UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO EM BANCO DE DADOS

EMANUEL EDUARDO DA SILVA OLIVEIRA

TRAJECTME: PLANNING SIGHTSEEING TOURS WITH HOTEL SELECTION FROM TRAJECTORY DATA

FORTALEZA

2018

EMANUEL EDUARDO DA SILVA OLIVEIRA

TRAJECTME: PLANNING SIGHTSEEING TOURS WITH HOTEL SELECTION FROM TRAJECTORY DATA

Dissertação apresentada ao Curso de Mestrado em Banco de Dados do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Banco de Dados. Área de Concentração: Banco de Dados

Orientador: Prof. Dr. José Antônio Fernandes de Macêdo

Coorientador: Dr. Igo Ramalho Brilhante

FORTALEZA

2018

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

O46t Oliveira, Emanuel Eduardo da Silva.
TrajectMe : Planning sightseeing tours with Hotel Selection from Trajectory Data / Emanuel Eduardo da Silva Oliveira. – 2018.
60 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2018.
Orientação: Prof. Dr. José Antônio Fernandes de Macêdo.
Coorientação: Prof. Dr. Igo Brilhante Ramalho.

1. Sightseeing tours planning. 2. Hotel selection. 3. Trajectories. 4. Genetic algorithm. 5. Trip Planning. I. Título.

CDD 005

EMANUEL EDUARDO DA SILVA OLIVEIRA

TRAJECTME: PLANNING SIGHTSEEING TOURS WITH HOTEL SELECTION FROM TRAJECTORY DATA

Dissertação apresentada ao Curso de Mestrado em Banco de Dados do Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Banco de Dados. Área de Concentração: Banco de Dados

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. José Antônio Fernandes de Macêdo (Orientador)
Universidade Federal do Ceará (UFC)

Dr. Igo Ramalho Brilhante (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Joao Jose Vasco Peixoto Furtado
Universidade de Fortaleza (Unifor)

Prof. Dr. João Paulo Pordeus Gomes
Universidade Federal do Ceará (UFC)

Prof. Dr. César Lincoln Cavalcante Mattos
Universidade Federal do Ceará (UFC)

À minha família, que se orgulha do meu empenho e da minha dedicação. Principalmente à minha mãe, Lúcia Bernardo, que me deu a oportunidade de ser uma pessoa do bem e por ser minha fonte de inspiração. À minha namorada, Ilanna Cabral, por ser paciente, amiga, companheira e por ser a melhor pessoa do universo.

AGRADECIMENTOS

Agradeço à minha família, pelo incentivo e pelo orgulho mútuo.

Agradeço à minha namorada, Ilanna Cabral, pelo companheirismo de todos os momentos, pela amizade e por não me deixar abater nessa árdua caminhada.

Agradeço ao meu orientador, José Antonio Macedo, pelas diversas oportunidades, pela precisão de suas orientações e por ser uma engrenagem imparável, que torna as pesquisas palpáveis, aplicáveis e úteis. Parabéns!

Agradeço ao meu amigo, parceiro de trabalho e coorientador, Igo Brilhante, pelas inúmeras conversas, conselhos e trocas de conhecimentos, coisas sem as quais este trabalho não seria viável. Muito obrigado!

Agradeço à instituição Universidade Federal do Ceará, bem como ao seu Programa de Mestrado e Doutorado em Ciências da Computação, aos seus professores e servidores, pela excelência, pelo esforço e por possibilitar, a mim e aos meus colegas discentes, uma oportunidade ímpar.

Agradeço aos meus amigos e companheiros de trabalho: Guilherme, Alex, Victor, Lucas, Felipe, Leonardo Ribeiro, Leonardo Anjos, Jonathan e Fugita, pela luta diária, pela empolgação e por primarem pela qualidade de seus trabalhos.

Agradeço também aos demais membros do Insight Data Science Lab.[1]: Abelardo, Andreza, Arina, Coutinho, Dayana, Erick, Florêncio, Francesco, Francisco, Hinessa, Holanda, Leopoldo, Luís, Lívia, Nicksson, Peres, Rosana, Ticiana, Vitória, Zschornack e todos os demais, pela dedicação em seus trabalhos e por tornar o laboratório um ambiente não só favorável às pesquisas de ponta, mas também um ambiente motivador, enriquecedor e que faz todos sentirem orgulho de escrever essa história. Ainda do Inisght, gostaria de agradecer ao gerente e amigo, David Araújo, por organizar toda essa galera e não deixar de primar pela excelência, pela confiança e pela sinceridade de sempre. Finalmente, agradeço imensamente ao Professor Regis Pires por me apresentar ao grupo e pelas constantes trocas de informação sempre que tenho a honra de encontrá-lo.

[1] http://www.insightlab.ufc.br/

“Perdemos nossa noção de propósito quando a espera se alonga.”

(HERBERT, 1965, p. 311)

RESUMO

Neste trabalho propomos o TRAJECTME, um algoritmo para resolver o problema de orientação com a seleção de hotéis (OPHS, *Orienteering Problem with Hotel Selection*) a partir das trajetórias de turistas extraídas de serviços baseados em localização. Este método é uma extensão do algoritmo memético proposto por Ali Divsalar em 2014, estado-da-arte do problema em questão, também escolhido como baseline para comparação frente a solução proposta. Coletamos dados de serviços como Foursquare e Flickr para reconstruir as trajetórias dos turistas. Em seguida, construímos um modelo de grafo de hotéis (HGM, Hotel Graph Model) usando um conjunto de trajetórias e um conjunto de hotéis para inferir sequências típicas de hotéis e pontos de interesse (PoI). O HGM é aplicado na fase de inicialização e nas operações genéticas do algoritmo memético para fornecer sequências de hotéis, enquanto a sequência de PoIs evolui pela aplicação de movimentos de busca local. Avaliamos nossa proposta usando datasets reais de três cidades italianas que possuem centenas de hotéis e PoIs. Os resultados mostram que o algoritmo proposto supera o estado-da-arte em até 208% no *score*. Nosso algoritmo também faz mais uso do *budget* disponível, sendo até 54% melhor do que o baseline nessa métrica.

Palavras-chave: planejamento de passeios turísticos; seleção de hotel; trajetórias; algoritmo genético; planejamento de viagens.

ABSTRACT

In this work, we propose TRAJECTME, an algorithm that solves the orienteering problem with hotel selection in several cities, taking advantage of the tourists' trajectories extracted from location-based services. This method is an extension of the state-of-the-art memetic-based algorithm proposed by Ali Divsalar in 2014. To this end, we collect data from services such as Foursquare and Flickr to reconstruct the trajectories of tourists. Next, we build a hotel graph model (HGM) using a set of trajectories and a set of hotels to infer typical sequences of hotels and point of interest (PoI). The HGM is applied in the initialization phase and in the genetic operations of the memetic algorithm to provide sequences of hotels, whereas the associated sequence of PoIs evolved by applying local search moves. We evaluate our proposal using a large and real dataset from three Italian cities using up to 1000 hotels. The results show that the proposed algorithm outperforms the state-of-the-art when using large real datasets. Our approach is better than the baseline algorithm by up to 208% concerning the solution score and proved to be more profitable toward PoI visiting time, being 54% better than state-of-the-art.

Keywords: sightseeing tours planning; hotel selection; trajectories; genetic algorithm; trip planning.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

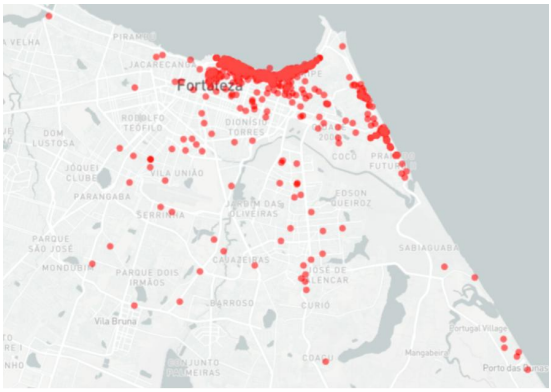
1 INTRODUCTION

Trip planning is of crucial importance to tourists in achieving a genuine travel experience. Presume a tourist who is planning to visit several cities within a region with many points of interests (PoIs) to be visited. This visit will endure several days, and only the hotel locations of departure and arrival are defined. For each day of this travel, the start and end hotel must be selected optimally, maximizing the satisfaction and the visiting time to the selected PoIs and minimizing the travel time among PoIs. The hotels can be selected from all suitable hotels available in the city. This example can be modeled as a variant of the orienteering problem known as the orienteering problem with hotel selection (OPHS) (DIVSALAR *et al.*, 2013). However, current OPHS state-of-the-art solutions (DIVSALAR *et al.*, 2013) (DIVSALAR *et al.*, 2014) (Van Hoek, 2016) (CASTRO *et al.*, 2015) (DUARTE *et al.*, 2016) do not scale when the number of hotels can reach the order of thousands hotels, which is the case of real applications. These solutions carried out experiments for synthetic datasets with up to 15 hotels and 100 PoIs. However, real-world problems have a higher scale. Let's suppose a traveler searching for hotels in Rome at Booking.com. He will find over 1000 hotel options available for her selection. In this regard, this work is the first attempt in the literature to provide a solution that scales given a large number of hotels. We have achieved this result using real data from tourist trajectories, which are collected and reconstructed from location-based services, such as Foursquare and Flickr.

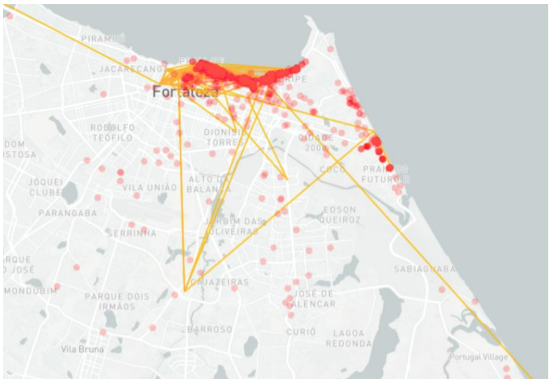
We leverage real tourist trajectories to find an initial solution to the OPHS more quickly. These trajectories carry the wisdom of the crowds regarding the most popular tourist routes. Besides, these trajectories help us to pick up to the most visited PoIs and identifying which regions are better to stay in a hotel closer to the attractions and tours. Figure 1 presents a scenario that helps us to understand the importance of using real trajectories to select hotels more efficiently. In Figure 1a the hotels in the city of Fortaleza are plotted whereas in Figure 1b the common trajectories of the tourists that visit the city are traced. We can see that the spatial extension of the trajectories match the distribution of the hotels, demonstrating that we can use the tourist trajectories to improve the selection of the hotels. It is worth noting that in Figure 1b the PoI that is distant from the hotel agglomeration refers to the airport of Fortaleza.

Figure 1 – Figure (a) shows the distribution of hotels in the city of Fortaleza while in Figure (b) the trajectories of tourists captured from location-based services are plotted together with the hotels.

(a) Hotels in Fortaleza



(b) Trajectories over hotels in Fortaleza



1.1 Contributions

The contributions of this work are three-fold: (1) an algorithm named Hotel Graph Model that models a set of hotels in the form of a graph based on historical trajectories of tourists in such a way that reduces the search space of solutions; (2) another algorithm named TRAJECTME that uses the HGM to find solutions to the OPHS problem and that scales to real-world cases generating solutions with quality comparable to competitors state-of-the-art solutions; (3) a set of experiments using real data that demonstrates the accuracy and efficiency of the proposal in scenarios with a large number of hotels.

We carried out experiments to validate our proposal using real datasets provided by location-based services. We compare our proposal TRAJECTME to the Memetic Algorithm (MA) presented in (DIVSALAR *et al.*, 2014). The experiment's results showed that our algorithm overcomes the competitor achieving higher scores and better usage of time budget available (travel utility metric), mainly when the number of hotels is large. Our approach is better than the competitor by up to 208% with respect to the solution score and proved to be more efficient in the use of visitation time, being 54% better than state-of-the-art.

1.2 Publications

As a result of this dissertation work, we have published the following paper:

- (OLIVEIRA *et al.*, 2018) OLIVEIRA, E.; BRILHANTE, I. R.; MACEDO, J. A. F. de. Trajectme: Planning sightseeing tours with hotel selection from trajectory data. In:

ACM. **Proceedings of the 2nd ACM SIGSPATIAL Workshop on Recommendations for Location-based Services and Social Networks.** [S.l.], 2018. p. 1..

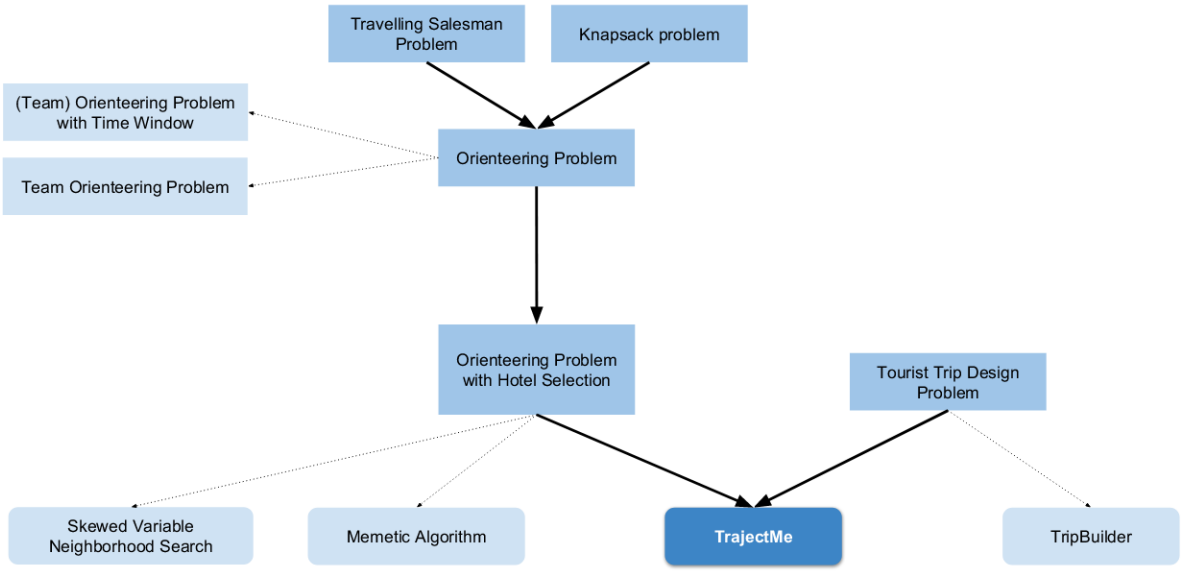
1.3 Organization

This work is structured as follows. Chapter 2 presents a review over related literature. This review explains the problems related to TRAJECTME and his main target problem. For each of these problems, we present the definition, practical applications, and approaches proposed by literature to solve them. The first kind, Orienteering Problem, is presented in Section 2.1 and its variations in the next ones: Team Orienteering Problem (Section 2.1.1), Orienteering Problem with Time Window and Team Orienteering Problem with Time Window (Section 2.1.2). Still, in this chapter, the Tourist Trip Design Problem (Section 2.2) is presented, since TRAJECTME is mainly applied to solve problems related to tourism routing planning. Next, in Section 2.3, the Orienteering Problem with Hotel Selection (OPHS), which is the target problem addressed by TRAJECTME, is detailed. We conclude the Chapter 2 in the Section 2.4 discussing about the related works. Chapter 3 presents the proposed approach of this work. Section 3.1 presents the Hotel Graph Model (HGM), its definition and how it is built. Section 3.2 presents the TRAJECTME , how it uses the HGM to extends the state-of-art algorithm and how it improves the results and the performance over big OPHS instances. Section 3.3 describes how to build the tourism knowledge base used by the proposed solution. Chapter 4 describes how was carried out the experiments and results obtained. Chapter 5 concludes the work.

2 LITERATURE REVIEW

TRAJECTME was developed to address the Orienteering Problem with Hotel Selection (OPHS), which is a kind of Orienteering Problem (OP) applied to the tourism context. Therefore OPHS can also be classified as a Tourist Trip Design Problem (TTDP). An overview of each problem related to this work will be given in this chapter. Each problem will be presented with a definition, practical applications and proposed solutions. In the last section, we will present the related works, their key features and how they relate to our work. Below, the Figure 2 presents a hierarchy of the works and problems related to TRAJECTME.

Figure 2 – Hierarchy of problems and works related to TRAJECTME. Regular rectangles are the problems. Rounded rectangles are the works. The darker the blue, the more it belongs to the TRAJECTME hierarchy. The arrows present the relations between them. Bold arrows are the hierarchy of TRAJECTME .



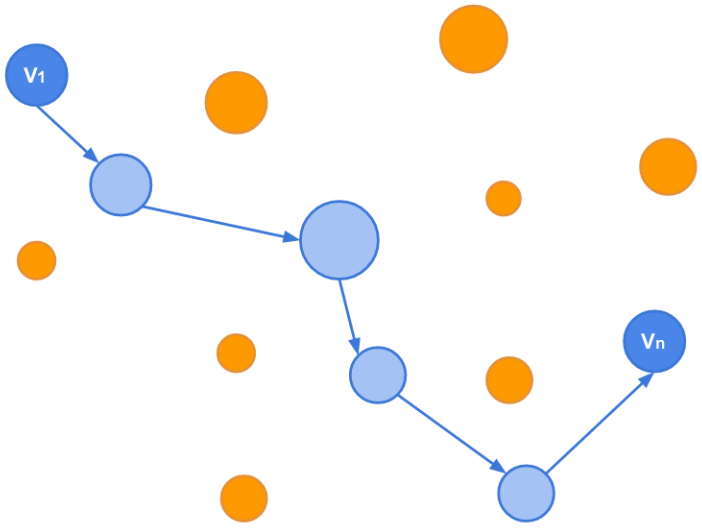
2.1 Orienteering Problem

The name "Orienteering Problem"(OP) originates from the sport-game of orienteering (CHAO *et al.*, 1993). In this game, individual competitors start at a specified control point, try to visit as many checkpoints as possible and return to the control point within a given time frame. Each checkpoint has a certain score and the objective is to maximize the total collected score. The OP is a combination of vertex selection and determining the shortest path between the selected vertices. As a consequence, the OP can be seen as a combination between the Knapsack Problem (KP) and the Travelling Salesperson Problem (TSP). The objective function of OP is to

maximize the total score collected, while the TSP tries to minimize the travel time or distance. Furthermore, not all vertices have to be visited in the OP. Determining the shortest path between the selected vertices will be helpful to visit as many vertices as possible in the available time. The OP is also known as the selective travelling salesperson problem (LAPORTE; MARTELLO, 1990)(GENDREAU *et al.*, 1998a)(THOMADSEN; STIDSEN, 2003), the maximum collection problem (KATAOKA; MORITO, 1988)(BUTT; CAVALIER, 1994) and the bank robber problem (ARKIN *et al.*, 1998).

In the OP, a set of N vertices i is given, each with a score S_i. The starting point (vertex 1 or v_1) and the end point (vertex N or v_N) are fixed. The time t_{ij} needed to travel from vertex i to j is known for all vertices. Not all vertices can be visited since the available time is limited to a given time budget T_{max}. The goal of the OP is to determine a path, limited by T_{max}, that visits some of the vertices, in order to maximize the total collected score. Each vertex can be visited at most once. Figure 3 illustrates an instance of OP.

Figure 3 – OP illustration. Circles' radius denote vertices' score.



In most cases, the OP is defined as a path to be found between distinct vertices, rather than a circuit or tour ($v_1 \equiv v_N$). In many applications, however, v_1 does coincide with v_N.

The OP arises in several applications, such as:

- *Limited traveling sales-person* (TSILIGIRIDES, 1984): A traveling sales-person problem when there is not enough time to visit all possible cities. He knows the number of sales to expect in each city and wants to maximize total sales while keeping the total travel time

limited to a day (or week). Then the problem is to pick a feasible set of cities, which is worth to visit in order to maximize the gains and minimize the losses.

- *Tourist guide* (SOUFFRIAU *et al.*, 2008): For tourists visiting a city or region, it is often impossible to visit everything they are interested in. Thus, they have to select what they believe to be the most valuable attractions. Making a feasible plan in order to visit these attractions in the available time span is often a difficult task. These planning problems are called Tourist Trip Design Problems (TTDP) and will be discussed in Section 2.2.
- *Home fuel delivery*(GOLDEN *et al.*, 1987): A fleet of trucks has to deliver to a large number of customers on a daily basis. The customers' fuel inventory level should be maintained at an adequate level at all times. The forecasted inventory level can be considered as a measure of urgency. A primary goal is to select a subset of customers to be visited each day who urgently require a delivery and are clustered in such a way that efficient truck paths can be constructed. This subset selection step is modeled as an OP where the urgency of delivery at a customer is used as the score. It is the first step of the larger inventory and routing problem of home fuel delivery. Other steps include the assignment of customers to vehicles and constructing efficient paths for each truck, between the assigned customers.
- *Military application* (WANG *et al.*, 2008): When a submarine or an unmanned aircraft is involved in surveillance activities, the length of the expedition is limited by a fuel or time constraint and the goal is to visit and photograph the best subset of all possible vertices.
- Another application mentioned in the literature is the single-ring design problem when building telecommunication networks (THOMADSEN; STIDSEN, 2003).

To solve OP, several researchers proposed algorithms. On the one hand, optimal approaches, like branch-and-bound (LAPORTE; MARTELLO, 1990)(RAMESH *et al.*, 1992) and branch-and-cut (GENDREAU *et al.*, 1998a)(FISCHETTI *et al.*, 1998). A branch-and-bound algorithm consists of a systematic enumeration of candidate solutions thought of as forming a rooted tree with the full set at the root and then the algorithm explores branches of this tree, which represent subsets of the solution set. A branch-and-cut involves running a branch and bound algorithm and using cutting planes, which are optimization methods that iteratively refine a feasible set, to tighten the linear programming relaxations. However, it only can handle instances only up to 500 vertices (VANSTEENWEGEN *et al.*, 2011b). In the other hand, many heuristics approaches was proposed to tackle bigger instance of OP, like stochastic and deterministic algorithms (TSILIGIRIDES, 1984), centre-of-gravity heuristic (GOLDEN *et*

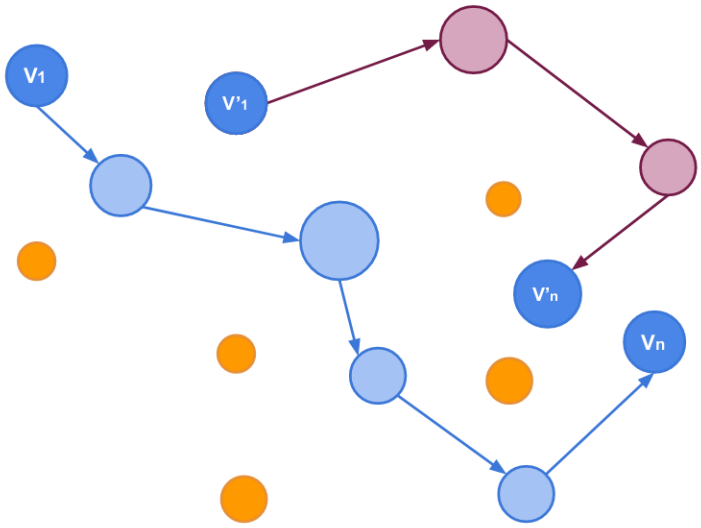
al., 1987), tabu search heuristic (GENDREAU *et al.*, 1998b), Pareto ant colony optimization algorithm (SCHILDE *et al.*, 2009) and multi-phase and multi-step heuristics frameworks (CHAO *et al.*, 1996b)(RAMESH; BROWN, 1991).

The OP has some variations. Among the most discussed in the literature are Team Orienteering Problem (TOP), Orienteering Problem with Time Window (OPTW), Team Orienteering Problem with Time Window (TOPTW). Moreover, the variation with hotel selection (OPHS) is the one of this work and will be discussed in Section 2.3.

2.1.1 Team Orienteering Problem

The Team Orienteering Problem (TOP) (CHAO *et al.*, 1996a) or multiple tour maximum collection problem (MTMCP) is an OP where the goal is to determine a set of paths, each limited by maximum budget, that maximizes the total collected score. The TOP corresponds to playing the game of orienteering by teams of several persons, each collecting scores during the same time span. Figure 4 illustrates an instance of TOP.

Figure 4 – TOP illustration. Circles' radius denote vertices' score. The colors differentiate the paths.



TOP was applied to solve some problems like:

- *Athlete recruitment from high schools* (BUTT; CAVALIER, 1994): A recruiter has to visit several schools in a given number of days. He can first assign a score to each school, based on its recruiting potential. As the recruiter's available time is limited, he has to choose the

schools to visit each day and try to maximize the recruiting potential.

- *Routing technicians to service customers* (TANG; MILLER-HOOKS, 2005): Each TOP path represents a single technician who can only work a limited number of hours in a day. Thus, not all customers requiring service can be included in the technicians' daily schedules. A subset of customers will have to be selected, taking into account customer importance and task urgency.

Column generation can be used as an exact algorithm to solve the TOP (BUTT; RYAN, 1999). Column generation is an efficient algorithm for solving larger linear programs: the overarching idea is that many linear programs are too large to consider all the variables explicitly. Since most of the variables will be non-basic and assume a value of zero in the optimal solution, only a subset of variables needs to be considered in theory when solving the problem. Column generation leverages this idea to generate only the variables which have the potential to improve the objective function, i.e., to find variables with negative reduced cost, assuming without loss of generality that the problem is a minimization problem. They were able to solve problems with up to 100 vertices when the number of vertices in each path remains small. Another exact method to deal with TOPs is starting with column generation and couple this with branch-and-bound to obtain a branch-and-price scheme (BOUSSIER *et al.*, 2007). Branch and price is a branch-and-bound method in which at each node of the search tree, columns may be added to the linear programming relaxation.

The first published heuristic for the TOP was developed in 1996 (CHAO *et al.*, 1996a) and is more or less the same as their five-step heuristic for the OP. The five-step heuristic only considers vertices that can be reached. In a Euclidean space, these vertices lie within an ellipse using start and end vertex as foci and the available budget as the length of the major axis. The initialization step creates many different paths, each starting with a vertex far away from the start and end vertex, and always assigns all other vertices to one of the paths using cheapest insertion. The best path is selected as the initial solution. Then more four steps are applied to improve the solution by swapping or replacing already included and adding non-included vertices. The difference between the first heuristic TOP and the five-steps heuristic is that, in the first one, instead of only selecting the best path, a set of best paths are selected and two reinitialization steps are used instead of one.

More metaheuristcs approaches were proposed to solve TOP, like tabu search heuristic and Variable Neighbourhood Search (VNS) (ARCHETTI *et al.*, 2007), Ant Colony Optimiza-

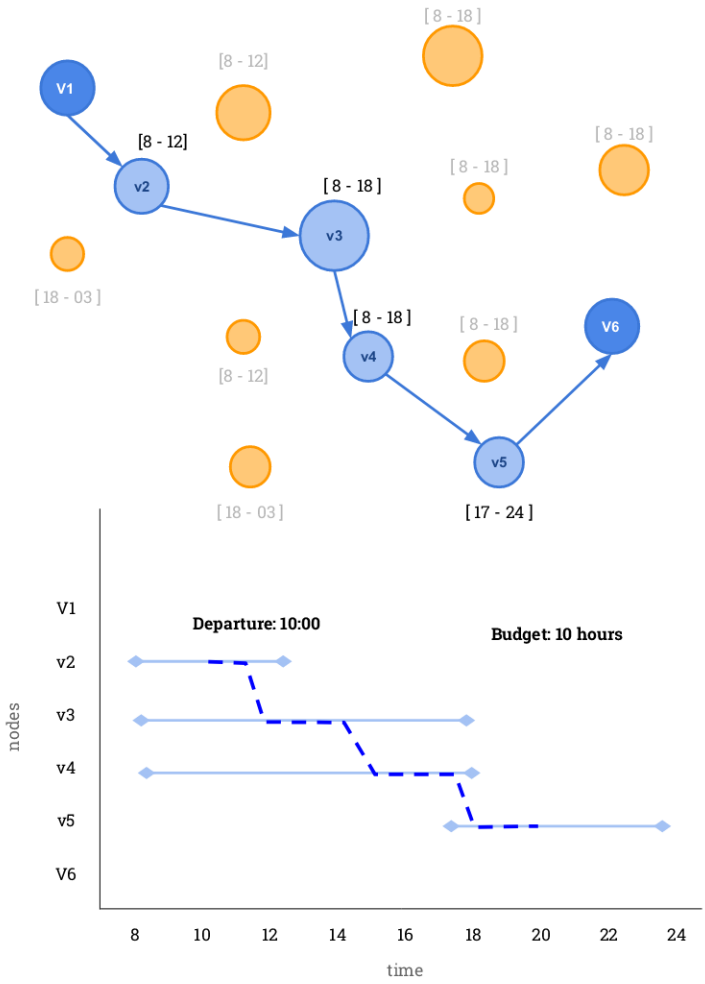
tion (ACO) (KE *et al.*, 2008), Guided Local Search (GLS), Skewed Variable Neighbourhood Search (SVNS) (VANSTEENWEGEN *et al.*, 2009a)(VANSTEENWEGEN *et al.*, 2009c) and Greedy Randomized Adaptive Search Procedure (GRASP) with Path Relinking (SOUFFRIAU *et al.*, 2010).

GLS (VANSTEENWEGEN *et al.*, 2009a) and SVNS (VANSTEENWEGEN *et al.*, 2009c) were the first to focus on obtaining good TOP solutions in only a few seconds of computational time. Both algorithms apply a combination of intensification and diversification procedures. Two diversification procedures simply remove a chain of attractions in each path. Another procedure tries to gather the available budget spread over different paths within the current solution, into a single path in the new solution. Two types of intensification procedures are designed. The first type tries to increase the score and the second type tries to decrease the travel time in a path. The SVNS algorithm outperforms the GLS algorithm. Furthermore, the computation time of the SVNS algorithm is lower. The success of the SVNS algorithm can be explained by a combination of factors. First of all, the SVNS framework appears suitable for this type of problem. Accepting a slightly worse intermediate solution when it is far from the incumbent, is a good strategy for selecting the vertices that will be part of the optimal solution. Additionally, the importance of a good diversification strategy is experimentally demonstrated and certain moves appear to be essential. The most important conclusion, however, is that it will always be the specific combination and sequence of different moves that determine the final quality of the algorithm.

2.1.2 Orienteering Problem with Time Window

In the Orienteering Problem with Time Window (OPTW), each vertex is assigned a time window and only can be visited during the specified time window (KANTOR; ROSENWEIN, 1992) (Figure 5). There's a variation of OPTW which multiple time-windows can be assigned to the vertices (SOUFFRIAU *et al.*, 2013). When first approached, Orienteering Problem with Time Window (OPTW) and Team Orienteering Problem with Time Window (TOPTW), discussed in the last paragraph of this section, received a lot of attention. The main reason is that instances with time windows should be solved in a very different way than instances without time windows. For instance, the well-known 2-Opt (LIN, 1965) move is indispensable to obtain high-quality results for the OP, but due to the time windows, it cannot be applied to efficiently solve the OPTW.

Figure 5 – OPTW illustration. Dashed lines denote the scheduled route, while range opening/-closing times.



Not many specific OPTW applications are mentioned in the literature, but most of the (T)OP applications mentioned before also require time windows and, as a consequence, the (T)OPTW model can be applied to many real-life situations. For example, the planning problems for tourist applications will have to take the opening hours of the tourist attractions into account. The routing of technicians or fuel delivery problems will also face opening hours in practice.

The first approach to solve OPTW (KANTOR; ROSENWEIN, 1992) starts with a straightforward insertion heuristic. The vertex with the highest ratio *score over insertion time* is inserted into the path, without violating time windows. Insertion time is the additional travel time incurred vertex inclusion in the path. Afterward, a depth-first search algorithm is proposed that constructs partial paths, using the insertion heuristic and beginning in the start vertex. Partial paths are abandoned if they are not feasible or if they are unlikely to yield the best total score.

Dynamic programming also was used to solve OPTW (RIGHINI, 2006)(RIGHINI; SALANI, 2008). Starting forward from the start vertex and backward from the end vertex, current

states are extended by adding an extra vertex at the end. Forward and backward states are matched if feasible and dominance tests are applied to record only non-dominated states. Decremental state space relaxation is used to reduce the number of states to be explored (RIGHINI; SALANI, 2009). A Special case of the OPTW in which the starting and end vertex are the same was addressed with a simple constructive heuristic and a granular variable neighborhood search (MANSINI *et al.*, 2006).

Team Orienteering Problem with Time Window (TOPTW), as mentioned earlier, is same as TOP, but with time window constraint in the vertex. Also as TOP, the goal of TOPTW is to maximize the sum of the collected scores by a fixed number of routes. The applications of TOP can be also extended to applications scenarios of TOPTW. For instance, a tourist visiting a city for some days and the museums only stay open in the afternoon and the nightlife attractions only start at midnight and end at sunrise. Some of solutions proposed to tackle TOPTW includes ant colony optimization (MONTEMANNI; GAMBARDELLA, 2009), Variable Neighbourhood Search (TRICOIRE *et al.*, 2010), Iterated Local Search (ILS) metaheuristic (VANSTEENWEGEN *et al.*, 2009b).

Many practical applications of the orienteering problem and its extensions are related to solutions of the touristic routing problem, also known as Tourist Trip Design Problem (TTDP). This problem is discussed in the next section.

2.2 Tourist Trip Desing Problem

The tourist trip design problem (TTDP) refers to a route-planning problem for tourists interested in visiting multiple points of interest (PoIs) (GAVALAS *et al.*, 2014). TTDP solvers derive daily tourist tours, i.e., ordered visits to PoIs, which respect tourist constraints and PoIs attributes. The main objective of the problem discussed is to select PoIs that match tourist preferences, thereby maximizing tourist satisfaction, while taking into account a multitude of parameters and constraints (e.g., distances among PoIs, visiting time required for each PoI, PoIs visiting days/hours, entrance fees, weather conditions) and respecting the time available for sightseeing on a daily basis.

Personalized Electronic Tourist guides (PETs) may be used to derive personalized tourist routes (BRILHANTE *et al.*, 2015)(GARCIA *et al.*, 2009)(KENTERIS *et al.*, 2009). Based on a list of personal interests and preferences, up-to-date information for the PoIs and information about the visit (e.g. date of arrival and departure, accommodation address, etc), a

PET can suggest feasible and near-optimal routes that include visits to a series of most interesting PoIs (VANSTEENWEGEN; OUDHEUSDEN, 2007).

A number of web and mobile applications have recently incorporated tourist route recommendations within their core functionality (VANSTEENWEGEN *et al.*, 2011a)(MTRIP, 2018)(GAVALAS *et al.*, 2012). In effect, they incorporate the main functionalities of PETS i.e., they generate personalized routes taking into account several user-defined parameters within their recommendation logic (days of visit, preferences upon POI categories, start/end location, visiting pace/intensity), while also allowing the user to manually edit the derived routes, e.g. add/remove PoIs. Recommended tours are visualized on maps (BRILHANTE *et al.*, 2015)(PLANNER, 2018)(MTRIP, 2018)(GAVALAS *et al.*, 2012), allowing users to browse informative content on selected PoIs. Some tools also offer augmented reality views of recommended attractions (MTRIP, 2018).

The generic problem of personalized tourist route generation which is mainly associated with the route generation functionality of mobile tourist guides and PETs has been defined as the "Tourist Trip Design Problem" (TTDP) (VANSTEENWEGEN; OUDHEUSDEN, 2007). The modeling of the TTDP is approached considering the following input data:

- A set of candidate PoIs, each associated with a number of attributes (e.g. type, location, opening days/hours, etc).
- The travel time among PoIs calculated using multi-modal routing information among PoIs, i.e. tourists are assumed to use all modes of transport available at the tourist destination, including public transportation, walking and/or bicycle.
- The score of each PoI, calculated as a weighted function of the objective and subjective importance of each PoI (subjectivity refers to the users' individual preferences and interests on specific PoI categories).
- The number of routes that must be generated, based upon the period of stay of the user at the tourist destination.
- The anticipated visiting duration of a user at a PoI which derives from the average duration and the user's potential interest for that particular PoI.
- The daily time budget that a tourist wishes to spend on visiting sights; the overall daily route duration (i.e. the sum of visiting times plus the overall time spent moving from a PoI to another which is a function of the topological distance) should be kept below this budget.

A basic classification of the TTDP variants may be based on the number of the derived routes as follows: (i) single tour TTDP variants aiming at finding a single tour that maximizes the collected profit while respecting certain tourist constraints and PoI attributes, and (ii) multiple tour TTDP variants aiming at finding multiple tours based upon the number of days the tourist's visit will last.

Single tour variants of the TTDP can be modeled using the Orienteering Problem (OP). Clearly, the OP may be used to model the simplest version of the TTDP wherein the PoIs are associated with a score and the goal is to find a single tour that maximizes the profit collected within a given time budget. Multiple tours variants of TTDP can be modeled using the Team versions of OP. Extensions of the OP have been successfully applied to model more complex versions both of the single and multiple tours of TTDP. The vast majority of the papers in the TTDP literature use the OP and its extensions to model different variants of the problem.

Given we describe the two main upper kinds of problems related to our approach, it's the time to detail the main kind of Orienteering Problem address by our work, its variation with hotel selection.

2.3 Orienteering Problem with Hotel Selection

The orienteering problem with hotel selection (OPHS) was defined by (DIVSALAR *et al.*, 2013) as a variant of the Orienteering Problem (OP). Is similar to Team Orienteering Problem (TOP), the difference is that in the TOP all trips have to start and end in the same vertex and no hotels need to be selected. OPHS is the main problem tackle by this work. To understand it better, we need to present a formal definition below.

In the OPHS, a set of hotels $\mathscr{H} = \{h_1, \dots h_M\}$, and a set of points of interest (PoIs) $\mathscr{P} = \{p_1, \dots p_N\}$ are given to form the set of vertices $\mathscr{V} = \{h_1, \dots h_M, p_1, \dots p_N\}$. Each PoI p_i is associated with a score S_i representing its relevance, while hotels have no score. The time t_{ij} to travel from vertex i to j is known for all pairs. Therefore, a trip T is defined by a start hotel h_s, an ordered set of PoIs and an end hotel h_e. Each trip T_i is limited to a given time budget T_i^d. Finally, the solution tour T^* is defined as D connected trips, where the end hotel of a trip is also the start hotel of the next one, and each PoI is visited at most once.

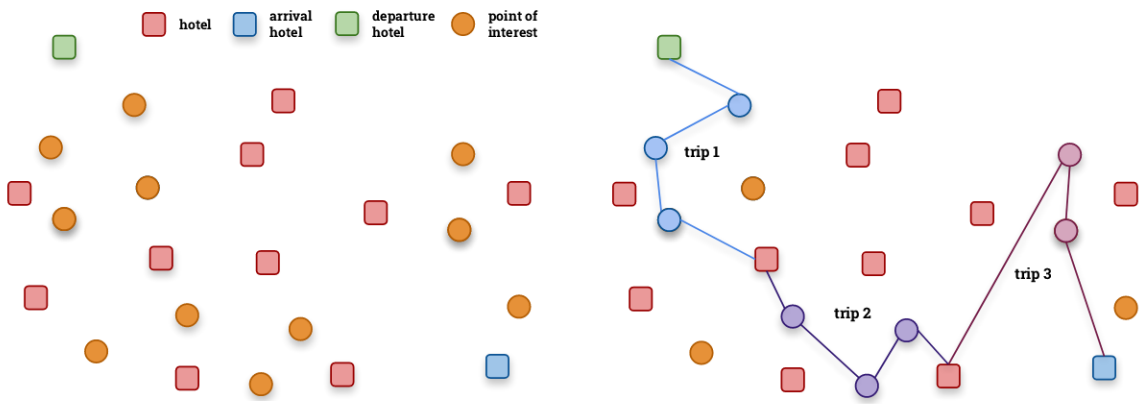
Given an integer D, the departure and arrival hotels, the goal of the orienteering problem with hotel selection is then to determine a tour T^* with D connected trips that maximize the total collected score of the visited PoIs.

As mentioned above, the OPHS is a new and challenging problem. It is a generalization of the OP and, therefore, NP-hard (GOLDEN *et al.*, 1987): no polynomial-time algorithm has been designed or is expected to be designed, to solve this problem optimally. This implies that exact solution algorithms are very time consuming for practical applications. Moreover, the OPHS is much more complex than the OP, since the starting and ending location for each trip need to be optimized as well.

In order to explain the nature of the OPHS, it's helpful to consider an example. Imagine a tourist who is planning to visit a certain region with various attractions. The tourist wants to select the combination of attractions that maximizes his pleasure. The visit will last several days and only the initial departure and the final arrival location are fixed. The departure and arrival locations (hotels) during the visit should be selected in an optimal way based on the attractions that are selected. Obviously, the hotel in which the tourist ends a given day has to be the same as the starting hotel of the next day. The hotels can be selected from all suitable hotels available in the region.

Figure 6 illustrate the OPHS, where the set of hotels, PoIs, and the departure and arrival hotels are given to find a tour of $D = 3$ connected trips. The solution is shown in Figure 6b where the three trips are highlighted in different colors, and the end of a trip is also the start hotel of the next one. The tour starts at the departure hotel (green vertex) and ends at the arrival hotel (blue vertex).

Figure 6 – Orienteering Problem with Hotel Selection illustration.
(a) The orienteering problem with hotel selection given the set of hotels, points of interests, departure, and arrival hotel.
(b) The tour, the solution for OPHS, composed by $D = 3$ connected trips. The paths were colored to distinct the trips.



A large number of practical applications can be modeled by the OPHS. For instance, a submarine performing a surveillance activity (tour) composed of consecutive missions (trips).

It requires to save zones (hotels) for provisioning between two missions. There are several possible points (vertices) to survey, but not all of them can be visited due to the limited available time. The submarine wants to maximize its benefit by selecting the most interesting combination of points. Only the initial departure and final arrival location of the whole surveillance activity are fixed. The departure and arrival save zone of each mission during the activity should be selected in an optimal way, considering the points that are selected for a visit. It is clear that when the submarine ends its current mission (trip) in a certain save zone, the next mission has to start in the same save zone.

Actually, depending on the exact practical circumstances, many of the applications for the orienteering problem or for the traveling salesperson problem with hotel selection (VANSTEENWEGEN *et al.*, 2011b) can be modeled more appropriately by the OPHS. For example, the well-known traveling salesperson problem turns into an OPHS under (realistic) circumstances: if the traveling salesperson needs to select which of his possible clients he will actually visit during his multiple day tours and he also needs to select the most appropriate hotels to stay every night. Other examples are truck drivers with limited driving hours wanting to reach an appropriate parking space, routing maintenance technicians with several depots to pick up spare parts, etc.

At this point, we describe all kind of problems related to our work. In the next section, we describe the works related to the TRAJECTME and the key differences between the approaches.

2.4 Related Works

Skewed Variable Neighborhood Search (SVNS) (DIVSALAR *et al.*, 2013) and Memetic Algorithm (MA) (DIVSALAR *et al.*, 2014) were the firsts works on OPHS. Brilhante et al. (BRILHANTE *et al.*, 2015) uses the trajectory drawn from the *wisdom-of-the-crowd* to solve Tourist Trip Design Problem (TTDP). TRAJECTME joins both the *state-of-art* solution construction steps and trajectories concept presented in (BRILHANTE *et al.*, 2015) to solve OPHS. A description of each work is given below, as well as the Table 1 comparing the main characteristics of each one.

SVNS (DIVSALAR *et al.*, 2013) introduced the hotel selection variant of the orienteering problem (OP). That work presents a mathematical formulation and possible applications of this problem. A modified variable neighborhood search framework (VNS), called Skewed

VNS (SVNS), was created to deal with OPHS. The SVNS obtains high-quality solutions for a lot of benchmark instances of varying size with known optimal solutions. However, hotel selection is a sensitive step of the problem, since choosing a different hotel significantly affects the entire solution. Thus, it is hard to determine which combination of hotels leads to a selection of high score vertices.

The structure of the OPHS, and the fact that it is not possible to predict which selection of hotels will result in a good solution, automatically leads to a two-level solution strategy. In this way, one level focuses on the hotel selection and the other part concentrates on the selection of PoIs. Thus, Divsalar et al. (DIVSALAR *et al.*, 2014) created a memetic algorithm (MA) that corresponds naturally to this two-level solution strategy.

Making use of historical data from tourists, collected from location-based services, can provide a way to reduce MA's computational search effort. A frequently visited sequence of PoIs is an indicator of quality and can be used as a starting point for the construction of better trips. Such sequence of PoIs is called trajectories in (BRILHANTE *et al.*, 2015).

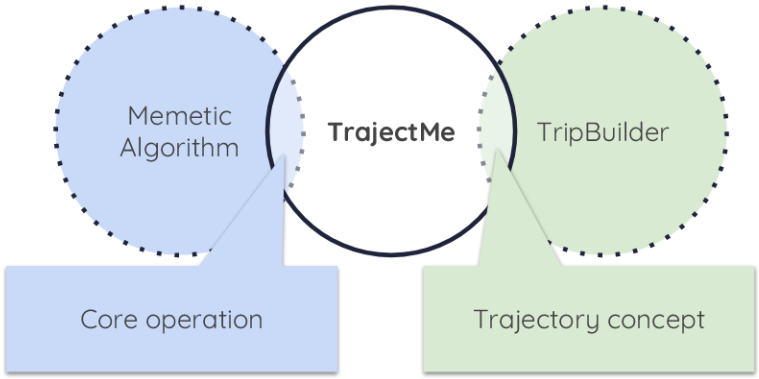
TRIPBUILDER (BRILHANTE *et al.*, 2015) mines data from location-based services, like Foursquare and Flickr, to build the tourist route that maximizes the tourist's interest based on his preferences. TRIPBUILDER is modeled as an instance of the Generalized Maximum Coverage (GMC) problem. Internally, (BRILHANTE *et al.*, 2015) extracts trajectories made by tourists in the past to create solution routes. A trajectory is a sequence of PoIs and must be fully considered to build a solution. However, TRIPBUILDER does not consider hotels as part of sightseeing planning.

The Table 1 compares the features of related works to TRAJECTME. SVNS and MA are computationally fast to solve OPHS with little variation for optimal solutions of their syntactic instances, but they are limited to handle large instances, like in real scenarios with hundreds of hotels and PoIs. On the other hand, TripBuilder takes advantage of trajectories that carries out the *wisdom-of-the-crowds* information to reduce the computational effort required to search for solutions in the solution space, but it doesn't deal with hotel selection. Therefore, TRAJECTME take advantage of characteristics from MA and TripBuilder to deal with large instances of OPHS (Figure 7), like real world tourism destinations, and including hotel as part of tour sightseeing planning. The next chapter presents a more detailed description of TRAJECTME and how it makes use of the key features of both related works to solve OPHS.

Table 1 – Related works

Work	Hotel	Trajectories	Real Datasets
Skewed VNS (SVNS) (DIVSALAR *et al.*, 2013)	X		
Memetic Algorithm (MA) (DIVSALAR *et al.*, 2014)	X		
TripBuilder (BRILHANTE *et al.*, 2015)		X	X
TRAJECTME	X	X	X

Figure 7 – TRAJECTME takes advantage of characteristics from MA and TripBuilder to deal with large instances of OPHS.



In this chapter, we considered which other kinds of problems are related to OPHS. The TTDP has also been considered since the TRAJECTME applies the OPHS especially in the case of tourist context. We also presented related works. Skewed Variable Neighborhood Search (SVNS) (DIVSALAR *et al.*, 2013) and Memetic Algorithm (MA) (DIVSALAR *et al.*, 2014) were the firsts facing the OPHS. To evolve the last one, TRAJECTME makes use of concepts well defined in TripBuilder (BRILHANTE *et al.*, 2015). Therefore, in the next chapter, we show how our approach works and how it solves the OPHS in an efficient way.

3 TRAJECTME: TRAJECTORY BASED MEMETIC ALGORITHM

In this chapter, we describe our proposal called TRAJECTME to solve the orienteering problem with hotel selection. TRAJECTME extends the original Memetic Algorithm by incorporating historical movements of tourists into a hotel graph model that summaries relevant pairs of hotels to boost the memetic algorithm procedures. We also present the Hotel Graph Model that is the key feature of the TRAJECTME and sets it apart from the state-of-art. Then, we detail the main components and the steps of the algorithm. First of all, we need to define some concepts.

Let us recall some definitions mentioned earlier: $\mathscr{H} = \{h_1, \ldots, h_M\}$ as the set of hotels and $\mathscr{P} = \{p_1, \ldots, p_N\}$ as the set of PoIs. Each PoI p_i and hotel h_j are univocally identified by its identifier, name and its geographic coordinates (latitude and longitude). The relevance score of the PoI p_i is given by S_i.

As mentioned early in the related works section, TRAJECTME takes advantage of the concepts from such works. Therefore, the following two definitions are strongly based on the definitions presented by (BRILHANTE *et al.*, 2015).

Definition 3.0.1 (User PoI History)**.** Given a user u and the PoIs $\mathscr{P}$, the PoI history of u is the temporally ordered sequence of m points of interest visited by u. Each PoI p_i of H_u is annotated with the two timestamps indicating the start time t_{1i} and the end time t_{2i} of the visit:

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

Let $\delta(t)$ be a function that returns a timestamp with time at the start of the day from a given timestamp t. For example:

$$\delta(\text{2018-10-20 23:10:00})$$
$$\text{2018-10-20 00:00:00}$$

Definition 3.0.2 (Trajectory)**.** Given a PoI history H_u, we define a trajectory $\mathscr{T}_u$ any subsequence of H_u

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

such that:

$$i \geq 1 \tag{3.1}$$

$$\delta(t_{1k}) \neq \delta(t_{2(k-1)}) \quad \text{, if } k > 1 \tag{3.2}$$

$$\delta(t_{1(k+i)+1}) \neq \delta(t_{2(k+i)}) \quad \text{, if } (k+i) < m \tag{3.3}$$

$$\delta(t_{1(k+j)}) = \delta(t_{2(k+j+-1)}) \quad \text{,} \forall j \text{ s.t. } 1 \geq j \leq i. \tag{3.4}$$

The constraint 3.1 ensures that such subsequence has at least two PoIs. The constraint 3.2 ensures that the start of the day of the first PoI of such subsequence is different from start of the day of the last PoI of the previous one (if exists, i.e., if $k > 1$). The constraint 3.3 ensures that the start of the day of the last PoI of such subsequence is different from the start of the day of the first PoI of the next one (if exists, i.e., if $(k+i) < m$). Finally, the constraint 3.4 ensures that the start of the day of all PoIs of such subsequence is the same.

The trajectories are intuitively sequences of PoIs visited consecutively on the same day by the user, from the first visited PoI to the last one on that day. For sake of simplicity, we denote $\mathscr{T}_u.first$ and $\mathscr{T}_u.last$ the first and last visited PoI in the trajectory $\mathscr{T}$, respectively. In addition, we can measure the relevance of trajectory by summing the relevance score of the PoIs as:

$$\sum_{j=k}^{k+i} S_j$$

Given the definitions, we can now describe precisely the Hotel Graph Model and the TRAJECTME operation. Let's start with HGM.

3.1 Hotel Graph Model

Divsalar et al. (DIVSALAR *et al.*, 2014) claim that the first problem to solve in OPHS is the selection of a good sequences of hotels, i.e., sequences that are feasible and are able to produce good sequence of PoIs to be visited within the time constraint. In our proposal, trajectories mined from the previous movement of tourists can highlight the best hotels in the city to generate the trips for OPHS. In fact, the trajectories represent the daily visited PoIs, where the users usually start and end their daily visitation. From this perspective, the start and end PoIs of the trajectories can be seen as important locations for selecting nearby hotels.

Therefore, our intuition is that the first and last PoIs of the trajectories represent good location candidates to find relevant hotels and, consequently, generate better sequences of Pois to visit.

Definition 3.1.1 (Hotel Graph Model)**.** The Hotel Graph Model (HGM) is a graph $G_h = (\mathscr{H}, E)$, where the set of vertices $\mathscr{H} = \{h_1, \ldots, h_M\}$ is the set of hotels, and the edges are defined as tuples $(h_i, h_j, c_{ij}, \mathscr{T}_u)$, where each edge carries a trajectory $\mathscr{T}_u$ and the cost c_{ij} of traversing and visiting PoIs in $\mathscr{T}_u$ starting at h_i and ending at h_j.

As we can see in the Definition 3.1.1, the edges of the HGM are essentially trips for the orienteering problem with hotel selection: a sequence of PoIs (trajectories) connected by two hotels.

In order to build the hotel graph model, we need to associate the trajectory set and the hotel set in such a way the closest hotels are preferable over far hotels. Hence, we look up the first and last PoIs of each trajectory to spatially match the closest hotels to each of those PoIs. We resort to R-tree data structures for spatial indexing and k nearest neighbor querying.

Algorithm 1 demonstrate how HGM is built given the set of trajectories $\mathscr{T}$, set of hotels $\mathscr{H}$ and an integer k. The main loop of the algorithm consists of iterating for each trajectory, applying k-nn over the first and last PoI of each trajectory, which are denoted by $\mathscr{T}_u.first$ and $\mathscr{T}_u.last$, to create the edge $(H_p^i, H_q^j, c_{ij}, \mathscr{T}_u)$, where H_p^i and H_q^j are the i-th and j-th closest hotels to PoIs p and q, respectively. The cost c_{ij} can be obtained from any arbitrary cost function, like the time to travel between two PoIs or the distance between them. Note that, in line 14 of the algorithm, we keep only the trajectory $\mathscr{T}_u$ with the highest score for each edge, when more than one trajectory is found for the pair of hotels.

Figure 8 depicts the overall process of building the hotel graph model from 8 trajectories, 12 hotels and $k = 1$. Given the trajectory and hotel sets (Figure 8a), the 1-nn is applied to find the closest hotel for the start and end PoIs of each trajectory (Figure 8b). Then, hotels connected by trajectory generate the edges in the HGM. Edges are then created between hotels when at least one trajectory connects them (Figure 8c). An example of trajectory is presented to help the understanding (Figure 8d). The connected trajectories are the User PoI History defined in the Definition 3.0.1.

In the next section, we present our proposal TRAJECTME that boosts the Memetic Algorithm by incorporating the hotel graph model built from trajectories of tourists.

Input : Set of trajectories $\mathscr{T}$, the set of hotels $\mathscr{H}$, an integer k
Output : $G_k = (\mathscr{H}', E_h)$

$u \leftarrow 1$;
while $u \leq |\mathscr{T}|$ **do**
  $p \leftarrow \mathscr{T}_u.first$;
  $q \leftarrow \mathscr{T}_u.last$;
  $H_p \leftarrow \eta_k(p)$; /* apply k-nn given p over $\mathscr{H}$ */
  $H_q \leftarrow \eta_k(q)$; /* apply k-nn given q over $\mathscr{H}$ */
  $i \leftarrow 1$;
  while $i \leq |H_p|$ **do**
    $\mathscr{H}' \leftarrow \mathscr{H}' \cup \{H_p^i\}$;
    $j \leftarrow 1$;
    while $j \leq |H_q|$ **do**
      $\mathscr{H}' \leftarrow \mathscr{H}' \cup \{H_q^j\}$;
      compute c_{ij};
      update E_h with $(H_p^i, H_q^j, c_{ij}, \mathscr{T}_u)$, keeping $\mathscr{T}_u$ with the highest score;
      $j \leftarrow j+1$;
    end
    $i \leftarrow i+1$;
  end
  $u \leftarrow u+1$;
end
return $G_k = (\mathscr{H}', E_h)$

ALGORITHM 1: Hotel Graph Model Construction.

3.2 TRAJECTME

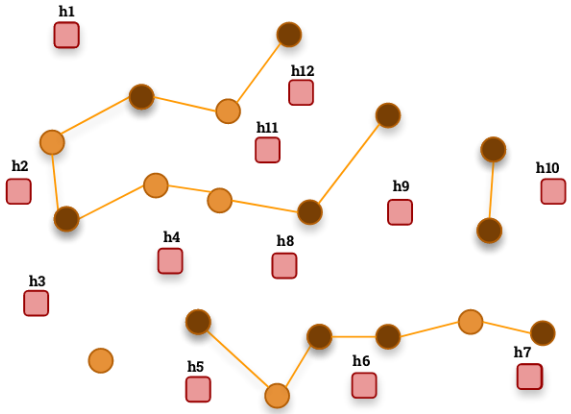
TRAJECTME is a genetic algorithm hybridized with local search techniques and boosted by a hotel graph model. The genetic algorithm focuses on the (re)combination of different sequences of hotels, where the local search technique is a Variable Neighborhood Descent (VND) (DUARTE *et al.*, 2016) which focuses on finding good sequences of PoIs between each pair of hotels in the tour for the OPHS. TRAJECTME takes advantage of HGM both for quickly generate of initial population and simplify the maintenance of data structure that keeps the trips between pairs of hotels.

Again, as mentioned early in the related works section, TRAJECTME takes advantage of the concepts from such works. Therefore, the general structure and the steps of this algorithm are strongly based on the Memetic Algorithm (MA) presented by (DIVSALAR *et al.*, 2014). The main difference between them it's the underlying presence of HGM during the initialization phase as a preprocessor and as a data structure that keeps the best trips between the hotels. By extends the MA, we need to describe it in all steps and details.

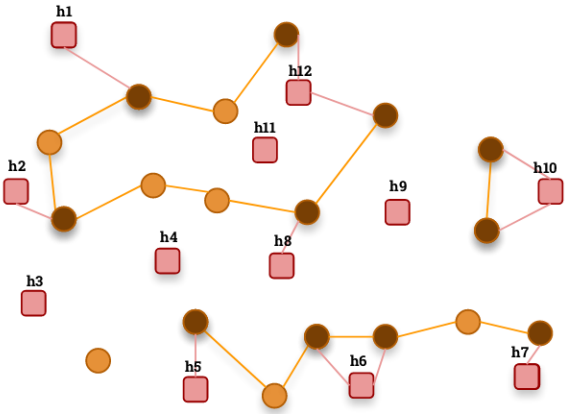
The general structure of TRAJECTME (Algorithm 2) consists of three main procedu-

Figure 8 – Example of the HGM building from 8 trajectories, 12 hotels and $k = 1$. (a) A set of hotels and a set of trajectories. (b) Each trajectory endpoint matches to the nearest hotel ($k = 1$ in this example). (c) By zooming out, we see the built hotel graph model summarizing pairs of hotels that are more likely to begin and end a trip, where each (d) edge carries the trajectory associated with the pair of hotels.

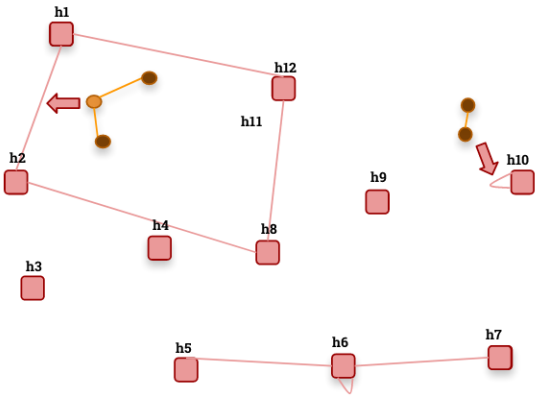
(a) Set of hotels and a set trajectories



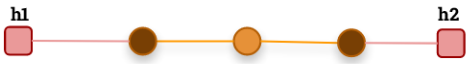
(b) Applying k-nn (1-knn)



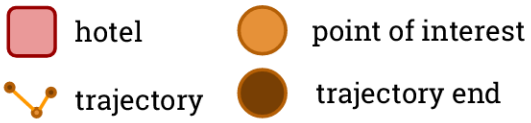
(c) Connecting hotels directly as a graph



(d) Example of a trip



(e) Legend



res: (i) the initialization, where the initial population (set of solutions for OPHS) is generated from HGM; (ii) the population of the pool of solutions, where genetic operations are applied to find new sequences of hotels and local search moves are used to improve the sequence of PoIs provided by HGM; (iii) the management of population to update the current population in main loop from the pool of solutions.

3.2.1 Generate initial population

Given the built HGM (Section 3.1), the creation of the initial population can be obtained through possible paths, i.e, a sequence of hotels, in the HGM between the hotel of departure and the hotel of arrival. Figure 9 depicts this procedure. (a) The overall process is to

Input : Hotel Graph Model $G_k = (\mathscr{H}', E_h)$, an integer D, an departure hotel h_d and an arrival hotel h_a
Result: *Best_Found_Solution*

```
Initialization:
    Generate initial population;
MainLoop:
    IterN ← 1;
    while IterN < MaxIterations do
        Populate the Pool;
        Sort the Pool according to quality;
        Save the Best_Found_Solution;
        Population management;
        IterN ← IterN + 1;
    end
```

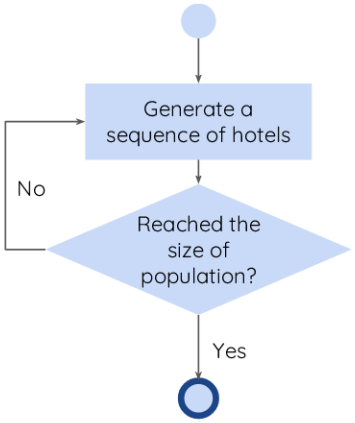
ALGORITHM 2: TRAJECTME general structure.

generate sequences until the number of solutions (sequences) reaches the population size. The size of the population is a parameter and will be discussed in Section 3.2.4. (b) A sequence of hotels is created starting from the hotel of departure and selecting the next hotels one by one. The HGM contains all possible next hotels from each start hotel through its edges. This process is carried out until the arrival hotel is reached. (c) The choice of the next hotel is made through the well-known selection method *roulette wheel* (GOLDBERG, 1989) (more about below). When selecting a hotel makes it impossible to arrive at the hotel of arrival, the last selected hotel is discarded and another is selected using the same probabilistic procedure. If there is no path of size D in the graph between the hotel of departure and the hotel of arrival, this sequence is given as infeasible.

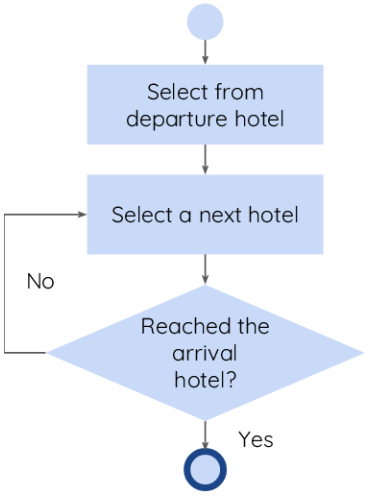
As mentioned above, *roulette wheel* is the method used to select a next hotel. To explain it, let H_i be a start hotel and $E_i \subset E_h$ the set of edges of H_i in which are its possible end hotels and f_i it fitness value, which is the unit of measure of this method. In the *roulette wheel* version used by TRAJECTME, the score of the edge between H_i and an end hotel H_j is used as the fitness value. The selection process starts by randomly choose value p such that $0 \leq p \leq \sum_{j \in E_i} f_j$. Then, a loop iterates over E_i through j until the accumulation of fitness values f_j reach p. Then, the j-est hotel whereupon the sum reach p is the selected one. Thus, the edge with the highest score is most likely to be selected. Figure 9c illustrates how the *roulette wheel* looks like: each instance (E_i) has its perimeter directly proportional to the fitness f_j and p is picked randomly within the roulette perimeter.

Figure 9 – Generate initial population components.

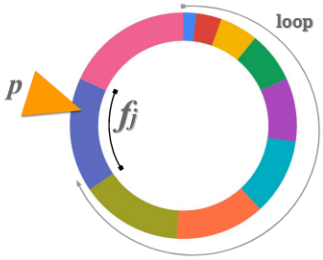
(a) Generate initial population.



(b) Generate a sequence of hotels.



(c) Roulette wheel selection method. Each colored segment represents an option, and its perimeter is its selection probability.

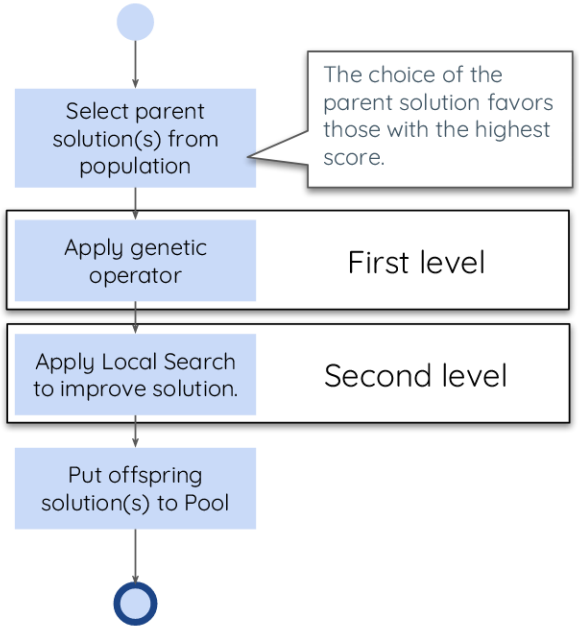


3.2.2 Populate the Pool

To find higher score solutions, we can derive them from the current population. Such finds are made through genetic operators and local search movements. We need to save the new solutions in a place to evaluate them in the future. Such place is called *Pool of solutions*. The Pool is cleaned at each iteration. At the begin of each iteration, the Pool is reset and the current population is transferred to it. After that, in order to generate new solutions, a two-level structure is applied (Figure 10): solutions are selected from current population as *parent solutions*, favoring the highest scores ones in the selection; then, at the first level, new sequences, called *offspring solutions*, are generated; after, at the second level, local search movements are applied to improve the score and reduce the costs; and then, the new solutions are placed in the Pool.

Algorithm 3 displays the whole process of populating the Pool. It has three loops one for each genetic operation: Crossover I, Crossover II, and Mutation (described below). These loops have the same structure: selects parent solutions from the current population (*roulette wheel*); applies genetic operator over parent solutions (first level) and gets offspring solutions; tries to improve offspring solutions by applying local search movements (second level); put offspring solutions in Pool; The difference is the number of offspring solutions: Crossovers generates two offspring solutions and Mutation generates only one. *CRI_R*, *CRII_R* and *PopSize* are parameters that control the number of solutions in the Pool. They'll be discussed in Section 3.2.4. Next, we describe how each level of this process works.

Figure 10 – Workflow of generating new solutions for the pool.



Add the *Current Population* to the *Pool* ;
$CRI_c \leftarrow 1$;
while $CRI_c \leq CRI\_R \times PopSize$ **do**
    Select solutions P_1 and P_2 from the *Current Population* (Roulette Whell);
    O_1 and $O_2 \leftarrow$ Apply Crossover I on P_1 and P_2 ; `// First level`
    Apply Local Search on O_1 and O_2 ; `// Second level`
    Add O_1 and O_2 to the *Pool*;
    $CRI_c \leftarrow CRI_c + 1$;
end
$CRII_c \leftarrow 1$;
while $CRII_c \leq CRII\_R \times PopSize$ **do**
    Select solutions P'_1 and P'_2 from the *Current Population* (Roulette Whell);
    O'_1 and $O'_2 \leftarrow$ Apply Crossover II on P'_1 and P'_2 ; `// First level`
    Apply Local Search on O'_1 and O'_2 ; `// Second level`
    Add O'_1 and O'_2 to the *Pool*;
    $CRII_c \leftarrow CRII_c + 1$;
end
$Mut_c \leftarrow 1$;
while $Mut_c \leq (PopSize - (CRI\_R + CRII\_R) \times PopSize)$ **do**
    Select solution P"$_1$ from the *Current Population* (Roulette Whell);
    O"$_1 \leftarrow$ Apply Mutation on P"$_1$; `// First level`
    Apply Local Search on O"$_1$; `// Second level`
    Add O"$_1$ to the *Pool*;
    $Mut_c \leftarrow Mut_c + 1$;
end

ALGORITHM 3: Populate the Pool.

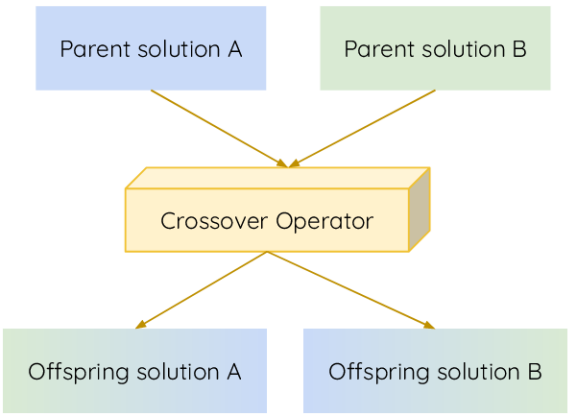
3.2.2.1 *First level: generating new sequences of hotels with genetic operators*

This level consists of two crossover operators and one mutation operator. This operators take solutions from current population as *parent solutions* and create new solutions, called *offspring solutions* (Figure 11).

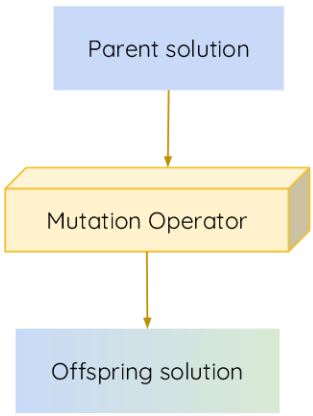
Crossover is a genetic function that takes two parent solutions to create two new ones (Figure 11a). TRAJECTME have two crossover operators. In both crossover operators, the sequence of hotels is handled in the same way: the sequence of hotels of the parent solutions are combined using a pivot-trip (one-point crossover) to create two new hotel sequences. Figure 12 depicts this process. T_2 and T_1 are pivot trips randomly selected for parent 1 and 2, respectively. Then, the offspring solution 1 is created combining $\langle h_d, h_1, h_2 \rangle$ from parent 1 with $\langle h_6, h_a \rangle$ from parent 2. The process is analogous for creating offspring 2.

Figure 11 – Genetic Operators.

(a) Crossovers I e II. Takes two solutions and generate two new ones.



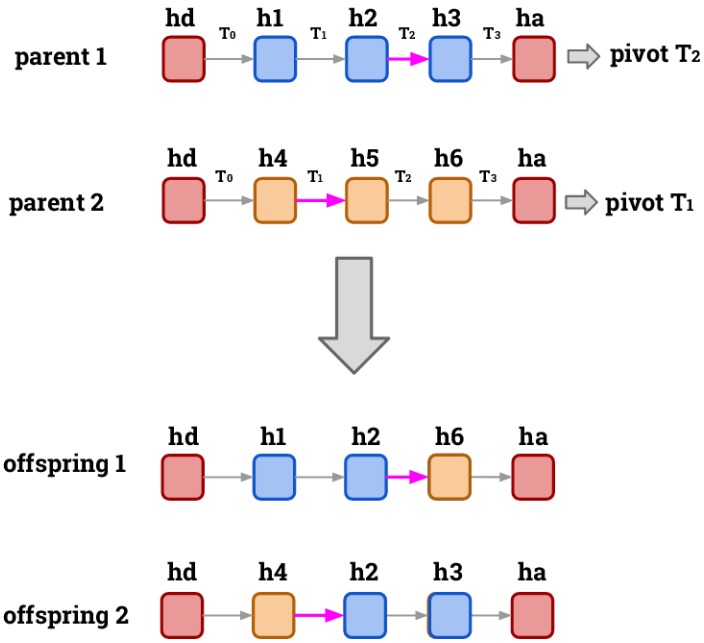
(b) Mutation. Takes one solution and generate a new one.



The difference between the two crossover operators lies in how they proceed from these two new sequences of hotels as presented as follows.

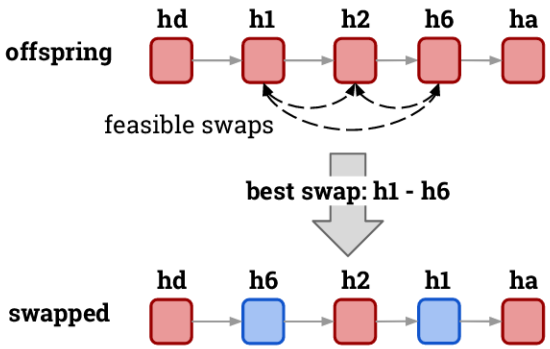
Crossover I. This crossover operator evaluates all feasible swaps between two hotels within each offspring solution (not including departure and arrival hotel). Based on the scores in the edges of HGM, the swap with the highest score is executed. For trips in which at least one hotel has been changed, the trajectory saved in the HGM is used as a path. Afterward, possible duplicated PoIs are removed, first from the trips that at least have one hotel changed and then from the other trips in the solution. In the end, local search, which will be detailed in the next level, is applied over the sequence of PoIs to further improve the solution. In this crossover operator, the

Figure 12 – Creation of offspring solutions from parent solutions based on two pivot trips T_2 and T_1.



connected hotels must also be connected in HGM, otherwise, the solution is infeasible. Figure 13 illustrate the swap in an offspring solution, where h_1 and h_6 are swapped, and the sequence of PoIs (trajectories) from HGM are used to form the trips.

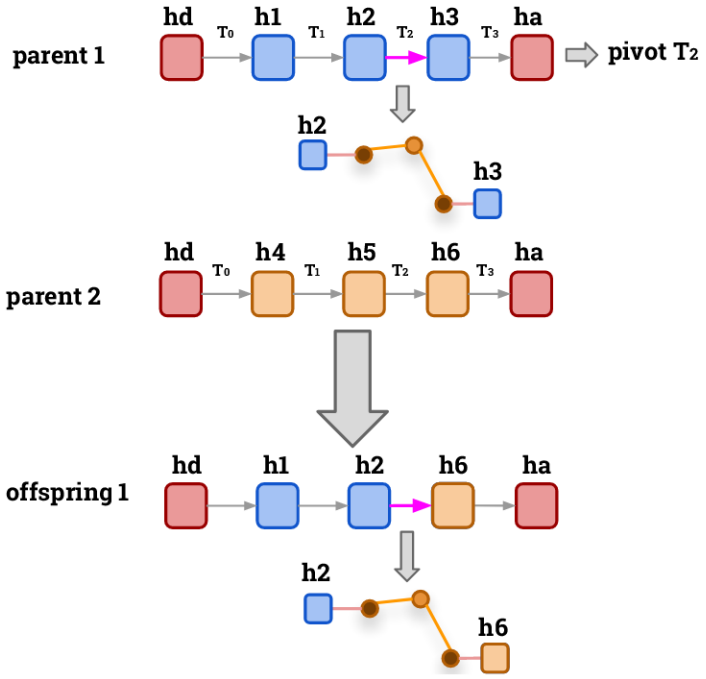
Figure 13 – Example of the Crossover I operator over an offspring solution with a swap between hotels h_1 and h_6.



Crossover II. On the offspring solution, this operator edits only the pivot trip by keeping the start hotel and sequence of PoIs from the first parent solution and using the end hotel from the second one. If the pivot trip turns infeasible, i.e., with the cost greater than the daily budget,

this operator removes the PoIs with the least ratio of the score over the cost until the trip fits the budget available. In the end, local search is applied to further improve the solution. Figure 14 depicts the Crossover II operator from two parents solutions and the pivot trip T_2 to generate the offspring solution.

Figure 14 – Crossover II example.



Mutation is another genetic function that takes one parent solution and creates a new one (Figure 11b). Such function *mutates* a solution by changing its sequence, replacing some item of the sequence by a new one. In our case, a sequence of hotels.

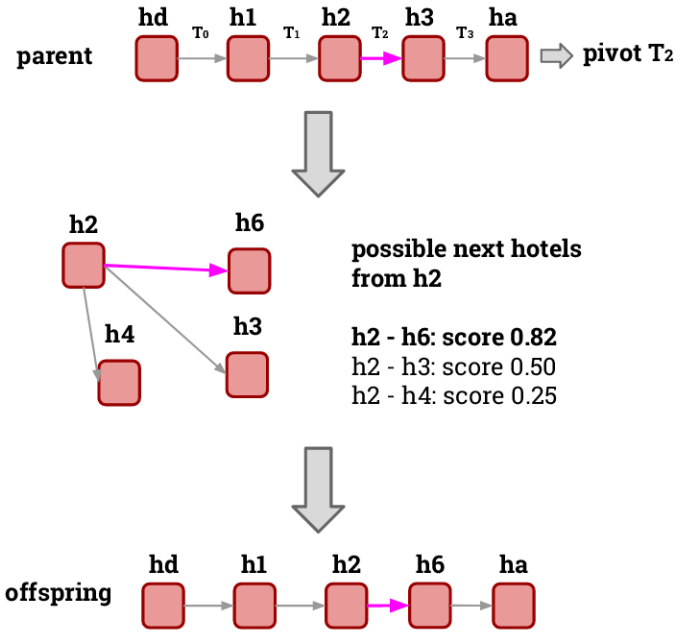
Mutation. In the mutation operator, one solution is selected from the population as a parent solution and it is mutated to create a new offspring. A pivot trip T_p is randomly selected where the end hotel of T_p is replaced by the hotel connected to the start hotel of T_p with the highest edge score in HGM. Starting from this new end hotel, it is verified that this replacement keeps the next trip feasible. Otherwise, the end hotel of the next trip is also considered to be replaced by using the same procedure.

In cases in which no feasible combination of end hotel for the pivot trip and end hotel for the next trip can be determined, the mutation operator will not modify the parent solution.

After replacing the hotel(s), the sequence of PoIs for the two or three affected trips are retrieved from HGM. To make the solution feasible, first, the duplicated PoIs are removed from the unchanged trips and then from the affected trips. Finally, local search is applied to

further improve the solution. Figure 15 depicts the mutation operator from one parent solution and one pivot trip T_2 to generate one offspring solution.

Figure 15 – Mutation example. The pair $\langle h_2, h_6 \rangle$ has the highest score between possible pairs starting from h_2



3.2.2.2 *Second level: improving the score and reducing the costs with Local Search*

This level consists of applying editions over the paths between the hotels that make up the sequences. Such editions are called *movements* or *moves*. All genetic operators presented at the first level apply local search moves with to improve the sequence of PoIs between the hotels. This movements either add new PoIs in the sequence or re-arranging the sequences of PoIs to decrease the time cost and, consequently, can add new PoIs to visit. A variable neighborhood descent with six well known local search moves is implemented in TRAJECTME 's local search. An overview of the local search operation is given in Algorithm 4. Next, all moves are briefly explained. The algorithm and the description of each move were originally created by Memetic Algorithm work (DIVSALAR *et al.*, 2014).

Insert: This move tries to include each of the available PoIs in the position with the minimum increase at tour cost. An inclusion only occurs if there's an available budget. Non-included PoIs are considered to inclusion until no more insertions are possible.

Move-Best: This move checks if by trying to take a PoI from its position and put in the other one

```
Input     :X ← The incumbent solution of the Local Search
N ← { Insert, Move-Best, Two-Opt, Swap-Best, Extract-Insert, Extract2-Insert } ;
k ← 0 ;
while k < 6 do
    X′ ← Apply neighborhood structure N_k on X;
    if X′ is better than X then
        X ← X′;
        k ← 0;
    else
        k ← k + 1;
    end
end
```

ALGORITHM 4: Memetic Algorithm Local Search.

it made the whole solution cheaper. Only the movement that leads to the highest cost decrease is applied.

Two-Opt: This move checks if the swap the order of two PoIs in the same trip leads to decrease trip cost.Only the movement that leads to the highest trip cost decrease is applied.

Swap-Best: This move tries to exchange two PoIs between two different trips. Such an exchange would execute only if one of the affected trips decreases its cost. To evaluate the exchange, this move needs to consider the amount of time saved in affected trips and the best positions to put on the PoIs. Only the movement that leads to the highest cost decrease is applied.

Extract-Insert: This move checks if by removing PoIs in each trip and then inserting available ones, the score of the trip can be increased. This process is sequentially applied. The excluded PoI is not considered again for insertion. Inserting the PoIs is done by Insert move, described above.

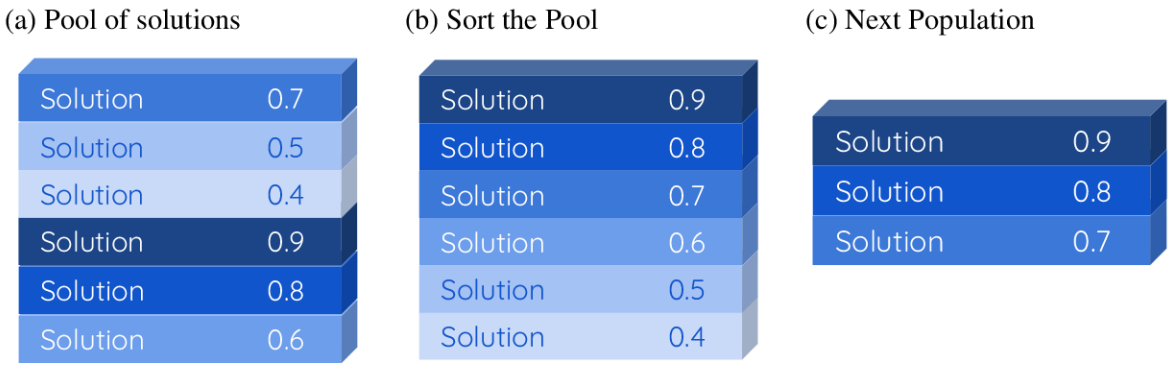
Extract2-Insert: This move is identical to the *Extract-Insert* move, except for the fact that it always considers two consecutive visited PoIs for exclusion.

In the main loop, at this point, due to the parameters *CRI_R* and *CRII_R* which controls how many offspring solutions must be created, the Pool reaches twice *PopSize*, the parameter which control the size of the population. Given that, a new generation should be elected to be the next population. That means which some solutions will be selected and others ones will be forgotten. This management will be explained next.

3.2.3 Management of population

At the end of each iteration in the main-loop, a new population is selected from the Pool. This selection is done based on a strategy to manage the population considering both the quality and the diversity of the solutions contained in it. Figure 16 illustrate this management. Each block represents a solution. Dark blues are solutions with higher scores, in contrast, light blues are solutions with lower scores. In order to apply our population management strategy, (a) all the solutions in the Pool are (b) sorted based on their quality (score). Afterward, (c) the solutions with the best solution quality are selected to be in the next population which becomes the current population of the next iteration. The same order is used to fill the rest of the population, but only solutions with a sequence of hotels different from the already selected solutions are chosen. If the number of solutions with different sequences of hotels is not enough to fill the population, the solutions with the minimum solution quality are selected to fill the population. In this way to we try to keep enough diversity in the population.

Figure 16 – Management of population illustration. Each block represents a solution. Dark blues are solutions with higher scores, in contrast, light blues are solutions with lower scores.



3.2.4 Parameter Settings

Since our proposal is based on the Memetic Algorithm (MA)(DIVSALAR *et al.*, 2014), TRAJECTME also depends on the same parameter setting: *MaxIteration* to handle the maximum number of iteration in the main loop; *PopSize* to define the size of the population in the algorithm; *CRI_R*, indicating the percentage of the population size that is used to create offspring solutions by the Crossover I operator; *CRII_R* is analogous for Crossover II operator; *BestSel_R* which is the percentage of the best solutions from the pool are selected to form the

new population; and *TabuSize* used in the mutation operator to handle the number of times a hotel is not selected for the same trip.

According to previous parameter sensitivity experiments (DIVSALAR *et al.*, 2014), the only changes in parameter values that have a significant impact on the performance of the algorithm are *MaxIteration* and *PopSize*. By using higher values for them the quality of the results are increased but at the cost of higher computation time.

In addition to the MA parameters, there is the TRAJECTME's k parameter, used in the hotel graph model. This parameter indicates how many hotels should be linked to trajectories' end. The intuition behind k's value is that higher values can increase the number of possible hotels and lower values decrease instead. A higher number of hotels indicates that more solutions can be found, which means that more arrangement of edges should be tested to find the better ones. However, it also means more iterations and CPU time. An analysis of about k's variation will be given in Chapter 4.

We detailed all concepts, operations, and parameters of TRAJECTME so far. However, to work appropriately, it relies on the available data about PoIs, hotels, and trajectories from regions and cities. The next section will present the procedures carried out to extract and process this data.

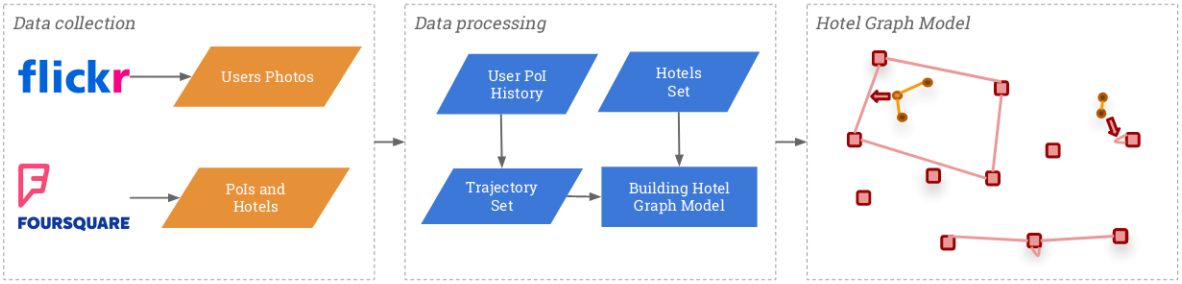
3.3 Building the Tourism Knowledge Base

To provide solutions to OPHS, TRAJECTME needs of a tourism knowledge base, which consists of the sets of PoIs, hotels, and trajectories, and the hotel graph model. The overall process to build this knowledge base is based on TripBuilder (BRILHANTE *et al.*, 2015). Figure 17 depicts an overview for building the hotel graph model. We organize the architecture into three components: (i) data collection, where the Flick photos, PoIs and hotel data are collected from location-based services: Flickr and Foursquare; (ii) data processing to generate PoI History for users to devise a trajectory set; and (iii) the construction of the hotel graph model given the trajectory set.

PoI and Hotel Sets. We collected points of interest for a given city by querying the Foursquare API[1] using the bounding box of the targeted city. Moreover, we filtered out PoIs by using their categories in order to keep only PoIs that are more relevant for tourism: architecture, arts, churches, food & drink, history, monuments, museums, nature, nightlife, shopping, sports,

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

Figure 17 – Components of Hotel Graph Model building process.



and transport. The set of hotels, in turn, we also query Foursquare API. However, we use the category filtering to retrieve only PoIs containing categories related to accommodations and hotels.

User PoI History. In order to collect long-term itineraries crossing the collected PoIs, we resort to Flickr API to retrieve photos taken in the given city. We query Flickr using the bounding box of the city to retrieve the metadata of the photos, such as user id, timestamp, tags, latitude, and longitude, etc [2]. Next, we spatially match the collected photos against the set of PoIs previously collected. Each photo is associated with a PoI when the photo has been taken within a distance of at most 100 meters from the PoI location. The photos taken by the same user are usually taken close to the same PoI, so we consider the timestamps associated with the first and last of these photos as the starting and ending time of the user as an estimation of her visit to the PoI. The average visit estimation is then computed to each PoI as its needed visiting time. In the end, the PoI history is reconstructed for each user.

Trajectory Set. Given the PoI History of the users, we construct the trajectories as presented in Definition 3.0.2. In this process we obtain a set of trajectories, i.e., sequences of PoIs, representing the daily visited PoIs of the users.

Hotel Graph Model. Once the sets of trajectories and hotels are obtained, we apply the Algorithm 1 detailed in Chapter 3 to build the HGM for each city. The parameter k is studied in our experimental evaluation present in Chapter 4.

In order to carry out the experiments, we apply this build process of knowledge base over three Italian cities. Table 2 shows the basic statistics for these three Italian cities regarding the number of PoIs, hotels[3] and trajectories for each one.

[2] https://www.flickr.com/services/api/flickr.photos.search.html
[3] The rounded number of hotels in the Rome and Florence was a Foursquare's API limitation.

Table 2 – Data collected from Italian cities.

City	PoIs	Hotels	Trajectories
Pisa, Italy	61	402	59
Florence, Italy	146	1000	593
Rome, Italy	302	1000	1685

4 EXPERIMENTS AND RESULTS

We conduct a number of experiments to validate our proposal TRAJECTME using real datasets provided by location-based services. For the best of our knowledge, this work is the first one to investigate the OPHS using real datasets. The algorithms have been implemented in Scala 2.11 and the experiments were conducted on a personal computer Intel© Core™ i5-5200U 2.20GHz CPU and 6GB of RAM.

We compare our proposal TRAJECTME to the memetic algorithm (MA) presented in (DIVSALAR *et al.*, 2014). To assess the effectiveness of both approaches, we consider two evaluation metrics described as follows.

4.1 Evaluation Metrics

Tour Score $S^{tour}(T^*)$. This metric aims at evaluating the total score of the PoIs in the solutions. In fact, this metric is the objective function of the OPHS. It is the ratio of the total score of the PoIs in the solution tour T^* over the sum of the scores of all the PoIs in $\mathscr{P}$ (Equation 4.1). We recall that the PoI score in our experiments is represented by the number of check-ins of the PoI collected from Foursquare.

$$S^{tour}(T^*) = \frac{\sum_{p \in T^*} S_p}{\sum_{p' \in \mathscr{P}} S_{p'}} \tag{4.1}$$

Tour Utility $U^{tour}(T^*)$. It evaluates how good is the solution in terms of visiting time. It is computed as the sum of the visiting time of the PoIs in the solution tour T^* over the sum of time budget of each day (Equation 4.2). Higher scored tours are preferable since they favor the time to enjoy attractions with respect to the traveling time.

$$U^{tour}(T^*) = \frac{\sum_{p \in T^*} vt(p)}{\sum_{T_i \in T^*} T_i^d} \tag{4.2}$$

In addition to the computed scores for the evaluation metrics, we present the improvements of TRAJECTME over MA w.r.t each metric by calculating the improvement percentage as

$$(\frac{\text{TM-}k_{result} - MA_{result}}{MA_{result}}) \times 100.$$

4.2 Experiments

We conduct a number of experiments by varying the number of trips D (days) and the parameter k used in the k-nn query during the construction of HGM. Since the parameters of TRAJECTME are the same as MA proposed by Divsalar et al. (DIVSALAR *et al.*, 2014), apart from k, we use the same configuration for these parameters as proposed by the authors in order to provide a balance between runtime and quality of the solutions. Further experiments with different parameter settings would be necessary to test the behavior of the baseline and our proposal. However, as we'll see in the performance results, the baseline takes a lot of processing time making it impossible to test various parameter settings for it. The departure and arrival hotels are randomly selected for each city but remain the same for all experiments in that city. Tables 3 - 5 show the results of the experiments for each metric in different settings.

We studied algorithms by varying the number of days and the parameter k for each city, except for Pisa where only considered $D = 2$ days because it is a small city where tourists can visit almost all the PoIs in one or two days. We also studied the parameter k TRAJECTME , named of TM-k for each k in $\{1,3,5,10,15\}$.

All results are the averages of the three executions for each set of input parameters. The best result is highlighted with bold, where we present the score and the improvement of TRAJECTME with respect to the MA result.

Table 3 – Average effectiveness of TRAJECTME compared to MA in Pisa.

Days	Algorithm	Tour Score	Tour Utility
1	MA	0.067	0.844
	TM-1	0.067 (0%)	0.844 (0%)
	TM-3	0.067 (0%)	0.844 (0%)
	TM-5	0.067 (0%)	0.844 (0%)
	TM-10	0.067 (0%)	0.844 (0%)
	TM-15	0.067 (0%)	0.844 (0%)
2	MA	**0.967**	0.802
	TM-1	0.960 (-0.698%)	**0.812 (1.299%)**
	TM-3	0.965 (-0.177%)	0.812 (1.299%)
	TM-5	0.966 (-0.025%)	0.802 (0%)
	TM-10	0.965 (-0.134%)	0.812 (1.299%)
	TM-15	0.964 (-0.235%)	0.812 (1.299%)

Table 4 – Average effectiveness of TRAJECTME compared to MA in Florence.

Days	Algorithm	Tour Score	Tour Utility
2	MA	0.242	0.873
	TM-1	0.413 (70.934%)	0.866 (-0.795%)
	TM-3	0.455 (88.465%)	0.861 (-1.392%)
	TM-5	0.465 (92.363%)	**0.891 (1.988%)**
	TM-10	0.455 (88.429%)	0.870 (-0.398%)
	TM-15	**0.481 (98.952%)**	0.878 (0.596%)
4	MA	0.790	0.871
	TM-1	0.813 (2.912%)	0.868 (-0.299%)
	TM-3	0.806 (2.049%)	**0.892 (2.493%)**
	TM-5	0.804 (1.803%)	0.867 (-0.399%)
	TM-10	0.827 (4.694%)	0.888 (1.994%)
	TM-15	**0.874 (10.597%)**	0.863 (-0.897%)
7	MA	0.974	0.872
	TM-1	0.982 (0.783%)	0.885 (1.536%)
	TM-3	0.983 (0.907%)	0.892 (2.275%)
	TM-5	**0.984 (1.032%)**	0.889 (1.991%)
	TM-10	0.983 (0.914%)	**0.892 (2.332%)**
	TM-15	0.981 (0.727%)	0.890 (2.105%)

Table 5 – Average effectiveness of TRAJECTME compared to MA in Rome.

Days	Algorithm	Tour Score	Tour Utility
2	MA	0.125	0.446
	TM-1	0.266 (113.313%)	0.583 (30.739%)
	TM-3	0.261 (109.099%)	0.623 (39.689%)
	TM-5	0.270 (116.421%)	0.559 (25.292%)
	TM-10	0.264 (111.071%)	0.611 (36.965%)
	TM-15	**0.385 (208.099%)**	**0.691 (54.864%)**
4	MA	0.331	0.602
	TM-1	**0.637 (92.364%)**	0.619 (2.886%)
	TM-3	0.574 (73.473%)	**0.707 (17.460%)**
	TM-5	0.567 (71.078%)	0.627 (4.185%)
	TM-10	0.578 (74.664%)	0.620 (3.030%)
	TM-15	0.568 (71.452%)	0.610 (1.443%)
7	MA	0.601	0.685
	TM-1	0.743 (23.712%)	0.682 (-0.434%)
	TM-3	0.782 (30.223%)	0.682 (-0.434%)
	TM-5	0.766 (27.585%)	**0.720 (5.141%)**
	TM-10	**0.790 (31.510%)**	0.703 (2.679%)
	TM-15	0.744 (23.838%)	0.676 (-1.376%)

4.3 Result analysis

We divided this analysis over four bias: (i) tour score, (ii) tour utility, (iii) k parameter sensitivity and (iv) CPU time.

4.3.1 Tour Score

We first study the tour score results in the three cities. Results for Pisa show that both algorithms achieved good results, where MA was slightly better than TRAJECTME (TM). This result highlights that for cities with a small set of PoIs, both algorithms tend to perform well since the tasks of selecting hotels and PoIs is simpler in this scenario. The high score 0.96 shows that the 2-days tours in Pisa the tourists visit almost all feasible PoIs.

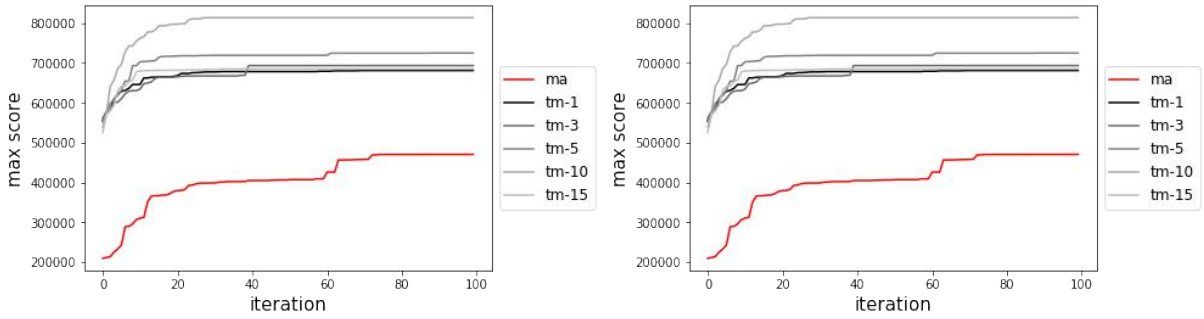
In Florence and Rome, on the other hand, we can see a different behavior. TM presented the highest scores, in special for 2-days tours, for both cities. As illustrated in Table 2, the number of PoIs and hotels are much larger than in Pisa. Consequently, finding good tours become a harder task to achieve. We notice that as the number of days D increases, the improvements of TM w.r.t MA decreases. However, the improvements are still significant, especially in Rome, where we got the largest number of PoIs and hotels.

The results for Florence and Rome highlight another important behavior to explain the reason why TM has shown important improvements over MA. The memetic algorithm combined with local search moves seem to be very affected for the initial population and the number of iterations in the main loop. Figure 18 compares the score convergence progress of algorithms through the iterations. The y-axis represents the maximum score from the current population for each iteration (x-axis). The red line represents the MA algorithm and the grey ones represent variations of TRAJECTME for different values of k. Due to the boosted initialization, TRAJECTME needs fewer iterations to reach high scores. The initialization with HGM provides high score solutions then a few iterations are needed to refine the final one. On the other hand, normal initialization generates delayed initial population leading MA to require more iterations compared to TRAJECTME to improve their results.

Therefore, these findings highlight the importance of HGM and trajectories of real tourists representing their daily sequence of visited PoIs. The inclusion of HGM into the memetic algorithm proves that the initial population created from the initialization step brings an enormous benefit when there a lot of PoIs and hotels to select. Thus more pleasurable and relevant tours

Figure 18 – Convergence score progress through the iterations. The y-axis represents the maximum score from the current population for each iteration (x-axis). The red line represents the MA algorithm and the grey ones represent variations of TRAJECTME for different k values. Due to the boosted initialization, TRAJECTME needs lesser iterations to archive high scores.

(a) Florence in 7 days (b) Rome in 4 days



can be generated for tourists in a city.

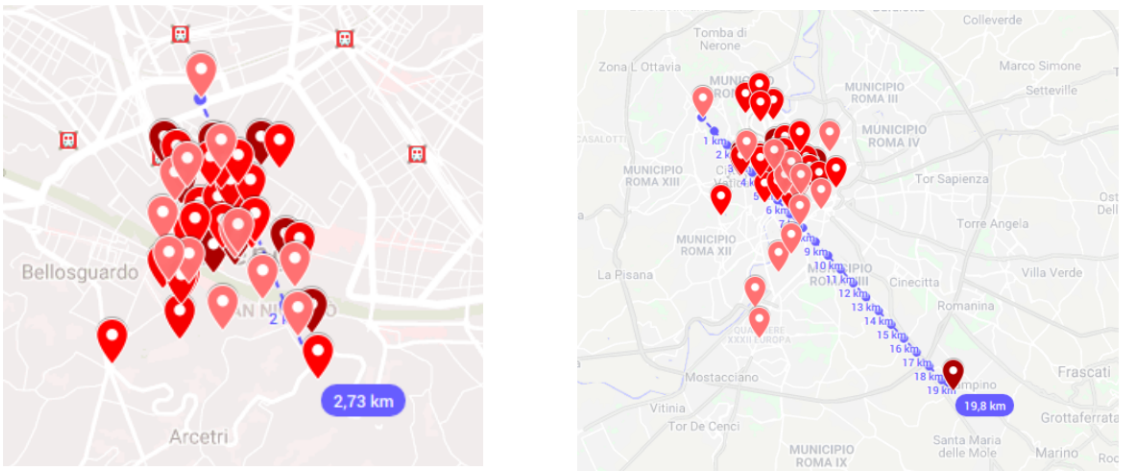
4.3.2 Tour Utility

For Pisa and Florence, the algorithms present similar results. The generated tours indeed favor for visiting/enjoying PoIs instead of traveling between them. Despite the similarity, TM shows a slightly better result in terms of tour utility. However, TRAJECTME outperforms MA in Rome, in particular for $D = 2$ days. The reason why this superiority for that case were trajectories. The trajectories carry the knowledge of the crowd, i.e., of the tourists who made it. Assuming that tourists enjoy the maximum amount of time available to visit, we can say that the trajectories are excellent initial solutions for time planning. In MA, the initial solutions generated by a greedy algorithm. Such algorithm focuses on having results that will be used only to get a preview about the possible score between the hotel pairs (DIVSALAR *et al.*, 2013), and can lead to solutions with PoIs distant from each other. Hence, the generated tours do not privilege the visit of PoIs and are penalized by a higher total traveling time. In summary, TRAJECTME significantly overcomes MA in terms of tour utility, reflecting the quality of the generated tours.

We also empirically observed the impact of the territorial extension of the city on Tour Utility. Figure 19 shows the top 50 most scored PoIs in Florence and Rome. We noticed that PoIs in Rome is more distant to each other than in Florence. For instance, there are PoIs 19.8 km away from each other in Rome, while in Florence, the longest distance found was 2.73 km. Thus, selecting relevant PoIs for the tour become harder, especially when the initial population does not contain good solutions. Distant PoIs means that the solutions can include costly paths

spending a lot of time on the trip and few for visitation, i.e., reducing the Tour Utility.

Figure 19 – Top 50 most scored PoIs in Florence (left) and Rome (right). The darker the higher the score. The highest distance between these PoIs is 2.73 and 19.8 km, respectively.



4.3.3 *Sensitivity of k*

The intuition behind k's value is that higher values can increase the number of hotels to be included in solutions, and lower values decrease instead. Therefore, k would affect the quality of solutions. However, the results show that for k values intuitively chosen, it does not present a default behavior on values. We observe great contrasts between the two general approaches: MA and TRAJECTME. But between TM-k versions, no pattern was observed in the results. Maybe the low variations between these versions were due to the randomness present in some steps of the algorithm. Some questions will be answered in the future about k variation. Would enough only the closest hotel? Would k a consequence of other instance properties, such as the amount or geologically distribution of hotels, trajectories, or PoIs?

Unfortunately, it wasn't possible to precisely determine the role of k on the quality of the solutions, but its impact on performance was easily noticed. The next section will present the results on algorithm performance and how k can reduce the time need to compute good solutions.

4.3.4 *CPU Time Performance*

MA is too fast when runs over tiny instances. It can solve instances with up to 15 hotels and one hundred of PoIs in a matter of seconds with the score close to optimal solutions. However, when it comes to real-world tourism context, due to the high complexity, i.e., the

quantity of possible combinations of PoIs-hotels-days, MA's performance slows down. Thanks to Hotel Graph Model (HGM), which is the main feature of TRAJECTME , we can handle big instance in less CPU time.

MA spends a lot of time in the initialization phase. It solves an Orienteering Problem between each pair of hotels through a greedy sub-op heuristic (DIVSALAR *et al.*, 2013). For example, in Rome instance, MA must compute one million OP solutions in the initialization phase (1000×1000 pairs of hotels). On the other hand, HGM allows TRAJECTME start quickly: it founds the k nearest hotels to each trajectory's end a use trajectory itself as the initial solution. This search is boosted by an R-Tree index. After that, the amount of hotels considered to next steps tends to be proportional according to k. High k values mean that more hotels will be considered to find solutions.

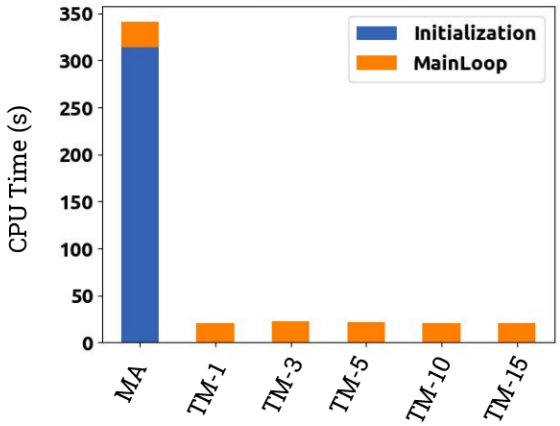
When k is high enough, and the amount of hotels grows, TRAJECTME approaches of MA, which considers all the hotels. Low k values instead mean that fewer hotels will be selected from the hotel set. Which in turn means fewer initial solutions will be found.

TRAJECTME makes the trajectory itself as the initial solution between each pair of hotels selected, i.e., it doesn't need any additional computation to find initial solutions between pairs of hotels. Moreover, the search space becomes smaller and more accurate: by using trajectories as the basis for selecting hotels, TRAJECTME prioritizes the hotels closest to the PoIs. This means that the resulting search space is formed mostly by well-located hotels close to the main tourist attractions, allowing good solutions to be found more quickly.

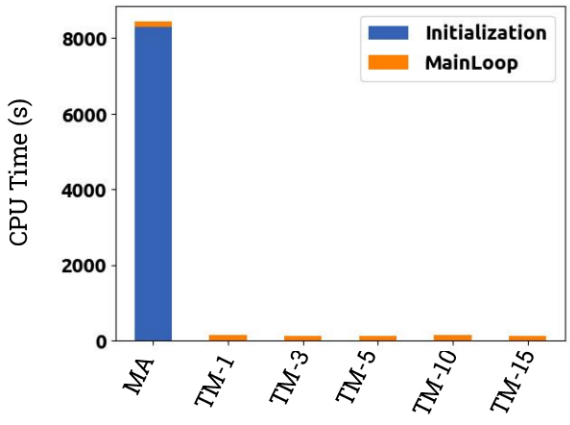
Figure 20 shows that TRAJECTME clearly outperforms MA in order of scale. While MA spend hours to initialize, computing its initial solutions, TRAJECTME initialize almost instantly. TRAJECTME's initialization is a tiny portion of its execution.

Figure 20 – CPU time performance benchmark. Time in seconds.

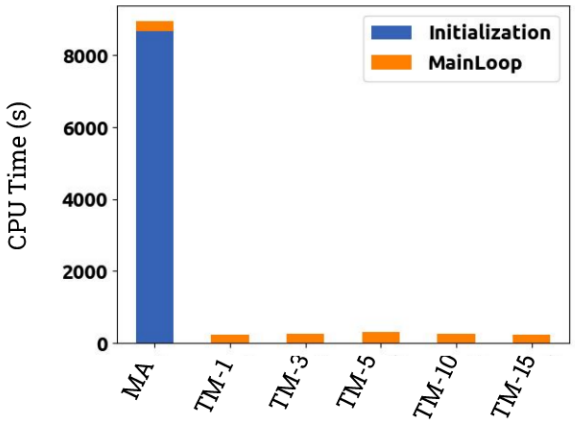
(a) Pisa, 2 days



(b) Florence, 4 days



(c) Rome, 4 days



5 CONCLUSION

In this work, we presented our proposal TRAJECTME, a memetic algorithm boosted by a hotel graph model (HGM) to solve the orienteering problem with hotel selection (OPHS) in a scenario with real datasets provided by location-based services and thousands of hotels. TRAJECTME extends the work proposed in (DIVSALAR *et al.*, 2014) by incorporating a hotel graph model that carries historical trajectories of tourists in the city to support the generation of tours for the OPHS in a more effective fashion. Trajectories of tourists were reconstructed combining photos collected from Flickr and PoIs from Foursquare.

In Chapter 2, we presented an extensive literature review of TRAJECTME's related problems and works. For each type of problem, we present the definition, the practical applications and the approaches proposed in the literature to solve it. We also presented the Tourism Trip Design Problem, which is a class of problems that mainly encompasses the task of touristic routing and, therefore, has been studied in this work. At the end of that review, we presented the related works and highlight two of them for their importance on the basic operation of the TRAJECTME. We related the key features of each work and how they influenced the design of the proposed algorithm.

We detailed our proposal in the next chapter (Chapter 3). We divided our approach presentation into three sections. In the first one, we presented the HGM and what its role in the proposed algorithm. In the following one, we presented the necessary definitions and detailed all the steps of the TRAJECTME's workflow and how it uses the HGM to optimize initialization and other phases. In the last one, we presented how and from where the data for the proper operation of TRAJECTME are obtained and processed.

To validate our work, we experimented it with real data collected for three cities of different sizes in terms of territorial extension, number of PoIs and hotels. We compared TRAJECTME to the Memetic Algorithm (DIVSALAR *et al.*, 2014) under two evaluation metrics to demonstrate the effectiveness of our proposal in solving OPHS with real datasets and thousands of hotels. The results showed that both proposal generate good solutions for a small city, such as Pisa. However, we significantly overcome the baseline achieving better efficiency, mainly when the number of hotels and PoIs are large. We also analyzed the time performance of both the algorithms over the same instances. As a result, we clearly see that TRAJECTME outperforms the MA in CPU time since MA spent a lot of time at initialization phase.

We envision several future works. First, we need to evaluate the applicability of this

work for real users, such as extend our proposal towards a personalized tour generation with hotel selection based on the preferences of the users and categories of the PoIs. We also need to understand the behavior of changing hotels, since it's uncommon to stay in more than one hotel in the same city. An interesting direction is to exploit our HGM to deal with regions, such as states and countries, thus a complete tour with hotel selection can be efficiently generated crossing many cities. Besides, we aim to figure out the significance of k and the other parameters used in our experiments to better understand the impact of the results in terms of performance and effectiveness for finding tours. Our proposal rely on the connectivity present in the HGM to be able to generate the tours. We intend to investigate the scenario when there's not enough trajectory available. New connections can be incrementally created to mitigate the sparse data problem using heuristics and link prediction approaches.

REFERENCES

ARCHETTI, C.; HERTZ, A.; SPERANZA, M. G. Metaheuristics for the team orienteering problem. **Journal of Heuristics**, Springer, v. 13, n. 1, p. 49–76, 2007.

ARKIN, E. M.; MITCHELL, J. S.; NARASIMHAN, G. Resource-constrained geometric network optimization. In: ACM. **Proceedings of the fourteenth annual symposium on Computational geometry**. [S.l.], 1998. p. 307–316.

BOUSSIER, S.; FEILLET, D.; GENDREAU, M. An exact algorithm for team orienteering problems. **4or**, Springer, v. 5, n. 3, p. 211–230, 2007.

BRILHANTE, I. R.; MACEDO, J. A.; NARDINI, F. M.; PEREGO, R.; RENSO, C. On planning sightseeing tours with TripBuilder. **Information Processing and Management**, v. 51, n. 2, p. 1–15, 2015. ISSN 03064573.

BUTT, S. E.; CAVALIER, T. M. A heuristic for the multiple tour maximum collection problem. **Computers & Operations Research**, Elsevier, v. 21, n. 1, p. 101–111, 1994.

BUTT, S. E.; RYAN, D. M. An optimal solution procedure for the multiple tour maximum collection problem using column generation. **Computers & Operations Research**, Elsevier, v. 26, n. 4, p. 427–441, 1999.

CASTRO, M.; SÖRENSEN, K.; VANSTEENWEGEN, P.; GOOS, P. A fast metaheuristic for the travelling salesperson problem with hotel selection. **4OR**, v. 13, n. 1, 2015. ISSN 16142411.

CHAO, I. *et al.* Algorithms and solutions to multi-level vehicle routing problems. University of Maryland at College Park, 1993.

CHAO, I.-M.; GOLDEN, B. L.; WASIL, E. A. EUROPEAN JOURNAL OF OPERATIONAL RESEARCH The team orienteering problem. **European Journal of Operational Research**, v. 88, p. 464–474, 1996.

CHAO, I.-M.; GOLDEN, B. L.; WASIL, E. A. A fast and effective heuristic for the orienteering problem. **European journal of operational research**, Elsevier, v. 88, n. 3, p. 475–489, 1996.

DIVSALAR, A.; VANSTEENWEGEN, P.; CATTRYSSE, D. A variable neighborhood search method for the orienteering problem with hotel selection. **International Journal of Production Economics**, Elsevier, v. 145, n. 1, p. 150–160, 2013.

DIVSALAR, A.; VANSTEENWEGEN, P.; SÖRENSEN, K.; CATTRYSSE, D. A memetic algorithm for the orienteering problem with hotel selection. **European Journal of Operational Research**, Elsevier B.V., v. 237, n. 1, p. 29–49, 2014. ISSN 03772217. Disponível em: <http://dx.doi.org/10.1016/j.ejor.2014.01.001>.

DUARTE, A.; MLADENOVIĆ, N.; SÁNCHEZ-ORO, J.; TODOSIJEVIĆ, R. Variable neighborhood descent. **Handbook of Heuristics**, Springer, p. 1–27, 2016.

FISCHETTI, M.; GONZALEZ, J. J. S.; TOTH, P. Solving the orienteering problem through branch-and-cut. **INFORMS Journal on Computing**, INFORMS, v. 10, n. 2, p. 133–148, 1998.

GARCIA, A.; LINAZA, M. T.; ARBELAITZ, O.; VANSTEENWEGEN, P. Intelligent routing system for a personalised electronic tourist guide. **Information and communication technologies in tourism 2009**, Springer, p. 185–197, 2009.

GAVALAS, D.; KENTERIS, M.; KONSTANTOPOULOS, C.; PANTZIOU, G. Web application for recommending personalised mobile tourist routes. **IET software**, IET, v. 6, n. 4, p. 313–322, 2012.

GAVALAS, D.; KONSTANTOPOULOS, C.; MASTAKAS, K.; PANTZIOU, G. A survey on algorithmic approaches for solving tourist trip design problems. **Journal of Heuristics**, Springer, v. 20, n. 3, p. 291–328, 2014.

GENDREAU, M.; LAPORTE, G.; SEMET, F. A branch-and-cut algorithm for the undirected selective traveling salesman problem. v. 32, p. 263–273, 12 1998.

GENDREAU, M.; LAPORTE, G.; SEMET, F. A tabu search heuristic for the undirected selective travelling salesman problem. **European Journal of Operational Research**, Elsevier, v. 106, n. 2-3, p. 539–545, 1998.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 1st. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 0201157675.

GOLDEN, B. L.; LEVY, L.; VOHRA, R. The orienteering problem. **Naval Research Logistics (NRL)**, Wiley Online Library, v. 34, n. 3, p. 307–318, 1987.

KANTOR, M. G.; ROSENWEIN, M. B. The orienteering problem with time windows. **Journal of the Operational Research Society**, Taylor & Francis, v. 43, n. 6, p. 629–635, 1992.

KATAOKA, S.; MORITO, S. An algorithm for single constraint maximum collection problem. **Journal of the Operations Research Society of Japan**, The Operations Research Society of Japan, v. 31, n. 4, p. 515–531, 1988.

KE, L.; ARCHETTI, C.; FENG, Z. Ants can solve the team orienteering problem. **Computers & Industrial Engineering**, Elsevier, v. 54, n. 3, p. 648–665, 2008.

KENTERIS, M.; GAVALAS, D.; ECONOMOU, D. An innovative mobile electronic tourist guide application. **Personal and ubiquitous computing**, Springer-Verlag, v. 13, n. 2, p. 103–118, 2009.

LAPORTE, G.; MARTELLO, S. The selective travelling salesman problem. **Discrete applied mathematics**, North-Holland, v. 26, n. 2-3, p. 193–207, 1990.

LIN, S. Computer solutions of the traveling salesman problem. **Bell System Technical Journal**, Wiley Online Library, v. 44, n. 10, p. 2245–2269, 1965.

MANSINI, R.; PELIZZARI, M.; WOLFER, R. **A granular variable neighbourhood search heuristic for the tour orienteering problem with time windows**. [S.l.], 2006.

MONTEMANNI, R.; GAMBARDELLA, L. M. An ant colony system for team orienteering problems with time windows. **Foundation Of Computing And Decision Sciences**, v. 34, n. 4, p. 287, 2009.

MTRIP. **mtrip Redefining Mobile in Travel**. 2018. Disponível em: <https://www.mtrip.com/>.

OLIVEIRA, E.; BRILHANTE, I. R.; MACEDO, J. A. F. de. Trajectme: Planning sightseeing tours with hotel selection from trajectory data. In: ACM. **Proceedings of the 2nd ACM SIGSPATIAL Workshop on Recommendations for Location-based Services and Social Networks**. [S.l.], 2018. p. 1.

PLANNER, C. T. **City Trip Planner The Ultimate City Break Guide!** 2018. Disponível em: <http://www.citytripplanner.com/>.

RAMESH, R.; BROWN, K. M. An efficient four-phase heuristic for the generalized orienteering problem. **Computers & Operations Research**, Elsevier, v. 18, n. 2, p. 151–165, 1991.

RAMESH, R.; YOON, Y.-S.; KARWAN, M. H. An optimal algorithm for the orienteering tour problem. **ORSA Journal on Computing**, INFORMS, v. 4, n. 2, p. 155–165, 1992.

RIGHINI, G. Dynamic programming for the orienteering problem with time windows. Università degli Studi di Milano-Polo Didattico e di Ricerca di Crema, 2006.

RIGHINI, G.; SALANI, M. New dynamic programming algorithms for the resource constrained elementary shortest path problem. **Networks: An International Journal**, Wiley Online Library, v. 51, n. 3, p. 155–170, 2008.

RIGHINI, G.; SALANI, M. Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming. **Computers & Operations Research**, Elsevier, v. 36, n. 4, p. 1191–1203, 2009.

SCHILDE, M.; DOERNER, K. F.; HARTL, R. F.; KIECHLE, G. Metaheuristics for the bi-objective orienteering problem. **Swarm Intelligence**, Springer, v. 3, n. 3, p. 179–201, 2009.

SOUFFRIAU, W.; VANSTEENWEGEN, P.; BERGHE, G. V.; OUDHEUSDEN, D. V. A path relinking approach for the team orienteering problem. **Computers & operations research**, Elsevier, v. 37, n. 11, p. 1853–1859, 2010.

SOUFFRIAU, W.; VANSTEENWEGEN, P.; BERGHE, G. V.; OUDHEUSDEN, D. V. The multiconstraint team orienteering problem with multiple time windows. **Transportation Science**, INFORMS, v. 47, n. 1, p. 53–63, 2013.

SOUFFRIAU, W.; VANSTEENWEGEN, P.; VERTOMMEN, J.; BERGHE, G. V.; OUDHEUSDEN, D. V. A personalized tourist trip design algorithm for mobile tourist guides. v. 22, p. 964–985, 10 2008.

TANG, H.; MILLER-HOOKS, E. A tabu search heuristic for the team orienteering problem. **Computers & Operations Research**, Elsevier, v. 32, n. 6, p. 1379–1407, 2005.

THOMADSEN, T.; STIDSEN, T. **The Quadratic Selective Travelling Salesman Problem**. Richard Petersens Plads, Building 305, DK-2800 Kgs. Lyngby, 2003. (IMM-Technical Report-2003-17). Disponível em: <http://www2.imm.dtu.dk/pubdb/p.php?2524>.

TRICOIRE, F.; ROMAUCH, M.; DOERNER, K. F.; HARTL, R. F. Heuristics for the multi-period orienteering problem with multiple time windows. **Computers & Operations Research**, Elsevier, v. 37, n. 2, p. 351–367, 2010.

TSILIGIRIDES, T. Heuristic methods applied to orienteering. **Journal of the Operational Research Society**, Springer, v. 35, n. 9, p. 797–809, 1984.

Van Hoek, S. Tabu Search for the Orienteering Problem with Hotel Selection. 2016.

VANSTEENWEGEN, P.; OUDHEUSDEN, D. V. The mobile tourist guide: an or opportunity. **OR insight**, Springer, v. 20, n. 3, p. 21–27, 2007.

VANSTEENWEGEN, P.; SOUFFRIAU, W.; BERGHE, G. V.; OUDHEUSDEN, D. V. A guided local search metaheuristic for the team orienteering problem. **European journal of operational research**, Elsevier, v. 196, n. 1, p. 118–127, 2009.

VANSTEENWEGEN, P.; SOUFFRIAU, W.; BERGHE, G. V.; OUDHEUSDEN, D. V. Iterated local search for the team orienteering problem with time windows. **Computers & Operations Research**, Elsevier, v. 36, n. 12, p. 3281–3290, 2009.

VANSTEENWEGEN, P.; SOUFFRIAU, W.; BERGHE, G. V.; OUDHEUSDEN, D. V. Metaheuristics for tourist trip planning. In: **Metaheuristics in the service industry**. [S.l.]: Springer, 2009. p. 15–31.

VANSTEENWEGEN, P.; SOUFFRIAU, W.; BERGHE, G. V.; OUDHEUSDEN, D. V. The city trip planner: an expert system for tourists. **Expert Systems with Applications**, Elsevier, v. 38, n. 6, p. 6540–6546, 2011.

VANSTEENWEGEN, P.; SOUFFRIAU, W.; OUDHEUSDEN, D. V. The orienteering problem: A survey. **European Journal of Operational Research**, Elsevier, v. 209, n. 1, p. 1–10, 2011.

WANG, X.; GOLDEN, B. L.; WASIL, E. A. Using a genetic algorithm to solve the generalized orienteering problem. In: **The vehicle routing problem: latest advances and new challenges**. [S.l.]: Springer, 2008. p. 263–274.