Personalized Itinerary Recommendation with Queuing Time Awareness

Kwan Hui Lim
The University of Melbourne, Australia
Data61, CSIRO, Australia
limk2@student.unimelb.edu.au

Jeffrey Chan
RMIT University, Australia
The University of Melbourne, Australia
jeffrey.chan@rmit.edu.au

Shanika Karunasekera
Christopher Leckie
The University of Melbourne, Australia
{karus,caleckie}@unimelb.edu.au

ABSTRACT

Personalized itinerary recommendation is a complex and time-consuming problem, due to the need to recommend popular attractions that are aligned to the interest preferences of a tourist, and to plan these attraction visits as an itinerary that has to be completed within a specific time limit. Furthermore, many existing itinerary recommendation systems do not automatically determine and consider queuing times at attractions in the recommended itinerary, which varies based on the time of visit to the attraction, e.g., longer queuing times at peak hours. To solve these challenges, we propose the PersQ algorithm for recommending personalized itineraries that take into consideration attraction popularity, user interests and queuing times. We also implement a framework that utilizes geo-tagged photos to derive attraction popularity, user interests and queuing times, which PersQ uses to recommend personalized and queue-aware itineraries. We demonstrate the effectiveness of PersQ in the context of five major theme parks, based on a Flickr dataset spanning nine years. Experimental results show that PersQ outperforms various state-of-the-art baselines, in terms of various queuing-time related metrics, itinerary popularity, user interest alignment, recall, precision and F1-score.

CCS CONCEPTS

• **Information systems** → **Personalization**; **Recommender systems**; *Location based services*; *Data mining*; Web applications;

KEYWORDS

Tour Recommendations; Trip Planning; Personalization; User Interests

ACM Reference format:
Kwan Hui Lim, Jeffrey Chan, Shanika Karunasekera, and Christopher Leckie. 2017. Personalized Itinerary Recommendation with Queuing Time Awareness. In *Proceedings of SIGIR '17, Shinjuku, Tokyo, Japan, August 07-11, 2017,* 10 pages.
https://doi.org/http://dx.doi.org/10.1145/3077136.3080778

SIGIR '17, August 07-11, 2017, Shinjuku, Tokyo, Japan

ACM ISBN 978-1-4503-5022-8/17/08…$15.00
https://doi.org/http://dx.doi.org/10.1145/3077136.3080778

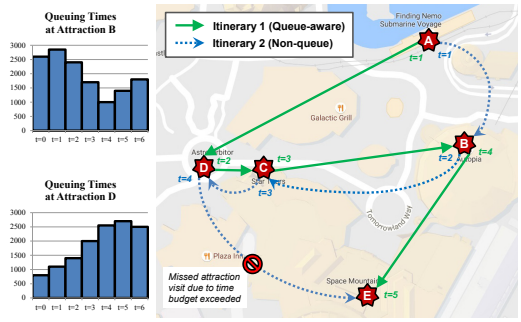

Figure 1: Example of itinerary recommendation with queue time consideration

1 INTRODUCTION

Touring and travelling are popular leisure activities for many people, as shown by the 1.18 billion annual tourists in 2015 [39]. A critical task for any tourist is to plan a trip itinerary that comprises popular and interesting attractions, which can be completed within a specific time limit. This task is especially complex and challenging due to: (i) the need to identify a set of popular attractions that are also aligned with the traveller's interests; (ii) the need to organize these attractions in the form of an itinerary with the constraints of a starting/ending place (e.g., near a traveller's hotel) and limited time for touring; and (iii) the need to plan for travelling, visiting and queuing times at the attractions, where queuing times are dependent on the time of attraction visit. In particular, neglecting to account for queuing times can create a frustrating experience for travellers as they spend an unnecessarily long time queuing instead of enjoying the attractions, and possibly miss attraction visits in their itineraries due to these queuing times exceeding their available touring time.

Figure 1 illustrates the importance of queuing time awareness in itinerary recommendation. Itinerary 2 (dotted blue line) does not consider queuing time and recommends an attraction visit sequence of $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$. However, this itinerary is unable to complete the visit to Attraction E, due to excessive queuing times at Attractions B and D that causes the time budget to be exceeded. Moreover, Itinerary 2 schedules visits to Attractions B and D at the peak (longest) of their queuing times. In contrast, Itinerary 1 (solid green line) considers queuing time and recommends that attractions be visited in the sequence of $A \rightarrow D \rightarrow C \rightarrow B \rightarrow E$, as Attraction D has a lower queuing time at $t = 2$, compared to Attraction B at the same time. By the time Itinerary 1 visits Attraction B (at $t = 4$), its queuing time has shortened drastically, compared to earlier at $t = 2$. In addition, Itinerary 1 schedules visits

to Attractions B and D when queues are the shortest, thus making the itinerary more enjoyable with less time wasted on queuing. In real-life, itinerary recommendation is even more complex due to: (i) the need to consider attraction popularity and user interest preferences; and (ii) since queuing times are time-dependent, itinerary recommenders need to also consider the time of visit to attractions and their queuing times.

In this work, we aim to address these issues and propose the PersQ algorithm for recommending itineraries that are popular, personalized to user interests, and minimize queuing times at attractions. To the best of our knowledge, our work is the first to use geo-tagged photos to automatically determine and incorporate queuing time in personalized itinerary recommendations. Our main contributions are:

- We introduce and formulate the QueueTourRec problem for recommending personalized itineraries that aim to maximize attraction popularity and user interest preferences and minimize queuing times, while adhering to a time constraint for completing the itinerary (Section 3).
- We propose the PersQ algorithm, where we have developed a novel implementation of Monte Carlo Tree Search (MCTS) for the domain of personalized itinerary recommendation with queuing time awareness (Section 5).
- We implement a framework for automatically extracting attraction popularity, user interests and queuing time from geo-tagged photos (Section 6). Using this framework, we have collected a dataset of user visits in five major theme parks across nine years, which is made publicly available at https://sites.google.com/site/limkwanhui/datacode#sigir17 (Section 7.1).
- We introduce various queuing-related evaluation metrics and compare PersQ against various state-of-the-art baselines. The results show that PersQ recommends itineraries with the lowest ratio of queuing time to total touring time, and schedules visits to popular attractions at times with the shortest queues. In addition, PersQ offered the best overall performance in terms of itinerary popularity, user interest, precision, recall and F1-scores (Sections 7 and 8).

For the rest of our paper, Section 2 discusses related work in itinerary recommendation, while Section 4 provides some background on MCTS.

2 RELATED WORK

General Itinerary Recommendation. Many early works on itinerary recommendation [10, 11, 19, 26] are based on the Orienteering problem [17, 38, 41], where the main objective is to recommend an itinerary that maximizes a global profit/reward and can be completed within a specific budget. Attraction popularity is frequently used as a global profit/reward in the context of recommending and planning an itinerary in touristic cities [10, 11, 19, 26] or theme parks [18, 24, 42]. While these are interesting works in itinerary recommendations, the use of a global profit/reward is common among all users and thus provides no personalization for user interest preferences. In recent years, more researchers have incorporated user interests and/or specific preferences to personalize such itinerary recommendations and we discuss these works next.

Table 1: Description of Key Notations

Notation	Description
A	The set of all attractions
a_x	A specific attraction, where $a_x \in A$
Cat_{a_x}	The category of attraction a_x
Dur_{a_x}	The visit duration at attraction a_x
$Trav_{a_x,a_y}$	Travelling time from a_x to a_y
$t^a_{a_x}$	The arrival time at attraction a_x
$t^d_{a_x}$	The departure time from attraction a_x
V_x	An user-attraction visit, $V_x = (a_x, t^a_{a_x}, t^d_{a_x})$
$Pop(a_x)$	The popularity of attraction a_x
$Int_u(c)$	The interest of user u in attraction a_x
$Queue^t(a_x)$	The queuing time at attraction a_x at time t

Personalized Itinerary Recommendation. Many recent works aim to recommend itineraries that are personalized to users based on their interest preferences [40] or personalized requirements such as a specific attraction visit sequence [15], mandatory attraction categories [8, 28], group interest satisfaction [14, 30], among others. For example, [49, 50] implemented a heuristic approximation algorithm to construct personalized itineraries that optimize for a heuristic of attraction popularity and user interests relative to the travelled distance, while [43] utilized a modified Ant Colony System [13] to recommend personalized itineraries that comprise less crowded attractions. Others such as [29, 31] modelled the personalized itinerary recommendation problem as an integer program with the objectives of recommending popular attractions and tailoring the attraction visit duration based on user interest. Similarly, [3, 5] aim to first identify a set of interesting and popular attractions, then construct an itinerary comprising these attractions using a variant of the Travelling Salesman Problem [32]. In addition, there have been many websites and applications [4, 8, 34, 44] developed for itinerary recommendations based on variants of the above-mentioned research.

Top-k POI Recommendation. An adjacent research area to itinerary recommendation are works on top-k POI recommendation, which has been extensively studied by the Information Retrieval community [25, 27, 46–48]. Most of these works are based on matrix factorization or collaborative filtering approaches and their main objective is to recommend a ranked list of top-k POIs to a user. To achieve this purpose, they utilize various sources of information such as social links [46], acitivity/user types [25], geographical/temporal aspects of POIs [27, 47, 48]. Although these works propose interesting uses of various POI information, the task of top-k POI recommendation is distinctly different from itinerary recommendation, which encompasses the added complexity of constructing a set of relevant POIs as a time-constrained connected itinerary with the consideration of travelling time and POI visit durations, among others.

Discussion of Related Work. While these earlier works are the state-of-the-art in itinerary and POI recommendation, our work differs from these earlier works in several important aspects, namely: (i) our QueueTourRec problem optimizes for attraction popularity, user interest preferences and queuing times distribution, which is automatically constructed based on geo-tagged photos and then included as part of the travelling cost along with transport time and attraction visit duration; (ii) unlike top-k POI recommendations,

our QueueTourRec problem involves recommending a set of POIs and planning it as a connected itinerary with the associated time constraints of travelling between POIs, visiting POIs and completing the itinerary within a time budget; and (iii) while MCTS is typically used for board game playing [6, 12][1], we propose a novel and non-trivial adaptation of MCTS (denoted the PersQ algorithm) for solving the QueueTourRec problem.

3 BACKGROUND AND PROBLEM DEFINITION

Table 1 highlights the key notations used in our work, which we elaborate more in the next section.

3.1 Preliminaries

For each theme park, there are are m attractions, and we represent this set of attractions as $A = \{a_1, ..., a_m\}$. Each attraction a_x is also associated with its latitude/longitude coordinates, category Cat_{a_x} (e.g., roller coaster, water rides, indoor), and the duration Dur_{a_x} needed to visit that attraction (e.g., 10 minutes for a roller coaster). For itinerary recommendation in theme parks, the primary mode of transportation is by walking. Thus, we use $Trav_{a_x,a_y}$ to denote the amount of time taken to walk from attractions a_x to a_y, which is derived from a leisure walking speed of 5km/hour [7] to cover the distance between attractions a_x and a_y.

While we define our problem in the context of a theme park, these definitions and problem formulation are generalizable to any general path planning related context. For example, in the context of a city, there are also various attractions that a tourist can visit, where each attraction is also associated with a specific category (e.g., shopping, beach, park) and visit duration. The travelling speed $Trav_{a_x,a_y}$ can also modified to reflect the transport modes of walking, cycling or driving. Similarly, in the context of a museum, the attractions would be in the form of an artifact or painting, each also associated with a specific category (e.g., classical, modern, gothic) and visit duration.

Definition 1: Attraction Visits. For a user u who has visited n attractions, we represent each visited attraction a_x as a tuple, $V_x = (a_x, t^a_{a_x}, t^d_{a_x})$. In this tuple, a_x denotes the visited attraction, while $t^a_{a_x}$ and $t^d_{a_x}$ denote the time that user u arrived at and departed from attraction a_x, respectively. Given $t^a_{a_x}$ and $t^d_{a_x}$, we can then derive the amount of time that user u spent at attraction a_x, by calculating the difference between the arrival and departure times, i.e., $t^a_{a_x}$ and $t^d_{a_x}$. Note that this time spent includes both the queuing time at attraction a_x and its visiting duration Dur_{a_x}. We later describe (in Definition 5) how these queuing times can be calculated.

Definition 2: Visit Sequence. Given the set of visited attractions (i.e., multiple tuples of $V_x = (a_x, t^a_{a_x}, t^d_{a_x})$) by a user u, we can construct his/her visit sequence by joining consecutive attraction visits to form a chronologically-ordered sequence, $S_u = (V_1, V_2, ..., V_n)$. A visit sequence is split into two separate visit sequences if two consecutive attraction visits take place more than 8 hours apart, i.e., $t^d_{a_x} - t^a_{a_{x+1}} > 8\ hours$. Other researchers have adopted a similar approach to divide such visit sequences based on an interval of 8 hours [10, 29, 31]. This list of visit sequences subsequently serves as the ground truth of real-life theme park visits, which is used as part of our experimentation and evaluation of the various algorithms and baselines.

Definition 3: Popularity of Attractions. Many earlier itinerary recommendation works [5, 10, 30] represent the popularity of a Point of Interest (POI) based on the number of visits that this POI has received. In our work, we employ a similar measure of popularity for each attraction based on the number of times this attraction has been visited, which is defined as:

$$Pop(a) = \sum_{u \in U} \sum_{a_x \in S_u} \delta(a_x = a),\ \forall\ a \in A \quad (1)$$

where $\delta(a_x{=}a) = 1$ if attraction a_x is the same as attraction a, and $\delta(a_x{=}a) = 0$ otherwise.

Definition 4: User-Attraction Interest Relevance. While attraction popularity is the same for all users, the interest relevance of an attraction differs from user to user, i.e., each user will have their own unique interest preferences. Given that C represents the set of all attraction categories and P the set of photos taken by user u, we define the interest level of user u in category c as:

$$Int_u(c) = \frac{1}{|P|} \sum_{p \in P} \delta(p = c),\ \forall\ c \in C \quad (2)$$

where $\delta(p = c) = 1$ if photo p is of an attraction that belongs to category c, and $\delta(p = c) = 0$ otherwise. In short, the interest level of a user u in attraction category c is based on the number of photos of attractions that belong to category c, relative to the total number of photos taken. The intuition behind this definition is that a user is more likely to take more photos of an attraction (category) that interests him/her, and less photos otherwise.

Definition 5: Queuing Times of Attractions. In general, each attraction is also associated with a certain queuing time before a user can visit the attraction, e.g., queuing to ride a roller coaster or to buy tickets for a show. Given that V_a is the set of visits to attraction a and t is the visiting time, we represent the queuing times at an attraction a as:

$$Queue^t(a) = \frac{1}{|V_a|} \sum_{u \in U} \sum_{a_x \in S_u} \delta(a_x = a)\big((t^d_{a_x} - t^a_{a_x}) - Dur_{a_x}\big) \quad (3)$$

where $\delta(a_x{=}a) = 1$ if attraction a_x is the same as attraction a, and $\delta(a_x{=}a) = 0$ otherwise. In short, we calculate the queuing time at an attraction a based on the total time spent at an attraction (which includes queuing time and visit duration), subtracted by its attraction visit duration (e.g., 10 minutes for a roller coaster). We derive this total time spent based on the departure time $t^d_{a_x}$ and arrival time $t^a_{a_x}$, which we then subtract by the attraction visiting time Dur_{a_x}. For simplicity, we consider the time t of $Queue^t(a)$ in terms of the hour of visit and use the average visit duration of attraction a at a time t to derive its queuing time at time t.[2]

[1] More background on MCTS is provided later in Section 4.

[2] Although we consider t in terms of the hour of visit, t can be easily generalized to smaller or larger units of time (e.g., 30 min or 2-hourly blocks)

3.2 Problem Definition

We first define the problem of recommending a personalized itinerary to a single user, with the main objectives of maximizing the popularity and interest relevance of recommended attractions in this itinerary, while minimizing the queuing times at these attractions. We denote this problem as the QueueTourRec problem. Our formulation of QueueTourRec is modelled based on a variant of the Orienteering problem [38, 41].

Given the set of attractions A, a time budget B, a starting point a_1 and destination point a_N, our objective is to recommend an itinerary $I = (a_1, ..., a_N)$ that maximizes the total reward $Rwrd(I)$ accumulated from the recommended itinerary I, while ensuring that the itinerary is completed within the time budget B. The reward function $Rwrd(I)$ is calculated based on the popularity, interest relevance and queuing times of the attractions $a_x \in A$ recommended in itinerary I, as denoted by $Pop(a_x)$, $Int(Cat_{a_x})$ and $Queue(a_x)$, respectively. The time budget B is calculated using the function $Cost(a_x, a_y) = Trav(a_x, a_y) + Dur(a_y) + Queue^t(a_y)$, i.e., considering for travelling time, attraction/ride duration and queuing time at $time = t$.

Unlike earlier works that do not automatically determine and consider queuing times, our work differs in the following ways: (i) we derive a queuing time distribution for each attraction (i.e., expected queuing duration at specific times), based on past tourist visits; (ii) we incorporate these queuing times into the itinerary based on the proposed visit time at each attraction; (iii) we recommend an itinerary that optimizes for attraction popularity, user interests and queuing times; and (iv) in contrast to the Orienteering problem with fixed costs, QueueTourRec includes a time-dependent cost that affects the chosen attractions and queuing times, thus we cannot utilize traditional algorithms designed for the former.

In essence, our main goal is to plan an itinerary $I = (a_1, ..., a_N)$ that maximizes the following objective function:

$$\max \sum_{a_i \in I} \sum_{a_j \in I, a_i \neq a_j} Path^t_{a_i,a_j} \Big(\frac{Int(Cat_{a_i}) + Pop(a_i)}{Queue^t(a_i)} \Big) \quad (4)$$

where $Path^t_{a_i,a_j} = 1$ if a path from attraction a_i to a_j is taken at time t (i.e., we visit attractions a_i and a_j in sequence), and $Path^t_{a_i,a_j} = 0$ otherwise.

Following which, we then attempt to solve for Eqn. 4, subject to the following constraints:

$$\sum_{a_i \in I, i \neq 1} Path^t_{a_1,a_i} = \sum_{a_j \in I, j \neq N} Path^{t+d}_{a_j,a_N} = 1 \quad (5)$$

$$\sum_{a_i \in I, k \neq N} Path^t_{a_i,a_k} = \sum_{a_j \in I, k \neq 1} Path^{t+d}_{a_k,a_j} \leq 1 \quad (6)$$

$$\sum_{a_i \in I} \sum_{a_j \in I, a_i \neq a_j} Path^t_{a_i,a_j} Cost^t(a_i, a_j) \leq B \quad (7)$$

where the cost function used in Equation 7 is defined as:

$$Cost(a_i, a_j) = Trav_{a_i,a_j} + Dur_{a_j} + Queue^t(a_j) \quad (8)$$

Equation 4 is a multi-objective function that aims to recommend attractions in an itinerary I that maximizes the popularity and interest relevance of all visited attractions, while minimizing the queuing times at these attractions. Equation 4 can also be enhanced with different weights for $Int(Cat_{a_i})$, $Pop(a_i)$ and $Queue^t(a_i)$ to provide varying levels of emphasis on each component. The optimization of Equation 4 is also subjected to Constraints 5 to 7. Constraint 5 ensures that the recommended itinerary begins from a specific attraction a_1, while terminating at attraction a_N.[3] Thereafter, Constraint 6 ensures that all paths in the recommended itinerary are connected and no attractions are visited multiple times. Constraint 7 ensures that the recommended itinerary can be completed within the time budget B. Specifically, the function $Cost(a_x, a_y)$ (i.e., Equation 8) is applied to all visited attractions to obtain the total time taken for the itinerary, which includes the consideration for travelling time $Trav_{a_i,a_j}$, attraction/ride duration Dur_{a_j} and queuing time $Queue^t(a_j)$.

The QueueTourRec problem formulation is based on a variant of the Orienteering problem, which has been shown to be NP-hard as the Orienteering problem is a specialized instance of the Travelling Salesman Problem [38, 41].[4] Furthermore, this problem formulation incorporates a time-dependent cost function $Cost^t(a_x, a_y)$, which adds to the complexity of this problem. Due to this NP-hard complexity, solving the QueueTourRec problem optimally is not feasible. To overcome these problems, we propose the PersQ algorithm for solving the QueueTourRec problem of recommending personalized and queue-aware itineraries. In the following sections, we provide some background on Monte Carlo Tree Search (MCTS) before describing our PersQ algorithm, which is partially based on the MCTS algorithm.

4 MONTE CARLO TREE SEARCH

MCTS is a popular search algorithm that has been successfully applied to many board games such as Chess, Go, Othello, among others [6, 12]. We first provide some background information on the MCTS algorithm in its typical application to board games. MCTS approaches the task of board game playing as a tree search problem, where nodes of the tree represent a specific board position and leaf nodes represent a terminal game state, i.e., a win or loss. Thus, moving from a node to a child node corresponds to making a game move that results in a new board position. The main objective is to perform a tree search that results in a set of optimal moves (nodes) leading to a win state. In the MCTS algorithm, the basic idea is that game play initially commences with iterations of random node selection to explore moves, and recording the outcome of choosing those moves. Thereafter, at subsequent game plays, MCTS departs from random moves and progressively builds upon previous successes by converging to moves that result in win states.

The MCTS algorithm typically runs for a fixed number of iterations (e.g., run for 100 iterations) or repeats the iterations for a specific running time (e.g., run as many iterations in 10 seconds). At each iteration, MCTS operates with four main steps:

[3] For a more general itinerary recommendation task, such as in a city, a_1 and a_N can be defined as the hotel that the tourist is residing in, and thus any recommended itinerary will start and end at the tourist's hotel.

[4] The QueueTourRec problem is also NP-hard as it can be generalized to the Orienteering problem, by using attraction popularity $Pop(a_i)$ as a global reward in Equation 4 and setting the queuing times $Queue^t(a_i)$ to be uniform. For a more detailed proof on the NP-hardness of the Orienteering problem, please refer to [2, 16].

(1) ***Selection***. MCTS starts at the root node r and recursively *selects* a child node c to expand based on a certain strategy, until reaching either a leaf/terminal node or an unvisited/unexpanded node. Thus, the root node r corresponds to the start state of the board, while the child node c corresponds to a move that results in a new board position.
(2) ***Expansion***. If the selected node c is not a leaf node (i.e., a terminal game condition leading to a win or loss), expand node c and randomly select one of its unvisited child nodes.
(3) ***Simulation***. Steps 1 and 2 are then continuously *simulated* (repeated) until reaching the end of the game, i.e., a leaf/terminal node.
(4) ***Back-propagation***. At this stage, the game would have ended, resulting in a win or loss to the player. The results of this game is then *back-propagated* to all the traversed nodes (board positions) during this iteration.

Steps 1 to 4 are considered one iteration of MCTS, which are repeated for a fixed number of iterations or for a specific running time. Each node (board position) will be labelled with the number of wins/losses (1/0) and the number of times it has been visited. Step 1 typically employs a selection strategy that exploits nodes with a high win-to-visited ratio, while Step 2 uses a random exploration strategy for previously unvisited nodes. Steps 3 and 4 then simulate the game play to the end and back-propagate the results to all the visited nodes, incrementing their win count (if game was won) and visited count by 1.

4.1 Motivations of MCTS for Itinerary Recommendation

Monte Carlo based methods have also been applied to various routing problems, such as variants of the Travelling Salesman Problem [33] and Vehicle Routing Problem [20], with the main purpose of identifying an appropriate next node to visit in the route. Although it is possible to solve such routing-related problems optimally as an Integer Linear Program, the problem complexity increases exponentially with the number of nodes (attractions). In the case of the QueueTourRec problem, there is the added complexity of a time-variable in the form of queuing times at attractions. More importantly, real-life applications typically require that solutions be generated in a short time-frame (minutes rather than hours). MCTS is well-poised for solving our QueueTourRec problem due to two main reasons [6]: (i) Instead of traversing the entire search tree, MCTS allows us to explore a smaller, more promising region of the solution space, thus leading to a shorter running time; and (ii) MCTS can be adapted to run only for a fixed amount of time, thus making it very suitable for real-life application;

However, the direct application of MCTS to the QueueTourRec problem is non-trivial for several reasons: (i) in board games, each move from a node to a child node has a uniform cost of 1, whereas this cost is variable for itinerary recommendation and can be based on distance or travel times; (ii) furthermore, this cost is time-dependent in the QueueTourRec problem, due to different queuing times at the same attraction based on the visit time; and (iii) each win/lose state in a board game corresponds to a binary reward of 1 or 0, which leads to a simple back-propagation strategy (Step 4). In contrast, itinerary recommendation results in a complex reward structure that is based on the attractions recommended and their corresponding popularity, interest relevance and queuing times. In the following sections, we describe our proposed PersQ algorithm, which is an adaptation of MCTS for the QueueTourRec problem, and we evaluate the effectiveness of PersQ against various state-of-the-art methods for personalized tour recommendations.

Algorithm 1: PersQ - Overview of Algorithm

input : $a_1 \in A$: Starting attraction, $a_N \in A$: Ending attraction, S: Starting time of itinerary, B: Total time budget, $maxLoop$: Number of iterations.
output: $I = (a_1, ..., a_N)$: Recommended itinerary.

```
1  begin
2      Initialize T_visits as a tree of visit count;
3      Initialize T_reward as a tree of reward collected;
4      Initialize I_list as a list of itineraries;
5      for iterations ← 1 to maxLoop do
6          Initialize I_temp as a list of attraction visits;
7          Append attraction a_1 to I_temp;
8          a_i ← a_1;
9          a_j ← ∅;
10         totalCost ← 0;
11         while totalCost < B do
12             a_j ← SelectNextNode(a_i, T_visits, T_reward);
13             Append attraction a_j to itinerary I_temp;
14             totalCost ←
                 totalCost + Trav_{a_i,a_j} + Dur_{a_j} + Queue^t(a_j);
15             if a_j == a_N then
16                 Break loop;
17             a_i ← a_j;
18         BackpropC(I_temp, T_visits);
19         if a_j == a_N then
20             R ← Simulate(I_temp);
21             BackpropR(I_temp, T_reward, R);
22             Append itinerary I_temp to itinerary list I_list;
23     Return best itinerary I from I_list;
```

5 OVERVIEW OF PERSQ ALGORITHM

Algorithm 1 gives an overview of our proposed PersQ algorithm, which takes as input a desired starting attraction a_1, ending attraction a_N, starting time S and time budget B for completing the itinerary. The output is in the form of a recommended itinerary $I = (a_1, ..., a_N)$, which starts from attraction a_1 at time S and finishes at attraction a_N by time $S + B$.

At the start of Algorithm 1 (Lines 2 and 3), two similar trees are initialized with the root node n_1 as the starting attraction a_1, with its child nodes as the set of attractions $a_i \in A$, up to a depth of $|A|$ (the number of attractions in the theme park). Thus, the traversal of nodes in this tree is equivalent to an itinerary $I = (a_1, ..., a_N)$, with a_1 as the node selected at $depth = 1$, a_2 at $depth = 2$ and so on, until a_N at $depth = N$. The two above-mentioned trees differ in terms of the values associated with each node, T_{visits} contains

Algorithm 2: PersQ - SelectNextNode()

input : $a_i \in A$: Current attraction; $I = (a_1, ...)$: Current itinerary; T_{visits}; Tree of visit counts; T_{reward}: Tree of accumulated reward.
output : $a_n \in A$: Next attraction.

1 **begin**
2 $visitCount_{a_i} \leftarrow GetVisitCount(a_i, T_{visits})$;
3 $a_n \leftarrow \emptyset$;
4 $UCT_{max} \leftarrow 0$;
5 **for** $a_j \in A$ *and* $a_j \notin I$ **do**
6 $visitCount_{a_j} \leftarrow GetVisitCount(a_j, T_{visits})$;
7 $totalReward_{a_j} \leftarrow GetTotalReward(a_j, T_{reward})$;
8 $exploit_{a_j} \leftarrow \frac{totalReward_{a_j}}{visitCount_{a_j}} + \frac{Int(Cat_{a_j})+Pop(a_j)}{Trav_{a_i,a_j}+Dur_{a_j}+Queue^t(a_j)}$;
9 $explore_{a_j} \leftarrow 2C_p\sqrt{\frac{2\ln visitCount_{a_i}}{visitCount_{a_j}}}$;
10 $UCT_{a_j} \leftarrow exploit_{a_j} + explore_{a_j}$;
11 **if** $UCT_{a_j} > UCT_{max}$ **then**
12 $a_n \leftarrow a_j$;
13 Return a_n;

the number of times a specific node has been selected (Lines 2), while T_{reward} contains the total reward accumulated by a specific node (Lines 3). Line 4 initializes I_{list}, which will contain a list of explored itineraries $I_{list} = (I_1, I_2, ..., I_{maxLoop})$ at the conclusion of our algorithm.

Algorithm 1 then runs for a fixed number of iterations *maxLoop* (Line 5) and we set $maxLoop = 1,000$ as this value allows the algorithm to complete in reasonable time (< 3 minutes). Each iteration (Lines 5 to 22) is equivalent to a single run of MCTS. At the start of each iteration, Lines 6 and 7 initialize an itinerary $I_{temp} = (a_1)$ as an ordered list with a_1 as the first attraction to be visited. Following which, Lines 11 to 17 construct an itinerary that can be completed within the time budget B based on travelling time, ride duration and queuing times (Line 14). This itinerary is constructed by iteratively calling the *SelectNextNode*() method (Line 12) and appending the next recommended attraction to I_{temp} (Line 13). The *SelectNextNode*() method reflects the Selection and Expansion stages of traditional MCTS and is discussed further in Section 5.1. Attractions are continuously appended to I_{temp} until either the time budget B is exceeded (Line 11) or the destination attraction a_N is reached (Line 15). After which, *BackpropC*() (Line 18) updates/increments the visit count of visited nodes in T_{visits} based on the recommended attractions in itinerary I_{temp}. If itinerary I_{temp} ends at attraction a_N (Line 19), we calculate the reward gained from I_{temp} (Line 20), update the accumulated rewards of visited nodes in T_{reward} accordingly (Line 21), and append I_{temp} to I_{list}. The calculation of reward and back-propagation of visit counts/rewards are described in more detail in Section 5.2. At the end of all iterations (Line 23), we examine all explored itineraries $I \in I_{list}$ and return the itinerary with the highest reward.

5.1 Selection and Expansion

Algorithm 2 describes the *SelectNextNode*() method, which is used for selecting the next attraction (node) to visit based on a current attraction a_i, visit count tree T_{visits}, and reward tree T_{reward}. Instead of examining all possible neighbours of attraction a_i, Line 5 focuses the search on attractions that have not been visited. Lines 6 to 10 are a variant of the Upper Confidence Bound [1] applied to Trees, commonly denoted as UCT [21]. The main aim of UCT is to choose a next attraction (node) a_j to visit, that maximizes:

$$UCT_{a_j}^{Orig} = \frac{totalReward_{a_j}}{visitCount_{a_j}} + 2C_p\sqrt{\frac{2\ln visitCount_{a_i}}{visitCount_{a_j}}} \quad (9)$$

In Equation 9, the first term $\frac{totalReward_{a_j}}{visitCount_{a_j}}$ controls for the *exploitation* of attractions (nodes) that results in itineraries with high rewards, relative to the number of times these nodes were chosen. The second term $2C_p\sqrt{\frac{2\ln visitCount_{a_i}}{visitCount_{a_j}}}$ controls for the *exploration* of nodes (attractions) that have not been previously selected, thus ensuring that different attractions (and hence, itineraries) are considered. The parameter C_p determines the emphasis to give to the *exploration* of nodes and we set $C_p = \frac{1}{\sqrt{2}}$, which Kocsis and Szepesvári [22] has proved to be the best value as it statisfies Hoeffding's inequality.

Our proposed PersQ algorithm improves upon the original UCT (Equation 9) by including an additional heuristic for next node (attraction) selection. Our version of UCT is defined as:

$$UCT_{a_j}^{PersQ} = \frac{Int(Cat_{a_j}) + Pop(a_j)}{Trav_{a_i,a_j} + Dur_{a_j} + Queue^t(a_j)} + \frac{totalReward_{a_j}}{visitCount_{a_j}} + 2C_p\sqrt{\frac{2\ln visitCount_{a_i}}{visitCount_{a_j}}} \quad (10)$$

Unlike the original MCTS that considers uniform cost (i.e., board games where each move has a cost of 1), the cost in itinerary recommendation is variable based on the selected next node (attraction). Thus, the addition of the heuristic $\frac{Int(Cat_{a_j})+Pop(a_j)}{Trav_{a_i,a_j}+Dur_{a_j}+Queue^t(a_j)}$ favours attractions with a higher interest relevance and popularity but with lower associated travelling and queuing time.

5.2 Simulation and Back-propagation

In traditional MCTS for game play, the simulation stage simulates a run from the root node to a terminal node and calculates the reward, which is either a 1 for a win or a 0 for a loss. For the purposes of itinerary recommendation, a simple binary value of 1 and 0 does not accurately reflect the rewards associated with different itineraries. Thus, we chose a reward that reflects the attraction popularity, user interest and queuing times associated with each itinerary, which is defined as:

$$Reward = \sum_{a \in I_{temp}} \left(\frac{Int(Cat_a) + Pop(a)}{Queue^t(a)}\right) \quad (11)$$

In Algorithm 1, this reward value is calculated in Line 20 after every iteration of itinerary generation, and subsequently back-propagated to our reward tree T_{reward} in Line 21. Specifically, this reward is back-propagated to all visited nodes only if the itinerary

ends at attraction a_N as specified in Constraint 5. The reason for this is so that we do not unnecessarily favour itineraries that do not satisfy our destination attraction constraint. On the other hand, we back-propagate the visit count (i.e., increment by one) to our visit tree T_{visits} (Line 18), regardless of whether the itinerary ends at our desired destination attraction a_N. As a result, nodes that are frequently visited but with low/no reward are less likely to be chosen based on our $UCT_{a_j}^{PersQ}$ formulation (Equation 10), due to the high denominator of total visit count.

6 ITINERARY RECOMMENDATION FRAMEWORK

Our framework for itinerary recommendation utilizes geo-tagged photos to derive attraction-related statistics and our PersQ algorithm for planning a personalized itinerary. This framework comprises the following steps:

(1) **Crawling of Geo-tagged Photos**. For each theme park, we used the Flickr API [45] to retrieve all geo-tagged photos that were taken within the theme park. The photos are tagged with 16 levels of geo-location accuracy and we only consider photos with the highest accuracy level of 16. Each of these photos is also tagged with the user ID of the taker, timestamp, and latitude/longitude geo-coordinates.
(2) **Mapping of Photos to Attractions**. Based on the geo-coordinates of the photos and theme park attractions, we map each photo to an attraction if they differ by <100m according to the Haversine formula [36]. If a photo is near to multiple attractions, we map this photo to the nearest attraction. At the end of this step, we have a set of attraction visits of all users to a theme park, as described in Definition 2 of Section 3.1.
(3) **Constructing Visit Sequences**. Following which, we chronologically order and join all attraction visits (from Step 2) of a single user to obtain the visit sequence of this user, as described in Definition 2 of Section 3.1.
(4) **Calculating Attraction Popularity, User Interests and Queuing Times**. Using the visit sequences from Step 3, we calculate the attraction popularity, user interests and queuing times according to Definitions 3, 4 and 5 of Section 3.1, respectively.
(5) **Recommending Personalized Itineraries With Queuing Time**. At the conclusion of Steps 1 to 4, we use the computed visit sequences, attraction popularity, user interests and queuing times to generate an itinerary recommendation based on our proposed PersQ algorithm, as described in Section 5.

One limitation of this framework is potential noise in the geo-tagged photos, which could affect the calculation of visit sequences and queuing times. An alternative is to use proprietary trajectory data from actual theme park operators, which are typically obtained via tracking devices issued to theme park visitors to monitor their trajectories (via sensors located throughout the theme park). While such datasets have high accuracy, these proprietary datasets are often not publicly available for researchers and furthermore, to construct a similar dataset for another theme park requires the installation of sensors and explicitly tracking visitors in the new theme park. As such, we utilized the above-mentioned framework that is built upon open-source data, i.e., publicly available geo-tagged photos, and can be easily extended for other theme parks or cities.

Table 2: Dataset description

Theme Park	No. of Photos	No. of Users	Attraction Visits	# Visit Sequences
Disneyland	181,735	3,704	119,987	11,758
Disney Epcot	90,435	2,725	38,950	5,816
California Adv.	193,069	2,593	57,177	6,907
Hollywood	57,426	1,972	41,983	3,858
Magic Kingdom	133,221	3,342	73,994	8,126

7 EXPERIMENTAL METHODOLOGY

7.1 Dataset

Our dataset comprises nine years of geo-tagged photos that were taken in five theme parks from Aug 2007 to Aug 2016. The five theme parks are Disneyland, Epcot, California Adventure, Disney Hollywood and Magic Kindgom. This dataset was collected using the Flickr API [45], then mapped to attraction visits and visit sequences, as described in Steps 1 to 3 of Section 6. Each attraction a_x is also assigned a category Cat_{a_x} (e.g., roller coaster, water rides, etc) based on its corresponding Wikipedia page. Similarly, each attraction a_x is also associated with a ride duration Dur_{a_x}, which can also be found on that attraction's Wikipedia page or a website like [37]. While earlier works have constructed datasets from geo-tagged photos, our dataset is the first that includes the queuing time distribution at each attraction based on these geo-tagged photos. The descriptive statistics of this dataset are shown in Table 2.

7.2 Baseline Algorithms

As there is no existing work that considers all three components of queuing time, attraction popularity and user interest, we select various state-of-the-art baselines that consider both attraction popularity and user interest for recommending personalized itineraries. These baselines are as follows:

- **Iterative Heuristic Approximation (IHA)**. A heuristic algorithm proposed in [49, 50] that commences with an itinerary from the starting attraction a_1 to destination attraction a_N and iteratively adds a new attraction between the current and destination attractions, until the budget is reached. The attraction selected to be added is the one with the maximum heuristic value of $\frac{Int(Cat_{a_j})+Pop(a_j)}{Trav_{a_i,a_j}+Dur_{a_j}}$, i.e., the next attraction with the highest profit relative to its travelling cost.
- **User Based Collaborative Filtering for Itineraries (UBCF-I)**. A variant of the popular User Based Collaborative Filtering (UCBF) [35, 47, 48] that utilizes user interest similarities (based on their ratings on items) to recommend a set of top-k items for another user. In our adaptation to itinerary recommendations, we define user ratings based on the number of posted photos on a specific attraction, hence more photos represent a higher rating. Thereafter,

we take the ranked top-k items (attractions) and iteratively add them to the starting attraction a_1 until the budget is reached, resulting in the recommended itinerary.

- **Personalized Tour Recommender (PersTour)**. An Integer Programming based algorithm proposed in [29, 31] is used, which recommends personalized itineraries that consider attraction popularity and user interest, with a variable visit duration to attractions based on a user's interest preferences. The PersTour algorithm defines user interest levels based on the length of time a user stays at attractions, relative to the average user.
- **Tour Recommendation With Interest Category (TourInt)**. The TourInt algorithm [28] formulates the tour recommendation problem with a mandatory category, which has to be visited at least once in the recommended itinerary. To personalize the recommended itinerary, TourInt defines this mandatory category as the attraction category that has been most frequently visited in the user's other visit sequences.
- **Trip Builder (TripBuild)**. We adapted the problem formulation in [3, 5] to fit our QueueTourRec problem, by adding in constraints of start and destination attractions. TripBuild recommends personalized itineraries that optimize for attraction popularity and user interest, where user interest is based on the number of times a user has visited attractions of a certain category, relative to his/her total attraction visits.

7.3 Evaluation

Like many recent itinerary recommendation works [5, 9, 29, 31, 43], our experimental evaluation uses the visit sequences of users as the ground truth of real-life visits by these users. Each visit sequence corresponds to the real-life attraction visits of users in a specific theme park, e.g., *Attraction* $a_4 \rightarrow a_{12} \rightarrow a_7 \rightarrow a_{23}$. To ensure a fair comparison with the baselines, which consider user interest, we perform our evaluation only on users with at least two visit sequences. Using these travel sequences, we then apply leave-one-out evaluation [23], where we use one visit sequence for evaluation and the other visit sequences to determine the interest preferences of this user. For each visit sequence, we use the first and last attraction of each visit sequence as input to PersQ and the baselines, along with the actual time spent in the visit sequence as the time budget. In addition, we also incorporate queuing time into the evaluation and thus attractions may be dropped from recommended itineraries due to the additional queuing time. We repeat the evaluation for all travel sequences in our dataset and compare PersQ against the various baselines using the following evaluation metrics:

(1) **Maximum Queue-time (MQ_I)**. The queuing times at attractions recommended in itinerary I, relative to their max. queuing time, defined as: $MQ_I = \frac{1}{|I|} \sum_{a \in I} \frac{Queue^t(a)}{Max_{t \in T}\left(Queue^t(a)\right)}$.

(2) **Queue:Cost Ratio (QC_I)**. The average ratio of queuing time to itinerary (time) cost of an itinerary I, defined as: $QC_I = \frac{1}{|I|} \sum_{a_i \in I, i \neq 1} \frac{Queue^t(a)}{Trav_{a_{i-1},a_i} + Dur_{a_i} + Queue^t(a_i)}$.

(3) **Queue:Popularity Ratio (QP_I)**. The average ratio of queuing time to attraction popularity of an itinerary I, defined as: $QP_I = \frac{1}{|I|} \sum_{a \in I} \frac{Queue^t(a)}{Pop(a)}$.

(4) **Popularity (Pop_I)**. The total popularity based on all attractions in an itinerary I, defined as: $Pop_I = \sum_{a \in I} Pop(a)$.

(5) **Interest (Int_I)**. The total interest alignment (to a user u) based on all attractions in an itinerary I, defined as: $Int_I^u = \sum_{a \in I} Int_u(Cat_a)$.

(6) **Recall: R_I.** The ratio of attraction visits in a user's real-life visit sequence that also exist in the recommended itinerary I. Given that A_r is the set of attractions in the recommended itinerary I and A_v is the set of attraction visits in the real-life travel sequence, we define recall as: $R_I = \frac{|A_r \cap A_v|}{|A_v|}$.

(7) **Precision: P_I.** The ratio of attractions in the recommended itinerary I that also exist in a user's real-life visit sequence. Using the same notations for A_r and A_v, we define precision as: $P_I = \frac{|A_r \cap A_v|}{|A_r|}$.

(8) **F-score (F_I)**. The harmonic mean of the precision P_I and recall R_I of an itinerary I, defined as: $F_I = \frac{2 \times P_I \times R_I}{P_I + R_I}$.

Metric 1 allows us to determine the extent to which an itinerary schedules visits to attractions at their maximum (longest) queuing times (MQ_I=1) or at their minimum (MQ_I=0), where a smaller value of MQ_I is preferred as it means we avoided attractions at their busiest time. Metric 2 measures the ratio of queuing time to the total time spent travelling to, queuing at and visiting/riding attractions, where a smaller QC_I value shows that the user spends less time queuing as part of the itinerary. Metric 3 measures the ratio of queuing time to the attraction popularity, and allows us to better differentiate between popular attractions (which are more likely to have longer queuing times) and unpopular ones with shorter queuing times. Similarly for Metric 3, a smaller value of QP_I is preferred as it indicates short queuing times at popular attractions. Metrics 4 and 5 are standard measures of itinerary popularity (Pop_I) and user-interest alignment (Int_I), respectively, while Metrics 6, 7 and 8 measure how well a recommended itinerary reflects the real-life attraction visits of a user.

8 RESULTS AND DISCUSSION

Queue-time Metrics. Table 3 gives an overview of the experimental results, in terms of the six evaluation metrics introduced in Section 7.3. PersQ outperforms all baselines with the lowest Maximum Queue-time (MQ_I), Queue:Cost Ratio (QC_I), Queue:Popularity Ratio (QP_I). In relative terms, PersQ out-performed all baselines with a reduction of 8.5% to 65.9% in Maximum Queue-time, 9.0% to 24.6% in Queue:Cost Ratio, and 21.6% to 45.1% in Queue:Popularity Ratio. These results indicate that PersQ recommends visits to popular attractions at times with the shortest queues, and constructs itineraries that include minimal time spent queuing at attractions.

Recall, Precision, F1-score. In terms of Recall (R_I) and F1-score (F_I), PersQ also out-performs all baselines for all datasets, with relative improvements of 24.2% to 45.5% for Recall, and 15.3% to 19.1% for F1-score. In terms of Precision (P_I), PersQ out-performs all baselines in 24 out of 25 cases, with a relative improvement

Table 3: Comparison between PersQ and various baselines, in terms of the mean and standard errors of Maximum Queue-time (MQ_I), Queue:Cost Ratio (QC_I), Queue:Popularity Ratio (QP_I), Popularity (Pop_I), Interest (Int_I), Recall (R_I), Precision (P_I) and F1-score (F_I). Lower values of MQ_I, QC_I and QP_I are preferred, while higher values of Pop_I, Int_I, R_I, P_I and F_I are better. The bold/blue numbers indicate the best result for each metric.

	Algorithm	Maximum Queue-time	Queue:Cost Ratio	Queue:Pop Ratio	Popularity	Interest	Recall	Precision	F1-score
Cali. Adv.	PersQ	**0.059±.002**	**0.257±.007**	**2515±121**	**1.794±.055**	**3.704±.105**	**0.483±.010**	**0.307±.007**	**0.338±.007**
	IHA	0.173±.006	0.346±.009	7669±904	1.394±.043	3.425±.083	0.332±.006	0.296±.006	0.287±.005
	UBCF-I	0.196±.007	0.330±.009	15679±1665	0.636±.025	1.530±.048	0.258±.005	0.282±.006	0.244±.004
	PersTour	0.188±.008	0.330±.011	5583±917	0.902±.033	1.423±.055	0.227±.007	0.223±.007	0.204±.006
	TourInt	0.188±.007	0.339±.011	4583±319	0.910±.033	1.444±.057	0.228±.007	0.225±.007	0.206±.006
	TripBuild	0.197±.008	0.321±.011	5182±360	0.871±.033	1.469±.055	0.225±.006	0.223±.007	0.205±.006
Hollywood	PersQ	**0.184±.005**	**0.298±.011**	**3585±335**	1.28±.053	**2.12±.088**	**0.482±.015**	**0.439±.014**	**0.431±.013**
	IHA	0.283±.009	0.423±.015	4570±223	**1.29±.044**	1.85±.070	0.367±.010	0.417±.012	0.371±.010
	UBCF-I	0.320±.012	0.410±.012	17493±1270	0.62±.029	1.09±.045	0.297±.008	0.370±.010	0.313±.008
	PersTour	0.263±.010	0.402±.013	10287±931	0.82±.033	1.23±.058	0.305±.009	0.367±.012	0.317±.010
	TourInt	0.256±.010	0.395±.013	9543±861	0.81±.033	1.22±.058	0.302±.010	0.364±.012	0.314±.010
	TripBuild	0.272±.011	0.412±.014	12144±1071	0.77±.034	1.24±.058	0.309±.010	0.374±.012	0.322±.009
Epcot	PersQ	**0.113±.004**	**0.284±.008**	**4088±152**	**0.891±.031**	2.407±.086	**0.472±.012**	**0.413±.012**	**0.407±.010**
	IHA	0.279±.006	0.344±.010	6861±285	0.852±.025	**2.511±.069**	0.380±.008	0.368±.008	0.353±.007
	UBCF-I	0.212±.007	0.331±.010	11664±823	0.500±.017	1.232±.044	0.291±.007	0.314±.008	0.283±.006
	PersTour	0.250±.008	0.356±.011	8034±360	0.604±.021	1.490±.060	0.310±.008	0.322±.009	0.297±.007
	TourInt	0.252±.007	0.352±.011	8112±340	0.599±.021	1.459±.058	0.317±.008	0.327±.009	0.303±.007
	TripBuild	0.246±.008	0.342±.011	8208±386	0.584±.021	1.561±.060	0.300±.008	0.312±.009	0.287±.008
Disneyland	PersQ	**0.161±.004**	**0.263±.005**	**5673±263**	**0.731±.017**	2.287±.055	**0.332±.006**	0.295±.006	**0.289±.005**
	IHA	0.198±.004	0.332±.006	13437±419	0.690±.020	**3.531±.068**	0.267±.004	0.270±.004	0.249±.003
	UBCF-I	0.176±.004	0.378±.007	14986±1072	0.495±.012	1.567±.037	0.235±.004	**0.296±.005**	0.240±.004
	PersTour	0.177±.005	0.322±.008	8255±579	0.623±.017	1.523±.045	0.190±.004	0.201±.005	0.180±.004
	TourInt	0.193±.005	0.313±.008	10267±1324	0.612±.017	1.498±.044	0.188±.004	0.200±.005	0.179±.004
	TripBuild	0.193±.005	0.289±.008	10601±1634	0.580±.017	1.454±.043	0.176±.004	0.181±.005	0.165±.004
Magic King.	PersQ	**0.132±.003**	**0.244±.005**	**4275±133**	**0.901±.025**	**3.330±.088**	**0.440±.008**	**0.326±.007**	**0.343±.006**
	IHA	0.240±.004	0.321±.008	6885±329	0.772±.017	3.159±.064	0.305±.005	0.312±.006	0.288±.004
	UBCF-I	0.206±.005	0.323±.007	11328±424	0.474±.012	1.389±.038	0.265±.004	0.304±.006	0.261±.004
	PersTour	0.208±.006	0.303±.010	8822±494	0.486±.016	1.305±.047	0.202±.005	0.201±.006	0.186±.005
	TourInt	0.195±.006	0.316±.010	9410±531	0.490±.017	1.311±.048	0.200±.005	0.200±.006	0.185±.005
	TripBuild	0.191±.006	0.287±.009	9074±456	0.479±.017	1.393±.048	0.193±.005	0.189±.006	0.177±.005

of up to 5.3%. The only exception is for the Disneyland dataset, where PersQ out-performs all baselines but under-performs UBCF-I by less than 0.34% in terms of Precision. Moreover, we are more interested in the F1-score, as it considers both precision and recall, and PersQ out-performs all baselines by at least 15.3% in terms of F1-score. These results indicate that itineraries recommended by PersQ are highly representative of the real-life visits of these users.

Popularity and Interests. In terms of Attraction Popularity (Pop_I) and User Interests (Int_I), PersQ also offers the best overall performance, while IHA offers the second best performance. PersQ has the highest Popularity scores for four of five theme parks and the highest Interest scores in three of five theme parks, while IHA leads in Popularity and Interest for one and two theme parks, respectively.

Discussion. The superior performance of PersQ is due to its three-fold consideration of attraction popularity, user interests and queuing times, which is automatically determined from geo-tagged photos. In contrast, the various baselines only consider attraction popularity and user interests, and hence may recommend visits to attractions that are popular and interesting but with excessive queuing times. This excessive queuing time consumes a large portion of the touring time, as indicated by the high QC_I scores, thus causing attractions later in the itinerary to be missed. Apart from PersQ, IHA is also able to offer a relatively good performance in terms of attraction popularity and user interests due to its use of the profit (popularity and interest) over cost (travelling time) heuristic.

9 CONCLUSION AND FUTURE WORK

In this paper, we proposed the QueueTourRec problem of recommending personalized itineraries of popular and interesting attractions, while minimizing queuing times. QueueTourRec is an NP-hard problem that includes time-dependent queuing times, which we then solve using our proposed PersQ algorithm that is adapted from MCTS. For determining queuing times, we also implemented a framework that utilizes geo-tagged photos to determine the distribution of queuing times at each attraction, as well as the attraction popularity and user interest preferences. We evaluated PersQ on a dataset of five major theme parks and show that PersQ out-performs the state-of-the-art in terms of maximum queuing times, queuing time to itinerary cost ratio, queuing time to attraction popularity ratio, attraction popularity, user interest alignment, recall, precision and F1-score.

In future, we intend to address other aspects of the QueueTourRec problem, such as: (i) addressing the cold-start problem by using social ties of users to solicit user interest; (ii) accounting for the effects of holidays, special events, weather and other uncertainties on attraction operations; (iii) developing a game-theoretic approach to itinerary recommendation that optimizes for global queuing times and crowdedness for the entire population of theme park visitors; (iv) incorporating a strict attraction category constraint for various user demographics (e.g., no dark rides for kids).

Acknowledgments. This work was supported in part by Data61, a Google PhD Fellowship in Machine Learning, a Google PhD Travel Scholarship and an ACM SIGIR Student Travel Grant. The authors thank the anonymous reviewers for their useful comments.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47, 2-3 (2002), 235–256.

[2] Avrim Blum, Shuchi Chawla, David R. Karger, Terran Lane, Adam Meyerson, and Maria Minkoff. 2007. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM J. Comput.* 37, 2 (2007), 653–670.

[3] Igo Brilhante, Jose Antonio Macedo, Franco Maria Nardini, Raffaele Perego, and Chiara Renso. 2013. Where shall we go today? Planning touristic tours with TripBuilder. In *Proc. of CIKM'13*. 757–762.

[4] Igo Brilhante, Jose Antonio Macedo, Franco Maria Nardini, Raffaele Perego, and Chiara Renso. 2014. TripBuilder: A Tool for Recommending Sightseeing Tours. In *Proc. of ECIR'14*. 771–774.

[5] Igo Ramalho Brilhante, Jose Antonio Macedo, Franco Maria Nardini, Raffaele Perego, and Chiara Renso. 2015. On planning sightseeing tours with TripBuilder. *Information Processing & Management* 51, 2 (2015), 1–15.

[6] Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Trans. on Computational Intel. and AI in Games* 4, 1 (2012), 1–43.

[7] Raymond C. Browning, Emily A. Baker, Jessica A. Herron, and Rodger Kram. 2006. Effects of obesity and sex on the energetic cost and preferred speed of walking. *Journal of Applied Physiology* 100, 2 (2006), 390–398.

[8] Luis Castillo, Eva Armengol, Eva Onaindía, Laura Sebastiá, Jesús González-Boticario, Antonio Rodríguez, Susana Fernández, Juan D. Arias, and Daniel Borrajo. 2008. SAMAP: An user-oriented adaptive system for planning tourist visits. *Expert Systems with Applications* 34, 2 (2008), 1318–1332.

[9] Dawei Chen, Cheng Soon Ong, and Lexing Xie. 2016. Learning Points and Routes to Recommend Trajectories. In *Proc. of CIKM'16*. 2227–2232.

[10] Munmun De Choudhury, Moran Feldman, Sihem Amer-Yahia, Nadav Golbandi, Ronny Lempel, and Cong Yu. 2010. Automatic construction of travel itineraries using social breadcrumbs. In *Proc. of HT'10*. 35–44.

[11] Munmun De Choudhury, Moran Feldman, Sihem Amer-Yahia, Nadav Golbandi, Ronny Lempel, and Cong Yu. 2010. Constructing travel itineraries from tagged geo-temporal breadcrumbs. In *Proc. of WWW'10*. 1083–1084.

[12] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proc. of CCG'06*. 72–83.

[13] Marco Dorigo, Mauro Birattari, and Thomas Stützle. 2006. Ant colony optimization. *IEEE Computational Intelligence Magazine* 1, 4 (2006), 28–39.

[14] Inma Garcia, Laura Sebastia, and Eva Onaindia. 2011. On the design of individual and group recommender systems for tourism. *Expert Systems with Applications* 38, 6 (2011), 7683–7692.

[15] Aristides Gionis, Theodoros Lappas, Konstantinos Pelechrinis, and Evimaria Terzi. 2014. Customized tour recommendations in urban areas. In *Proc. of WSDM'14*. 313–322.

[16] Bruce L. Golden, Larry Levy, and Rakesh Vohra. 1987. The Orienteering problem. *Naval Research Logistics* 34, 3 (1987), 307–318.

[17] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. 2016. Orienteering Problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research* 255, 2 (2016), 315–332.

[18] Aldy Gunawan, Zhi Yuan, and Hoong Chuin Lau. 2014. A Mathematical Model and Metaheuristics for Time Dependent Orienteering Problem. In *Proc. of PATAT'14*. 202–217.

[19] Hsun-Ping Hsieh, Cheng-Te Li, and Shou-De Lin. 2012. TripRec: recommending trip routes from large scale check-in data. In *Proc. of WWW'12*. 529–530.

[20] Astrid S. Kenyon and David P. Morton. 2003. Stochastic vehicle routing with random travel times. *Transportation Science* 37, 1 (2003), 69–8.

[21] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *Proc. of ECML'06*. 282–293.

[22] Levente Kocsis, Csaba Szepesvári, and Jan Willemson. 2006. *Improved monte-carlo search*. Technical Report. University of Tartu, Institute of Computer Science.

[23] Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proc. of IJCAI'95*. 1137–1145.

[24] Hoong Chuin Lau, William Yeoh, Pradeep Varakantham, Duc Thien Nguyen, and Huaxing Chen. 2012. Dynamic Stochastic Orienteering Problems for Risk-Aware Applications. In *Proc. of UAI'12*. 448–458.

[25] Kenneth Wai-Ting Leung, Dik Lun Lee, and Wang-Chien Lee. 2011. CLR: a collaborative location recommendation framework based on co-clustering. In *Proc. of SIGIR'11*. 305–314.

[26] Xun Li. 2013. Multi-day and multi-stay travel planning using geo-tagged photos. In *Proc. of GeoCrowd'13*. 1–8.

[27] Xutao Li, Gao Cong, Xiao-Li Li, Tuan-Anh Nguyen Pham, and Shonali Krishnaswamy. 2015. Rank-GeoFM: a ranking based geographical factorization method for point of interest recommendation. In *Proc. of SIGIR'15*. 433–442.

[28] Kwan Hui Lim. 2015. Recommending Tours and Places-of-Interest based on User Interests from Geo-tagged Photos. In *Proc. of SIGMOD'15 PhD Symposium*. 33–38.

[29] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2015. Personalized Tour Recommendation based on User Interests and Points of Interest Visit Durations. In *Proc. of IJCAI'15*. 1778–1784.

[30] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2016. Towards Next Generation Touring: Personalized Group Tours. In *Proc. of ICAPS'16*. 412–420.

[31] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2017. Personalized Trip Recommendation for Tourists based on User Interests, Points of Interest Visit Durations and Visit Recency. *Knowledge and Information Systems* (2017), In Press.

[32] Clair E. Miller, Albert W. Tucker, and Richard A. Zemlin. 1960. Integer programming formulation of traveling salesman problems. *J. ACM* 7, 4 (1960), 326–329.

[33] Edward J. Powley, Daniel Whitehouse, and Peter I. Cowling. 2013. Monte Carlo tree search with macro-actions and heuristic route planning for the multiobjective physical travelling salesman problem. In *Proc. of CIG'13*. 1–8.

[34] Ioannis Refanidis, Christos Emmanouilidis, Ilias Sakellariou, Anastasios Alexiadis, Remous-Aris Koutsiamanis, Konstantinos Agnantis, Aimilia Tasidou, Fotios Kokkoras, and Pavlos S. Efraimidis. 2014. myVisitPlanner GR: Personalized Itinerary Planning System for Tourism. In *Proc. of SETN'14*. 615–629.

[35] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *Proc. of the CSCW'94*. 175–186.

[36] Roger W. Sinnott. 1984. Virtues of the Haversine. *Sky and Telescope* 68, 158 (1984).

[37] TouringPlans.com. 2016. Disney's Hollywood Studios Attraction Durations. (2016). https://touringplans.com/hollywood-studios/attractions/duration.

[38] Theodore Tsiligirides. 1984. Heuristic methods applied to Orienteering. *Journal of the Operational Research Society* 35, 9 (1984), 797–809.

[39] UNWTO. 2016. United Nations World Tourism Organization (UNWTO) Annual Report 2015. (2016). http://www2.unwto.org/annual-reports.

[40] Pieter Vansteenwegen, Wouter Souffriau, Greet Vanden Berghe, and Dirk Van Oudheusden. 2011. The city trip planner: An expert system for tourists. *Expert Systems with Applications* 38, 6 (2011), 6540–6546.

[41] Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. 2011. The Orienteering problem: A survey. *Euro. J. of Operational Rsch.* 209, 1 (2011), 1–10.

[42] Pradeep Varakantham and Akshat Kumar. 2013. Optimization approaches for solving chance constrained stochastic orienteering problems. In *Proc. of ADT'13*.

[43] Xiaoting Wang, Christopher Leckie, Jeffery Chan, Kwan Hui Lim, and Tharshan Vaithianathan. 2016. Improving Personalized Trip Recommendation to Avoid Crowds Using Pedestrian Sensor Data. In *Proc. of CIKM'16*.

[44] Wolfgang Wörndl and Alexander Hefele. 2016. Generating Paths Through Discovered Places-of-Interests for City Trip Planning. In *Information and Communication Technologies in Tourism*. Springer International Publishing, 441–453.

[45] Yahoo. 2016. Flickr API. (2016). https://www.flickr.com/services/api/.

[46] Lina Yao, Quan Z. Sheng, Yongrui Qin, Xianzhi Wang, Ali Shemshadi, and Qi He. 2015. Context-aware Point-of-Interest Recommendation Using Tensor Factorization with Social Regularization. In *Proc. of SIGIR'15*. 1007–1010.

[47] Mao Ye, Peifeng Yin, Wang-Chien Lee, and Dik-Lun Lee. 2011. Exploiting geographical influence for collaborative point-of-interest recommendation. In *Proc. of SIGIR'11*. 325–334.

[48] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat Thalmann. 2013. Time-aware point-of-interest recommendation. In *Proc. of SIGIR'13*.

[49] Chenyi Zhang, Hongwei Liang, and Ke Wang. 2016. Trip Recommendation Meets Real-World Constraints: POI Availability, Diversity, and Traveling Time Uncertainty. *ACM Transactions on Information Systems* 35, 1 (2016), 5.

[50] Chenyi Zhang, Hongwei Liang, Ke Wang, and Jianling Sun. 2015. Personalized Trip Recommendation with POI Availability and Uncertain Traveling Time. In *Proc. of CIKM'15*. 911–920.