Dipartimento di Informatica
Università degli Studi di Verona

Rapporto di ricerca
Research report
May 2018

RR 107/2018

Adaptive Trip Recommendation System

Alberto Belussi
Damiano Carra
Sara Migliorini

Questo rapporto è disponibile su Web all'indirizzo:
This report is available on the web at the address:
http://www.di.univr.it/report

Abstract

Travel recommendation systems provide suggestions to the users based on different information, such as user preferences, needs, or constraints. The recommendation may also take into account some characteristics of the *points of interest* (POIs) to be visited, such as the opening hours, or the peak hours. Although a number of studies have been proposed on the topic, most of them tailor the recommendation considering the user viewpoint, without evaluating the impact of the suggestions on the system as a whole. This may lead to oscillatory dynamics, where the choices made by the recommendation system generate new peak hours.

This paper considers the trip planning problem that takes into account the balancing of users among the different POIs. To this aim, we consider the estimate of the level of crowding at POIs, including both the historical data and the effects of the recommendation. We formulate the problem as a multi-objective optimization problem, and we design a recommendation engine that explores the solution space in near real-time, through a distributed version of the Simulated Annealing approach. Through an experimental evaluation on a real dataset of users visiting the POIs of a touristic city, we show that our solution is able to provide high quality recommendations, yet maintaining the attractions not overcrowded.

Keywords: Trip recommendation, Simulated Annealing, Map Reduce, SpatialHadoop

1 Introduction

Traveling is part of many people leisure activities, and an increasing fraction of the economy comes from the tourism. Vising a city is a common choice for a short-term trip: besides the well known destinations, such as New York or Paris, many cities are becoming popular destinations, for instance, during the weekend, or as an intermediate stop while reaching other places. Each destination contains many attractions, or *Points of Interests* (POIs), which are listed in different sources. For instance, travel guides, such as Lonely Planet, provide different suggestions based on the available time. Other options are the Location Based Social Networks (LBSNs) [1], which collect the travellers' experiences to derive popular attractions. Many cities have tourist offices, which provide general information about the city and its POIs. In order to facilitate the visiting experience, sometimes popular POIs are bundled, so that the tourists have a single "city pass" that can be used to guide their choices. Still, once the tourists have a list of POIs to visit, how can they make the most of them given their available time?

Trip recommendation systems deal with this kind of issues. In their essence, these systems need to solve an optimization problem [2], such as the Traveling Salesman Problem, which is NP-hard. Very often, the trip recommendation systems try to provide a solution taking into account not only the tourist available time, but also other elements, such as their personal interests, or budget [3]. Therefore, these systems need to solve a multi-objective optimization problem, whose complexity is further increased. The solutions proposed in the literature usually deal with well-known heuristics for local optimization: they translate the user requirements to an utility function, and they adopt different techniques (e.g., gradient descent) to explore the solution space. Since the heuristics have been proven to approximate efficiently the optimal solution, the literature has recently focused on more complex scenarios that take into accounts the user needs and constraints: for instance, the user previous experiences may influence the POI choice, or the POI opening hours may determine its position in the sequence of POIs to be visited [4]. The aim of the trip recommendation system becomes to *tailor* the suggestions to the specific user.

Limitation of the prior works. The proposed solutions concentrate their attention to the user needs and viewpoints: the systems take as input the user preferences and some information about the POIs, and provide a recommended trip. The information about the POIs are "static", such as the opening hours, or an estimate of the busy periods (see Sect. 2 for a review of the literature). The fact that the suggestions have an impact on the status of the POIs is not considered in the recommendation engine. In other words,

the optimization is based on the users, not on the system as a whole. For instance, if the recommendation system considers the busy hours of the POIs of the previous day or week, it will generate trips trying to avoid the busy hours of the different POIs. This may generate different busy hours, since many of the users may be directed to a specific POI at the same time. This oscillatory dynamics have been observed in routing algorithms that take into accounts the current state of the routes [5]. In order to avoid such dynamics, the system should estimate the *effect of the trip recommendation on the system itself*. Considering the whole system in turn has an impact on the user experience: balancing the user among the different POIs means avoiding the formation of crowds, with the corresponding possible delays due to the increased number of people to manage.

Proposed approach. In this paper we consider the trip planning problem that takes into account, besides the user preferences and the system constraints, the balancing of users among the different POIs. The recommendation engine needs to consider the prediction of the user presence at the POIs. The quality of the prediction determines the quality of the recommendation: the prediction should include historical data, as well as the recommendations made so far by the system itself. There are a number of challenges that need to be faced to design such a system. First, the user requests are usually issued by a mobile application, where the user expects a near real-time response: the solution space, therefore, should be explored in a limited timeframe. Another issue regards the necessity to understand the impact of the estimation error – due to some unpredictable user behavior – on the effectiveness of the balancing process. Finally, in order to increase the effectiveness of the recommendations, the constraints used by the system for comparing possible solution instances should include spatial properties, like for example the total trip distance computed on a network with different traveling modes.

Motivating example. Figure 1 shows an example taken from a real-world case regarding the city of Verona in northern Italy. The map reports a set of POIs covered by a "city pass", called *VeronaCard*. Each POI is represented by a circle of variable size which indicates an estimation of the current number of visiting tourists. The figure displays also a trip performed by a tourist (the green one), who travels from p_1 = "Arena" to p_2 = "Casa di Giulietta", without immediately visiting it; in fact, she goes over to reach p_3 = "Chiesa di San Fermo Maggiore" and comes back to p_2 at the end. The POI p_2 is visited after p_3 due to the number of tourists currently on p_2, producing a trip which is not optimal w.r.t. to the distance travelled. Namely, if the tourist is able to know in advance the level of crowding in each POI, she can plan a better trip (e.g., the blue one).

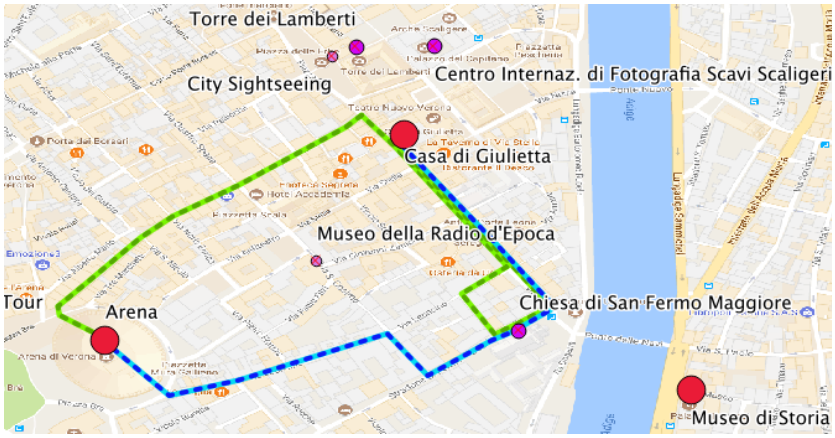

Figure 1: Example of trip performed by using the "city pass" VeronaCard and influenced by the number of tourists currently visiting the POIs. The level of crowding is represented by the circle size. The green line is the real trip, and the blu line is a recommended trip that avoids the crowded POI.

Key contributions. The contributions of our work are the following: (1) we formulate the online optimization problem, where we consider the current estimation of the user visiting the different POIs as part of the input of the recommendation system. (2) We design and implement an efficient solution engine that works in near real-time. The solution is based on a parallel version of the Simulated Annealing approach, using the MapReduce programming framework. (3) We evaluate the trip recommendation system with a dataset collected from the tourist information office of the city of Verona. The dataset contains the visits of different POIs included in a set of city passes, whose duration is pre-determined (e.g, 1-day pass, 2-day pass).

The remainder of this paper is organized as follows. In Section 2 we discuss the related works. Section 3 is devoted to the problem formulation. In Section 4 we present the system, the algorithms for trip recommendation, and their implementation in MapReduce. Finally, in Section 5 we discuss the experimental results and we conclude the paper in Section 6.

2 Related Work

This section reviews the related works focusing on two main topics: (i) trip or itinerary recommendation, and (ii) computational aspects of the solution of the optimization problem.

Recommendation systems. This topic has received a lot of attention in recent years, therefore the related literature is vast. Here, due to space

constraints, we highlight some representative works based on the taxonomy provided in two recent surveys [1] [6]. The interested reader can find more details in such surveys and the references therein.

The main problem to consider is the identification of the POIs and their relevance. The data used to find POIs can be gathered from different sources, such as user check-in behaviours [7,8], crowdsourced digital footprints [9,10], GPS data [11,12], or it can be inferred by using geographical or social correlations of visited POIs [13–15]. Once the system has the list of POIs, it needs to select the subset of POIs that are relevant to the user. The recommendation may take into account multiple constraints [2,16] or constraints related to time [4,17]. The POIs can be used to build semantically enriched trajectories – for a survey on the topic, please refer to [18]. All the above systems are focused on the user viewpoint to provide a tailored recommendation. Only some of them include geographical consideration in building the itinerary, and none of them adapts the solution considering the number of users that can be present in the POIs.

The only work that considers how much a POI may be crowded is [3]. Nevertheless, the proposed system bases its recommendations on instantaneous information, therefore it may generate new peak hours at the different POIs. Moreover, the authors do not consider the geographical aspects in building the itinerary. To the best of our knowledge, our work is the first that takes into consideration the impact of the recommendations on the current and future level of crowding, so that to balance the users among the POIs.

Optimization problem. Approximate solutions of optimization problems have been extensively treated in the literature. Hoos et al. [19] provide a broad view of the techniques and the solutions adopted so far. Since we are interested in a near real-time system, we focus on some works that deal with the parallel implementation of a specific technique, i.e., the Simulated Annealing (SA).

A common approach is to adopt an Asynchronous Approach [20, 21], where different workers executes independent SA using different starting solutions, and the best solution among them is reported. Inspired by such an approach, the authors in [22, 23] propose different MapReduce implementations, where the computations is divided among Map and Reduce tasks in different ways. The solution of multi-objective optimization problems using SA have been considered in [24, 25], and its parallel implementation in [26]. To the best of our knowledge, these parallel implementations have never been adapted to the MapReduce framework. In our work, we take inspirations from the above mentioned works to design a MapReduce implementation of the solution of a multi-objective optimization problem.

3 Problem formulation

In this section we provide the necessary definitions and formalize the trip planning problem we want to consider.

Definition 1 (Point of interest)**.** *A point of interest (POI) p represents an attraction reachable by users. It is characterized by several attributes, such as the admission fee, or the opening hours. Among these, we consider in particular the spatial coordinates defining its position on the Earth surface, which we denote with p^c. We also consider the* duration of a visit*, denoted by $p^v(t)$, which depends on the instant t when the visit starts.*

The dependency on t is necessary since $p^v(t)$ is influenced by many factors, such as the day of the week, and the number of people currently visiting the POI p. We will show in Sect. 4 how we compute (and update) the value of $p^v(t)$. For the purposes of this paper, the set of POIs that can be considered for building a trip is assumed to be known and fixed, and is denoted by $\mathcal{P}$.

Definition 2 (Trip)**.** *A trip τ is an ordered collection of POIs, i.e., $\tau = \langle p_1, p_2, \ldots, p_n \rangle$, where n indicates the number of POIs contained in τ, $|\tau| = n$.*

Given the set of POIs, $\mathcal{P}$, the set of all possible trips, denoted by $\mathcal{T}$, contains all the possible ordered combination of POIs, for any cardinality of τ.

Definition 3 (Path)**.** *Given any two spatial coordinates c_i and c_j, and a travel mode m (e.g, walking, public transportation), a path $\pi(c_i, c_j, m)$ is a continuous portion of a transport network that connects the points whose location is defined by c_i and c_j. The path is characterized by the travel distance, $\pi_{td}(c_i, c_j, m)$, and by the travel time, $\pi_{tt}(c_i, c_j, m)$.*

Notice that, in order to maintain the notation simple, we may not indicate the dependency of π_{td} (π_{tt}) on the travel mode, which is specified by the user when she/he submits the query to the system.

Definition 4 (Recommendation query)**.** *Users looking for a recommendation submit a query $\mathcal{Q}$ to the system by specifying the following constraints:*

- *the initial coordinates c_0 where the trip begins (user position);*
- *the time at which the trip will start t_0;*
- *the maximum trip duration $TD_{\max}$;*
- *the travel mode m.*

In order to reply to such a query, the system needs to compute a set of values that drives the trip selection. We start considering the main constraint, i.e., the total time of the duration of the trip should be less than $\mathrm{TD}_{\max}$. To this aim, we introduce a fictional POI p_0, which corresponds to the user initial position, and we set $p_0^c = c_0$ and $p_0^v(t_0) = 0$. We denote with t_i the time of arrival at p_i, the i-th POI of the trip, which can be computed considering the time t_{i-1}, the visit time of the previous POI and the travel time between the two POIs, i.e.,

$$t_i = t_{i-1} + p_{i-1}^v(t_{i-1}) + \pi_{tt}(p_{i-1}^c, p_i^c), \quad i \geq 1. \tag{1}$$

Note that $t_1 = t_0 + p_0^v(t_0) + \pi_{tt}(p_0^c, p_1^c) = t_0 + \pi_{tt}(c_0, p_1^c)$, which represents the staring time of the trip plus the travel time between the user position and the first POI. We can now define the total trip time λ_τ for a trip τ as:

$$\lambda_\tau(c_0, t_0) = \sum_{i=1}^{n} \left(\pi_{tt}(p_{i-1}^c, p_i^c) + p_i^v(t_i) \right), \tag{2}$$

where $n = |\tau|$. When exploring the solution space, the system will consider the trips for which $\lambda_\tau(c_0, t_0) < \mathrm{TD}_{\max}$. The exploration is guided by the values of the objective function. In the following, we consider a set of possible optimization criteria that can be minimized. For simplicity, we focus mainly of three criteria: adding more objective functions is cumbersome.

Definition 5 (Objective functions)**.** *Given a trip τ, the objective functions f_n, f_{tt} and f_{td} denotes the number of locations* not *visited during the trip, the estimated trip travel time, and the total distance travelled during the trip respectively. They are computed as:*

$$\begin{aligned} f_n &= |\mathcal{P}| - |\tau| \\ f_{tt} &= \sum_{i=1}^{n} \pi_{tt}(p_{i-1}^c, p_i^c) \\ f_{td} &= \sum_{i=1}^{n} \pi_{td}(p_{i-1}^c, p_i^c) \end{aligned} \tag{3}$$

where $n = |\tau|$.

When defining f_n, we use the number of locations *not* visited (instead of the ones visited), so that all objective functions need to be minimized.

We are now ready to define the trip planning problem, which can be cast as an optimization problem:

$$\begin{aligned} \underset{\tau}{\text{Minimize}} \quad & \langle f_n, f_{tt}, f_{td} \rangle, \\ \text{subject to} \quad & \lambda_\tau(c_0, t_0) < \mathrm{TD}_{\max} \end{aligned} \tag{4}$$

Note that the global objective function we would like to minimize is a composition of objective functions, and it can be defined as $\bar{f} : \mathcal{T} \rightarrow \mathbb{R}^3$. We are therefore in the context of *multi-objective* optimization, in which is not possible to define a total order. We need to introduce a *dominance* relation to partially order the set of possible solutions. A trip τ_i dominates a trip τ_j, denoted $\tau_i \prec \tau_j$, if at least one of the composing objective functions is smaller for τ_i than for τ_j, while the other are equivalent. The results of the optimization problem will be the set of mutually non-dominating trips, i.e.,

$$res(\mathcal{Q}) = \{\tau \in \mathcal{T} \mid \nexists \tau_0 \in \mathcal{T} \text{such that } \tau_0 \prec \tau\}$$

Considering the cardinality of the set containing all the possible trips, $\mathcal{T}$, the solution space to explore in order to provide a recommendation is very large. In addition, note that the total trip time depends on the POI visit duration, which depends on the time when the visit starts, therefore the solution space further increases. For this reason, our search for the solution is based on well known heuristics for solving the optimization problem.

4 Proposed system

4.1 Overview

Our proposed recommendation system has two main components: an offline analysis of the user presence in the different POIs, and a recommendation engine based on a parallel implementation divided into two main stages. Figure 2 shows the overall architecture. User presence at the POIs is collected in a database. Overnight, the new records are processed by an offline procedure that extracts the main information and updates the data structures that are used by the recommendation engine. In particular, a set of common trips are stored back in the database, while the POI profiles (see Sect. 4.2) are stored in a distributed cache. The recommendation engine itself is composed by two steps: given a query, it first extracts a set of possible solutions from the set of common trips, then it explores the solution space looking for improved solutions by taking into account the optimization criteria, which include the time spent in each POI.

Once the system has issued the recommendation, it updates the POI profiles to reflect the impact of such recommendation on the POI crowding. Therefore, when the next user will submit a query, the recommendation engine will use the updated values of POI profiles while computing the optimal solution.

In the following sections, we describe in detail the different system components.

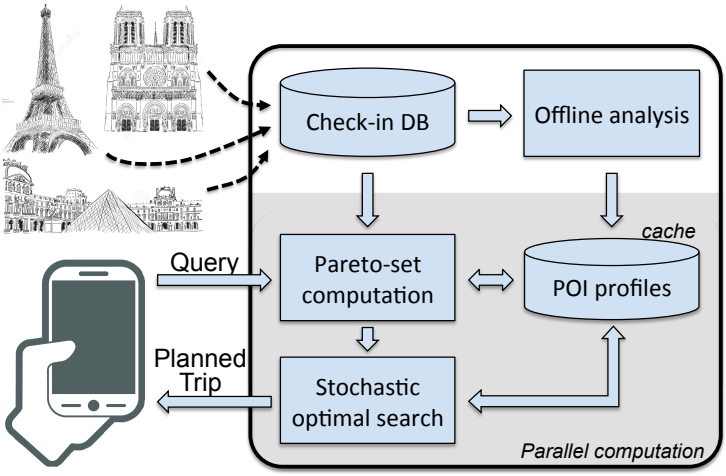

Figure 2: System architecure.

4.2 Offline analysis of the check-ins

The POIs record the entering and the exiting visitors: in fact, for security reasons, it is necessary to know how many people are inside a POI. Every time a new entrance is recorded, the POI sends to the database a record $r = \langle t^{ci}, p, u^{id}, n^p \rangle$, where t^{ci} is the check-in (entrance) timestamp, $p \in \mathcal{P}$ is the POI, u^{id} is the user identifier (if it exists), and n^p is the current number of visitors inside the POI p.

There are two types of users: anonymous and registered. Registered users are people who use, for instance, a bundle offer, where they receive an identifier and they can access to a set of POIs with reduced prices (e.g., a city-pass, or an app for their smartphones). All the other users that cannot be identified, such as local users who visit a single POI, are anonymous.

Having registered users is important, since it is possible to reconstruct the set of POIs that they visited. The offline analysis of the registered users allows to build a set of popular trips that can be used by the recommendation engine as a starting point in the search of the optimal solution when replying to a query. The set of popular trips are stored back in the main database, and accessed by the recommendation engine when it processes a query.

Another advantage derived from registered users is the possibility to compute an important metric: the visiting time. Given a registered user u^{id}, for any two consecutive visited POIs, p_j and p_k, the offline analysis can compute the actual time spent at POI p_j by subtracting from the interval between the timestamps t_j^{ci} and t_k^{ci} the travel time from p_j to p_k, assuming a given travel mode m (e.g., the most used m). In such way, the record corresponding to

the check in of u^{id} at p_j, formally $r_j = \langle t_j^{ci}, p_j, u^{id}, n_j^p \rangle$ can be enriched by a new element, $vt_j = t_k^{ci} - t_j^{ci} - \pi_{tt}(p_j^c, p_k^c, m)$, where t_k^{ci} is taken from the record $r_k = \langle t_k^{ci}, p_k, u^{id}, n_k^p \rangle$. The records that have been enriched now contain a direct relation between the number of people inside a POI and the visiting time for that POI.

In summary, given a POI p_i, it is possible to build a set of characterizing measures, which we call *profiles*:

- **Average Time Occupancy**, ATO(d, h): for each day d of the week, it represents the average number of visitors inside the POI at time h (h has the granularity of hours); the average is calculated considering the same day for a given interval (e.g., last year);

- **Average Visiting time**, AVT(n^p): it provides the average visiting time given a number of visitors inside the POI; the average is computed considering all the enriched records, by grouping the records for the same value of n^p.

The Average Time Occupancy ATO(d, h) reflects how much crowded a POI is on average, and it should show some peaks at specific hours (e.g. mid morning). As for the Average Visiting Time AVT(n^p), intuitively it should be an increasing function, i.e, as the number of user inside a POI increases, the visiting time should increase, since the crowding slows the visit. Note that these profiles can be updated incrementally: it is sufficient to maintain some additional information in the database and use such information during the update.

With the profiles defined above, the duration of a POI visit at time t, introduced in Definition 1 and denoted by $p^v(t)$, can be computed as follow. Given the time t we can derive the day d and the hour h, we then compute the average number of user for that day at that hour, $n^p = \text{ATO}(d, h)$, and then we derive the average visiting time from AVT(n^p), i.e.,

$$p^v(t) = \text{AVT}(\text{ATO}(d, h)), \tag{5}$$

with d day and h hour of the request derived from t.

The above definition may suggest that the system does not adapt to the estimated level of crowding as more and more recommendation are provided by the system, since ATO(d, h) is computed offline. We will show in Sect. 4.6 that the actual ATO$'(d, h)$ profile used by the recommendation engine contains a dynamic variable component, which is continuously updated during the day. The system, therefore, is able to tailor the recommendation to the estimated level of crowding.

4.3 Exploration of the solution space

Looking for the exact solution of the optimization problem defined in Eq. (4) is computationally expensive, therefore we need to resort to well known heuristics. Our solution builds trip recommendations using a dominance-based Multi-Objective Simulated Annealing (MOSA) [25] technique.

Multi-Objective Simulated Annealing. At each step of the simulated annealing procedure, the current solution is replaced with a random one with a probability that depends both on the difference between the corresponding objective values and a global parameter T (*temperature*), which is progressively decreased during the process. This behaviour avoids the stuck on local optima, which is the main drawback of greedy algorithms. It has been proven that, using a simulated annealing approach, the solution converges to the global optimum if annealed sufficiently slow [27].

The exploration of the solution space is based on the comparison between the current solution τ_{curr} with a new potential solution τ_{new}, obtained through a *perturbation* of the current solution τ_{curr}. The perturbation could be, for instance, a POI removal or addition, or a change in the order of the POI. The comparison is done by considering the objective function $\bar{f} = \langle f_n, f_{tt}, f_{td} \rangle$ defined in Eq. (3). As stated in Sect. 3, with multi-objective optimization, we can define a partial order on the solution based on the concept of dominance: A trip τ_{new} dominates another trip τ_{curr}, denoted as $\tau_{\text{new}} \prec \tau_{\text{curr}}$, if it is better in at least one objective function and equivalent in the remaining ones.

Trips τ_{curr} and τ_{new} are mutually non-dominating if and only if neither dominates the other. The set of mutually non-dominating solutions is called *Pareto-set*, and it is denoted by $\mathcal{S}$. A solution not dominated by any other solution is called *Pareto-optimum*. From the Pareto-set $\mathcal{S}$ we can compute the *Pareto-front* $\mathcal{F} \subseteq \mathbb{R}^3$, which is the set of points in the objective space, i.e., $\mathcal{F} = \{\bar{f}(\tau) \mid \tau \in \mathcal{S}\}$

The goal of a MOSA algorithm is to move the current Pareto-front towards the optimal Pareto-front (the Pareto-front of the Pareto-optimum set) while encouraging the diversification of the candidate solutions. In particular, the probability of making a transition from the current solution τ_{curr} towards a new solution τ_{new} is specified by an acceptance probability function $P(\tau_{\text{curr}}, \tau_{\text{new}}, T)$ which depends upon the global parameter T (*temperature*) and the energy of the two solutions. The energy of a solution τ, denoted by $E(\tau, \mathcal{F})$, measures the portion (number of solutions) of the current Pareto-front that dominates τ, i.e.,

$$E(\tau, \mathcal{F}) = |\{v \in \mathcal{F} \mid v \prec \bar{f}(\tau)\}|. \tag{6}$$

Note that the energy of a solution τ belonging to the Pareto-front is 0.

Given two solutions τ_{curr} and τ_{new}, where τ_{curr} is part of the Pareto-set, and therefore $\bar{f}(\tau_{\text{curr}})$ is part of the Pareto-front $\mathcal{F}$, we can compute the energy difference between τ_{curr} and τ_{new} by considering the extended Pareto-front $\mathcal{F}' = \mathcal{F} \cup \bar{f}(\tau_{\text{new}})$ as following:

$$\Delta_E(\tau_{\text{new}}, \tau_{\text{curr}}, \mathcal{F}') = \frac{E(\tau_{\text{new}}, \mathcal{F}') - E(\tau_{\text{curr}}, \mathcal{F}')}{|\mathcal{F}'|} \tag{7}$$

The acceptance probability $P(\tau_{\text{curr}}, \tau_{\text{new}}, T)$ is then

$$P(\tau_{\text{curr}}, \tau_{\text{new}}, T) = \min\left(1, \exp\left(-\frac{\Delta_E(\tau_{\text{new}}, \tau_{\text{curr}}, \mathcal{F}')}{T}\right)\right) \tag{8}$$

Note that it is possible to escape local optima, since a candidate solution τ_{new} that is dominated by one or more members of the current estimated Pareto-front, may still be accepted with a probability defined in Eq. (8). On the other hand, candidate solutions that move the estimated front towards the true front are always accepted, since $P(\tau_{\text{curr}}, \tau_{\text{new}}, T) = 1$. The temperature T is actually a monotonically decreasing function, that decreases at every iteration at a very slow rate till it reaches a minimum value.

Basic building block: the TRSA Algorithm. The exploration of the solution space is based on a building block called TRSA (Trip Recommendation Simulated Annealing). Starting from a given Pareto-set $\mathcal{S}_{\text{init}}$ and a trip $\tau \in \mathcal{S}_{\text{init}}$, the algorithm looks for potential new trips to be added to $\mathcal{S}_{\text{init}}$ in order to advance the Pareto-front $\mathcal{F}$. TRSA is illustrated in Algorithm 1.

The function ComputeParetoFront() uses the input Pareto-set $\mathcal{S}$ for initializing the Pareto-front $\mathcal{F}$. As long as the temperature T is greater than the minimum value T_{min}, the algorithm explores the solution space by perturbating the current solution τ. The possible perturbations are:

- adding a POI in a random position (except the first);
- removing a POI (except the first);
- replacing a POI (except the first) with another one;
- shifting the position between two POIs (except the first).

The function Perturb(), while looking for a new potential trip τ', evaluates its total trip time, $\lambda_{\tau'}$, and considers only the trips for which $\lambda_{\tau'} < \text{TD}_{\max}$.

The algorithm then computes the energy of τ' (line 7), and the probability of accepting τ' (line 8). If τ' is accepted, we remove from $\mathcal{S}$ the trips

Algorithm 1: TRSA algorithm.

Data: $\mathcal{S}_{\text{init}}$, τ, $\text{TD}_{\max}$, $T_{\min}$, T_{init}
Result: $\mathcal{S}$

```
1  S ⟵ S_init;
2  F ⟵ ComputeParetoFront(S);
3  T ⟵ T_init;
4  while T > T_min do
5      τ' ⟵ Perturb(τ, TD_max);
6      F' ⟵ F ∪ f̄(τ');
7      Δ_E ⟵ ComputeEnergyDiff(τ', τ, F');
8      P ⟵ min(1, exp(−Δ_E/T));
9      if rand(0, 1) < P then
10         RemoveDominated(S, τ', F, f̄(τ'));
11         S ⟵ S ∪ τ';
12         F ⟵ F ∪ f̄(τ');
13         τ ⟵ τ';
14     UpdateTemperature(T);
15 return S
```

dominated by τ', and from $\mathcal{F}$ the corresponding points (line 10), we add the trip to $\mathcal{S}$ and continue to explore. At each iteration, the temperature is updated according to a cooling strategy such as the ones defined in [28].

Using the TRSA Algorithm. The initial Pareto-set $\mathcal{S}_{\text{init}}$ contains a set of equivalent solutions, and the aim of TRSA is to look for better solutions starting from a trip $\tau \in \mathcal{S}_{\text{init}}$. This exploration can be done on a single machine. In parallel, other machines may try to improve the Pareto-front $\mathcal{F}$ starting from other trips $\tau' \in \mathcal{S}_{\text{init}}$. Therefore TRSA represents the basic building block of the overall parallel computation. In Sect. 4.5 we will show how these parallel computation is done with MapReduce. Before, we need to determine the main input of the TRSA algorithm: the initial Pareto-set $\mathcal{S}_{\text{init}}$. Such input can be determined in parallel using the MapReduce framework.

4.4 Initial Pareto-set

The initial Pareto-set $\mathcal{S}_{\text{init}}$ is built using the popular trips computed offline (see Sect. 4.2) and stored in the main database. The evaluation of these potential solutions include the verification of the main constraint, i.e., the duration of the selected trips cannot be longer than the total trip duration $\text{TD}_{\max}$. This is done with the help of the profiles $\text{ATO}(d, h)$ and $\text{AVT}(n^p)$

stored in the cache. Moreover, we need to check that each potential trip is not-dominated by the trips currently inserted in $\mathcal{S}_{\text{init}}$.

The evaluation of the potential trips to be added to $\mathcal{S}_{\text{init}}$ may be expensive. Nevertheless, it can be done easily in parallel, since each trip is independent from the others (see Algorithm 2). Given a query $\mathcal{Q}$ that defines the start point, the travel mode and the desired duration interval, we extract all the popular trips that have similar duration interval: the duration interval of the popular trips has been computed offline considering as a starting point the first POI of the trip, and visit time for each POI equal to the average visit time computed over all the visits at that POI. For each of these popular trips, the MAP method checks if the trip satisfies $\mathcal{Q}$, it actually computes the total duration considering the staring point specified in the query and the time at which the trip will start. If the trip satifies the query, it is added to $\mathcal{S}$. The addition is done with the help of the function UPDATE$(\mathcal{S}, \tau)$, which ensures that $\mathcal{S}$ does not contain duplicates and that dominated values are removed. Since multiple MAP calls may be executed by the same JVM, we return the $\mathcal{S}$ in the CLEANUP method called at the end of the task.

Algorithm 2: MapReduce job for the initialization of the Pareto-set $\mathcal{S}_{\text{init}}$. The key *id* read by the mapper is a generic identifier associated to each row and can be safely ignored.

```
1  class MAPPER
2      method SETUP()
3          S_map ⟵ ∅
4      method MAP(id, τ)
5          if τ satisfies Q then
6              S_map ⟵ UPDATE(S_map, τ)
7      method CLEANUP()
8          return (Q, S_map)
9  class REDUCER
10     method REDUCE(Q, ⟨S_1, S_2, ...⟩)
11         S_init ⟵ ∅
12         foreach S_i ∈ ⟨S_1, S_2, ...⟩ do
13             foreach τ ∈ S_i do
14                 S_init ⟵ UPDATE(S_init, τ)
15         return (Q, S_init)
```

The REDUCE method collects the partial Pareto-sets computed by the MAP tasks, and merge them using the UPDATE($\mathcal{S}, \tau$). Since it is necessary to verify that the merged Pareto-sets do not contain dominated solutions, there could be only one reducer. Nevertheless, most of the work is done by the mappers, then the reduce simply compares the proposed solutions.

4.5 Stochastic parallel search of the optimum

As reported in Sect. 2, different approaches have been proposed in literature for parallelizing the Simulated Annealing algorithm with MapReduce. The approach we adopt is similar to the one presented in [20]: we use different mappers for executing independent iterations of the TRSA algorithm, starting from different solutions $\tau \in \mathcal{S}_{\text{init}}$, and we then use the reducer to compute the final result.

The initial Pareto-set $\mathcal{S}_{\text{init}}$ computed by Algorithm 2 is stored both in the cache (so that all mappers can access to it) and in a parallel data structure that makes easy to access to each $\tau \in \mathcal{S}_{\text{init}}$ in parallel by the different mappers. The MapReduce pseudo-code is shown in Algorithm 3.

Each mapper performs an execution of the TRSA algorithm, i.e., it explores the solution space starting from a trip τ. The output of each mapper contains the improved Pareto-set S_i, and these sets are combined together by a single reducer to eliminate dominated and redundant solutions. Note that, also for this job, most of the work is done by the mappers, which explores the solution space through perturbations.

4.6 Closing the loop: Profile update

The recommendation system takes as input a set of popular trips and compute the best solutions to the user query $\mathcal{Q}$. To this aim, since the query contains the maximum trip duration $\text{TD}_{\max}$, the system uses the profiles stored in the cache to compute the duration of the potential solutions. If we use the profiles $\text{ATO}(d, h)$ and $\text{AVT}(n^p)$ (defined in Sect. 4.2) computed offline, we obtain a static system that redistribute the users according to average values. This approach may be still important, since not all tourists make use of the recommendation system, therefore the averages may be a good indication of the level of crowding.

Nevertheless, with the diffusion of smartphones, we may expect that an increasing number of tourists will use the recommendation system, therefore we should be able to update the profiles to reflect the actual tourist distribution over the POIs. In particular, we consider the Average Time Occupancy $\text{ATO}(d, h)$ profiles. While building offline these profiles, it is possible to

Algorithm 3: MapReduce job for the execution of the TRSA algorithm. The key *id* read by the mapper is a generic identifier associated to the trip and can be safely ignored.

```
1  class Mapper
2      method Map(id, τ)
3          S_map ⟵ ∅
4          S_init ⟵ retrieve from cache
5          S_map ⟵ TRSA(S_init, τ, TD_max, T_min, T_init)
6          return (Q, S_map)
7  class Reducer
8      method Reduce(Q, ⟨S_1, S_2, ...⟩)
9          S ⟵ ∅
10         foreach S_i ∈ ⟨S_1, S_2, ...⟩ do
11             foreach τ ∈ S_i do
12                 S ⟵ Update(S, τ)
13         return (Q, S)
```

identify the two main components for such profiles: the occupancy due to (i) registered and (ii) anonymous users. While we can not have any impact on the anonymous users, we may be able to influence the registered users, since they are the tourists that make use of the recommendation system. Therefore, in our system we consider the following *Average Time Occupancy* profiles:

$$\mathrm{ATO}'(d,h) = \mathrm{ATO}_{\mathrm{anon}}(d,h) + \mathrm{ETO}_{\mathrm{reg}}(d,h) \tag{9}$$

where $\mathrm{ATO}_{\mathrm{anon}}(d,h)$ is the component due to anonymous users, and $\mathrm{ETO}_{\mathrm{reg}}(d,h)$ is the *Estimated Time Occupancy* computed considering the registered users and the recommendations done so far by the system.

The profiles $\mathrm{ETO}_{\mathrm{reg}}(d,h)$ are reset at the beginning of each day with the average behaviour of the user in the past. As the system issues recommendations, it records the choices of the users, and it updates the estimation of the POI occupancy assuming that the user will follow the recommended trip, spending the estimated time in each POI and for traveling from one POI to the next. Even if the actual user behaviour may vary, the overall estimation of the POI occupancies should not be significantly affected, since they are the results of aggregated values.

4.7 Discussion

The current recommendation system considers a set of objective functions in the search for the optimal solution. Using the Avergage Time Occupancy $\mathrm{ATO}'(d, h)$ defined in Eq. (9), for a given trip τ we can estimate the number of visitors inside a POI at the time the tourist should reach the POI (and this estimate includes the recommendations done so far), and we can compute the time spent inside the POI from the $\mathrm{AVT}(n^p)$ profile. The level of crowding influences the number of visitors inside a POI, and the corresponding computation of the visiting time. Selecting POIs without considering the current level of crowding may result in spending time in crowded POIs, which in turn influences the number of POIs that can be visited. Since one of the objective functions includes the number of POI visited, optimizing using such objective function means finding the solution where the time spent in the visited POI is at its minimum.

We can consider additional objective functions that take into consideration the affinity with the user interests, or the contraints related to the tourist budget, or the relevance of a POI using rankings. All these extensions are straightforward to implement, and we consider here only the functions defined in Eq. (3) in order to show the effectiveness of our approach.

5 Case Study and Experiments

We evaluated the recommendation system described in this paper using real-world traces collected for registered tourists visiting the city of Verona, Italy.

Available dataset. The tourist office of the city of Verona offers a *sightseeing city pass* called "VeronaCard": for a given fee, the tourist may visit up to 22 POIs around the city within a specific time-frame (e.g., 24 hours, or 48 hours). Every time a tourist with the VeronaCard enters in a POI, a record is created: it contains the VeronaCard number (unique identifier), the timestamp of the entrance and the POI identifier. The dataset includes approximately 1,200,000 records that spans 5 years.

From this dataset, we derive a set of data and measurements that we use in our experiments. We start by building the trips followed by the tourists with a VeronaCard, i.e., the sequence of visited POIs, obtaining approximately 250,000 trips. For each trip, given two consecutively visited POIs, with the help of the Google APIs, we compute the travel time and distance of the path connecting such two POIs. Since Verona is a small city and all the POIs are within walking distance, we assume walking as main travel mode. Knowing the travel time, we derive the visit time for each POI, at different times of

Table 1: Statistics about the collected trips. The columns report: the number of visited POIs, the number of trips with such number of POIs, the average duration of the trip (hour:min), the average travel time, and the average travel distance.

$\|\tau\|$	# trips	$\bar{\lambda}_\tau$	avg trav. time	avg trav dist.
2	14,520	04:10	00:10	750m
3	31,455	04:20	00:17	1,5Km
4	40,878	06:00	00:26	2,0Km
5	37,900	07:50	00:34	2,7Km
6	28,261	09:00	00:42	3,4Km
7	16,139	10:30	00:51	4,0Km
8	7,823	11:30	00:60	4,7Km
9	3,060	12:00	01:10	5,5Km

the day. In addition, we compute the number of tourists inside each POI[1].

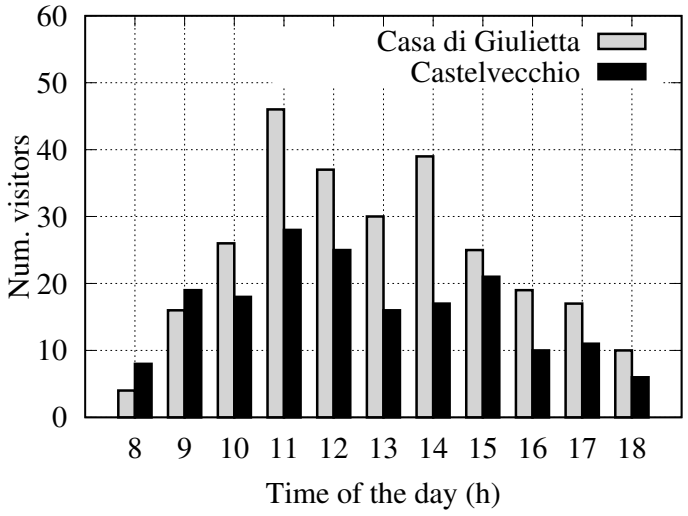

Figure 3: Average Time Occupancy ATO(d, h) for two POIs in Verona. The averages have been computed considering the Sundays in July.

Table 1 show some statistics related to trips. We grouped trips with the same number of visited POIs, $|\tau|$: for each group, we show the number of trips with that number of POIs, the average duration of the trips (considering

[1]The information about the exact number of tourists visiting a POI is available partially for a subset of POIs, therefore, for the purpose of our experiments, we prefer to compute in the same way the number of visitors with the described approach for all the POIs.

the first POI as the starting POI), the average travel time and travel distance – we first sum the travel times and travel distances of the paths for each trip, and then compute the average.

We use the processed dataset to build the POI *profiles* defined in Sect.4.2: for any POI, we compute the Average Time Occupancy ATO(d, h) and the Average Visiting Time AVT(n^p). Figures 3 and 4 shows sample ATO(d, h) and AVT(n^p) profiles for two POIs called "Casa di Giulietta" and "Castelvecchio" respectively. As for the average time occupancy (Fig. 3), we show the curves for July's Sundays (the average number of visitors computed considering the Sundays in July). As expected, there are two peak hours, in the morning and the afternoon. Interestingly, the peak hours for the two POIs in the afternoon are slightly different.

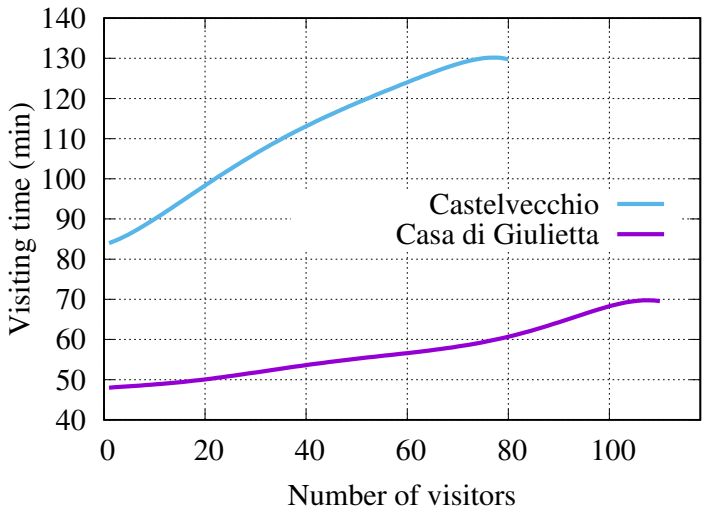

Figure 4: Average Visiting Time AVT(n^p) for two POIs. The averages are computed considering the whole dataset.

As for the average visiting time (ATV curves shown in Fig. 4) we notice an increasing visiting time as the number of visitor increases, which indicates the impact of crowding in the visiting time.

As a final step of our offline processing, we collect the most popular trips, which will be used by the recommendation engine as a starting point when a new query is submitted.

Experimental methodology. In order to test our recommendation system, we need to provide a set of queries. To this aim, we consider out dataset and the trips we built from such a dataset. For a given day, we consider the trips collected that day: for each trip, we create a query where (i) the initial

coordinates at which the trip starts are the coordinates of the first POI, (ii) the time at which the trip begins is the time of the access to the first POI, and (iii) the maximum trip duration is given by the computed trip duration time augmented with an estimate duration of the last visit[2].

For that set of queries, we observe the output from two possible perspectives: the POI viewpoint and the trip viewpoint. To do so, we assume that the users actually behave as expected, i.e., if we estimate a visit time for a POI or a travel time for a path, it will take exactly those estimated times to visit that POI or to travel along that path. As a future work, we plan to introduce some variability in the user behaviour, and study the impact of such variability on the observed output.

From the POI viewpoint, we record for each POI the number of visitors over time, and we build the time occupancy curve for that day. From the trip viewpoint, we record the values of the objective functions defined in Eq. (3).

We compare three different approaches:

- *No recommendation*: we simply consider that the user follows the trip as built from the dataset, i.e., the query result is actually the trip from which the query has been derived, which is the trip performed by the user autonomously;
- *Recommendation based on averages*: we use the static version of the $\mathrm{ATO}(d, h)$ profiles, i.e., the recommendation are based on the average occupancy of the previous observation interval (e.g., last year);
- *Adaptive Recommendation*: we use the $\mathrm{ATO}'(d, h)$ profiles, which are updated after every recommendation.

For the last case, since we do not have the records from the anonymous users, we assume that half of the users recorded that day are anonymous, i.e., $\mathrm{ATO}_{\mathrm{anon}}(d, h) = 0.5 \cdot \mathrm{ATO}(d, h)$. We tested different percentage, not reported here for space constraints, obtaining similar qualitative results. The MapReduce TRSA algorithm has been implemented using SpatialHadoop [29], an extension of Apache Hadoop [30] which provides a native support for spatial data, in terms of spatial data types, operations and indexes. SpatialHadoop has been successfully applied in order to efficiently perform spatial analysis and validation of huge amount of geographical data [31].

POI viewpoint. Figure 5 shows the number of visitors over time for the POI called "Casa di Giulietta" on February 14th, 2015, with or without a recommendation system. It is interesting to note that a static recommendation

[2]For any trip, we are not able to know the visit time of the last POI, since we do not have the next visited POI used to compute such value. The estimated duration of the last visit is simply the average visit time for that POI.

simply changes the peak hour with respect to a system with no recommendation, since it uses the average peak hour of the past days, but it does not adapt to the estimated number of users in the POI. Instead, our dynamic recommendation spread the tourists over time.

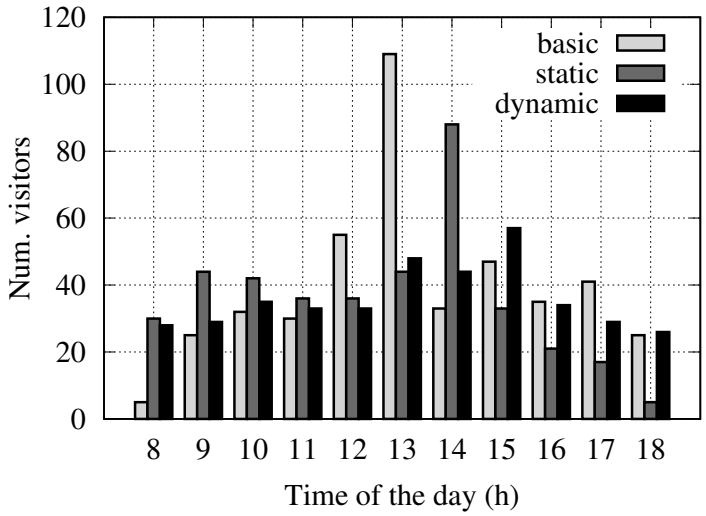

Figure 5: Number of visits at "Casa di Giulietta" both considering the behavior of the tourists without recommendation (basic) and with the two approaches based on average (static) and adaptive (dynamic) profiles for POIs.

Trip viewpoint. Figure 6 illustrates three different alternative trips: the red one is an original trip performed by a user without any recommendation. It starts from the POI named "Torre dei Lamberti" at 11:06, then it stops at "Casa di Giulietta" at 12:33, and finally it reaches "Arena" at 14:15. The blue path is a solution produced by the TRSA algorithm using only *recommendation based on average* crowding information. As you can notice, the trip starts from the same POI and at the same time (query parameters), then it stops at "City Sightseeing" at 12:16, it proceeds towards "Casa di Giulietta" at 13:15, and it finally arrives at "Arena" at 14:15. In this case, the tourist arrived at "Casa di Giulietta" during the peak hour (13:15) for this specific day, since the algorithm considers only average historical static information about the POI occupancies. Finally, the green line represents the trip produced by the TRSA algorithm considering *adaptive recommendation*. In this case, the trip starts from the same POI and at the same hour, but it proceed towards "Museo Conte" at 12:16, then it visits "Centro Internaz. di Fotografia" at 13:04 and it arrives at "Casa di Giulietta" at 14:40 when the peak hour is passed. Notice that, the recommendation system has improved

the values of the objective functions, indeed the blu trip enhanced f_n since it includes an additional POI, while the green trip enhanced all objective functions: it has an additional POI, f_{tt} is decreased of 36% and f_{td} of 49% with respect to the red trip.

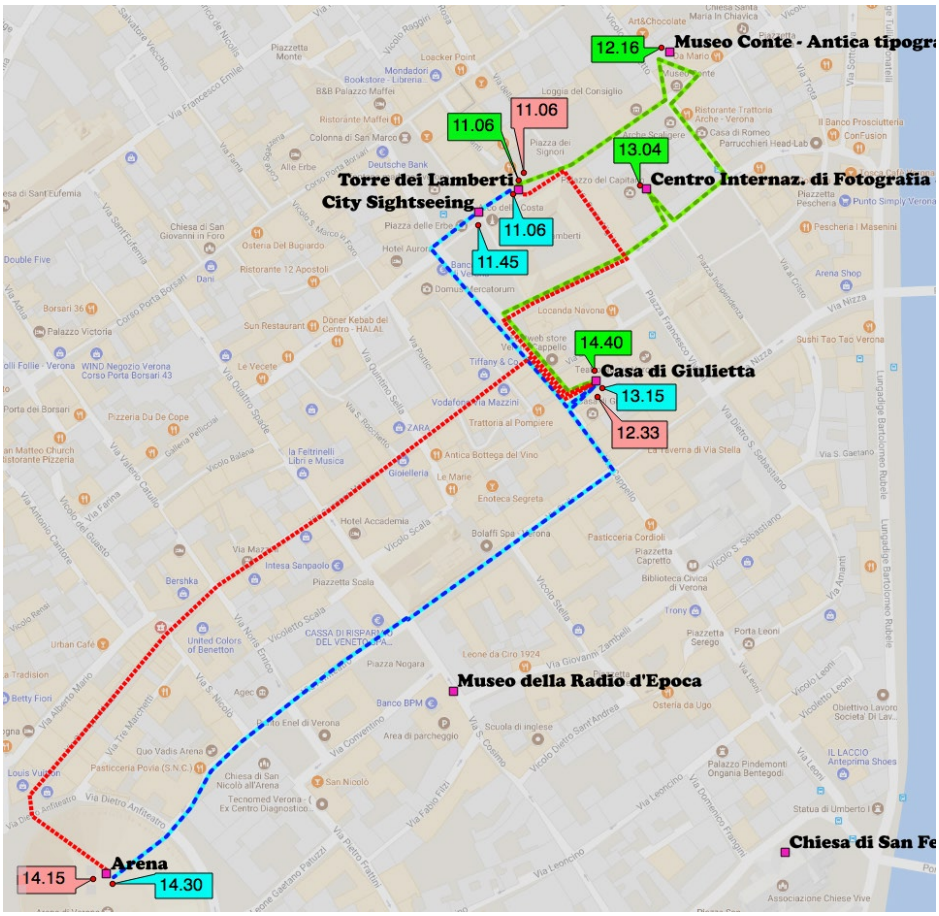

Figure 6: An original path (the red one) together with two paths produced by our TRSA algorithm, the blue one obtained by using only historical statistical data about the level of crowding, while the green one considering also dynamic and adaptive information about the level of crowding.

Table 2 reports some data about the quality of the produced recommendations. More specifically, we have considered the three POIs that have been most frequently chosen as the starting point for a trip: "Arena" (p_s^1), "Casa di Giulietta" (p_s^2) and "Castelvecchio" p_s^3. For each of these POIs, the table reports the number of historical records, the size of the initial estimated Pareto-front built from these records, the percentage of improvement of the various objective functions w.r.t. the original trips. The table shows

Table 2: Statistics about the recommendations. For each starting point we show the number of available trips, the size of the initial Pareto-front, the percentage of improvement of each objective function w.r.t. its original trip, the percentage of cases in which at least one or two objective functions are improved.

p_s	$\#\tau$	$\mid \mathcal{F} \mid$	f_n	f_{tt}	f_{td}	$\lvert f_{improved} \rvert$	
						≥ 1	≥ 2
p_s^1	124,800	11,300	4%	67%	64%	71%	64%
p_s^2	28,000	3,930	1%	68%	66%	69%	66%
p_s^3	21,175	3,359	1%	73%	70%	74%	70%

that each component of the objective function is improved by the TRSA algorithm. Moreover, the improvements in each component of the objective function increases with the number of available historical trips.

6 Conclusion and future work

Personalized trip recommendation systems tailor the suggestions to the users based on their constraints and requirements. Nevertheless, they do not consider the impact of the recommendations on the whole system. In this paper we took a step to fill this gap. In particular, we proposed a system that efficiently searches the solution space through a MapReduce implementation of the multi-objects optimization problem and balances the users among different POIs by including the predicted level of crowding. We evaluate our implementation using a real dataset, showing consistent improvements over the paths usually followed by the tourists.

Our road-map includes the evaluation of the impact that errors may have on the predictions of the level of crowding, and the corresponding quality of the recommendations. Moreover, we plan to extend the system to support approximated spatial query [32].

References

[1] Y. Yu and X. Chen, "A survey of point-of-interest recommendation in location-based social networks," in *Workshops at the 29th AAAI Conference on Artificial Intelligence*, 2015.

[2] E.-C. Lu, C.-Y. Chen, and V. Tseng, "Personalized trip recommendation with multiple constraints by mining user check-in behaviors," in *Proc. of*

the 20th Intern. Conf. on Advances in Geographic Information Systems, 2012, pp. 209–218.

[3] X. Wang, C. Leckie, J. Chan, K. Lim, and T. Vaithianathan, "Improving personalized trip recommendation by avoiding crowds," in *Proc. of the 25th ACM Intern. Conf. on Information and Knowledge Management*, 2016, pp. 25–34.

[4] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. Thalmann, "Time-aware point-of-interest recommendation," in *Proc. of the 36th Intern. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2013, pp. 363–372.

[5] K. Varadhan, R. Govindan, and D. Estrin, "Persistent route oscillations in inter-domain routing," *Computer networks*, vol. 32, no. 1, pp. 1–16, 2000.

[6] S. Zhao, I. King, and M. Lyu, "A survey of point-of-interest recommendation in location-based social networks," *arXiv preprint arXiv:1607.00647*, 2016.

[7] E. Cho, S. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *Proc. of the 17th ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining*, 2011, pp. 1082–1090.

[8] H. Gao, J. Tang, X. Hu, and H. Liu, "Exploring temporal effects for location recommendation on location-based social networks," in *Proc. of the 7th ACM Conf. on Recommender Systems*, 2013, pp. 93–100.

[9] C. Chen, D. Zhang, B. Guo, X. Ma, G. Pan, and Z. Wu, "Tripplanner: Personalized trip planning leveraging heterogeneous crowdsourced digital footprints," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1259–1273, 2015.

[10] H. Gao, J. Tang, X. Hu, and H. Liu, "Content-aware point of interest recommendation on location-based social networks," in *AAAI*, 2015, pp. 1721–1727.

[11] Y. Zheng and X. Xie, "Learning travel recommendations from user-generated gps traces," *ACM Trans. on Intelligent Systems and Technology*, vol. 2, no. 1, p. 2, 2011.

[12] H. Yoon, Y. Zheng, X. Xie, and W. Woo, "Smart itinerary recommendation based on user-generated gps trajectories," in *Intern. Conf. on Ubiquitous Intelligence and Computing*, 2010, pp. 19–34.

[13] Y. Liu, W. Wei, A. Sun, and C. Miao, "Exploiting geographical neighborhood characteristics for location recommendation," in *Proc. of the 23rd ACM Intern. Conf. on Information and Knowledge Management*, 2014, pp. 739–748.

[14] J.-D. Zhang and C.-Y. Chow, "igslr: personalized geo-social location recommendation: a kernel density estimation approach," in *Proc. of ACM SIGSPATIAL Intern. Conf. on Advances in Geographic Information Systems*, 2013, pp. 334–343.

[15] M. Ye, P. Yin, and W.-C. Lee, "Location recommendation for location-based social networks," in *Proc. of the 18th ACM SIGSPATIAL Intern. Conf. on Advances in Geographic Information Systems*, 2010, pp. 458–461.

[16] G. Adomavicius and Y. Kwon, "Multi-criteria recommender systems," in *Recommender Systems Handbook*. Springer, 2015, pp. 847–880.

[17] J.-D. Zhang and C.-Y. Chow, "TICRec: A probabilistic framework to utilize temporal influence correlations for time-aware location recommendations," *IEEE Trans. on Services Computing*, vol. 9, no. 4, pp. 633–646, 2016.

[18] C. Parent, S. Spaccapietra, C. Renso, G. Andrienko, N. Andrienko, V. Bogorny, M. Damiani, A. Gkoulalas-Divanis, and et al., "Semantic trajectories modeling and analysis," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, p. 42, 2013.

[19] H. Hoos and T. Stützle, *Stochastic local search: Foundations and applications*. Elsevier, 2004.

[20] E. Onbaşoğlu and L. Özdamar, "Parallel simulated annealing algorithms in global optimization," *Journal of Global Optimization*, vol. 19, no. 1, pp. 27–50, 2001.

[21] Z. Czech and P. Czarnas, "Parallel simulated annealing for the vehicle routing problem with time windows," in *Proc. of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 2002, pp. 376–383.

[22] A. Radenski, "Distributed simulated annealing with mapreduce," in *European Conf. on the Applications of Evolutionary Computation*. Springer, 2012, pp. 466–476.

[23] H. Li and C. Liu, "Prediction of protein structures using a map-reduce hadoop framework based simulated annealing algorithm," in *2013 IEEE Intern. Conf. on Bioinformatics and Biomedicine*, 2013, pp. 6–10.

[24] B. Suman and P. Kumar, "A survey of simulated annealing as a tool for single and multiobjective optimization," *Journal of the operational research society*, vol. 57, no. 10, pp. 1143–1160, 2006.

[25] B. Suman, "Study of simulated annealing based algorithms for multiobjective optimization of a constrained problem," *Computers & Chemical Engineering*, vol. 28, no. 9, pp. 1849–1871, 2004.

[26] P. McMullen and G. Frazier, "Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations," *Intern. Journal of Production Research*, vol. 36, no. 10, pp. 2717–2741, 1998.

[27] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York, NY, USA: John Wiley & Sons, Inc., 1989.

[28] Y. Nourani and B. Andresen, "A comparison of simulated annealing cooling strategies," *Journal of Physics A: Mathematical and General*, vol. 31, no. 41, p. 8373, 1998.

[29] A. Eldawy, "SpatialHadoop: Towards Flexible and Scalable Spatial Processing Using Mapreduce," in *Proc. of the 2014 SIGMOD PhD Symposium*, 2014, pp. 46–50.

[30] T. White, *Hadoop: The Definitive Guide*, 4th ed. O'Reilly Media, Inc., 2015.

[31] S. Migliorini, A. Belussi, M. Negri, and G. Pelagatti, "Towards massive spatial data validation with spatialhadoop," in *Proc. of the 5th ACM SIGSPATIAL Intern. Workshop on Analytics for Big Geospatial Data*, 2016, pp. 18–27.

[32] A. Belussi, B. Catania, and S. Migliorini, *Approximate Queries for Spatial Data*. Springer Berlin Heidelberg, 2013, pp. 83–127.

University of Verona
Department of Computer Science
Strada Le Grazie, 15
I-37134 Verona
Italy

http://www.di.univr.it