found. The several POIs in the city become the desired pick-up or drop-off locations for the users. In this way, the user may visit places, such as school, swimming pool, train station, etc., located along the shared path. In addition, users are allowed to specify their own POIs (e.g., home, work place), and constraints on the visiting order of these POIs of their trips. We wish to point out that we assume that users of the ride need to spend a low amount of time in each POI, e.g., they have to be merely picked-up or dropped-off, or they need to run very short errands (e.g., take a child to school, buy a coffee or a newspaper). On the contrary, we consider the final destination of each trip as a stay point for the corresponding user. Although some papers in the literature propose solutions for ridesharing, to the best of the authors' knowledge, a standard method for determining the best ride-matching criterion does not exist [14]. Thus, the problem is formulated as follows. Each user needs to visit a set of POIs distributed in the city road network according to a given order. The set of POIs contains an origin point, a destination point, and possible intermediate stops. The proposed approach aims to recommend to each user (i.e., the *driver* of the ride) a set of similar users (i.e., *passengers*) with whom to share the ride, based on the geographic characteristics of their paths. The ride-matching criterion is based on the idea of dissimilarity between paths: the *dissimilarity value* is the additional travel distance that needs to be travelled by the driver of the shared ride to visit the points of his/her path and those of the passengers' paths, by respecting the constraints on the visiting orders within each path. The resulting travelling path is called *shared path* and it is optimized based on the total travelled distance. It is found by computing the shortest path on a Directed Acyclic Graph (DAG) representing the POIs of the paths taken into account and the connections between them, i.e., the road

distance between the POIs. Stated in other words, we aim to solve the Sequential Ordering Problem (SOP), first formulated in [15]. SOP is a combinatorial optimization problem, which consists of determining the minimum cost Hamiltonian tour between a source node and a destination node on a directed graph, satisfying a set of ordering constraints between pairs of nodes. We wish to point out that, in the computation of the DAG, the use of travelling times in place of the distances could also be easily adopted. Therefore, users are grouped together to share the ride at the minimal cost, in terms of additional distance to be travelled by the driver. The greater the number of passengers involved in each ride-share, the lower the number of cars moving in the city. Such an approach could be very useful to reduce traffic in the city, as usual/frequent activities of citizens (e.g., grocery shopping, children pick-up and drop-off, theatre/cinema/sport events attendance) often share the same place. For the above reasons, it becomes easy to find people with the same transport needs. Furthermore, the respect of the order of sequences within each path becomes of the utmost importance, e.g., in the case of typical urban paths such as {1. *get out of work,* 2. *pick-up children at school,* 3. *go home*}, where strong ordering constraints are present. A possible real-world context that could benefit from the proposed approach is ridesharing for children. Ridesharing services for children have spread in recent years to support families in driving their kids to/from school or sport activities, and have become extremely useful especially in the case of families with multiple children all engaged in various activities or attending different schools. Such services, e.g., Carpool Kids [16], HopSkipDrive [17], KidsLyft "*For kids and parents who are going places*" [18], Kango [19], *Zum* [20], allow families to reserve safe rides (since parents serve as drivers) and specify single or multiple pick-up and drop-off requests for their children. In the following, we provide an example of a simple early morning scenario involving two users (user α and user β), e.g., having children attending the same school. User α's path is {1. *leave home*, 2. *drop-off child at school*, 3. *pick-up a colleague*, 4. *get to work*}. User's β path is {1. *leave home*, 2. *drop-off child at school*}. In the shared ride, user α, already giving a ride to work to a colleague, could easily make one additional stop to serve the ride request of user β, i.e., to pick-up (at home) and to drop-off (at school) user's β child. Thus, the proposed approach can be employed in existing (children) ridesharing services to find the best ride-share recommendation, i.e., the best matching, and the shortest shared path, by respecting the ordering constraints within each path. The greater the number of common or close POIs (e.g., school, workplace) in the paths, the higher the efficiency of the ride-share.

The paper is organized as follows. Section 2 discusses the related work about trajectory data mining and ridesharing recommenders. Section 3 and Appendix A present the proposed approach for urban ridesharing recommendations. Section 4 discusses the experimental analysis performed. Finally, Section 5 provides the conclusions and the future research.

2. Related work

This section reviews the related work about trajectory data mining and ridesharing recommendation systems.

2.1 Trajectory data mining

In the following, we review some definitions of trajectory/path similarity functions proposed in the literature. Trajectory data mining is a wide research area referring to the extraction of knowledge from the analysis of

traces of moving objects [21]-[22], for several purposes, e.g., traffic detection/prediction [11],[23] discovery of hot spots and frequent routes [24]-[25], identification of similar traces or users [26]-[27], trajectory prediction [28], path routing [29]. In this paper, trajectory data mining is targeted at friend recommendation, and carpooling or ridesharing services. A *trajectory* of a moving object is represented as a series of discrete spatiotemporal points, indicating both the position in the road network, and the corresponding time. A *path* can be considered, instead, as a spatial trajectory [30]. Similarity functions between trajectories typically take into account time, speed, and direction of the object, while in the case of paths, they only take into account the spatial dimension (e.g., the starting node, the ending node, the length of the path, the intermediate nodes, etc.). Similarity functions are used to evaluate the amount of similarity between objects. The class of similarity functions chosen, e.g., distance-based, statistic-based, point-based, depends on the kind of object and context considered. The dissimilarity between objects, represented by a set of numerical attributes, is usually measured with point-based distances, e.g., L_p-norm distance metric (Euclidean distance for $p = 2$). Some drawbacks of this distance metric make it not well-suited for trajectories or paths. The drawbacks include the impossibility of considering trajectories with different lengths, or shifted in time, and the bad management of outliers. The edit distance metric [30]-[31] is defined on two strings as the minimal number of operations needed to transform one string into the other. Variants of this metric have been proposed for their application to paths/trajectories: i) the edit distance on real sequences [32] captures the similarity in shape between two trajectories; ii) the longest common subsequence (LCS) [33] is based on the matching of two sequences of points by stretching and rearranging the elements in time and space. Among statistic-based metrics, the odds metric is used to measure the likelihood that a sequence belongs to a set of sequences based on the underlying statistical distribution of sequences [30]. The inter-cluster distance metric (e.g., the maximum, the minimum) is used to represent the distance between paths represented as clusters of points [30]. The dynamic time warping metric [34] allows measuring the similarity between two temporal sequences, which may vary in time or speed. The main drawback of this measure is its inefficiency for noisy data, but it can be applied to trajectories of different lengths. The similarity between two paths p_1 and p_2, computed according to the perimeter-based similarity metric, corresponds to the perimeter of the region formed by the paths, the shortest path from the starting node of p_1 and the starting node of p_2, and the shortest path from the ending node of p_1 and the ending node of p_2 [30]. This metric is well suited to check the similarity between paths based on the starting and ending points.

The majority of the similarity functions mentioned above are not suited to work with paths or trajectories. In fact, in this case, the function should take into account also the underlying road network/graph, and problems such as graph connectivity and compliance with the order in the sequences. Other similarity metrics handle only standalone trajectories, or are not able to compare paths of different length, or with different sampling frequencies. In this paper, we propose a novel dissimilarity function that tries to overcome the aforementioned problems. The intuition behind this function is to consider as dissimilarity measure the additional road that needs to be travelled by the user of a given path to visit also the POIs of at least another path: if the length of this additional road is small, then the effort requested to the driver for the shared ride is negligible.

2.2 Ridesharing recommendation systems

Hereafter, we recall some ridesharing recommendation systems from the literature. In [13], the authors propose a recommender capable of identifying opportunities for ridesharing. The system collects GPS mobility

data, identifies users' routine behaviors by employing text mining techniques, and finally discovers similarities among rides. He *et al.* [35] propose an intelligent routing scheme for carpooling recommendation. The system extracts frequent routes of users, searches for qualified riders and generates a commonly accepted route, which minimizes the driving distance, the walking distance, and the travel costs. Xiao *et al.* [26] estimate the similarity between users according to the semantic location histories extracted from GPS traces, with the aim of enabling friend and location recommendation. In [33], the authors tackle the ride-matching problem and perform an automatic classification of similar trajectories using the nearest neighbor classifier, and the LCS as similarity function between trajectories, and by allowing stretching in time and translating in space. In [14], the authors identify suitable matches between users based on preferred characteristics (age, gender, smoking preferences, pet restrictions, etc.) and by satisfying constraints such as vehicle occupancy, waiting time to pick-up, number of connections, detour distance. In [36], the authors propose a dynamic ridesharing system for taxis, by employing a shortest path algorithm and a dynamic matching criterion, based on historical data. Each trip is defined in terms of only the origin and destination points, and the constraints about waiting times.

In [37], driver and passengers with similar itineraries are matched by means of an agent based model, that considers both single- and multi-passenger matches. Herbawi and Weber [38] tackle a ride-matching problem of multi-hop trips with detours. They solve a multi-objective optimization problem, taking into account the minimization of the overall travelled distance, the minimization of the travel times, and the maximization of the number of ride matches. In [39], the authors develop a ridesharing framework for matching users with similar mobility patterns. First, they infer users' mobility patterns and social relations, then they provide ride matches based on spatial, temporal and social constraints. In [40], the authors focus on a carpooling application aimed at identifying pairs of users that might share their vehicle during their routine trips. They extract the habitual mobility profiles from user location history, employing trajectory clustering, and a route similarity function. In [41], the authors propose knowledge-based carpooling matches between drivers and passengers based on characteristics of the users' routes, the users' preferences, and the shared rides. Stiglic *et al.* [42] propose an algorithm to match drivers with riders willing to walk to/from meeting points. Multiple-rider matches are obtained in a recursive way, by adding to the match one rider at a time.

The ride-matching criterion proposed in this paper is based on the geographic dissimilarities (distances) of the POIs composing the trips and not merely on the origin-destination matrix, that is, we take into account also multiple-stops trips, i.e., trips containing intermediate stops along them in addition to the destination point. This means that paths belonging to the same region, city, district, or area of a city (as in our case) will be matched according to the distances between the POIs composing the trips. In addition, with the proposed ride-matching criterion, the starting points (or ending points) of the matched paths do not have to be the same or very close in space: they can be anywhere in the city. With respect to the other works in the literature, our approach takes into account a very general situation whose strengths are summarized as follows: i) a driver can give a ride to multiple passengers; ii) passengers' pick-up or drop-off points are exactly on their paths, i.e., no need for the passengers to walk to a common meeting point; iii) users' trips are multiple-stops with ordering constraints; iv) origin and destination of the trips do not have to be common for the users; v) the driver is not dedicated, i.e., the driver visits also his/her own POIs; vi) the urban context is taken into account. This work is an extension of the work in [43], in which only one passenger per ride (in addition to the driver) was taken into account. In the present work, instead, we propose a novel framework for the recommendation of ride-shares with multiple

passengers, by reducing even more the number of private cars moving in the city. To the best of the authors' knowledge, a similar approach does not exist in the literature.

3. The proposed approach for urban ridesharing

In this section, we describe the proposed approach for urban ridesharing recommendation aimed at grouping multiple passengers interested in travelling in the same ride, approximately in the same time interval (e.g., early morning). The aim of the work is to suggest to drivers, travelling in the city by car, a set of users with whom to share the ride, based on the similarity between the driver's path and the passengers' paths. The recommendation also suggests the optimal *shared path* to follow, i.e., the path with the smallest detour distance for the driver. Each user trip is defined in terms of a list of POIs that the user needs to visit in the given order. The set of POIs includes the path's origin and destination points, and the stops along it. The shared path should visit all the POIs of each user involved in the ride-share in the correct order.

Consider a set $\Phi = \{p_1, \dots, p_P\}$ of P paths approximately belonging to the same time interval. Let $p_\alpha = \{a_1^{(\alpha)}, \dots, a_{Q_\alpha}^{(\alpha)}\}$ be a path in Φ defined in terms of Q_α POIs $a_q^{(\alpha)}$, $q = 1, \dots, Q_\alpha$, $Q_\alpha \geq 2$. The problem consists in finding for each path p_α (namely, the driver's path) the optimal group of R_α paths in Φ for an efficient ridesharing recommendation, being R_α the number of passengers accepted by the driver. Stated in other terms, being the user of path p_α the driver of the shared ride, we want to recommend him/her the optimal ride sharing combination, namely the optimal set of R_α passengers. The recommendation is based on the concept of dissimilarity between paths. The dissimilarity is evaluated according to a novel distance-based dissimilarity function. The *dissimilarity value* represents the additional distance to be travelled by the driver of the shared ride to visit also the POIs of one or more passengers' paths. It is computed as the difference between the length of the shared path and the length of the driver's path. To compute the shared path, and thus its length, we exploit a well-known algorithm for computing the shortest path on the edge-weighted DAG whose nodes represent the POIs of the paths.

Obviously, to find the best recommendation for a given driver would require to exhaustively try each possible combination of the driver's path with R_α passengers' paths, resulting in a high computational effort. Indeed, the greater the number of users, the greater the number of possible matches driver-passengers to check. On the other hand, in a given ride-share, it may be useless to take into account paths very far on the map (e.g., belonging to very distant areas of the city). Thus, in order to reduce the computational complexity of the problem, a preliminary phase is performed in which the paths in Φ are organized in clusters based on the dissimilarity criterion explained above. This allows to search for the best recommendation for a given driver among passengers' paths belonging to the same cluster of the driver's path, making the entire process more efficient. The methodology consists of four different phases: 1) *Preprocessing,* 2) *Grouping of paths*, 3) *Analysis of multi-passenger ride matches*, and 4) *Ridesharing recommendation*. The steps of the elaboration are shown in Fig. 1.

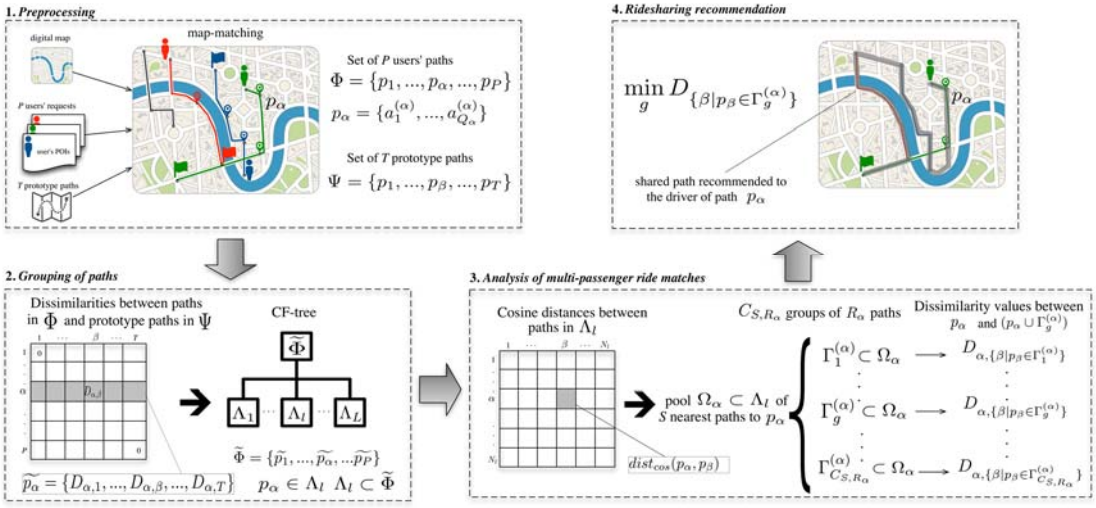


Fig. 1. The elaboration performed on the paths to find the best recommendation for the driver of path p_α.

The first phase includes the representation of the city road network, the collection and elaboration of the ride requests from the users, and the definition of a set Ψ of *T prototype* paths. Prototype paths are used as reference paths for the computation of the similarity between the paths in Φ.

The second phase mainly consists in arranging the paths in Φ in clusters, on the basis of a dissimilarity criterion computed between the paths and the prototype paths defined on the same city road network. Prototype paths can be considered as fake-passenger paths. More in detail, the paths in Φ and the prototype paths in Ψ are analyzed in a pairwise manner, by taking into account ride-shares consisting of two paths: a driver's path in Φ and a fake passenger's path in Ψ. Then, the paths are organized in clusters $\Lambda_l \subset \Phi$, $l = 1, \dots, L$, with L the number of clusters identified, on the basis of a hierarchical tree data structure. The aim of this phase is twofold. On the one hand, we aim to reduce the computational complexity of the following phase, in which the multi-passenger ride matches will be found within each cluster of paths. On the other hand, the use of the prototype paths allows transforming the paths in Φ into objects described in terms of the same number of features, i.e., the T features representing the pairwise dissimilarity values between the paths in Φ and the prototype paths in Ψ.

The third phase is aimed at analyzing multi-passenger ride matches within each cluster of paths $\Lambda_l \subset \Phi$. The analysis is applied in turn on each path p_α in Λ_l as follows. Being p_α the path taken into account, a *pool* Ω_α of S paths near p_α is selected in Λ_l. The pool Ω_α contains the candidate paths to be matched with path p_α. The nearness is computed exploiting the elaboration made in the previous phase. More precisely, being each path described in terms of the T dissimilarity values, the nearness is evaluated according to the cosine distance in $\mathbb{R}^T$ between path p_α and each other path in the same cluster Λ_l. The recommender is supposed to find within the pool the best *set of paths* to recommend to the driver of path p_α for a ride-share. More precisely, being R_α the number of passengers accepted by the driver, a DAG is built for each possible combination of R_α paths in the pool Ω_α, until the best set of paths is found. The quality of the set of paths is checked by computing a *dissimilarity value* between the driver's path p_α and the resulting shared path. Here too, the shared path among $R_\alpha + 1$ paths is computed based on the shortest path algorithm referred to above. The dissimilarity represents the additional travel distance requested to the driver (the user of path p_α) to satisfy also the passenger requests, i.e.,

to visit the POIs (in the correct order) of the R_α paths selected.

Finally, in the last phase, the driver of path p_α is provided with the best ridesharing recommendation. The recommendation consists of the best set of passengers with whom to share the ride, and the optimal shared path to follow. The best set of passengers is determined by applying a ride-matching criterion based on the idea of dissimilarity between paths, expressing the detour for the driver. Hence, this set consists of the passengers who allow the driver the smallest detour distance.

In the following, each phase of the methodology is described in more detail, also providing the pseudo-code for each phase. Please note that we considered the elaboration from a high-level point of view, by identifying and discussing the main subroutines. In addition, Table 1 summarizes the meaning of the main symbols employed in this paper.

3.1 Preprocessing

To represent the city road network and the positioning of the POIs in the city, we exploit the digital map provided by the well-known OpenStreetMap (OSM) [44] framework for digital maps. OSM is an open source, free-license project aimed at collecting geographic data to create freely available maps of the world with free content. A digital map is a graph composed of a set of vertices, defined as GPS positions, and a set of edges, each one defined in terms of length, bearing and endpoint vertices. With this structure, a road is described as the conjunction of consecutive edges (also called segments) identified in correspondence of intersections, changes in bearing of the road, traffic lights, pedestrian crossings, and other relevant points. The use of the digital map of the city will allow computing the real travel distances between different points of the map according to the road network constraints (one-ways, limited traffic zones, etc.).

Each POI of a user trip can be expressed with the name of the place, the complete address, or directly in terms of the GPS position (latitude and longitude). Then, each POI is map-matched on the closest OSM map segment. The resulting *path* is described as a sequence of POIs, corresponding to a sequence of segments on the digital map.

Further, we define a set Ψ of T *prototype* paths. Prototype paths are fake-passenger paths uniformly distributed in the city road network. They are used as a reference to arrange users' paths in groups, based on a dissimilarity criterion. To define the prototype paths in Ψ, we considered the north, south, east, and west areas of the city, and for each area, we identified the paths visiting the most relevant POIs of the city (e.g., hospitals, train/bus stations, airport, malls, schools, supermarkets, etc.). The value of T depends on the desired granularity of the groups of paths, and on the size of the urban context considered. The greater the value of T, the finer the granularity of the representation of the paths in the city road network.

The preprocessing phase is performed by the following two subroutines:

Subroutine 1.1: map-match the P users' paths**:=**

- **given** a digital map of the city
- **given** a set Φ of P users' paths
- **for each** user path p_α **in** Φ, $\alpha = 1, \dots, P$ **do**
  - map-match each POI of p_α on the city map (path p_α has Q_α POIs)
- **end for**
- **return** the set Φ of P users' map-matched paths

Subroutine 1.2: map-match the *T* prototype paths**:=**

- **given** a digital map of the city
- **given** a set Ψ of *T* prototype paths
- **for each** prototype path p_β **in** Ψ, $\beta = 1, \dots, T$ **do**
  - map-match each POI of p_β on the city map (path p_β has Q_β POIs)
- **end for**
- **return** the set Ψ of *T* prototype map-matched paths

3.2 Grouping of paths

This phase focuses on the organization of users' paths in clusters. To this aim, we act as if each prototype path were a passenger path, and we measure the dissimilarity between each path in Φ and each prototype path in Ψ, by computing the shared path for the ride involving two users: the driver of path $p_\alpha \in \Phi$ and the passenger of path $p_\beta \in \Psi$. More precisely, for each path $p_\alpha \in \Phi$, $\alpha = 1, \dots, P$, we define the *dissimilarity value* $D_{\alpha,\beta}$ between p_α and p_β, $p_\beta \in \Psi$, $\beta = 1, \dots, T$, as the detour distance, i.e., the additional distance, with respect to path p_α, to be travelled by user of path p_α (the driver of the shared path) to visit the POIs of paths p_α and p_β, respecting the visiting orders of POIs in both paths. In addition, being the user of path p_α the driver of the shared path, an additional constraint on the visiting order of the POIs is that $a_1^{(\alpha)}$ and $a_{Q_\alpha}^{(\alpha)}$ are the first and the last visited points of the shared path, respectively. More precisely, the shared path should visit the POIs of both paths respecting the order of the POIs in each path, and maintain in the shared path the driver's origin and destination. More formally, $D_{\alpha,\beta}$ is computed as:

$$D_{\alpha,\beta}(p_\alpha, p_\beta) = length(p_\alpha \cup p_\beta) - length(p_\alpha), \quad (1)$$

where $(p_\alpha \cup p_\beta)$ is the shortest *shared path* travelled by the user of path p_α for visiting each point of paths p_α and p_β, preserving the order of the visited POIs in p_α and p_β, $length(p_\alpha \cup p_\beta)$ is the length of the shared path $(p_\alpha \cup p_\beta)$, and $length(p_\alpha)$ is the length of path p_α. The term $length(p_\alpha)$ is computed as the sum of the distances (in meters) on the city road network between consecutive points in path p_α:

$$length(p_\alpha) = \sum_{q=1}^{Q_\alpha - 1} dist\left(a_q^{(\alpha)}, a_{q+1}^{(\alpha)}\right), \quad (2)$$

where $a_q^{(\alpha)}$ and $a_{q+1}^{(\alpha)}$ are two consecutive POIs of path p_α, and $dist(a_q^{(\alpha)}, a_{q+1}^{(\alpha)})$ is the length (in meters) of the shortest path between $a_q^{(\alpha)}$ and $a_{q+1}^{(\alpha)}$ computed on the map by using the Graph Hopper Route Planner [45], thus corresponding to the real travel distance between the POIs $a_q^{(\alpha)}$ and $a_{q+1}^{(\alpha)}$ on the city road network. The routing algorithm employed generates the shortest path according to the classical Dijkstra's algorithm [46] for route planning. Similarly, $length(p_\alpha \cup p_\beta)$ is computed as:

$$length(p_\alpha \cup p_\beta) = \sum_{k=1}^{Q_\alpha + Q_\beta - 1} dist\left(a_k^{(\alpha \cup \beta)}, a_{k+1}^{(\alpha \cup \beta)}\right), \quad (3)$$

where $a_k^{(\alpha \cup \beta)}$ and $a_{k+1}^{(\alpha \cup \beta)}$ are two consecutive POIs of the shared path $(p_\alpha \cup p_\beta)$.

The dissimilarity function in (1) respects the coincidence axiom, that is, $D_{\alpha,\beta}(p_\alpha, p_\beta) = 0$ if and only if $p_\alpha = p_\beta$,

but not the symmetry property, that is, $D_{\alpha,\beta} \neq D_{\beta,\alpha}$. With the definition in (1), the closer p_α and p_β (according to the previously defined distance) to each other in the road network, the lower the corresponding dissimilarity value. Of course, if p_α and p_β are very far from each other, the dissimilarity value will be higher.

Table 1. List of symbols

Symbol	Meaning
Φ	Set of users' paths
P	Number of paths in Φ, $\lvert\Phi\rvert = P$
Ψ	Set of prototype paths
T	Number of prototype paths in Ψ, $\lvert\Psi\rvert = T$
Λ_l	l-th cluster of paths, $l = 1, \dots, L$
L	Number of clusters in Φ
N_l	Number of paths in Λ_l, $\lvert\Lambda_l\rvert = N_l$
p_α	α-th path, $p_\alpha = \{a_1^{(\alpha)}, \dots, a_{Q_\alpha}^{(\alpha)}\}$, $\alpha = 1, \dots, P$
$a_q^{(\alpha)}$	q-th POI in path p_α, $q = 1, \dots, Q_\alpha$
Q_α	Number of POIs in path p_α
$D_{\alpha,\beta}$	Dissimilarity value between driver's path p_α and passenger's path p_β, $\beta = 1, \dots, P$
$\widetilde{p_\alpha}$	α-th path represented as the object, $\widetilde{p_\alpha} = \{D_{\alpha,1}, \dots, D_{\alpha,\beta}, \dots, D_{\alpha,T}\}$.
$(p_\alpha \cup p_\beta)$	Shared path built on the driver's path p_α and the passenger's path p_β
$DAG_{\alpha,\beta}$	DAG for the paths p_α (driver's path) and p_β (passenger's path)
Ω_α	Pool of similar paths to p_α
S	Number of paths in the pool Ω_α
$length(p_\alpha \cup p_\beta)$	Length (in meters) of the shared path $(p_\alpha \cup p_\beta)$
$length(p_\alpha)$	Length (in meters) of path p_α
$dist(a_q^{(\alpha)}, a_{q+1}^{(\alpha)})$	Length (in meters) of the shortest path between two POIs $a_q^{(\alpha)}$ and $a_{q+1}^{(\alpha)}$
$dist_{cos}(p_\alpha, p_\beta)$	Cosine distance between paths p_α and p_β, represented by $\widetilde{p_\alpha}$ and $\widetilde{p_\beta}$, respectively
C_{S,R_α}	Number of possible sets of passengers' paths for the driver of path p_α
R_α	Number of passengers accepted by the driver of path p_α
$\Gamma_g^{(\alpha)}$	g-th set of passengers' paths for the driver of path p_α, $g = 1, \dots, C_{S,R_\alpha}$, $\lvert\Gamma_g^{(\alpha)}\rvert = R_\alpha$
$length(p_\alpha \cup \Gamma_g^{(\alpha)})$	Length (in meters) of the shared path $(p_\alpha \cup \Gamma_g^{(\alpha)})$
$D_{\alpha,\{\beta \mid p_\beta \in \Gamma_g^{(\alpha)}\}}$	Dissimilarity value between the driver's path p_α and the set $\Gamma_g^{(\alpha)}$
sp_α	Optimal shared path recommended to the driver of path p_α
$DAG_{\alpha,\{\beta \mid p_\beta \in \Gamma_g^{(\alpha)}\}}$	DAG for the set of paths $\{p_\alpha, \Gamma_g^{(\alpha)}\}$
$X_{i,j}$	Node of the DAG associated with a driver's POI in the single-passenger ridesharing
$Y_{i,j}$	Node of the DAG associated with a passenger's POI in the single-passenger ridesharing
$X_{i,j_1,\dots,j_r,\dots,j_{R_\alpha}}$	Node of the DAG associated with a driver's POI in the multi-passenger ridesharing, $r = 1, \dots, R_\alpha$
$Y^r{}_{i,j_1,\dots,j_r,\dots,j_{R_\alpha}}$	Node of the DAG associated with a passenger's POI in the multi-passenger ridesharing, $r = 1, \dots, R_\alpha$
(i,j)	Indexes indicating the state of progress achieved in completing the shared path: i refers to the driver's path, and j to the passenger's path
$(i, j_1, \dots, j_r, \dots, j_{R_x})$	Indexes indicating the state of progress achieved in completing the shared path: i refers to the driver's path, and j_r, $r = 1, \dots, R_\alpha$, refers to the r-th passenger's path

An example of this reasoning is shown in Fig. 2. The figure shows three paths: paths p_α and p_β are close to each other, while path p_γ is far from both p_α and p_β. The dissimilarity values $D_{\alpha,\beta}(p_\alpha, p_\beta)$ and $D_{\beta,\alpha}(p_\beta, p_\alpha)$ are lower than the dissimilarity values computed taking into account path p_γ. This results in a longer detour (visible also from the map) for the designated driver, when path p_γ is matched with path p_α or with path p_β.

The computation of $length(p_\alpha \cup p_\beta)$ is not trivial since we have to determine the *shortest path* travelled by the user of path p_α for: i) visiting each point of paths p_α and p_β, ii) preserving the order of the visited POIs in p_α and p_β, and iii) starting and ending in the driver's first and last POIs. To this aim, we solve the problem as a SOP, by finding the shortest path between a source node (the driver's first POI) and a destination node (the driver's last POI) on a DAG, satisfying a set of ordering constraints between pairs of nodes (visiting order of POIs on the two paths). Thus, being the user of path p_α the driver of the shared path, we build an edge-weighted DAG (called from now on $DAG_{\alpha,\beta}$) and we compute the shortest path $(p_\alpha \cup p_\beta)$, using the algorithm described in [47]-[48].

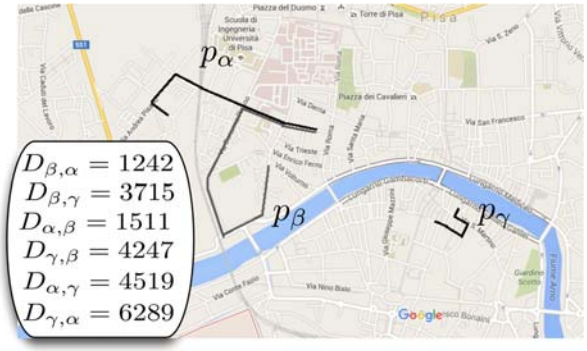


Fig. 2. Three paths on the city map of Pisa, Italy, and the corresponding pairwise dissimilarity values computed in meters. The map is provided by Google® Maps.

Hereafter we describe in detail the architecture and the semantics of $DAG_{\alpha,\beta}$. $DAG_{\alpha,\beta}$ has two kinds of nodes: $X_{i,j}$ and $Y_{i,j}$. The nodes of type $X_{i,j}$ are associated with the POIs of the user acting as driver of the shared path, i.e., POIs $a_i^{(\alpha)}$ of path p_α, while the nodes of type $Y_{i,j}$ are associated with the user acting as passenger, i.e., POIs $a_j^{(\beta)}$ of path p_β. The indices i and j indicate the state of progress achieved in completing the shared path, on the driver's path, and on the passenger's path, respectively. More in detail, staying in node $X_{i,j}$ of the DAG means that the shared path has just reached the POI $a_i^{(\alpha)}$ of the driver's path p_α and has already visited all the previous POIs $a_1^{(\alpha)}, \dots, a_{i-1}^{(\alpha)}$ of path p_α and the POIs $a_1^{(\beta)}, \dots, a_j^{(\beta)}$ of path p_β. Similarly, staying in node $Y_{i,j}$ means that the shared path has just reached the POI $a_j^{(\beta)}$ of the passenger's path p_β and has already visited all the previous POIs $a_1^{(\beta)}, \dots, a_{j-1}^{(\beta)}$ of path p_β and the POIs $a_1^{(\alpha)}, \dots, a_i^{(\alpha)}$ of path p_α.

The DAG has four possible types of edge. The type of edge depends on the extreme nodes of the edge, and the weight associated with each edge is the minimal distance computed on the road network between the POIs associated with the extreme nodes. The four types of edges are:

i) edge $X_{i,j} \rightarrow X_{i+1,j}$ is defined between two POIs of the driver's path, and has weight $dist(a_i^{(\alpha)}, a_{i+1}^{(\alpha)})$;

ii) edge $X_{i,j} \rightarrow Y_{i,j+1}$ is defined between one POI of the driver's path and one of the passenger's path, and has weight $dist(a_i^{(\alpha)}, a_{j+1}^{(\beta)})$;

iii) edge $Y_{i,j} \rightarrow X_{i+1,j}$ is defined between one POI of the passenger's path and one of the driver's path, and has weight $dist(a_j^{(\beta)}, a_{i+1}^{(\alpha)})$;

iv) edge $Y_{i,j} \rightarrow Y_{i,j+1}$ is defined between two POIs of the passenger's path, and has weight $dist(a_j^{(\beta)}, a_{j+1}^{(\beta)})$.

Edges and nodes are defined in the DAG according to specific subsets of values for *i* and *j*, as clarified in detail in Appendix A.1.

Therefore, the shared path $(p_\alpha \cup p_\beta)$ starts from point $a_1^{(\alpha)}$ (corresponding to node $X_{1,0}$ in $DAG_{\alpha,\beta}$, i.e., the source node of the DAG) and ends in point $a_{Q_\alpha}^{(\alpha)}$ (corresponding to node X_{Q_α,Q_β} in $DAG_{\alpha,\beta}$, i.e., the sink node of the DAG). Then the shared path is computed as the shortest path on $DAG_{\alpha,\beta}$ from the source node to the sink node, and the dissimilarity value $D_{\alpha,\beta}$ is computed according to (1). Fig. 3 shows an example of the generation of the shared path including the driver's path p_α (with $Q_\alpha = 4$) and the passenger's path p_β (with $Q_\beta = 3$). The optimal shared path $(p_\alpha \cup p_\beta)$ is shown in yellow on the map and on the DAG. The formal description of how to build the single-passenger DAG $DAG_{\alpha,\beta}$ is discussed in Appendix A.1.

The elaboration is repeated for each pair of paths (p_α, p_β), $\{(p_\alpha, p_\beta): p_\alpha \in \Phi, p_\beta \in \Psi\}$ obtaining as a result the $P \times T$ *dissimilarity matrix* containing the values $D_{\alpha,\beta}$. More in detail, each path p_α is now represented as an object $\widetilde{p_\alpha}$ described in terms of *T* features, with feature β corresponding to the dissimilarity value between p_α and prototype path p_β. We indicate this representation with the vector $\widetilde{p_\alpha} = \{D_{\alpha,1}, \dots, D_{\alpha,\beta}, \dots, D_{\alpha,T}\}$, corresponding to row α of the dissimilarity matrix, as shown in Fig. 1. In this way, each path is represented in terms of the same set of features, making it easy to compare the similarity (nearness) between paths, i.e., similar paths will have similar vector representations.

Then, to organize paths in clusters we consider the paths in Φ as objects described in terms of *T* features (the dissimilarities with the prototype paths), and then we employ the tree data structure used in the Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) algorithm [49] to represent and cluster the paths. BIRCH deals with the efficient incremental and dynamical clustering of high dimensional datasets. Thus, it is particularly suited to quickly and efficiently find clusters in large sets of data. The algorithm is based on the definition of the clustering feature (CF), a summary information for each cluster of objects in the dataset. A CF is defined for each cluster as a triple CF = (*n*, *LS*, *SS*), where *n* is the number of objects belonging to the cluster, *LS* is the linear sum of the objects, and *SS* is the square sum of the objects. In addition, a centroid is associated with each cluster. BIRCH incrementally constructs a hierarchical data structure called CF-tree. More precisely, it stores (and groups) the CFs associated with the clusters, instead of storing the objects belonging to the clusters. The dataset is thus represented by means of a CF-tree, a height-balanced tree, reproducing a hierarchical representation of the data. Each non-leaf node of the tree contains at most b_1 entries: each entry contains a CF and a pointer to a child node. A non-leaf node corresponds to a cluster of sub-clusters, and stores the sum of the CFs of its children. Each leaf node of the tree contains at most b_2 entries: each entry contains a CF, corresponding to an actual cluster of objects. All entries in a leaf must satisfy the following requirement: the maximum diameter of the cluster represented by the entry in each leaf node has to be smaller than a pre-fixed threshold *t*. The larger *t*, the smaller the tree, as clusters will contain a higher number of objects. The parameters

b_1 and b_2 are called *branching factors* and limit the number of sub-clusters in a non-leaf node and a leaf node, respectively. The lower their values, the lower the number of nodes of the tree. As new objects are added to the tree, the tree is dynamically adjusted by means of splitting and merging operations on nodes, i.e., objects are redistributed among clusters, and the CF values are recalculated. To insert a new object in the tree, the following operations are performed:

1. first, by visiting the tree starting from the root node, the closest leaf node is identified based on the CF values (according to the average inter-cluster distance);
2. then, the closest CF entry in the leaf is identified:
   i) if the CF entry can absorb a new object without violating the threshold condition t, the object is inserted into the leaf;
   ii) else, if the maximum size of entries b_2 is not reached yet, a new CF entry is added to the leaf, and the new object is inserted into it;
   iii) otherwise, the leaf is split, a new CF entry is added to the parent node (if this is not possible, the parent node is split as well), and the CF entries are redistributed among the leaves;
3. finally, after insertion, the CFs of the leaf nodes and of all their ancestors are updated.

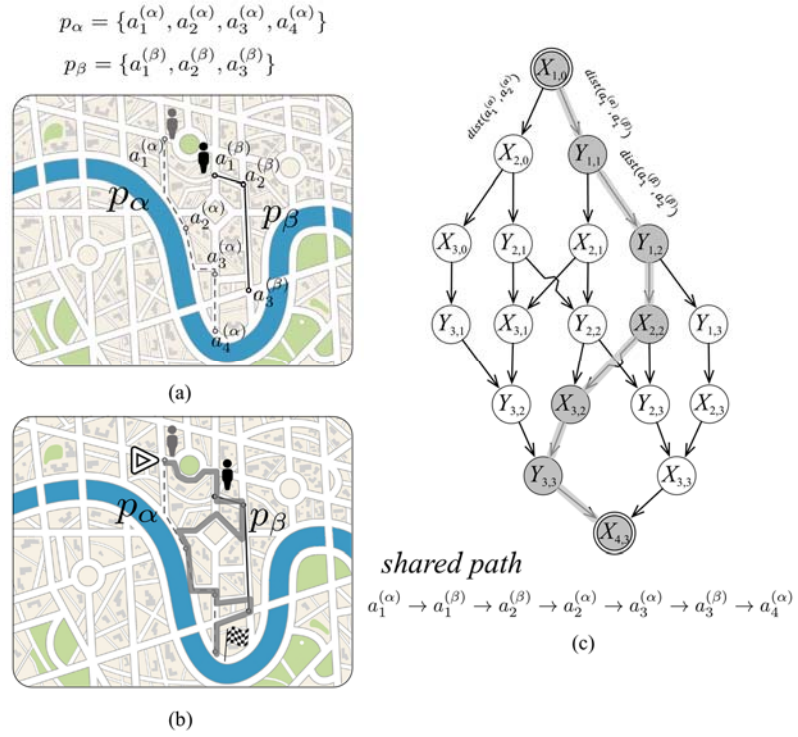


Fig. 3. The generation of the shared path for single-passenger ridesharing. (a) The driver's path p_α (dashed line) and the passenger's path p_β (continuous line). (b) The shared path $(p_\alpha \cup p_\beta)$ (semi-transparent bold line), going from the start symbol to the checkered flag. (c) The corresponding DAG built to find the shared path $(p_\alpha \cup p_\beta)$ (semi-transparent bold line). Please note that, for the sake of simplicity, we show the weights only on a few edges, but all the weights are computed as explained in the text.

With reference to our context, given an initial set Φ of paths represented in $\mathbb{R}^T$, we build a CF-tree employing

the BIRCH algorithm, and we organize the given paths in clusters. At any time a new ride request arrives, the associated path is directly assigned to the closest node, and the CF-tree is dynamically adjusted.

Finally, we need to check if the size of each cluster obtained is in a reasonable interval. In fact, a too small size may prevent from correctly performing the subsequent analysis due to an insufficient number of paths. On the contrary, a too big size leads to a high computational effort to analyze the paths within the cluster. Therefore, in order to obtain clusters having a reasonable number of paths, we fix an interval for the size of the clusters, and we define the actions to perform if the size of the cluster obtained with BIRCH is out of this interval. More in detail:

i) if a cluster does not contain at least N_{min} paths, the cluster is merged with the nearest cluster according to the distance between the centroids. In order to perform the subsequent analysis, the value of N_{min} may depend, e.g., on the size of the pool of paths, or on the number of passengers of the ride-share;

ii) instead, if a cluster contains more than N_{max} paths, the cluster is split to limit the calculation time. More precisely, the cluster undergoes a new analysis with BIRCH, i.e., a new CF-tree for the paths in the cluster is built. The value of the threshold t in this case can be adjusted based on the distribution of paths, i.e., if the cluster is highly compact, the value of t is diminished so as to generate clusters with a lower number of objects. The sub-clusters originated from this analysis are treated as the other clusters. The value of N_{max} may depend, e.g., on the available computing capacities, or on some Quality of Service requirements for the system.

Thus, at the end of this elaboration the paths in Φ result to be organized in L clusters Λ_l, $l = 1, \dots, L$, as follows:

$$\Phi = \bigcup_l \Lambda_l \,,$$

with each cluster containing N_l paths, $N_{min} \leq N_l \leq N_{max}$.

The following subroutines summarize the work performed in the "Grouping of paths" phase:

Subroutine 2.1: compute the $P \times T$ matrix of dissimilarity values between users' path and prototype paths**:=**

- **given** a set Φ of P users' map-matched paths
- **given** a set Ψ of T prototype map-matched paths
- **for each** user path p_α **in** Φ, $\alpha = 1, \dots, P$ **do**
  - compute the length of path p_α: $length(p_\alpha)$
  - **for each** prototype path p_β **in** Ψ, $\beta = 1, \dots, T$ **do**
    - build the DAG for single-passenger ridesharing for path p_α and prototype path p_β: $DAG_{\alpha,\beta}$
    - compute on $DAG_{\alpha,\beta}$ the *shared path* $(p_\alpha \cup p_\beta)$ as the shortest path that starts in p_α's first POI, respects the visiting orders of POIs in p_α and p_β, and ends in p_α's last POI
    - compute the length of $(p_\alpha \cup p_\beta)$: $length(p_\alpha \cup p_\beta)$
    - compute the dissimilarity value: $D_{\alpha,\beta}(p_\alpha, p_\beta) = length(p_\alpha \cup p_\beta) - length(p_\alpha)$
  - **end for**
- **end for**
- **return** the $P \times T$ dissimilarity matrix of values $D_{\alpha,\beta}(p_\alpha, p_\beta)$

Subroutine 2.2: identify L clusters of users' paths**:=**

- **given** the $P \times T$ dissimilarity matrix of values $D_{\alpha,\beta}(p_\alpha, p_\beta)$
- apply the BIRCH algorithm to identify L clusters of paths in Φ
- **return** the L clusters of users' paths

3.3 Analysis of multi-passenger ride matches

In the following we describe the analysis aimed at finding multi-passenger ride matches. The analysis is performed within each cluster of paths Λ_l, $l = 1, \dots, L$, found after the elaboration made in the previous phase. The elaboration is applied in turn on each path p_α in Λ_l, with p_α represented as $\widetilde{p_\alpha} = \{D_{\alpha,1}, \dots, D_{\alpha,T}\}$. First, the similarity between $p_\alpha \in \Lambda_l$ (the driver's path) and each other path $p_\beta \in \Lambda_l$, $\beta = 1, \dots, N_l$, is computed exploiting the cosine distance. We chose this kind of distance as it is known to be best suited to cope with the high-dimensionality of the objects. In fact, when dealing with high-dimensional domains, the use of other distance metrics (e.g., the Euclidean distance) could flatten the distances between the objects due to the "curse of dimensionality" effect [50]. Thus, the cosine distance between p_α and p_β is computed according to the following equation:

$$dist_{cos}(p_\alpha, p_\beta) = 1 - \frac{\sum_{k=1}^{T} D_{\alpha,k} \cdot D_{\beta,k}}{\sqrt{\sum_{k=1}^{T} (D_{\alpha,k})^2} \cdot \sqrt{\sum_{k=1}^{T} (D_{\beta,k})^2}} \tag{4}$$

being p_α and p_β represented as the objects $\widetilde{p_\alpha} = \{D_{\alpha,1}, \dots, D_{\alpha,T}\}$ and $\widetilde{p_\beta} = \{D_{\beta,1}, \dots, D_{\beta,T}\}$, respectively. Then, the N_l values $dist_{cos}(p_\alpha, p_\beta)$ computed are sorted in ascending order, and a *pool* $\Omega_\alpha \subset \Lambda_l$, of $S \geq R_\alpha$ paths near p_α is selected as those having the smaller cosine distance from p_α. We recall that R_α is the number of passengers accepted by the driver of the ride-share. The pool is defined as $\Omega_\alpha = \{p_\beta | p_\beta \in \Lambda_l, p_\beta \neq p_\alpha, dist_{cos}(p_\alpha, p_\beta) \leq \vartheta_S\}$, with ϑ_S the cosine distance computed between p_α and the *S*-th path (after the sorting). Stated in other words, the paths in the set $\{\Lambda_l \backslash \{p_\alpha\}\}$ are sorted in ascending order based on the value computed using (4), and the first *S* paths are selected to form the pool Ω_α.

Among the *S* paths selected we are supposed to find the best group of paths of size R_α to recommend to the driver of path p_α for a ride-share. The minimum value for R_α is 1, while the maximum value for R_α depends on the number of free seats in the driver's vehicle, or on other driver's preferences. In addition, for the sake of simplicity, we assume that the users asking for a ride need to travel alone. In this way R_α matches exactly the actual number of passengers present in the car. Obviously, to adapt the methodology to the case of users travelling with someone else is straightforward. E.g., let us consider a ride-share satisfying the requests of three users (including the driver): if one of the users travels with a child, the methodology is applied with $R_\alpha = 2$, even though the actual number of passengers in the car is four.

Actually, to find the best group of R_α paths for the driver, each possible group of passengers' paths in Ω_α should be examined. We indicate with $\Gamma_g^{(\alpha)}$, $\Gamma_g^{(\alpha)} \subset \Omega_\alpha$, the *g*-th *group of paths* in Ω_α, defined as $\Gamma_g^{(\alpha)} = \{p_\beta | \, p_\beta \in \Omega_\alpha, |\Gamma_g^{(\alpha)}| = R_\alpha\}$, $g = 1, \dots, C_{S,R_\alpha}$. C_{S,R_α} is the number of possible groups of R_α paths in Ω_α, and is computed as the number of combinations of *S* paths into R_α distinct elements as follows:

$$C_{S,R_\alpha} = \binom{S}{R_\alpha} = \frac{S!}{(S-R_\alpha)!R_\alpha!}. \tag{5}$$

Fig. 4 clarifies the relationship among the sets of paths Φ, Λ_l, Ω_α, and $\Gamma_g^{(\alpha)}$ involved in the elaboration.

Then, for each possible group of paths $\Gamma_g^{(\alpha)}$:

i) a DAG (denoted with $DAG_{\alpha,\{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}}$) is built on the set of paths $\{p_\alpha, \Gamma_g^{(\alpha)}\}$, which includes the driver's path p_α and the R_α passengers' paths in $\Gamma_g^{(\alpha)}$;

ii) the candidate shared path ($p_\alpha \cup \Gamma_g^{(\alpha)}$) is calculated, corresponding to the shortest path on $DAG_{\alpha,\{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}}$;

iii) the dissimilarity value between path p_α and the shared path ($p_\alpha \cup \Gamma_g^{(\alpha)}$) is computed.

The dissimilarity value is computed as a generalization of (1)-(3) to multi-passenger ride-shares as follows:

$$D_{\alpha,\{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}}\left(p_\alpha, \Gamma_g^{(\alpha)}\right) = length\left(p_\alpha \cup \Gamma_g^{(\alpha)}\right) - length(p_\alpha). \quad (6)$$

In (6), $length\left(p_\alpha \cup \Gamma_g^{(\alpha)}\right)$ denotes the length of the shared path and is defined as:

$$length\left(p_\alpha \cup \Gamma_g^{(\alpha)}\right) = \sum_{k=1}^{Q_\alpha - 1 + Q_{\Gamma_g^{(\alpha)}}} dist\left(a_k^{\left(\alpha \cup \{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}\right)}, a_{k+1}^{\left(\alpha \cup \{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}\right)}\right), \quad (7)$$

where $a_k^{\left(\alpha \cup \{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}\right)}$ and $a_{k+1}^{\left(\alpha \cup \{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}\right)}$ are two consecutive POIs of the shared path ($p_\alpha \cup \Gamma_g^{(\alpha)}$), and $Q_{\Gamma_g^{(\alpha)}} = \sum_{\beta \in \Gamma_g^{(\alpha)}} Q_\beta$ is the sum of the number of POIs of all the passengers' paths. The shared path is computed as the shortest path on the corresponding multi-passenger DAG built, i.e., the g-th $DAG_{\alpha,\{\beta|p_\beta \epsilon \Gamma_g^{(\alpha)}\}}$, a generalization of the single-passenger DAG presented before. The formal rules for the creation of the DAG in the case of a multiple passengers ridesharing are described in Appendix A.2.

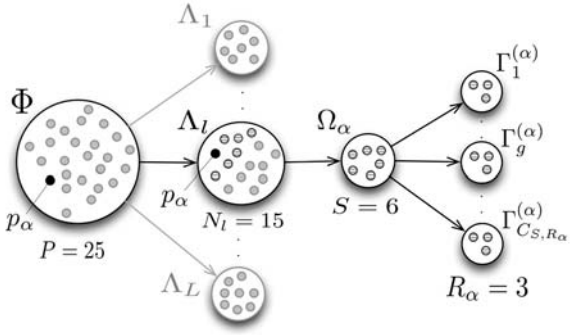


Fig. 4. The sets of paths built to find the best recommendation for the driver of path p_α: the initial set Φ, the cluster Λ_l, the pool Ω_α, and the groups of paths $\Gamma_g^{(\alpha)}$, $g = 1, \ldots, C_{S,R_\alpha}$. To show the reasoning, we set P = 25, S = 6 and R_α = 3, thus $C_{S,R_\alpha} = 20$.

The selection of a pool of candidate paths is needed to guarantee to recommend the optimal ride match to the driver. In fact, merely taking into account the nearest R_α paths to p_α will not typically lead to the best solution, as Fig. 5 shows. In Fig. 5 the driver's path p_α is shown along with the three nearest paths p_β, p_γ, and p_δ ordered by ascending dissimilarity, i.e., with $D_{\alpha,\beta} < D_{\alpha,\gamma} < D_{\alpha,\delta}$. By considering no pool and, for the sake of simplicity, $R_\alpha = 2$, we can easily see that the detour for the driver when the two nearest paths p_β and p_γ are taken into

account is clearly bigger than that computed when taking into account p_β, matched with the third nearest path p_δ. More precisely, $D_{\alpha,\Gamma_1^{(\alpha)}} \ll D_{\alpha,\Gamma_2^{(\alpha)}}$, being $\Gamma_1^{(\alpha)} = \{p_\beta, p_\delta\}$ and $\Gamma_2^{(\alpha)} = \{p_\beta, p_\gamma\}$.

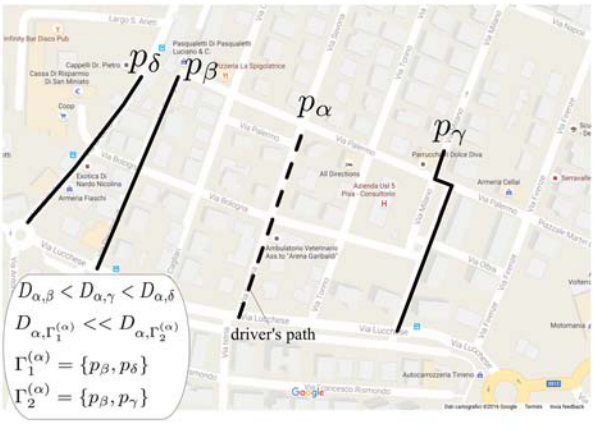


Fig. 5. A situation showing the requirement for a pool of $S \geq R_\alpha$ paths (continuous lines) to compute the optimal shared path for the driver of path p_α (dashed line). The map is provided by Google® Maps.

The following subroutines summarize the work performed in the "Analysis of multi-passenger ride matches" phase:

Subroutine 3.1: identify a pool of *S* paths for each user's path**:=**

- **given** the L clusters of users' paths
- **for each** cluster Λ_l, $l = 1, \dots, L$, **do**
  - **for each** user path p_α **in** Λ_l, $\alpha = 1, \dots, N_l$ **do**
    - **for each** user path p_β **in** Λ_l, $\beta = 1, \dots, N_l$, **do**
      - compute the pairwise cosine distance $dist_{cos}(p_\alpha, p_\beta)$
    - **end for**
    - sort the N_l values $dist_{cos}(p_\alpha, p_\beta)$
    - form the *pool* Ω_α of paths by selecting the *S* paths p_β, $p_\beta \neq p_\alpha$, **in** Λ_l, having smaller cosine distance from p_α
  - **end for**
- **end for**
- **return** a pool Ω_α of *S* paths for each user's path p_α

Subroutine 3.2: examine the groups of paths in each user's pool**:=**

- **given** the pool Ω_α of *S* paths for each user's path p_α
- **for each** user's pool of paths Ω_α, $\alpha = 1, \dots, P$ **do**
  - form the C_{S,R_α} possible groups of paths in Ω_α, $\Gamma_g^{(\alpha)}$, $g = 1, \dots, C_{S,R_\alpha}$
  - **for each** group of paths $\Gamma_g^{(\alpha)}$ **in** Ω_α, $g = 1, \dots, C_{S,R_\alpha}$, **do**
    - build the DAG for multi-passenger ridesharing for the path p_α and the group of paths $\Gamma_g^{(\alpha)}$: $DAG_{\alpha,\{\beta | p_\beta \in \Gamma_g^{(\alpha)}\}}$
    - compute on $DAG_{\alpha,\{\beta | p_\beta \in \Gamma_g^{(\alpha)}\}}$ the *shared path* $(p_\alpha \cup \Gamma_g^{(\alpha)})$ as the shortest path that starts in p_α's first POI, respects the visiting orders of POIs in p_α and $\Gamma_g^{(\alpha)}$, and ends in p_α's last POI
    - compute the length of $(p_\alpha \cup \Gamma_g^{(\alpha)})$: $length\left((p_\alpha \cup \Gamma_g^{(\alpha)}\right)$
    - compute the dissimilarity value: $D_{\alpha,\{\beta | p_\beta \in \Gamma_g^{(\alpha)}\}}\left(p_\alpha, \Gamma_g^{(\alpha)}\right) = length\left(p_\alpha \cup \Gamma_g^{(\alpha)}\right) - length(p_\alpha)$.
  - **end for**
- **end for**
- **return** the groups $\Gamma_g^{(\alpha)}$ of paths and the dissimilarity values $D_{\alpha,\{\beta | p_\beta \in \Gamma_g^{(\alpha)}\}}\left(p_\alpha, \Gamma_g^{(\alpha)}\right)$ for each user's path p_α

3.4 Ridesharing recommendation

In the last phase, a ridesharing recommendation is finally produced for each driver of the paths in Φ.

More precisely, the best group of paths $\Gamma_{g=\bar{g}}^{(\alpha)}$ for the driver of path p_α is found among the C_{S,R_α} possible groups, as the one corresponding to the shortest detour for the driver, i.e., the smallest dissimilarity value:

$$g = \bar{g} \ : \ \min_{g} \left(D_{\alpha,\{\beta | p_\beta \in \Gamma_g^{(\alpha)}\}} \left(p_\alpha, \Gamma_g^{(\alpha)} \right) \right).$$

The recommendation for the driver of path p_α consists of:

i) the best group of paths $\Gamma_{\bar{g}}^{(\alpha)}$ (i.e., users with whom to share the ride), corresponding to the dissimilarity value $D_{\alpha,\{\beta | p_\beta \in \Gamma_{\bar{g}}^{(\alpha)}\}} \left(p_\alpha, \Gamma_{\bar{g}}^{(\alpha)} \right)$;

ii) the corresponding optimal shared path $sp_\alpha = \left(p_\alpha \cup \Gamma_{\bar{g}}^{(\alpha)} \right)$ to follow in order to satisfy the order constraints on each path.

The quality of a recommendation for the driver of path p_α can then be evaluated in terms of the dissimilarity value itself, measuring the detour for the driver, and in terms of the mileage saving obtained exploiting the ride-sharing opportunity with respect to the case in which the R_α+1 users drive alone.

The following subroutine summarizes the work performed in the "Ridesharing recommendation" phase:

Subroutine 4.1: find and recommend the optimal shared path to each user**:=**

- **given** the groups $\Gamma_g^{(\alpha)}$ of paths and the dissimilarity values $D_{\alpha,\{\beta | p_\beta \in \Gamma_g^{(\alpha)}\}} \left(p_\alpha, \Gamma_g^{(\alpha)} \right)$ for each user's path p_α
- **for each** user path p_α **in** Φ, $\alpha = 1, \dots, P$ **do**
  - select the best group of paths $\Gamma_{g=\bar{g}}^{(\alpha)}$ corresponding to the smallest dissimilarity value $D_{\alpha,\{\beta | p_\beta \in \Gamma_g^{(\alpha)}\}} \left(p_\alpha, \Gamma_g^{(\alpha)} \right)$, $g = 1, \dots, C_{S,R_\alpha}$
  - provide the optimal shared path $sp_\alpha = \left(p_\alpha \cup \Gamma_{\bar{g}}^{(\alpha)} \right)$
- **end for**
- **return** the best group of paths $\Gamma_{g=\bar{g}}^{(\alpha)}$ for each user's path p_α and the optimal shared path sp_α

The underlying idea of the proposed approach could be of the utmost importance in the effective management of ridesharing services, as it is able to take into account multiple-stops trips. Users may indicate their trips, defined in terms of the set of ordered POIs, scheduled in the near future, and even a preferred role for the trip (passenger, driver or both). In the case that a user wants to act exclusively as driver (or as passenger), the elaboration regarding that user can be adapted accordingly, e.g., by excluding the user from the list of passengers (or drivers). In the experiments, we take into account the most general case, in which the users typically are car owners and may act indifferently as drivers or passengers, and we produce a recommendation for each driver.

3.5 Flowchart of the overall process

With the objective of summarizing the overall process performed by our ridesharing recommender, Fig. 6 shows the flowchart of the overall process, pointing out the activation of the subroutines in each phase. Appendix B presents some considerations on the computational complexity of each phase.

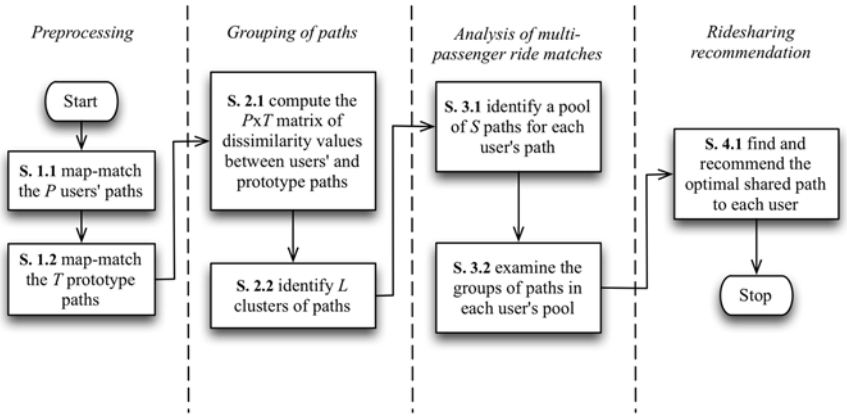


Fig. 6. Flowchart of the main operations of the elaboration, by phase.

4. Experimental results and discussion

In this section, we describe the experimental analysis performed on a case study for evaluating the proposed approach. Table 2 summarizes the parameters' values of the experimental analysis.

Table 2. List of symbols

Parameter	P	$length(p_\alpha)$	T	L	R_α	S	Q_α	b_1=b_2	t	N_{min}	N_{max}
Value	100	200 m - 4 km	10	17	1 - 4	R_α - 10	2 - 5	5	5000	S+1	30

The set Φ of paths employed as case study contains P = 100 trips belonging to the road network of the city of Pisa, Italy (covering an area of about 70 km^2), and representing the ride requests coming from P users in a given time interval. The trip length ranges from 200 m to about 4 km. They represent an example of the most frequent or usual trips of users moving in the city by car in a given time interval. In fact, the aim is to match trips with similar temporal constraints, i.e., users with similar needs, and moving approximately in the same area. Each trip is described in terms of a set of POIs, which can be of common interest (e.g., school, grocery, department store, drugstore) or relevant for the corresponding user (e.g., home, work place). The number of POIs of each trip ranges from 2 (meaning that the trip is defined only in terms of its origin and destination points) to 5. Each POI can be provided as the name of the place (e.g. Pisa Airport), the complete address (e.g, Piazzale D'Ascanio, 1, 56121 Pisa, Italy), or directly by means of its GPS coordinates (e.g., {43.688479, 10.397713}). The trips were generated within a specific interview campaign involving 30 voluntary users. The users indicated some of their habitual paths in the city along with the set of POIs to visit. Table 3 shows the GPS coordinates of the POIs of each path. We recall that users of the ride have to be merely picked-up or dropped-off, i.e., they will spend a negligible amount of time in each POI.

During preprocessing, the POIs of each trip are map-matched, exploiting the OSM digital map related to the city of Pisa, Italy. The set Ψ of prototype paths was selected in order to uniformly cover the city. By taking into account the size of the city, we generated T = 10 prototype paths, with the same ranges for length and number of

POIs of the paths in Φ. Next, we grouped the paths into clusters. First, we computed a dissimilarity value between each path in Φ, and each prototype path in Ψ, by building the corresponding DAG for single-passenger ridesharing, and by computing the shortest path on it. Then, the paths, represented in terms of T dissimilarities values, were clustered by means of the CF-tree data structure of BIRCH. In the experimental analysis, we set the following values for the model parameters. The values for the branching factors b_1 and b_2 were both set to 5. The value for the threshold t was heuristically set, in order to obtain clusters of a reasonable size: we tried values between 0 and the average pairwise distance between paths in Φ, and we found 5000 to be a reasonable value for t. From the CF-tree we obtained $L = 17$ clusters. Afterward, we applied the additional splitting and merging operations to obtain clusters of reasonable sizes, by setting $N_{min} = S + 1$, and $N_{max} = 30$. The choice of the value of S is discussed hereunder.

Then, we performed the elaboration to obtain multi-passenger ridesharing recommendations, by taking into account one cluster at a time, say cluster Λ_l. For each driver's path, say $p_\alpha \in \Lambda_l$, we took into account a pool of S candidate passenger paths. Within the pool, we considered all the possible groups of R_α passenger paths. The value of R_α may be different for each driver, but in the experiments, for the sake of simplicity, we took into account the same value for all the drivers. Thus, for each possible group of R_α paths, we built the corresponding DAG for multiple-passenger ridesharing, and we computed the dissimilarity value and the resulting shared path. The experiments for multi-passenger ride-sharing were performed for $R_\alpha = \{1,2,3,4\}$. We did not consider higher values for R_α as the majority of cars moving in the city has typically five seats (four taken by the passengers and one by the driver). Regarding the value of S, we experimented with and without the use of the pool, by proving that the use of the pool improves the results obtained in terms of the length of the shared path. More in detail, computing the recommendations without employing the pool of paths means choosing the first R_α paths in a greedy way (i.e., setting $S = R_\alpha$). Thus, for each value of R_α, we heuristically tried several values for S: from R_α (corresponding to the absence of the pool) to 10. In fact, we found that for higher values of S the results did not change in a considerable way. Further, the value of S may be set based on the distribution of paths in the city road network. More precisely, if the paths to be processed are all close together, increasing the value of S may help to find a better shared path; on the contrary if the paths are very far away from each other, even a low value of S may be sufficient to find the best shared path.

Finally, the recommender matched path p_α (and thus its user) with the least dissimilar group of paths, corresponding to the shared path with the shortest detour for the driver. The recommendation results of the experiment performed with $R_\alpha = 1$ and S =5 are provided in Table 4.

Even though the effort requested to the driver may be big, several benefits rise from ridesharing. Traffic problems, e.g., queues and congestions, are reduced as only one vehicle moves in the city instead of R_α+1 vehicles. Pollution, caused by cars travelling for a given amount of distance, may be reduced in the case of vehicle mileage savings, i.e., when the shared path is shorter than the sum of the lengths of the paths of the R_α+1 users driving alone. Besides, the monetary cost of parking fees, fuel, restricted traffic areas tickets, etc. can also be shared among users. In the experiments performed, the recommendation for each driver in Φ was evaluated in terms of the following measures: i) mileage saving (if achieved), ii) length of the resulting shared path, and iii) dissimilarity value, i.e., a measure of the detour for the driver. In this way, the convenience of the ride-share can be assessed both in terms of the overall economy of the ride, and in terms of the discomfort for the driver. Table 5 shows the results of the experiments performed for different combinations of the values of S and R_α. In each

experiment, we first computed the measures for each of the 100 recommendations, and then we calculated the averaged values. More precisely, the length of the shared path, and the dissimilarity value were computed according to (6)-(7), for each recommendation, and then averaged over the 100 paths. The mileage saving (*MS*) was calculated for each recommendation as the difference between the sum of the distances travelled by the R_α+1 users driving alone, and the length of the shared path sp_α of the ride-share:

$$MS = length(p_\alpha) + \sum_{p_\beta \in \Gamma_{\bar{g}}^{(\alpha)}} \big(length\,(p_\beta) \big) - length(sp_\alpha).$$

Then, the average mileage saving was obtained considering only the *positive* mileage savings obtained in each experiment: we obtained a positive mileage saving in *Z* cases over 100 (with *Z* going from 39 to 76). The reason of this result is trivial: the detour for the driver is unavoidable, as he/she needs to alter his/her path to visit the POIs of the passengers picked-up. The only (lucky) case in which a detour is not needed is when all the POIs of the passengers are exactly on the driver's path. In addition, we wish to point out that the detour for the driver may be big, due to the presence of several one-ways, or limited traffic zones, which require, e.g., to go back over the same route, by increasing the length of the shared path. This is particularly evident in the city road network of Pisa, Italy.

Table 5 shows an increasing average mileage saving, an increasing value of *Z*, a decreasing average length of the shared path, and a decreasing average dissimilarity value as the size of the pool increases.

Furthermore, in order to better highlight the reasoning, we also show in more detail (Fig. 6) an example of multi-passenger ridesharing with $R_\alpha = 4$, i.e., with five users. Fig. 6(a) shows the position and the information about the five paths involved in this ride-share on the city map: the driver's path p_6 (depicted with a black dashed line) and four passengers' paths $p_1,\ p_2,\ p_3,\ p_4\ \in \Omega_6$ (depicted with continuous colored lines). The set of passengers' paths represents the best group of paths found in the pool of paths Ω_6 for the driver of path p_6. Fig. 6(b) shows the resulting shared path to follow $sp_6 = (p_6 \cup \{p_1, p_2, p_3, p_4\})$ (depicted on the map with a bold grey semi-transparent line). The length of the shared path is 8 km, and it requires a detour for the driver of about 4.3 km. From Fig. 6 we can observe that the distance travelled by the driver along the shared path (8 km in the example) is lower than the sum of the distances travelled by the single vehicles (9.5 km in the example), with a mileage saving of 1.5 km. We would like to point out that the considerable detour for the driver is mainly due to the limited number of users considered in the experimental analysis. When the recommender will be deployed, we expect that a higher number of users will allow generating more convenient shared paths for the driver. Anyway, we have to consider that, just in the described example, we have a significant benefit since only one car will be travelling in the city rather than five, thus ensuring traffic reduction.

As a final remark, we wish to point out that, in the case the driver thinks the detour proposed by the recommender system is too big, the system might also be designed to reduce his/her discomfort by suggesting a shared-ride with: i) a lower number of passengers, or ii) a smaller detour value (the driver may indicate a maximum detour value allowed, e.g., corresponding to a given percentage of the length of the driver's path). For the sake of simplicity, however, we have limited ourselves to consider only shared rides with the number of passengers indicated by the driver, and without limitation on the detour value.

The proposed approach is intended to be integrated in a ridesharing recommendation service for smart cities. A user willing to use the service sends a ride request and he/she is recommended with an efficient ridesharing

solution. In order to handle the computational effort needed to find the best recommendation for the users, the service can be built on a Service Oriented Architecture (SOA). SOAs allow exploiting easy integration with other services and scalability of the service itself. The integration property allows the ridesharing recommender to be integrated with other services (e.g., traffic news, weather information). The scalability property allows distributing the users' requests among different elaboration servers, dedicated, e.g., to different city areas, or clusters of paths. Further, the use of a cloud computing architecture and of different instances of the service could help manage scalability dynamically, and reduce the time needed to compute a recommendation. In fact, this time depends both on the number of passengers of the ride-share, and on the number of POIs composing each path involved: the greater the total number of POIs to visit in the shared ride, the bigger the order of the DAG, and thus the greater the time needed to compute the shortest path on it.

Finally, we would like to highlight that a comparison with ridesharing works existing in the literature is not straightforward as the proposed system is novel and has peculiar features that make it different from the previous works and hardly comparable with these. Basically, the proposed approach differs from existing works because it is characterized by all the following aspects: i) it takes the urban context into account; ii) the driver is not dedicated and can give a ride to multiple passengers; iii) it matches multiple-stops trips, and allows each user to specify ordering constraints between the POIs to be visited; iv) the ride-matching criterion is based on the definition of a novel dissimilarity function between paths, which takes into account the geographic distances between the POIs; v) the shared path recommended to the driver is the shortest path satisfying the users' ride requests; vi) passengers' pick-up or drop-off points are exactly on their paths, i.e., no need for the passengers to walk to a common meeting point; vii) origin and destination of the trips do not have to be common for the users. In particular, to the best of our knowledge, the aspect iii) is considered in no previous ridesharing work: our system allows users to specify POIs to be visited and the order of the visit, thus making it unique among the different approaches proposed in the literature. Since the optimization is based on this constraint, comparisons with ridesharing approaches, which do not consider to visit POIs in a user-specified order, does not appear to be fair. Further, although there exist a number of ridesharing services available on the web, these services allow users to indicate the origin and destination of the trip and then suggest possible shared rides by considering origins and destinations of the trips already stored in the repository. Thus, no experimental evaluation can be performed by using these services because they do not allow populating the database independently.

Since we could not prove the effectiveness of our system in terms of optimal choice of the passengers, we decided at least to show that our system is able to generate the optimal shared ride for the selected passengers. To this aim, we used RouteXL [51], a publicly available route planner service for multiple destinations. Given a fixed starting point, a fixed ending point, and a set of intermediate stops in random order, the service generates the best path in terms of duration of the route by choosing the best visiting order of the intermediate stops. We recall that in the case of negligible amount of time spent in each POI and considering the urban context, the quickest path corresponds actually to the shortest path. We adopted as starting and ending points for RouteXL the corresponding points of the driver. Further, the intermediate stops were initialized with the starting and ending points of the trips of the passengers, and with the POIs of the trips of the driver and passengers. We aimed to verify whether RouteXL determines the same optimal path as our system, when the choice of the minimal shared path is not affected by the constraints on the visiting order of the POIs. For the sake of simplicity, we selected cases with one driver and at most two passengers. We showed and discussed four cases

(Fig. 7) in which the shared path of a given recommendation suggested by our algorithm and depicted through Google Maps (on the right) matches the route generated by the RouteXL service (on the left). Conversely, we show two cases (Fig. 8) in which the optimal path computed by the RouteXL service does not match the recommended path produced by our algorithm, due to the presence of unfavorable constraints on the visiting order of the POIs. In these cases, however, it can be observed that the optimal path determined by our system and the one determined by RouteXL are not very different and the increase in length is actually due to the constraints imposed by the single paths.

5. Conclusions

We have presented an approach for ride-sharing recommendation. The ride-matching criterion is based on a novel *dissimilarity function* computed between the driver's path and the *shared path*. The dissimilarity function measures the detour requested to the driver to visit also the POIs of the passengers' paths, by respecting the ordering constraints defined between the POIs of each path. By formulating the problem as a SOP, we are able to find the shortest shared ride that satisfies the ordering constraints. We have applied our approach on a case study of 100 paths in the city of Pisa, Italy.

As a future work we aim to include, in the computation of the shared path, also temporal constraints for visiting the POIs, e.g., to let the users specify a specific time window to visit each POI. In addition, we are planning to add live road traffic information to the recommender system. In this way, the suggested shared path recommended to the users can avoid traffic jams, by taking into account the live traffic conditions.

Funding

This work was partially supported by the project "IoT e Big Data: metodologie e tecnologie per la raccolta e l'elaborazione di grosse moli di dati", id. PRA_2017_37, funded by "Progetti di Ricerca di Ateneo - PRA 2017" of the University of Pisa.

Acknowledgements

We would like to thank Prof. Fabio Schoen, Dept. of Information Engineering, University of Florence, Italy, and Dr. David Di Lorenzo, Fleetmatics Research, Florence, Italy, for their technical support.

Table 3. Description of paths

Path id	*List of POIs*
1	$a_1^1 = (43.71878,10.38510)$ $a_2^1 = (43.71829,10.39224)$ $a_3^1 = (43.71790,10.39440)$
2	$a_1^2 = (43.71677,10.38217)$ $a_2^2 = (43.71726,10.38916)$ $a_3^2 = (43.72235,10.39205)$ $a_4^2 = (43.72346,10.39199)$ $a_5^2 = (43.72463,10.39397)$
3	$a_1^3 = (43.71616,10.39147)$ $a_2^3 = (43.71404,10.39061)$ $a_3^3 = (43.71410,10.38845)$ $a_4^3 = (43.71974,10.38520)$

4	$a_1^4 = (43.71606,10.39266)\ a_2^4 = (43.71974,10.38520)$
5	$a_1^5 = (43.71616,10.39147)\ a_2^5 = (43.71344,10.38908)\ a_3^5 = (43.71751,10.38902)\ a_4^5 = (43.71790,10.39440)$
6	$a_1^6 = (43.71685,10.40731)\ a_2^6 = (43.71864,10.40864)\ a_3^6 = (43.71668,10.40422)$
7	$a_1^7 = (43.72386,10.41012)\ a_2^7 = (43.71923,10.41089)\ a_3^7 = (43.71636,10.40453)\ a_4^7 = (43.71756,10.40431)$ $a_5^7 = (43.71829,10.40498)$
8	$a_1^8 = (43.71563,10.40683)\ a_2^8 = (43.71582,10.40714)\ a_3^8 = (43.72076,10.40405)$
9	$a_1^9 = (43.71563,10.40683)\ a_2^9 = (43.71642,10.40666)\ a_3^9 = (43.71636,10.40453)\ a_4^9 = (43.71668,10.40422)$
10	$a_1^{10} = (43.71973,10.40521)\ a_2^{10} = (43.72521,10.40561)$
11	$a_1^{11} = (43.71181,10.39370)\ a_2^{11} = (43.71386,10.40358)$
12	$a_1^{12} = (43.71281,10.40507)\ a_2^{12} = (43.71369,10.40394)\ a_3^{12} = (43.71868,10.40922)$
13	$a_1^{13} = (43.71250,10.39799)\ a_2^{13} = (43.71226,10.40109)\ a_3^{13} = (43.71267,10.40239)\ a_4^{13} = (43.71363,10.40324)$
14	$a_1^{14} = (43.71066,10.40008)\ a_2^{14} = (43.70992,10.39951)\ a_3^{14} = (43.71032,10.40555)\ a_4^{14} = (43.71313,10.40579)$
15	$a_1^{15} = (43.71051,10.40256)\ a_2^{15} = (43.71313,10.40579)$
16	$a_1^{16} = (43.72821,10.39396)\ a_2^{16} = (43.72731,10.39267)\ a_3^{16} = (43.72397,10.38362)$
17	$a_1^{17} = (43.71010,10.38334)\ a_2^{17} = (43.71335,10.38872)\ a_3^{17} = (43.71726,10.38916)\ a_4^{17} = (43.72278,10.39222)$ $a_5^{17} = (43.72440,10.40236)$
18	$a_1^{18} = (43.72322,10.37953)\ a_2^{18} = (43.71746,10.38265)\ a_3^{18} = (43.72003,10.38677)$
19	$a_1^{19} = (43.71564,10.37992)\ a_2^{19} = (43.71797,10.38172)\ a_3^{19} = (43.72631,10.37824)$
20	$a_1^{20} = (43.72464,10.39140)\ a_2^{20} = (43.72489,10.39253)\ a_3^{20} = (43.71971,10.39739)\ a_4^{20} = (43.71588,10.40187)$
21	$a_1^{21} = (43.71066,10.40543)\ a_2^{21} = (43.71059,10.40127)$
22	$a_1^{22} = (43.70148,10.42042)\ a_2^{22} = (43.70604,10.40925)\ a_3^{22} = (43.70727,10.40759)\ a_4^{22} = (43.70995,10.40173)$
23	$a_1^{23} = (43.70693,10.38581)\ a_2^{23} = (43.70805,10.38553)\ a_3^{23} = (43.71048,10.38778)$
24	$a_1^{24} = (43.71386,10.40186)\ a_2^{24} = (43.71422,10.40289)\ a_3^{24} = (43.71444,10.40314)$
25	$a_1^{25} = (43.72996,10.41945)\ a_2^{25} = (43.72729,10.41555)\ a_3^{25} = (43.72847,10.40269)$
26	$a_1^{26} = (43.69945,10.39177)\ a_2^{26} = (43.70627,10.39677)$
27	$a_1^{27} = (43.70024,10.40181)\ a_2^{27} = (43.70398,10.39638)\ a_3^{27} = (43.70379,10.39787)$
28	$a_1^{28} = (43.70700,10.39609)\ a_2^{28} = (43.69964,10.39223)$
29	$a_1^{29} = (43.69856,10.41001)\ a_2^{29} = (43.70674,10.40295)$
30	$a_1^{30} = (43.69962,10.40070)\ a_2^{30} = (43.69965,10.40011)\ a_3^{30} = (43.69979,10.39840)\ a_4^{30} = (43.70011,10.39790)$ $a_5^{30} = (43.70152,10.38869)$
31	$a_1^{31} = (43.73065,10.39719)\ a_2^{31} = (43.72212,10.39767)\ a_3^{31} = (43.71640,10.37406)\ a_4^{31} = (43.71206,10.38249)$
32	$a_1^{32} = (43.72464,10.40785)\ a_2^{32} = (43.72104,10.42930)$
33	$a_1^{33} = (43.69945,10.39177)\ a_2^{33} = (43.69818,10.39102)\ a_3^{33} = (43.69762,10.39612)$
34	$a_1^{34} = (43.70666,10.39717)\ a_2^{34} = (43.70484,10.40104)\ a_3^{34} = (43.70343,10.40362)\ a_4^{34} = (43.70431,10.40653)$
35	$a_1^{35} = (43.71168,10.42055)\ a_2^{35} = (43.71602,10.41420)\ a_3^{35} = (43.72749,10.42690)$
36	$a_1^{36} = (43.72330,10.39212)\ a_2^{36} = (43.72012,10.39664)\ a_3^{36} = (43.71906,10.39387)\ a_4^{36} = (43.71903,10.38603)$
37	$a_1^{37} = (43.71591,10.39847)\ a_2^{37} = (43.70986,10.39365)\ a_3^{37} = (43.70912,10.40131)\ a_4^{37} = (43.71073,10.40346)$
38	$a_1^{38} = (43.71296,10.40127)\ a_2^{38} = (43.70912,10.40131)$
39	$a_1^{39} = (43.71790,10.39440)\ a_2^{39} = (43.72001,10.38666)\ a_3^{39} = (43.71677,10.38217)\ a_4^{39} = (43.71396,10.38337)$
40	$a_1^{40} = (43.71183,10.39863)\ a_2^{40} = (43.70992,10.40046)\ a_3^{40} = (43.71246,10.40519)\ a_4^{40} = (43.71313,10.40579)$
41	$a_1^{41} = (43.71066,10.39176)\ a_2^{41} = (43.70976,10.40404)$
42	$a_1^{42} = (43.71588,10.40187)\ a_2^{42} = (43.71077,10.39834)\ a_3^{42} = (43.70112,10.40021)$
43	$a_1^{43} = (43.72009,10.38644)\ a_2^{43} = (43.71762,10.39108)\ a_3^{43} = (43.71589,10.39347)$
44	$a_1^{44} = (43.70614,10.39793)\ a_2^{44} = (43.70524,10.39984)\ a_3^{44} = (43.70511,10.40164)$
45	$a_1^{45} = (43.70608,10.39809)\ a_2^{45} = (43.71091,10.39990)$
46	$a_1^{46} = (43.72347,10.41364)\ a_2^{46} = (43.72260,10.41484)\ a_3^{46} = (43.72170,10.41213)$

47 $a_1^{47} = (43.72025,10.41987)\ a_2^{47} = (43.72057,10.41784)\ a_3^{47} = (43.71809,10.41806)\ a_4^{47} = (43.71789,10.41625)\ a_5^{47} = (43.71970,10.41336)$

48 $a_1^{48} = (43.71966,10.41288)\ a_2^{48} = (43.71930,10.41706)\ a_3^{48} = (43.72098,10.42070)$

49 $a_1^{49} = (43.71010,10.41326)\ a_2^{49} = (43.71226,10.42386)\ a_3^{49} = (43.70993,10.42391)$

50 $a_1^{50} = (43.72307,10.40884)\ a_2^{50} = (43.71938,10.41753)\ a_3^{50} = (43.71517,10.41064)\ a_4^{50} = (43.71164,10.40948)$

51 $a_1^{51} = (43.72267,10.40964)\ a_2^{51} = (43.72208,10.41032)\ a_3^{51} = (43.72212,10.41331)\ a_4^{51} = (43.72376,10.41080)$

52 $a_1^{52} = (43.71883,10.41102)\ a_2^{52} = (43.71917,10.41535)\ a_3^{52} = (43.72114,10.41639)\ a_4^{52} = (43.72202,10.41504)$

53 $a_1^{53} = (43.72829,10.39425)\ a_2^{53} = (43.72900,10.39869)\ a_3^{53} = (43.72886,10.40548)$

54 $a_1^{54} = (43.72886,10.40542)\ a_2^{54} = (43.72796,10.39978)\ a_3^{54} = (43.72720,10.39486)\ a_4^{54} = (43.72771,10.39345)$

55 $a_1^{55} = (43.72263,10.40668)\ a_2^{55} = (43.72146,10.40316)\ a_3^{55} = (43.72372,10.40265)\ a_4^{55} = (43.72435,10.40231)$

56 $a_1^{56} = (43.71419,10.40792)\ a_2^{56} = (43.71364,10.40976)\ a_3^{56} = (43.71463,10.40974)\ a_4^{56} = (43.71585,10.40909)\ a_5^{56} = (43.71865,10.40858)\ a_6^{56} = (43.72355,10.40224)$

57 $a_1^{57} = (43.71399,10.40833)\ a_2^{57} = (43.71046,10.40977)\ a_3^{57} = (43.71004,10.41279)$

58 $a_1^{58} = (43.72587,10.40339)\ a_2^{58} = (43.72896,10.40618)\ a_3^{58} = (43.73313,10.41120)$

59 $a_1^{59} = (43.72789,10.39984)\ a_2^{59} = (43.73090,10.40211)\ a_3^{59} = (43.73226,10.40778)$

60 $a_1^{60} = (43.73096,10.39661)\ a_2^{60} = (43.72761,10.39090)\ a_3^{60} = (43.72877,10.38865)$

61 $a_1^{61} = (43.71183,10.39863)\ a_2^{61} = (43.70992,10.40046)\ a_3^{61} = (43.71246,10.40519)\ a_4^{61} = (43.71313,10.40579)$

62 $a_1^{62} = (43.71066,10.39176)\ a_2^{62} = (43.70976,10.40404)$

63 $a_1^{63} = (43.71588,10.40187)\ a_2^{63} = (43.71077,10.39834)\ a_3^{63} = (43.70112,10.40021)$

64 $a_1^{64} = (43.72009,10.38644)\ a_2^{64} = (43.71762,10.39108)\ a_3^{64} = (43.71589,10.39347)$

65 $a_1^{65} = (43.70614,10.39793)\ a_2^{65} = (43.70524,10.39984)\ a_3^{65} = (43.70511,10.40164)$

66 $a_1^{66} = (43.70608,10.39809)\ a_2^{66} = (43.71091,10.39990)$

67 $a_1^{67} = (43.72347,10.41364)\ a_2^{67} = (43.72260,10.41484)\ a_3^{67} = (43.72170,10.41213)$

68 $a_1^{68} = (43.72025,10.41987)\ a_2^{68} = (43.71789,10.41625)\ a_3^{68} = (43.71970,10.41336)$

69 $a_1^{69} = (43.71966,10.41288)\ a_2^{69} = (43.72098,10.42070)$

70 $a_1^{70} = (43.71010,10.41326)\ a_2^{70} = (43.71226,10.42386)\ a_3^{70} = (43.70993,10.42391)$

71 $a_1^{71} = (43.72307,10.40884)\ a_2^{71} = (43.71938,10.41753)\ a_3^{71} = (43.71517,10.41064)\ a_4^{71} = (43.71164,10.40948)$

72 $a_1^{72} = (43.72267,10.40964)\ a_2^{72} = (43.72208,10.41032)\ a_3^{72} = (43.72376,10.41080)$

73 $a_1^{73} = (43.71883,10.41102)\ a_2^{73} = (43.71917,10.41535)\ a_3^{73} = (43.72114,10.41639)\ a_4^{73} = (43.72202,10.41504)$

74 $a_1^{74} = (43.72829,10.39425)\ a_2^{74} = (43.72886,10.40548)$

75 $a_1^{75} = (43.72796,10.39978)\ a_2^{75} = (43.72771,10.39345)\ a_3^{75} = (43.72886,10.40542)$

76 $a_1^{76} = (43.72263,10.40668)\ a_2^{76} = (43.72146,10.40316)\ a_3^{76} = (43.72435,10.40231)$

77 $a_1^{77} = (43.71419,10.40792)\ a_2^{77} = (43.71364,10.40976)\ a_3^{77} = (43.71865,10.40858)\ a_4^{77} = (43.72355,10.40224)$

78 $a_1^{78} = (43.71399,10.40833)\ a_2^{78} = (43.71046,10.40977)\ a_3^{78} = (43.71004,10.41279)$

79 $a_1^{79} = (43.72896,10.40618)\ a_2^{79} = (43.73313,10.41120)\ a_3^{79} = (43.72587,10.40339)$

80 $a_1^{80} = (43.72789,10.39984)\ a_2^{80} = (43.73090,10.40211)\ a_3^{80} = (43.73226,10.40778)$

81 $a_1^{81} = (43.73096,10.39661)\ a_2^{81} = (43.72761,10.39090)\ a_3^{81} = (43.72877,10.38865)$

82 $a_1^{82} = (43.72464,10.40785)\ a_2^{82} = (43.72104,10.42930)$

83 $a_1^{83} = (43.69945,10.39177)\ a_2^{83} = (43.69762,10.39612)$

84 $a_1^{84} = (43.70666,10.39717)\ a_2^{84} = (43.70484,10.40104)\ a_3^{84} = (43.70343,10.40362)\ a_4^{84} = (43.70431,10.40653)$

85 $a_1^{85} = (43.71602,10.41420)\ a_2^{85} = (43.72749,10.42690)\ a_3^{85} = (43.71168,10.42055)$

86 $a_1^{86} = (43.72330,10.39212)\ a_2^{86} = (43.72012,10.39664)\ a_3^{86} = (43.71903,10.38603)$

87 $a_1^{87} = (43.71591,10.39847)\ a_2^{87} = (43.70986,10.39365)\ a_3^{87} = (43.70912,10.40131)$

88 $a_1^{88} = (43.71296,10.40127)\ a_2^{88} = (43.70912,10.40131)$

89 $a_1^{89} = (43.72821,10.39396)\ a_2^{89} = (43.72731,10.39267)\ a_3^{89} = (43.72397,10.38362)$

90	$a_1^{90} = (43.71010,10.38334)$ $a_2^{90} = (43.71335,10.38872)$ $a_3^{90} = (43.71726,10.38916)$ $a_4^{90} = (43.72278,10.39222)$ $a_5^{90} = (43.72440,10.40236)$
91	$a_1^{91} = (43.72322,10.37953)$ $a_2^{91} = (43.71746,10.38265)$ $a_3^{91} = (43.72003,10.38677)$
92	$a_1^{92} = (43.71564,10.37992)$ $a_2^{92} = (43.72631,10.37824)$
93	$a_1^{93} = (43.72464,10.39140)$ $a_2^{93} = (43.72489,10.39253)$ $a_3^{93} = (43.71971,10.39739)$ $a_4^{93} = (43.71588,10.40187)$
94	$a_1^{94} = (43.71066,10.40543)$ $a_2^{94} = (43.71059,10.40127)$
95	$a_1^{95} = (43.70604,10.40925)$ $a_2^{95} = (43.70995,10.40173)$ $a_3^{95} = (43.70148,10.42042)$
96	$a_1^{96} = (43.70693,10.38581)$ $a_2^{96} = (43.70805,10.38553)$ $a_3^{96} = (43.71048,10.38778)$
97	$a_1^{97} = (43.71386,10.40186)$ $a_2^{97} = (43.71422,10.40289)$
98	$a_1^{98} = (43.72996,10.41945)$ $a_2^{98} = (43.72729,10.41555)$ $a_3^{98} = (43.72847,10.40269)$
99	$a_1^{99} = (43.69945,10.39177)$ $a_2^{99} = (43.70627,10.39677)$
100	$a_1^{100} = (43.69945,10.39177)$ $a_2^{100} = (43.70627,10.39677)$

Table 4. Recommendation results for the experiment with R_α=1 and S=5.

Driver's path id	*Passenger's path id*	*Order of POIs in shared path*
1	5	$a_1^1 a_2^1 a_1^5 a_2^5 a_3^5 a_4^5 a_3^1$
2	1	$a_1^2 a_1^1 a_2^1 a_3^1 a_2^2 a_3^2 a_4^2 a_5^2$
3	4	$a_1^3 a_2^3 a_1^4 a_3^3 a_2^4 a_4^3$
4	3	$a_1^4 a_1^3 a_2^3 a_3^3 a_4^3 a_2^4$
5	3	$a_1^5 a_1^3 a_2^3 a_2^5 a_3^3 a_3^5 a_4^3 a_4^5$
6	9	$a_1^6 a_2^6 a_1^9 a_2^9 a_3^9 a_4^9 a_3^6$
7	9	$a_1^7 a_2^7 a_1^9 a_2^9 a_3^9 a_3^7 a_4^9 a_4^7 a_5^7$
8	9	$a_1^8 a_1^9 a_2^8 a_2^9 a_3^9 a_4^9 a_3^8$
9	6	$a_1^9 a_1^6 a_2^6 a_2^9 a_3^9 a_3^6 a_4^9$
10	8	$a_1^{10} a_1^8 a_2^8 a_3^8 a_2^{10}$
11	41	$a_1^{11} a_1^{41} a_2^{41} a_2^{11}$
12	6	$a_1^{12} a_2^{12} a_1^6 a_2^6 a_3^6 a_3^{12}$
13	41	$a_1^{13} a_1^{41} a_2^{41} a_2^{13} a_3^{13} a_4^{13}$
14	41	$a_1^{14} a_1^{41} a_2^{14} a_2^{41} a_3^{14} a_4^{14}$
15	14	$a_1^{15} a_1^{14} a_2^{14} a_3^{14} a_4^{14} a_2^{15}$
16	18	$a_1^{16} a_2^{16} a_1^{18} a_2^{18} a_3^{18} a_3^{16}$
17	90	$a_1^{17} a_1^{90} a_2^{90} a_2^{17} a_3^{90} a_3^{17} a_4^{90} a_4^{17} a_5^{90} a_5^{17}$
18	1	$a_1^{18} a_2^{18} a_1^1 a_2^1 a_3^1 a_3^{18}$
19	18	$a_1^{19} a_1^{18} a_2^{18} a_3^{18} a_2^{19} a_3^{19}$
20	1	$a_1^{20} a_2^{20} a_1^1 a_2^1 a_3^1 a_3^{20} a_4^{20}$
21	24	$a_1^{21} a_1^{24} a_2^{24} a_3^{24} a_2^{21}$
22	38	$a_1^{22} a_2^{22} a_3^{22} a_1^{38} a_2^{38} a_4^{22}$
23	41	$a_1^{23} a_2^{23} a_1^{41} a_2^{41} a_3^{23}$
24	21	$a_1^{24} a_1^{21} a_2^{21} a_2^{24} a_3^{24}$
25	55	$a_1^{25} a_2^{25} a_1^{55} a_2^{55} a_4^{55} a_3^{55} a_3^{25}$
26	99	$a_1^{26} a_1^{99} a_2^{99} a_2^{26}$
27	44	$a_1^{27} a_1^{44} a_2^{44} a_3^{44} a_2^{27} a_3^{27}$
28	44	$a_1^{28} a_1^{44} a_2^{44} a_3^{44} a_2^{28}$

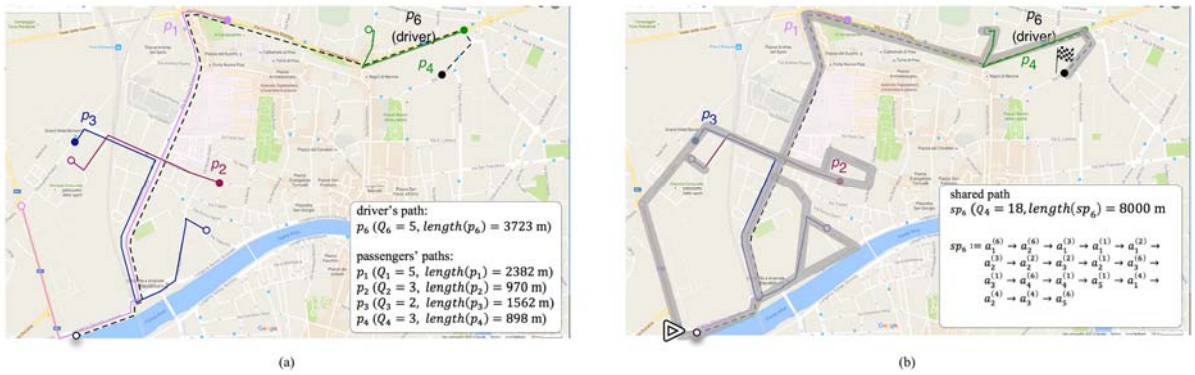
29	44	$a_1^{29}a_1^{44}a_2^{44}a_3^{44}a_2^{29}$
30	27	$a_1^{30}a_2^{30}a_1^{27}a_2^{27}a_3^{27}a_3^{30}a_4^{30}a_5^{30}$
31	93	$a_1^{31}a_1^{93}a_2^{93}a_2^{31}a_3^{93}a_4^{93}a_3^{31}a_4^{31}$
32	46	$a_1^{32}a_1^{46}a_2^{46}a_3^{46}a_2^{32}$
33	83	$a_1^{33}a_1^{83}a_2^{33}a_2^{83}a_3^{33}$
34	84	$a_1^{34}a_1^{84}a_2^{34}a_2^{84}a_3^{84}a_3^{34}a_4^{84}a_4^{34}$
35	12	$a_1^{35}a_1^{12}a_2^{12}a_3^{12}a_2^{35}a_3^{35}$
36	1	$a_1^{36}a_1^{1}a_2^{1}a_3^{1}a_2^{36}a_3^{36}a_4^{36}$
37	41	$a_1^{37}a_1^{41}a_2^{37}a_3^{37}a_2^{41}a_4^{37}$
38	21	$a_1^{38}a_1^{21}a_2^{21}a_2^{38}$
39	3	$a_1^{39}a_1^{3}a_2^{3}a_3^{3}a_2^{39}a_4^{3}a_3^{39}a_4^{39}$
40	61	$a_1^{40}a_1^{61}a_2^{61}a_2^{40}a_3^{61}a_3^{40}a_4^{61}a_4^{40}$
41	62	$a_1^{41}a_1^{62}a_2^{62}a_2^{41}$
42	63	$a_1^{42}a_1^{63}a_2^{63}a_2^{42}a_3^{63}a_3^{42}$
43	1	$a_1^{43}a_1^{1}a_2^{1}a_3^{1}a_2^{43}a_3^{43}$
44	65	$a_1^{44}a_1^{65}a_2^{65}a_2^{44}a_3^{65}a_3^{44}$
45	66	$a_1^{45}a_1^{66}a_2^{66}a_2^{45}$
46	67	$a_1^{46}a_1^{67}a_2^{67}a_2^{46}a_3^{67}a_3^{46}$
47	68	$a_1^{47}a_1^{68}a_2^{47}a_3^{47}a_4^{47}a_2^{68}a_3^{68}a_5^{47}$
48	69	$a_1^{48}a_1^{69}a_2^{48}a_2^{69}a_3^{48}$
49	70	$a_1^{49}a_1^{70}a_2^{49}a_2^{70}a_3^{70}a_3^{49}$
50	71	$a_1^{50}a_1^{71}a_2^{50}a_2^{71}a_3^{50}a_3^{71}a_4^{71}a_4^{50}$
51	72	$a_1^{51}a_1^{72}a_2^{51}a_2^{72}a_3^{51}a_3^{72}a_4^{51}$
52	69	$a_1^{52}a_2^{52}a_1^{69}a_2^{69}a_3^{52}a_4^{52}$
53	59	$a_1^{53}a_2^{53}a_1^{59}a_2^{59}a_3^{59}a_3^{53}$
54	55	$a_1^{54}a_1^{55}a_2^{55}a_4^{55}a_3^{55}a_2^{54}a_3^{54}a_4^{54}$
55	59	$a_1^{55}a_2^{55}a_4^{55}a_1^{59}a_2^{59}a_3^{59}a_3^{55}$
56	55	$a_1^{56}a_2^{56}a_3^{56}a_4^{56}a_5^{56}a_1^{55}a_2^{55}a_4^{55}a_3^{55}a_6^{56}$
57	24	$a_1^{57}a_1^{24}a_2^{24}a_3^{24}a_2^{57}a_3^{57}$
58	59	$a_1^{58}a_1^{59}a_2^{59}a_2^{58}a_3^{59}a_3^{58}$
59	53	$a_1^{59}a_1^{53}a_2^{53}a_2^{59}a_3^{53}a_3^{59}$
60	54	$a_1^{60}a_1^{54}a_2^{54}a_3^{54}a_4^{54}a_2^{60}a_3^{60}$
61	40	$a_1^{61}a_1^{40}a_2^{61}a_2^{40}a_3^{61}a_3^{40}a_4^{40}a_4^{61}$
62	41	$a_1^{62}a_1^{41}a_2^{41}a_2^{62}$
63	42	$a_1^{63}a_1^{42}a_2^{63}a_2^{42}a_3^{42}a_3^{63}$
64	91	$a_1^{64}a_1^{91}a_2^{91}a_3^{91}a_2^{64}a_3^{64}$
65	44	$a_1^{44}a_1^{65}a_2^{65}a_2^{44}a_3^{44}a_3^{65}$
66	45	$a_1^{66}a_1^{45}a_2^{45}a_2^{66}$
67	46	$a_1^{67}a_1^{46}a_2^{67}a_2^{46}a_3^{46}a_3^{67}$
68	47	$a_1^{68}a_1^{47}a_2^{47}a_3^{47}a_4^{47}a_2^{68}a_5^{47}a_3^{68}$
69	48	$a_1^{69}a_1^{48}a_2^{48}a_3^{48}a_2^{69}$
70	49	$a_1^{70}a_1^{49}a_2^{49}a_2^{70}a_3^{49}a_3^{70}$
71	50	$a_1^{71}a_1^{50}a_2^{50}a_2^{71}a_3^{50}a_3^{71}a_4^{50}a_4^{71}$

72	51	$a_1^{72}a_1^{51}a_2^{51}a_2^{72}a_3^{51}a_4^{51}a_3^{72}$
73	52	$a_1^{52}a_1^{73}a_2^{52}a_2^{73}a_3^{52}a_3^{73}a_4^{52}a_4^{73}$
74	80	$a_1^{74}a_1^{80}a_2^{80}a_3^{80}a_2^{74}$
75	74	$a_1^{75}a_2^{75}a_1^{74}a_2^{74}a_3^{75}$
76	80	$a_1^{76}a_2^{76}a_1^{80}a_2^{80}a_3^{80}a_3^{76}$
77	76	$a_1^{77}a_2^{77}a_3^{77}a_1^{76}a_2^{76}a_3^{76}a_4^{77}$
78	57	$a_1^{78}a_1^{57}a_2^{78}a_2^{57}a_3^{57}a_3^{78}$
79	80	$a_1^{79}a_1^{80}a_2^{80}a_3^{80}a_2^{79}a_3^{79}$
80	74	$a_1^{80}a_1^{74}a_2^{80}a_2^{74}a_3^{80}$
81	89	$a_1^{81}a_1^{89}a_2^{89}a_2^{81}a_3^{89}a_3^{81}$
82	32	$a_1^{82}a_1^{32}a_2^{32}a_2^{82}$
83	33	$a_1^{83}a_1^{33}a_2^{33}a_3^{33}a_2^{83}$
84	34	$a_1^{84}a_1^{34}a_2^{34}a_2^{84}a_3^{84}a_3^{34}a_4^{34}a_4^{84}$
85	69	$a_1^{85}a_1^{69}a_2^{69}a_2^{85}a_3^{85}$
86	64	$a_1^{86}a_1^{64}a_2^{64}a_3^{64}a_2^{86}a_3^{86}$
87	64	$a_1^{87}a_1^{64}a_2^{64}a_3^{64}a_2^{87}a_3^{87}$
88	38	$a_1^{38}a_1^{88}a_2^{38}a_2^{88}$
89	81	$a_1^{89}a_1^{81}a_2^{89}a_2^{81}a_3^{81}a_3^{89}$
90	17	$a_1^{90}a_1^{17}a_2^{90}a_2^{17}a_3^{90}a_3^{17}a_4^{90}a_4^{17}a_5^{17}a_5^{90}$
91	92	$a_1^{91}a_1^{92}a_2^{92}a_2^{91}a_3^{91}$
92	91	$a_1^{92}a_1^{91}a_2^{91}a_3^{91}a_2^{92}$
93	64	$a_1^{93}a_2^{93}a_1^{64}a_2^{64}a_3^{64}a_3^{93}a_4^{93}$
94	21	$a_1^{94}a_1^{21}a_2^{21}a_2^{94}$
95	88	$a_1^{95}a_1^{88}a_2^{88}a_2^{95}a_3^{95}$
96	23	$a_1^{96}a_1^{23}a_2^{23}a_2^{96}a_3^{23}a_3^{96}$
97	24	$a_1^{97}a_1^{24}a_2^{24}a_3^{24}a_2^{97}$
98	76	$a_1^{98}a_2^{98}a_1^{76}a_2^{76}a_3^{76}a_3^{98}$
99	26	$a_1^{99}a_1^{26}a_2^{26}a_2^{99}$
100	26	$a_1^{100}a_1^{26}a_2^{26}a_2^{100}$

Table 5. Averaged results for the multi-passenger ridesharing recommendations by varying the number of passengers R_α and the size of the pool *S*.

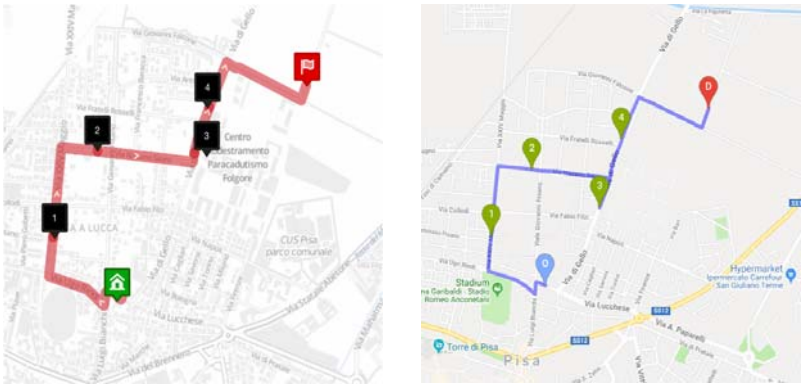
R_α	S	Average shared path's length [m] over 100 paths	Average dissimilarity [m] over 100 paths	Average mileage saving MS [m] over Z paths (Z)
1	10	2185	604	1129 (70)
	9	2195	614	1134 (70)
	8	2281	700	1090 (65)
	7	2303	722	1103 (64)
	6	2318	737	1121 (63)
	5	2476	895	1054 (57)
	4	2583	1002	1107 (51)
	3	2707	1126	1116 (49)
	2	2842	1261	1101 (48)
	1	3236	1655	1103 (48)
2	10	3178	1597	1245 (58)
	9	3205	1624	1256 (57)
	8	3331	1751	1214 (55)

	7	3409	1828	1113 (54)
	6	3451	1870	1072 (56)
	5	3847	2266	995 (48)
	4	3999	2418	987 (46)
	3	4218	2637	903 (45)
	2	4907	3326	824 (36)
3	10	3918	2337	1563 (67)
	9	3955	2374	1533 (68)
	8	4223	2642	1400 (69)
	7	4324	2743	1278 (68)
	6	4474	2893	1230 (69)
	5	4937	3356	1121 (64)
	4	5172	3591	1223 (51)
	3	6137	4556	982 (39)
4	10	4870	3289	1624 (76)
	9	4927	3346	1665 (76)
	8	5312	3731	1607 (70)
	7	5580	3999	1476 (64)
	6	5882	4301	1625 (63)
	5	6585	5004	1307 (56)
	4	7746	6165	1052 (49)



(a) (b)

Fig. 7. Example of multi-passenger ridesharing. (a) The driver (user of path p_6, dashed line) and $R_6 = 4$ passengers (the users of paths p_1, p_2, p_3, $p_4 \in \Omega_6$, continuous lines). Empty dots and full dots indicate the starting point and the ending point of each path, respectively. (b) The resulting shared path sp_6 (bold grey semi-transparent line) and the visiting order of the POIs in sp_6. The start symbol and the checkered flag indicate the starting point and the ending point of the shared path suggested to the driver. The map is provided by Google® Maps.



(a)

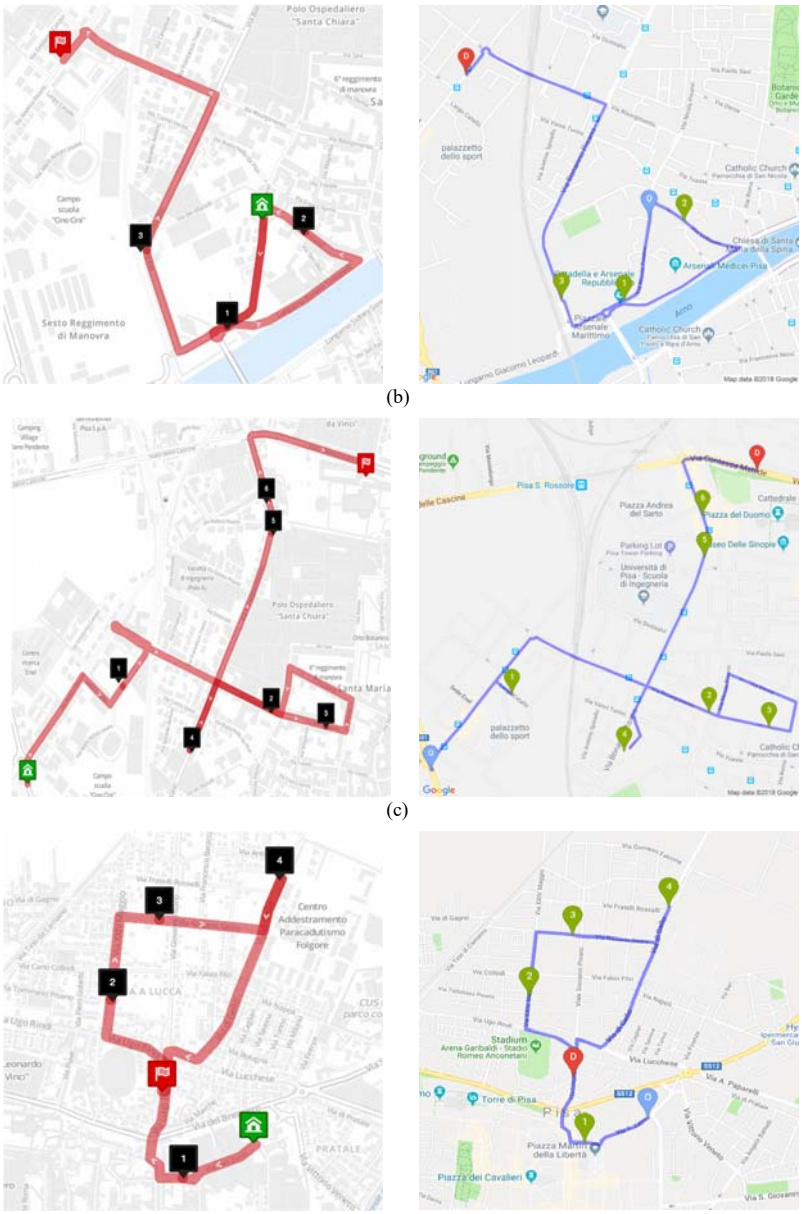


Fig. 8. Four examples of exact matches involving (a) driver of path #58 and passenger of path #59, (b) driver of path #3 and passenger of path #4, (c) driver of path #2 and passenger of path #1, (d) driver of path #76 and passenger of path #80. On the left, route generated by the RouteXL service from the initial POI (house label) to the final POI (flag label). On the right, shared path recommended by our system from the initial POI (O label) to the final POI (D label). In both figures the number on each intermediate POI identifies the visiting order.

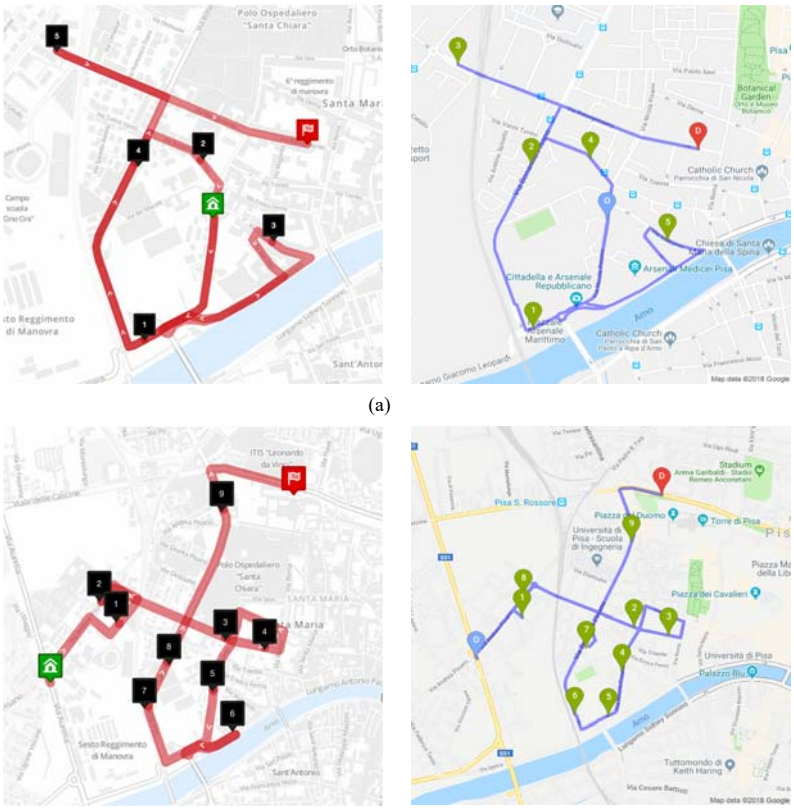


Fig. 9. Two examples of inexact matches involving (a) driver of path #5 and passenger of path #43, (b) driver of path #2 and passengers of paths #1 and #3. On the left, route generated by the RouteXL service from the initial POI (house label) to the final POI (flag label). On the right, shared path recommended by our system from the initial POI (O label) to the final POI (D label). In both figures the label number on each intermediate POI identifies the visiting order within the path.

Appendix A. Generation of the DAG

In the following we describe more formally how to build the single-passenger DAG and the multi-passenger DAG.

A.1 How to build a single-passenger DAG

For reasons of readability, we will index with *x* all that is concerned with the driver's path, and with *y* all that is concerned with the passenger's path, with the aim of maintaining the notation consistent with that of nodes $X_{i,j}$ and $Y_{i,j}$. Thus, we define $DAG_{x,y} = (\nu, \varepsilon)$, where ν is the set of nodes, and ε is the set of edges, i.e., ordered pairs of nodes. The set of nodes ν is defined as $\nu = \nu_X \cup \nu_Y$, where ν_X is the set of nodes of type $X_{i,j}$:

$$\nu_X = \{X_{i,j} | (i,j) = \{(i = 1, j = 0) \vee (i = Q_x, j = Q_y) \vee (i = 2, \dots, Q_x - 1, j = 0, \dots, Q_y, \text{if } Q_x \geq 3)\}\},$$

and ν_Y is the set of nodes of type $Y_{i,j}$:

$\nu_Y = \{Y_{i,j} | (i,j) = (i = 1, \dots, Q_x - 1, j = 1, \dots, Q_y\}$.

More precisely, the nodes of type $X_{i,j}$ are defined for the following sets of values of *i* and *j*:

- $i = 1$, and $j = 0$ corresponding to the source node of the DAG,
- $i = Q_x$, and $j = Q_y$, corresponding to the sink node of the DAG, and
- $i = 2, \dots, Q_x - 1$, and $j = 0, \dots, Q_y$ corresponding to the other nodes. This set is valid only if $Q_x \geq 3$;

while the nodes of type $Y_{i,j}$ are defined for the following values of *i* and *j*:

- $i = 1, \dots, Q_x - 1$, and $j = 1, \dots, Q_y$.

The order of the DAG, i.e., the total number of nodes $|\nu|$, is defined as:

$$|\nu| = |\nu_X| + |\nu_Y| = [(Q_x - 2) \cdot (Q_y + 1) + 2] + [(Q_x - 1) \cdot Q_y] = 2 \cdot Q_x \cdot Q_y + Q_x - 3 \cdot Q_y \quad \text{(A.1)}$$

being $|\nu_X|$ and $|\nu_Y|$ the number of nodes of type $X_{i,j}$, and of type $Y_{i,j}$, respectively.

The set of edges $\varepsilon = \varepsilon_I \cup \varepsilon_{II} \cup \varepsilon_{III} \cup \varepsilon_{IV}$ is built according to the rules defined in Table A.1, with $\varepsilon_I, \varepsilon_{II}, \varepsilon_{III}, \varepsilon_{IV}$ referring to four different sets of edge. More in detail, each type of edge is built according to a set of specific values for *i* and *j*. Sometimes an additional condition on the number of POIs Q_x of the driver's path is required. In Table A.1, conditions (*) derive from the values assumed by index *i*, while conditions (§) are needed to avoid moving between two POIs of the driver's path in the case of $Q_x = 2$. In this case, in fact, the edges of type $X_{i,j} \rightarrow X_{i+1,j}$ do not exist in the DAG, and there is only one possible shared path: the one starting in node $X_{1,0}$, visiting in sequence the nodes of the passenger's path, and ending in node X_{Q_x,Q_y}.

The size of the single-passenger DAG, i.e., the number of edges, corresponds to the sum of the number of edges by type, $|\varepsilon| = |\varepsilon_I| + |\varepsilon_{II}| + |\varepsilon_{III}| + |\varepsilon_{IV}|$. These values are shown in Table A.2. E.g., in the case of $Q_x \geq 4$, the number of edges is $|\varepsilon| = 4 \cdot Q_x \cdot Q_y - 8 \cdot Q_y + 2$, while in the case of $Q_x = 2$, $|\varepsilon| = Q_x + (Q_x - 1) \cdot (Q_y - 1)$.

A.2 How to build a multiple-passenger DAG

Hereafter we describe in detail how to build the *g*-th DAG in the case of a multiple passengers ridesharing. This DAG is a generalization of the single-passenger DAG. For the sake of simplicity, as done before, we will index with *x* all that is concerned with the driver's path p_α, and with y^r, $r = 1, \dots, R_x$, all that is concerned with the *r*-th passenger's path of the *g*-th group $\Gamma_g^{(\alpha)}$ taken into account. Thus the ride-share is defined in terms of the driver's path p_x and a set of R_x passengers' paths $p_{y^r}, \forall r \in \{1, \dots, R_x\}$.

The DAG corresponding to the ride-share has two kinds of nodes: the nodes of kind $X_{i,j_1,\dots,j_r,\dots,j_{R_x}}$, and the nodes of kind ${Y^r}_{i,j_1,\dots,j_r,\dots,j_{R_x}}$, defined $\forall r \in \{1, \dots, R_x\}$. The former nodes are associated with the POIs of the user acting as driver of the shared path, while the latter nodes are associated with the POIs of the users acting as passengers. The superscript *r* in node $Y^r_{(\dots)}$ indicates the corresponding *r*-th passenger. The indices $(i, j_1, \dots, j_r, \dots, j_{R_x})$, indicate the state of progress achieved in completing the shared path on all the paths taken into account in the ride-share. More precisely, the index i refers to the progress related to the driver's path, while each index j_r

($r = 1, \dots, R_x$) refers to the progress related to the r-th passenger path. More in detail, staying in node $X_{i,j_1,\dots,j_r,\dots,j_{R_x}}$ of the DAG means that the shared path has:

i) just reached the POI $a_i^{(x)}$ of the driver's path p_x,

ii) already visited all the previous POIs $a_1^{(x)}, \dots, a_{i-1}^{(x)}$ of path p_x, and

iii) already visited the POIs $a_1^{(y^r)}, \dots, a_{j_r}^{(y^r)}$ of the passenger's path p_{y^r}, $\forall r = 1, \dots, R_x$.

Similarly, staying in node $Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}}$, with $\tau \in \{1, \dots, R_x\}$ means that the shared path has:

i) just reached the POI $a_{j_\tau}^{(y^\tau)}$ of path p_{y^τ},

ii) already visited the previous POIs $a_1^{(y^\tau)}, \dots, a_{j_\tau-1}^{(y^\tau)}$ of path p_{y^τ},

iii) already visited the POIs $a_1^{(x)}, \dots, a_i^{(x)}$ of path p_x, and

iv) already visited the previous POIs $a_1^{(y^r)}, \dots, a_{j_r}^{(y^r)}$ of path p_{y^r}, $\forall r = 1, \dots, R_x$, with $r \neq \tau$.

The edges of the DAG are built according to a set of rules valid for specified subsets of values for $(i, j_1, \dots, j_r, \dots, j_{R_x})$, as explained later. In the case of the multi-passenger DAG, we have five types of edge instead of four:

i) $X_{i,j_1,\dots,j_r,\dots,j_{R_x}} \rightarrow X_{i+1,j_1,\dots,j_r,\dots,j_{R_x}}$, with weight $dist(a_i^{(x)}, a_{i+1}^{(x)})$, is defined between two POIs of the driver's path p_x;

ii) $X_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \rightarrow Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau+1,\dots,j_{R_x}}$, with weight $dist(a_i^{(x)}, a_{j_\tau+1}^{(y^\tau)})$, is defined between one POI of the driver's path p_x and one POI of a passenger's path p_{y^τ} with $\tau \in \{1, \dots, R_x\}$;

iii) $Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \rightarrow Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau+1,\dots,j_{R_x}}$, with weight $dist(a_{j_\tau}^{(y^\tau)}, a_{j_\tau+1}^{(y^\tau)})$, is defined between two POIs of the same passenger's path p_{y^τ} with $\tau \in \{1, \dots, R_x\}$;

iv) $Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_\omega,\dots,j_{R_x}} \rightarrow Y^{\omega}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_\omega+1,\dots,j_{R_x}}$, with weight $dist(a_{j_\tau}^{(y^\tau)}, a_{j_\omega+1}^{(y^\omega)})$, is defined between one POI of the passenger's path p_{y^τ} and one POI of another passenger's path p_{y^ω} with $\tau, \omega \in \{1, \dots, R_x\}, \tau \neq \omega$;

v) $Y^{\tau}{}_{i,j,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \rightarrow X_{i+1,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}}$, with weight $dist(a_{j_\tau}^{(y^\tau)}, a_{i+1}^{(x)})$ is defined between one POI of the driver's path p_x and one POI of a passenger's path p_{y^τ} with $\tau \in \{1, \dots, R_x\}$.

The DAG is defined as $DAG_{x,\{y|p_y \epsilon \Gamma_g^{(x)}\}} = (\nu, \varepsilon)$, where $\nu = \nu_X \cup \nu_Y$ is the set of nodes, and ε is the set of edges. The order of the DAG is $|\nu| = |\nu_X| + |\nu_Y|$, where $|\nu_X|$ is the number of nodes of kind $X_{i,j_1,\dots,j_r,\dots,j_{R_x}}$, and $|\nu_Y|$ is the number of nodes of kind $Y^r_{i,j_1,\dots,j_r,\dots,j_{R_x}}$ $\forall r \in \{1, \dots, R_x\}$. Thus, $|\nu_X|$ is defined as:

$$|\nu_X| = (Q_x - 2) \cdot \prod_{r=1}^{R_x} (Q_{y^r} + 1) + 2, \tag{A.2}$$

where Q_x is the number of POIs of p_x, and Q_{y^r} is the number of POIs of p_{y^r}, $\forall r \in \{1, \dots, R_x\}$; $|\nu_Y|$ is defined as:

$$|\nu_Y| = \sum_{r=1}^{R_x} |\nu_Y^r|, \tag{A.3}$$

being $|\nu_Y^\tau|$, with $r = \tau, \tau \in \{1, \dots, R_x\}$, defined as:

$$|v_Y^\tau| = (Q_x - 1) \cdot Q_{y^\tau} \cdot \prod_{r=1,\ r\neq\tau}^{R_x} (Q_{y^r} + 1). \quad \text{(A.4)}$$

Equations (A.2)-(A.4) are a generalization of Equation (A.1).

The nodes of type $X_{i,j_1,\dots,j_r,\dots,j_{R_x}}$ are defined for the following sets of values of *i* and *j*:

- $i = 1$, and $j_r = 0, \forall r = 1, \dots, R_x$, corresponding to the source node of the DAG,
- $i = Q_x$, and $j_r = Q_{y^r}, \forall r = 1, \dots, R_x$ corresponding to the destination node of the DAG, and
- $i = 2, \dots, Q_x - 1$, and $j_r = 0, \dots, Q_{y^r}, \forall r = 1, \dots, R_x$, corresponding to the other nodes.

The node of type $Y^\tau{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}}$, $\forall \tau \in \{1, \dots, R_x\}$ are defined for the following values of *i* and *j*:

- $i = 1, \dots, Q_x - 1$, and $j_r = \begin{cases} 1, \dots, Q_{y^r}, & (r = \tau) \\ 0, \dots, Q_{y^r}, & (\forall\, r \neq \tau) \end{cases}$.

The size of the multi-passenger DAG is the sum of the number of edges by type, $|\varepsilon| = |\varepsilon| + |\varepsilon_{\text{II}}| + |\varepsilon_{\text{III}}| + |\varepsilon_{\text{IV}}| + |\varepsilon_{\text{V}}|$. These values are shown in Table A.3. E.g., in the case of $Q_x = 2$, and $R_x = 2$, the number of edges is $|\varepsilon| = 4 \cdot Q_{y^1} \cdot Q_{y^2} + 2$.

The set of edges $\varepsilon = \varepsilon_I \cup \varepsilon_{II} \cup \varepsilon_{III} \cup \varepsilon_{IV} \cup \varepsilon_V$ is built according to the rules (i.e., sets of values of *i* and *j*) defined in Table A.4. These rules are a generalization to R_x passengers of the rules in Table A.1. More in detail, each type of edge is built according to the corresponding conditions on the values for the indexes $(i, j_1, \dots, j_r, \dots, j_{R_x})$. In addition, some conditions on the indexes must undergo an additional condition related to the number of POIs of the driver's path Q_x. In Table A.4, conditions (*) derive from the values assumed by index *i*, while conditions (§) are needed to avoid moving between two POIs of the driver's path in the case of $Q_x = 2$. In fact, in this case edges of type $X_{i,j_1,\dots,j_r,\dots,j_{R_x}} \rightarrow X_{i+1,j_1,\dots,j_r,\dots,j_{R_x}}$ cannot exist in the DAG.

Table A.1. Existence rules of the edges in the single-passenger DAG

Edge's type	Set	Edge's weight	Set of values for *i* and *j*	Conditions on Q_x
$X_{i,j} \rightarrow X_{i+1,j}$ (driver's POI→ driver's POI)	ε_{I}	$dist(a_i^{(x)}, a_{i+1}^{(x)})$	$i = Q_x - 1$, and $j = Q_y$	if $Q_x \geq 3$ (§)
			$i = 1$, and $j = 0$	if $Q_x \geq 3$ (§)
			$i = 2, \dots, Q_x - 2$, and $j = 0, \dots, Q_y$	if $Q_x \geq 4$ (*)
$X_{i,j} \rightarrow Y_{i,j+1}$ (driver's POI→ passenger's POI)	ε_{II}	$dist(a_i^{(x)}, a_{j+1}^{(y)})$	$i = 1, \dots, Q_x - 1$, and $j = 0$	–
			$i = 2, \dots, Q_x - 1$, and $j = 1, \dots, Q_y - 1$	if $Q_x \geq 3$ (*)
$Y_{i,j} \rightarrow X_{i+1,j}$ (passenger's POI→ driver's POI)	ε_{III}	$dist(a_j^{(x)}, a_{i+1}^{(y)})$	$i = Q_x - 1$, and $j = Q_y$	–
			$i = 1, \dots, Q_x - 2$, and $j = 1, \dots, Q_y$	if $Q_x \geq 3$ (*)
$Y_{i,j} \rightarrow Y_{i,j+1}$ (passenger's POI→ passenger's POI)	ε_{IV}	$dist(a_j^{(y)}, a_{j+1}^{(y)})$	$i = 1, \dots, Q_x - 1$, and $j = 1, \dots, Q_y - 1$	–

Table A.2. Number of edges by type in the single-passenger DAG

Edge type	Set	Number of edges
$X_{i,j} \to X_{i+1,j}$ (driver's POI → driver's POI)	ε_{I}	$\lvert\varepsilon_{\mathrm{I}}\rvert = \begin{cases} 0 & (\text{if } Q_x = 2) \\ 2 & (\text{if } Q_x = 3) \\ 2 + (Q_x - 3) \cdot (Q_y + 1) & (\text{if } Q_x \geq 4) \end{cases}$
$X_{i,j} \to Y_{i,j+1}$ (driver's POI → passenger's POI)	$\varepsilon_{\mathrm{II}}$	$\lvert\varepsilon_{\mathrm{II}}\rvert = \begin{cases} Q_x - 1 & (\text{if } Q_x = 2) \\ Q_x - 1 + (Q_x - 2) \cdot (Q_y - 1) & (\text{if } Q_x \geq 3) \end{cases}$
$Y_{i,j} \to X_{i+1,j}$ (passenger's POI → driver's POI)	$\varepsilon_{\mathrm{III}}$	$\lvert\varepsilon_{\mathrm{III}}\rvert = \begin{cases} 1 & (\text{if } Q_x = 2) \\ 1 + (Q_x - 2) \cdot Q_y & (\text{if } Q_x \geq 3) \end{cases}$
$Y_{i,j} \to Y_{i,j+1}$ (passenger's POI → passenger's POI)	$\varepsilon_{\mathrm{IV}}$	$\lvert\varepsilon_{\mathrm{IV}}\rvert = (Q_x - 1) \cdot (Q_y - 1)$

Table A.3. Number of edges by type in the multi-passenger DAG

Edge type	Set	Number of edges
$X_{i,j_1,\dots,j_r,\dots,j_{R_x}} \to X_{i+1,j_1,\dots,j_r,\dots,j_{R_x}}$ (driver's POI → driver's POI)	ε_{I}	$\lvert\varepsilon_{\mathrm{I}}\rvert = \begin{cases} 0 & (\text{if } Q_x = 2) \\ 2 & (\text{if } Q_x = 3) \\ 2 + (Q_x - 3) \cdot \prod_{r=1}^{R_x} (Q_{y^r} + 1), & (\text{if } Q_x \geq 4) \end{cases}$
$X_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \to Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau+1,\dots,j_{R_x}}$ (driver's POI → passenger τ 's POI)	$\varepsilon_{\mathrm{II}}$	$\lvert\varepsilon_{\mathrm{II}}\rvert = \begin{cases} 1 & (\text{if } Q_x = 2) \\ 1 + (Q_x - 2) \cdot Q_{y^\tau} \cdot \prod_{r=1,\ r\neq\tau}^{R_x} (Q_{y^r} + 1) & (\text{if } Q_x \geq 3) \end{cases}$, $\tau \in \{1, \dots, R_x\}$
$Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \to Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau+1,\dots,j_{R_x}}$ (passenger τ 's POI → passenger τ 's POI)	$\varepsilon_{\mathrm{III}}$	$\lvert\varepsilon_{\mathrm{III}}\rvert = (Q_x - 1) \cdot (Q_{y^\tau} - 1) \cdot \prod_{r=1,\ r\neq\tau}^{R_x} (Q_{y^r} + 1), \quad \tau \in \{1, \dots, R_x\}$
$Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_\omega,\dots,j_{R_x}} \to Y^{\omega}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_\omega+1,\dots,j_{R_x}}$ (passenger τ 's POI → passenger ω 's POI)	$\varepsilon_{\mathrm{IV}}$	$\lvert\varepsilon_{\mathrm{IV}}\rvert = (Q_x - 1) \cdot Q_{y^\tau} \cdot Q_{y^\omega} \cdot \prod_{r=1,\ r\neq\tau\neq\omega}^{R_x} (Q_{y^r} + 1)$, $\tau, \omega \in \{1, \dots, R_x\}, \tau \neq \omega$
$Y^{\tau}{}_{i,j,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \to X_{i+1,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}}$ (passenger τ 's POI → driver 's POI)	ε_{V}	$\lvert\varepsilon_{\mathrm{V}}\rvert = \begin{cases} 1 & (\text{if } Q_x = 2) \\ 1 + (Q_x - 2) \cdot Q_{y^\tau} \cdot \prod_{r=1,\ r\neq\tau}^{R_x} (Q_{y^r} + 1) & (\text{if } Q_x \geq 3)' \end{cases}$ $\tau \in \{1, \dots, R_x\}$

Table A.4. Existence rules of the edges in the multi-passenger DAG

Edge type	Edge's weight	Sets of values for i and j	Conditions on Q_x
$X_{i,j_1,\dots,j_r,\dots,j_{R_x}} \to X_{i+1,j_1,\dots,j_r,\dots,j_{R_x}}$ (driver's POI → driver's POI)	$dist(a_i^{(x)}, a_{i+1}^{(x)})$	$i = Q_x - 1;\ j_r = Q_{y^r}, \forall r \in \{1, \dots, R_x\};$	if $Q_x \geq 3$ (§)
		$i = 1;\ j_r = 0\ , \forall r \in \{1, \dots, R_x\};$	if $Q_x \geq 3$ (§)
		$i = 2, \dots, Q_x - 2;\ j_r = 0, \dots, Q_{y^r}, \forall r \in \{1, \dots, R_x\};$	if $Q_x \geq 4$ (*)
$X_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \to Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau+1,\dots,j_{R_x}}$ (driver's POI → passenger τ 's POI)	$dist(a_i^{(x)}, a_{j_\tau+1}^{(y^\tau)})$	$i = 1;\ j_r = 0, \forall r \in \{1, \dots, R_x\};$	–
		$i = 2, \dots, Q_x - 1;\ j_r = \begin{cases} 0, \dots, Q_{y^r} - 1, (r = \tau) \\ 0, \dots, Q_{y^r}, (\forall\, r \neq \tau) \end{cases}$, $r \in \{1, \dots, R_x\};$	if $Q_x \geq 3$ (*)
$Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_x}} \to Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau+1,\dots,j_{R_x}}$ (passenger τ 's POI → passenger τ 's POI)	$dist(a_{j_\tau}^{(y^\tau)}, a_{j_\tau+1}^{(y^\tau)})$,	$i = 1, \dots, Q_x - 1;\ j_r = \begin{cases} 1, \dots, Q_{y^r} - 1, (r = \tau) \\ 0, \dots, Q_{y^r}, (\forall r \neq \tau) \end{cases}$, $r \in \{1, \dots, R_x\};$	–
$Y^{\tau}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_\omega,\dots,j_{R_x}} \to Y^{\omega}{}_{i,j_1,\dots,j_r,\dots,j_\tau,\dots,j_\omega+1,\dots,j_{R_x}}$ (passenger τ 's POI → passenger ω 's	$dist(a_{j_\tau}^{(y^\tau)}, a_{j_\omega+1}^{(y^\omega)})$,	$i = 1, \dots, Q_x - 1;\ j_r = \begin{cases} 1, \dots, Q_{y^r}, (r = \tau) \\ 0, \dots, Q_{y^r} - 1, (r = \omega) \\ 0, \dots, Q_{y^r}, (\forall r \neq \tau \neq \omega) \end{cases}$,	–

POI)		$r \in \{1, \dots, R_x\}$;	
$Y^{\tau}{}_{i,j,\dots,j_r,\dots,j_\tau,\dots,j_{R_X}} \rightarrow X_{i+1,j_1,\dots,j_r,\dots,j_\tau,\dots,j_{R_X}}$ (passenger τ 's POI → driver 's POI)	$dist(a_{j_\tau}^{(y^\tau)}, a_{i+1}^{(x)})$	$i = Q_x - 1;\ \ j_r = Q_{y^r}, \forall r \in \{1, \dots, R\};$	–
		$i = 1, \dots, Q_x - 2;\ \ j_r = \begin{cases} 1, \dots, Q_{y^r}\ (r = \tau) \\ 0, \dots, Q_{y^r}\ (\forall r \neq \tau) \end{cases}$, $r \in \{1, \dots, R_x\}$;	if $Q_x \geq 3$ (*)

Appendix B. Computational complexity of the elaboration

In this appendix, we present the approximated computational complexity for each phase (expressed using the "Big O" notation).

B.1 Computational complexity of the "Preprocessing" phase

The complexity of the subroutine 1.1 depends on the number of paths P and on the number of POIs Q_α involved in each path. The approximated complexity can be computed considering the same value of $Q = Q_\alpha$ for all paths, and the map-matching operation having negligible complexity. Hence, the complexity of the first subroutine is $O(P) \cdot O(Q) = O(P \cdot Q)$.

The computation of the complexity of subroutine 1.2 is similar to that of subroutine 1.1. By applying the same reasoning, we determine that the complexity of the second subroutine is $O(T \cdot Q)$.

The overall complexity of this phase can be approximated as the sum of the complexities of the composing subroutines, that is, $O(P \cdot Q) + O(T \cdot Q) = O(Q \cdot (T + P))$. Now, T has typically a lower order of magnitude than P (indeed, in the experiments we set T = 10 and P = 100), hence we can state that $(T + P) \approx P$. Further, Q can be considered as a negligible constant (indeed, in the experiments we employed a number of POIs per path ranging from 2 to 5). Thus, we can approximate the complexity of the "Preprocessing" phase to $\approx O(P)$.

B.2 Computational complexity of the "Grouping of paths" phase

In subroutine 2.1 the computation of the matrix of dissimilarity values between each user's path and each prototype path involves a series of operations. For the sake of simplicity, we consider only the most time-consuming operation, that is, the computation of the shortest path on the single-passenger DAG. This operation is O(|ν|+|ε|) with |ν| being the number of nodes and |ε| being the number of edges of the DAG. Now according to the building rules of the single-passenger DAG (see Appendix A.1), O(|ν|) and O(|ε|) can both be approximated with $O(Q^2)$ with Q the number of POIs per path. The shortest path is computed for each pair of user's path and prototype path, thus the overall complexity of subroutine 2.1 is $O(Q^2) \cdot O(P \cdot T) = O(Q^2 \cdot P \cdot T)$.

The complexity of subroutine 2.2 can be approximated with the complexity of the BIRCH algorithm that is $O(P)$.

The overall complexity of the "Grouping of paths" phase can be approximated with $O(Q^2 \cdot P \cdot T) + O(P) \approx O(Q^2 \cdot P \cdot T)$. The simplification is possible as the additive factor $O(P)$ has a lower order of complexity than $O(Q^2 \cdot P \cdot T)$ from the asymptotic point of view.

B.3 Computational complexity of the "Analysis of multi-passenger ride matches" phase

The complexity of subroutine 3.1 depends on the number N_l of paths per cluster, and on the number L of clusters. The approximated complexity can be computed considering, for the sake of simplicity, the same value $N = N_l$ for all clusters. Hence, the complexity of the first subroutine is $O(N^2) \cdot O(L) = O(P \cdot N)$, as $N \cdot L = P$.

To compute the complexity of the subroutine 3.2, we consider only the most time-consuming operation, that is, the computation of the shortest path on the multi-passenger DAG. This operation is O(|ν|+|ε|) with |ν| being the number of nodes and |ε| being the number of edges of the DAG. Now, according to the building rules of the multi-passenger DAG (see Appendix A.2), O(|ν|) and O(|ε|) both depend on the number of POIs Q of each path, and on the number R_α of passengers of the ride-share (for the sake of simplicity we can consider the same value of $R = R_\alpha$ for all users). Thus, $O(|\nu|) \approx O(Q^{R+2})$, and $O(|\varepsilon|) \approx O(Q^{R+3})$. The operation is $O(|\nu|+|\varepsilon|) \approx O(Q^{R+2} + Q^{R+3}) \approx O(Q^{R+3})$. The shortest path is computed for each user and for each group of paths among $C_{S,R}$ possible ones in the user's pool of paths, thus the overall complexity is $O\left(P \cdot C_{S,R}\right) \cdot O(Q^{R+3}) = O(Q^{R+3} \cdot P \cdot C_{S,R})$.

The overall complexity of the "Analysis of multi-passenger ride matches" phase can be approximated with $O(Q^{R+3} \cdot P \cdot C_{S,R}) + O(P \cdot N) \approx O(Q^{R+3} \cdot P \cdot C_{S,R})$, as the additive factor $O(P \cdot N)$ has a lower order of complexity than $O(Q^{R+3} \cdot P \cdot C_{S,R})$ from the asymptotic point of view.

B.4 Computational complexity of the "Ridesharing recommendation" phase

The complexity of subroutine 4.1 depends only on the number P of paths and can be approximated with $O(P)$, which also corresponds to the complexity of the "Ridesharing recommendation" phase. The overall complexity of the ridesharing approach is therefore $O(P) + O(Q^2 \cdot P \cdot T) + O(Q^{R+3} \cdot P \cdot C_{S,R}) + O(P) \approx O(Q^2 \cdot P \cdot T) + O(Q^{R+3} \cdot P \cdot C_{S,R})$. We observe that the complexity depends linearly on P, T and $C_{S,R}$ and exponentially on Q. We have to consider that Q is the number of POIs for one path and typically this number is very small. Further, R is the number of passengers of the ride-share and can assume at most the value of 4. Thus, the weight of Q on the computation of the complexity is comparable with the weight of P, T and $C_{S,R}$.

Appendix C. Glossary

User's POI: the desired pick-up or drop-off location for the user. A *user's path* is an ordered sequence of user's POIs.

Shared path: the path suggested to the driver. The *shared path* visits all the POIs of the paths involved in the ride-share, and respects the ordering constraints between POIs in each path.

Dissimilarity value: the additional travel distance (i.e., the detour) for the driver of the *shared path*. It is computed as the difference between the length of the *shared path* and the length of the driver's path.

Ridesharing recommendation: ride-matching suggestion for the driver of the shared ride. It consists of the best group of passengers' paths, and the corresponding shortest *shared path* to follow.

Multi-passenger ridesharing: ridesharing involving the driver, and at least two other users (i.e., passengers).

Pool of paths: set of candidate paths to be matched with the driver's path. The pool contains the best *group of paths* to recommend to the driver.

Prototype paths: reference paths used to arrange users' paths in groups, for efficiency reasons.

References

[1] European Union (EU). Regulation (EU) No 333/2014 of the European Parliament and of the Council of 11 March 2014 amending Regulation (EC) No 443/2009 to define the modalities for reaching the 2020 target to reduce CO2 emissions from new passenger cars, 2014, http://ec.europa.eu/clima/policies/transport/vehicles/cars/documentation_en.htm. Accessed June 30th, 2017.

[2] Eurostat European Commission. Energy, Transport and Environment Indicators, 2016, 2016, http://ec.europa.eu/eurostat/documents/3217494/7731525/KS-DK-16-001-EN-N.pdf/cc2b4de7-146c-4254-9521-dcbd6e6fafa6. Accessed June 30th, 2017.

[3] Piórkowski, M. (2010) Collaborative transportation systems. *Proc. 2010 IEEE Wireless Communication and Networking Conf., WCNC*, Sydney, Australia, 18-21 April, pp. 1–6, IEEE.

[4] BlaBlaCar, www.blablacar.com/. Accessed June 30th, 2017.

[5] Lin, C., Choy, K.L., Ho, G.T.S., Chung, S.H., and Lam, H.Y. (2014) Survey of green vehicle routing problem: past and future trends. *Expert Systems with Applications,* **41**(4), 1118–1138.

[6] Furuhata, M., Dessouky, M., Ordóñez, F., Brunet, M.-E., Wang, X., and Koenig, S. (2013) Ridesharing: The state-of-the-art and future directions. *Transportation Research Part B: Methodological,* **57**, 28–46.

[7] Uber, www.uber.com/. Accessed June 30th, 2017.

[8] Flinc, www.flinc.org/. Accessed June 30th, 2017.

[9] RideshareOnline, www.rideshareonline.com/. Accessed June 30th, 2017.

[10] Galland, S., Knapen, L., Yasar, A.-U.-H., Gaud, N., Janssens, D., Lamotte, O., Koukam, A., and Wets, G. (2014) Multi-agent simulation of individual mobility behavior in carpooling. *Transportation Research Part C: Emerging Technologies,* **45**, 83–98.

[11] Agatz, N., Erera, A., Savelsbergh, M., and Wang, X. (2012). Optimization for dynamic ride-sharing: A review. *European J. of Operational Research*, **223**, 295–303.

[12] Taghavi, M., Bentahar, J., Bakhtiyari, K., and Hanachi, C. (2017) New insights towards developing recommender systems. *The Computer J.*, **61**(3), 319–348.

[13] Bicocchi, N., and Mamei, M. (2014) Investigating ride sharing opportunities through mobility data analysis. *Pervasive and Mobile Computing,* **14**, 83–94.

[14] Teodorović, D., and Dell'Orco, M. (2008) Mitigating traffic congestion: solving the ride-matching problem by bee colony optimization. *Transp. Planning and Technology,* **31**(2), 135–152.

[15] Escudero, L.F. (1988) An inexact algorithm for the sequential ordering problem. *European J. of Operational Research,* **37**, 236–249.

[16] CarpoolKids, https://carpool-kids.com/. Accessed June 30th, 2017.

[17] HopSkipDrive, http://www.hopskipdrive.com/. Accessed June 30th, 2017.

[18] KidsLyft, http://www.kidslyft.com/. Accessed June 30th, 2017.

[19] Kango, http://www.kangoapp.co/. Accessed June 30th, 2017.

[20] Zum, https://ridezum.com/. Accessed June 30th, 2017.

[21] Pan, G., Qi, G., Zhang, W., Li, S., Wu, Z., and Yang, L.T. (2013) Trace analysis and mining for smart cities: issues, methods, and applications. *IEEE Communications Magazine,* **51**, 120–126.

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

[23] D'Andrea, E., and Marcelloni, F. (2017) Detection of traffic congestion and incidents from GPS trace analysis. *Expert Systems with Applications,* **73**, 43–56.

[24] Zheng, Y., Zhang, L., Xie, X., and Ma, W.-Y. (2009) Mining interesting locations and travel sequences from GPS trajectories. *Proc. 18th Int. Conf. on World Wide Web, WWW*, Madrid, Spain, 20-24 April, pp. 791–800, ACM, New York, NY, USA.

[25] Hung, C.-C., Peng, W.-C., and Lee, W.-C. (2015) Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *The VLDB J.,* **24**, 169–192.

[26] Xiao, X., Zheng, Y., Luo, Q., and Xie, X. (2010) Finding similar users using category-based location history. *Proc. 18th SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems, GIS*, San Jose, CA, USA, 3-5 November, pp. 442–445, ACM, New York, NY, USA.

[27] Li, Y., Han, and J. Yang, J. (2004) Clustering moving objects. *Proc. 10th ACM Int. Conf. Knowledge Discovery and Data Mining, SIGKDD*, Seattle, WA, 22-25 August, pp. 617–622, ACM, New York, NY, USA.

[28] Chen, L., Lv, M., Ye, Q., Chen, G., and Woodward, J. (2011) A personal route prediction system based on trajectory data mining. *Information Sciences,* **181**, 1264–1284.

[29] Rahaman, M.S., Mei, Y., Hamilton, M., and Salim, F.D. (2017) CAPRA: a contour-based accessible path routing algorithm. *Information Sciences,* **385–386**, 157–173.

[30] Lu, Q., Chen, F., and Hancock, K. (2009) On path anomaly detection in a large transportation network. Computers, Environment and Urban Systems. *Spatial Data Mining–Methods and Applications,* **33**, 448–462.

[31] Levenshtein, V. (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics—Doklady,* **10**(10), 707–710.

[32] Chen, L., Özsu, M.T., and Oria, V. (2005) Robust and fast similarity search for moving object trajectories. *Proc. 2005 ACM SIGMOD Int. Conf. on Management of Data, SIGMOD*, Baltimore, MD, USA, 13-17 June, pp. 491–502, ACM, New York, NY.

[33] Vlachos, M., Gunopoulos, D., and Kollios, G. (2002) Discovering similar multidimensional trajectories. *Proc. 18th Int. Conf. on Data Engineering, ICDE*, San Jose, CA, USA, 26 February - 1 March, pp. 673–684, IEEE Computer Society, Washington, DC, USA.

[34] Berndt, D.J., and Clifford, J. (1994) Using dynamic time warping to find patterns in time series. *Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, AAAIWS*, Seattle, WA, 31 July – 1 August, pp. 359–370, AAAI Press.

[35] He, W., Hwang, K., and Li, D. (2014) Intelligent carpool routing for urban ridesharing by mining GPS trajectories. *IEEE Trans. on Intelligent Transportation Systems,* **15**, 2286–2296.

[36] Tian, C., Huang, Y., Liu, Z., Bastani, F., and Jin, R. (2013) Noah: a dynamic ridesharing system. *Proc. 2013 ACM SIGMOD Int. Conf. on Management of Data, SIGMOD*, New York, NY, USA, 22-27 June, pp. 985–988, ACM, New York, NY.

[37] Nourinejad, M., and Roorda, M.J. (2016) Agent based model for dynamic ridesharing. *Transportation Research Part C: Emerging Technologies,* **64**, 117–132.

[38] Herbawi, W., and Weber, M. (2012) Modeling the multihop ridematching problem with time windows and solving it using genetic algorithms. *Proc. 2012 IEEE 24th Int. Conf. on Tools with Artificial Intelligence, ICTAI*, Athens, Greece, 7-9 November, pp. 89–96, ACM, New York, NY, USA.

[39] Cao, B., Alarabi, L., Mokbel, M.F., and Basalamah, A. (2015) SHAREK: a scalable dynamic ride sharing system. *Proc. 2015 16th IEEE Int. Conf. on Mobile Data Management, MDM*, 15-18 June, pp. 4–13, IEEE Computer Society, Washington DC, USA.

[40] Trasarti, R., Pinelli, F., Nanni, M., and Giannotti, F. (2011) Mining mobility user profiles for car pooling. *Proc. 17th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, KDD*, San Diego, CA, USA, 21-24 August, pp. 1190–1198, ACM, New York, NY, USA.

[41] Dimitrakopoulos, G., Demestichas, P., and Koutra, V. (2012) Intelligent management functionality for improving transportation efficiency by means of the car pooling concept. *IEEE Trans. on Intelligent Transportation Systems,* **13**, 424–436.

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

[43] D'Andrea, E., Lorenzo, D.D., Lazzerini, B., Marcelloni, F., and Schoen, F. (2016) Path clustering based on a novel dissimilarity function for ride-sharing recommenders. *Proc. 2016 IEEE Int. Conf. on Smart Computing, SMARTCOMP,* 18-20 May, St. Louis, MI, USA, pp. 1–8, IEEE.

[44] OpenStreetMap, www.openstreetmap.org/. Accessed June 30th, 2017.

[45] Graph Hopper. Route Planner, www.graphhopper.com/. Accessed June 30th, 2017.

[46] Dijkstra, E.W. (1959) A note on two problems in connexion with graphs. *Numerische Mathematik,* **1**, 269–271.

[47] Library algs4, http://algs4.cs.princeton.edu/code. Accessed June 30th, 2017.

[48] Sedgewick, R., and Wayne, K. (2011) *Algorithms*, 4th Edition, Addison-Wesley Professional, Boston, MA.

[49] Zhang, T., Ramakrishnan, R., and Livny, M. (1996) BIRCH: an efficient data clustering method for very large databases. *Proc. 1996 ACM SIGMOD Int. Conf. on Management of Data, SIGMOD*, Montreal, QC, Canada, 4–6 June, pp. 103–114, ACM, New York, NY, USA.

[50] Bernecker, T., Houle, M.E., Kriegel, H.-P., Kröger, P., Renz, M., Schubert, E., and Zimek, A. (2011) Quality of similarity rankings in time series. In: D. Pfoser et al. (eds.), *Advances in Spatial and Temporal Databases* (pp. 422–440). Springer, Berlin, Heidelberg.

[51] RouteXL. Route planner service for multiple destinations, www.routexl.com, Accessed August 20th, 2018.